Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla
Avda Reina Mercedes, s/n. 41012 SEVILLA
Fax : 95 455 71 39. **Tlf**: 95 455 71 39. **E-mail**: lsi@lsi.us.es

A MDA proposal to integrate the measurement lifecycle Into the process lifecycle

"Una propuesta basada en el paradigma dirigido por modelos para la integración del ciclo de vida de la medición al ciclo de vida del proceso"

A DISSERTATION SUBMITTED TO

UNIVERSITY OF SEVILLA IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE

March|2019

By

Ayman Meidan

Advisors: Dr. Julián Alberto García García
Dr. María José Escalona Cuaresma
Dr. Isabel Ramos Roman

Acknowledgments

In the first place, I would like to thank my Ph.D. directors, Julián A. García-García, María José Escalona, and Isabel Ramos, who guided me through my research, encouraged me in difficult times and spent numerous hours proof-reading my manuscripts. This research would not have been possible without you. You have greatly influenced my skill in software engineering and research methodology.

I would like to thank my family for their unlimited patience, understanding, and love which motivated me to complete this work. I would like to thank my friends, for being always there and available to listen and advice. Lastly, I would also like to thanks all the colleagues of the research group (IWT2) and the department of computer languages and information systems for all their support throughout the study period.

Contents

Tables

Figures

ABSTRACT

Context: Measurement enables organizations to gain knowledge about its processes and projects, also to reach predictable performance and high capability processes, which places organizations in better positions to make appropriate decisions. Measuring the software development process supports organizations in its endeavor to understand, evaluate, manage, and improve its development processes and projects. In the last decades, the software development process has evolved to meet the market needs and to keep abreast of modern technologies and infrastructures that have influenced the product development and its use. These changes in the development processes have increased the importance of the measurement and caused changes in the measurement process and the used measures.

Objective: This thesis aims to contribute to the software process measurement domain in two main aspects; first, propose a novel solution to support the identification and the operational definition of the measurement concepts and objectives. The second is defining a measurement lifecycle and integrate it into the process lifecycle.

Method: We have carried out a survey and mapping study to understand the current state of the art, and to identify existing gaps. After that, we have proposed a theoretical solution to support the software process measurement, and finally, we have developed this solution to allow its practical use in real environments, enabling its application and evaluation in a real project.

Results: The proposed solution consists of three main components: (i) Measurement lifecycle; which define the measurement activities throughout the process lifecycle, (ii) Measurement metamodels; these metamodels support the measurement lifecycle and its integration into the process lifecycle, (iii) Transformation process; which allow the derivation of the necessary measurement models, artifacts, and activities throughout the process lifecycle.

Conclusion: The solution presented in this dissertation allows organizations to manage and improve their processes and projects; the proposed information model supports the unification of the measurement concepts vocabulary, coherently connects them, and ensures the traceability between these concepts. The defined measurement process lifecycle provides a clear and comprehensive guide for the organizations to establish the measurement objectives and carry out the necessary activities to achieve them. The proposed measurement definition metamodel support and guide the engineers to define the measurement concepts and their relationships completely and operationally. Moreover, the proposed transformations use this metamodel to support the measurement process and to derive the necessary measurement artifacts and activities throughout the process lifecycle.

RESUMEN

Contexto: la medición permite a las organizaciones obtener conocimiento sobre sus procesos y proyectos, también alcanzar un rendimiento predecible y procesos de alta capacidad, lo que pone las organizaciones en mejores posiciones para tomar decisiones apropiadas. La medición del proceso de desarrollo de software apoya a las organizaciones en su esfuerzo para comprender, evaluar, gestionar y mejorar sus procesos y proyectos de desarrollo.

Objetivo: Esta disertación propone una solución novedosa para respaldar la identificación y la definición de los conceptos y objetivos de medición en una forma operativa. Además, busca definir un ciclo de vida de la medición e integrarlo en el ciclo de vida del proceso.

Método: Hemos llevado a cabo una encuesta y estudios de mapeo para comprender el estado del arte e identificar brechas existentes. Posteriormente, hemos propuesto una solución teórica para respaldar la medición del proceso del software y, finalmente, hemos desarrollado esta solución para permitir su uso práctico en entornos reales, permitiendo su aplicación y evaluación en un proyecto real.

Resultados: La solución propuesta consta de tres componentes principales: (i) Ciclo de vida de la medición; que define las actividades de medición a lo largo del ciclo de vida del proceso, (ii) Metamodelos de medición; estos metamodelos apoyan el ciclo de vida de la medición y su integración en el ciclo de vida del proceso, (iii) Proceso de transformación; que permite la derivación de los modelos de medición, artefactos y actividades necesarios a lo largo del ciclo de vida del proceso.

Conclusión: la solución presentada en este trabajo permite a las organizaciones gestionar y mejorar sus procesos y proyectos; El modelo de información propuesto apoya la unificación del vocabulario de los conceptos de medición, los conecta de forma coherente y garantiza la trazabilidad entre estos conceptos. El ciclo de vida del proceso de medición proporciona una guía clara y completa para que las organizaciones establezcan los objetivos de medición y realicen las actividades necesarias para lograrlos. El metamodelo de definición de la medición apoya y guía a los ingenieros para definir los conceptos de medición y sus relaciones de manera completa y operativa; además, las transformaciones propuestas utilizan este metamodelo para respaldar el proceso de medición y derivar los artefactos y las actividades de medición necesarios durante el ciclo de vida del proceso.

Chapter I Introduction

This chapter describes the context of this work. For this, the first section presents a brief introduction to the research domain of the thesis. Next section describes the structure of this document, and finally, this chapter concludes with a summary.

1. Introduction

Defining and improving the development process is one of the most important strategies used by organizations to enhance productivity and improve the quality of the developed software. The development process is the primary guide for the management of the work teams and the production process. It is also used as a basis for project planning and monitoring. Defining, monitoring and improving the software development process (SDP) aims to produce high-quality software products and more predictive and productive projects.

Software development is considered to be comprised of three essential components: products, processes, and resources (N. E. Fenton 1991). Developing software is a long, costly and complex process. The outcome of this process is not only the final product but the production of many intermediate and supplementary artifacts during the development endeavor. The quality of this development process significantly impacts the quality of the resulting product (Cugola and Ghezzi 1998; Fuggetta 2000; Barbara Kitchenham and Pfleeger 1996).

In the last decades, the SDP has evolved to meet the market needs and to keep abreast of modern technologies and infrastructures that have influenced the product development and its use. These changes in the development processes have increased the importance of the measurement (Bourgault et al. 2002) and caused changes in the measurement process and the used measures (Tihinen et al. 2012).

For instance, cloud computing allowed to merge the software development, deployment, and operation in what is known as DevOps. Measurement is one of the four DevOps perspectives (Collaboration culture, automation, measurement, and sharing) (Bang et al. 2013). In this context, measurement promotes the communication and the common understanding between development and operations. On the other side, today's software is increasingly developed by teams working in different geographic locations, time zones, and cultures. Management of these kinds of projects is more challenging and complicated than traditional on-site development. The measurement is an essential element for the success of these development projects (Tihinen et al. 2012).

These evolutions in the development process, technologies and infrastructures create new challenges and obstacles for the measurement regarding data collection, storage,

analysis, interpretation and decision-making based on the measurement results. These challenges and difficulties emphasize the importance of the measurement in the context of the SDP.

Measuring the SDP and its outcomes is the only way to gain knowledge about them. Besides, the obtained measurements could be used in models for prediction purposes (Lennselius, Wohlin, and Vrana 1987). Moreover, software process measurement provides support for better understanding, evaluation, and control of the development process, project and the resulting product (Ebert et al. 2007). Measurement also enables organizations to have insight into its processes, predict, and improve its quality and performance, which give organizations a better position to make appropriate and informed decisions as early as possible during the development process (Abreu Fernando Brito and Carapuça 1994; García et al. 2006).

Software process measurement is essential in controlling the development process, verifying that it is performing expectedly, satisfying the requirements with its outcome, supporting the organizations in its endeavor to achieve predictable performance and higher capability, as well as improving the processes in a continuous and effective manner (Florac and Carleton 1999; Paulish and Carleton 1994). In high maturity organizations, measurement is used to support the management's decisions and goals (Chrissis, Konrad, and Shrum 2011).

The Capability Maturity Model (CMM) (Paulk et al. 1994), the Capability Maturity Model Integration (CMMI) (Ahern, Clouse, and Turner 2004) (Team 2002) and ISO/IEC 15504 (also known as SPICE) (ISO/IEC 15504-5:2006, Information Technology - Process Assessment - Part 1-5: 2006) (Emam, Melo, and Drouin 1997) aim to support organizations in improving continuously their maturity and capability (Boehm 2006) in order to improve its productivity and competitiveness. The constant improvement of software processes makes them more efficient and increases product quality.

The effective and quantitative management of the software processes promotes and support the organization efforts towards continuous improvement. Therefore, introducing and integrating the process measurement into the software process is essential for its quantitative management, assessment, and continuous improvement. Table I.1 demonstrates the relationship between measurement and management.

Table I.1. Measurement, management and improvement (Bellini, Pereira, and Becker 2008).

«What is correctly measured is correctly managed (Feigenbaum 2001); moreover, without constant measurement there is no process management, and, with no process management, there are no improvements (Avison and Fitzgerald 1999; Gardner 2001). In other words, management needs the measurement for being accurate, but measurement needs management for having a purpose (Bourque et al. 1999).»

After presenting the essential role, measurement plays in software development. The following section provides an overview of the solution proposed in this work.

The work presented in this dissertation is part of a series of thesis works whose objective is to improve different aspects of process management throughout the process lifecycle. The context of this proposal and its relation to previous theses are detailed in chapter III.

The proposal presented in this thesis aims to use the Model-Driven Engineering (MDE) paradigm (Schmidt 2006) to integrate the measurement process into the process lifecycle in the way that allows the definition and modeling of the process measures explicitly and operationally during the process modeling phase. It also aims to use the MDE transformations to derive the measurable process execution model from the process definition model.

The result of this thesis is a theoretical solution guided by models to improve the measurement of the software processes, as well as the design and development of a support software tool that allows the application of this theoretical solution in practical environments. Achieving this goal requires the following steps:

In the beginning, investigate the measurement process (e.g., inputs, activities, artifacts and outputs) and link it with the development process lifecycle. Then, identify and define the necessary measurement concepts to support the measurement process throughout its lifecycle.

The second step includes integrating the measurement lifecycle into the process lifecycle by defining the measurement activities, roles, and outputs to be performed and produced in each phase of the process lifecycle.

The third step consists of using the MDE paradigm to support the integrated management of both lifecycles by developing a metamodel to define the measurement concepts in order to link this definition with the process metamodel. This link allows process modeler to model the measurement concepts together with the process elements. Furthermore, using the MDE transformations to derive the measurement execution model from the measurement definition model (MDM) and using it to enrich the process execution model with the measurement concerns in order to consider the measurement subjects during the process execution.

And finally, develop the proposed solution in the form that allows its use in real environments. The feedback from these experiences is evaluated to validate and improve the proposed framework.

Before going through this thesis, it is convenient to introduce the main concepts related to the measurement domain.

Measurement is the process that assigns values, according to specific rules, to describe the properties of entities (N. E. Fenton and Pfleeger 1996). In software engineering, the entity might be an object or event (e.g., program or project). Entities are commonly classified into three categories: products, processes, and resources (N. E. Fenton 1991). An attribute is a measurable characteristic of an entity (e.g., quality of a program or cost of a project, and the experience of the development staff). Two types of attributes can be distinguished: internal and external (Shepperd 1995). Internal attributes

are those which can be measured based on the entity itself, such as the elapsed time of the development process and the project size. On the other hand, the measurement of external attributes such as product reliability and resource productivity depends on the measured entity and its environment.

Software measurement involves broad activities; the measurement process is comprised of determining the required data, designing measurement methods and plans, applying the measurement methods, using tools and techniques to gather and analyze the data and exploiting measurement results.

Software engineering aims to ensure that software development produces the expected outcomes within the estimated cost and schedule (Sommerville 2004). Accomplishing this objective is the main concern of software process engineering, through the provision of improved approaches and methodologies to better conduct the development process.

Software process - defined in Table I.2- provides a unified environment that integrates the production and management activities. This integration controls and improves the information flow by which the management activities control the production activities (Ruiz-gonzález and Canfora 2004).

Table I.2. Software process (Fuggetta 2000).

«Coherent set of policies, organizational structures, technologies, procedures and artifacts that are needed to conceive, develop, deploy, and maintain a software product»

Defining and executing the software process as a sequence of work activities allows the measurement, control, and improvement of this process. When the execution consistently follows the process definition, the expected outcomes will be produced within the planned quality, cost and time. What allows the prediction of the performance of the process and the results, and keeps it under statistical control. When the process is under statistical control, every process enactment will produce approximately the same results. In this context, the only way to improve the process outcome is by improving the process itself. It is not possible to obtain sustained improvement until the process is under statistical control (Watts S. Humphrey 1989; Watts S Humphrey 1988).

Measuring the software process is essential to monitor, control, assess, manage and improve it. Besides, it is also relevant to gain insight into the process to early detect problems and identify risks, in addition to facilitating communication and enabling objective planning, estimation, decision-making processes, and improvements (Jethani 2013).

There are two types of process measurement: in-process measures, which are carried out within the process stage, and post-activity measures, which measure the effectiveness by which the customer expectations were met at the end of the process. (Bhide 1990).

Process Modeling is an approach to defining the process, by describing the involved activities, entities, data, resources, work sequence, work-performers, etc., and the relationship between them(Curtis, Kellner, and Over 1992; Gill 1999). Due to the

complexity of the SDP, the use of process modeling has become increasingly important, also recognized as a good software engineering practice (Alegría et al. 2010) to describe, document and communicate the process elements, policies, and structure. Besides that, the process model ensures the common understanding of the process among all shareholders (e.g., domain experts, process analysts, and software developers).

Furthermore, the process model could be used as a reference to produce the project plan, support the process management and improvement also support its automated execution (Curtis, Kellner, and Over 1992).

Due to the importance and widespread use of the process modeling, there are many process modeling languages in the literature such as the languages proposed by Chou (Chou 2002), Di Nitto (Nitto et al. 2002), xSPEM (Bendraou et al. 2007) and PLM_4BS (Garcia-Garcia et al. 2017). There are also industrial standard languages such as SPEM 2.0 (Object Management Group 2002b) and BPMN 2.0 (Object Management Group 2011).

These languages define the main process elements (e.g., activities, sequence flow, performer, etc.). Our review of these languages and standards revealed that there is a lack of the definition and integration of measurement concepts into the process lifecycle(Hill et al. 2006). For example, these languages do not support the operational definition of the measurement concepts and objectives.

2. Thesis Structure

After this introduction, the dissertation continues describing how the rest of the thesis is organized into chapters.

Chapter II This chapter presents an overview of the state of the art of the thesis topic, also discusses the related works to this thesis. On the one hand, describes the previous research performed by the Ph.D. student to understand the current state of the art, and to identify existing gaps; this section comprises a survey on business processes management suites and a mapping study related to the measurement in the context of the SDP. On the other hand, the related proposals section discusses the main proposals related to this thesis.

Chapter III provides a detailed description of the problem, motivation, influences, and the thesis objectives. It also provides a brief description of the main components of the proposed solution. Besides, it introduces the proposed measurement lifecycle and concepts.

Chapter IV introduces the proposed metamodels to support the measurement throughout the process lifecycle; The Measurement Definition Metamodel, support the operational definition of the measurement concepts proposed in the previous chapter. The measurement execution metamodel, support the measurement issues during the process execution. And finally, the monitoring metamodel, which allows monitoring the measurement objectives to support the process management and improvement.

Chapter V This chapter starts by providing an overview of the MDE transformation concepts, and the transformation languages used in this work. After that, analyzes the relationships between the proposed metamodels. And finally, describes the transformation process and the required rules to derive the necessary artifacts (e.g., models, code, and documentation) to support the measurement process throughout its lifecycle.

Subsequently, **chapter VI** begins by describing the influences that affected the development and validation of the proposal presented in this thesis. Then, introduces the project through which we have validated the proposed solution. After that, describes how we have developed and integrated the proposal into the project tool to enable its practical use. Next, explains how the project team used the solution during the project. And finally, describe the results of this experience.

Later, **chapter VII** offers an overview of the research framework in which this work is carried out. After that, it describes the fundamental contributions of this thesis. Then, presents the principal future works and outline the main relationships established during this work with other research groups and the business sector. And finally, states the final conclusion of this research.

Finally, this document concludes with two appendixes; **appendix A**, which list a glossary of the terms used throughout this document, and **appendix B,** which enumerate the publications of the doctoral student.

3. Chapter summary

This chapter presents an overview of the main concepts related to the thesis domain and highlights the need to define and model the software process. And finally, reveals the essential role that measurement plays in software development.

Besides, this chapter states the main problem of this thesis work which is the lack of defining the measurement concepts and integrating the measurement issues into the process lifecycle as the thesis problem.

Furthermore, this chapter mentions the main objectives of this research; on the one hand, supporting the operational definition of the measurement concepts and integrating it into the process lifecycle in the form that allows the process designer and modeler to define and model the process measures operationally and explicitly within the process definition model. On the other hand, derive the process execution model considering the measurement concepts defined in the process definition model.

Finally, it presents the organization of this dissertation with a brief description of the content of each chapter.

Chapter II Related Work

This chapter is divided into two sections: the first presents the previous research carried out by the author to comprehend the current state of the art and to discover the existing gaps in the domain; this section comprises a survey on the process management suites and a mapping study on the measurement in the context of the software process. The second section addresses the existing proposals related to this work, the modeling languages, and tools. Also, describes the existing methods for selecting and defining the measures, furthermore, explains the process lifecycles and measurement lifecycles found in the literature.

1. Understand the current state of the art

This section describes the previous research performed by the Ph.D. student to understand the current state of the domain. The first section presents a survey on business processes management suites, the second research describes a systematic mapping study of the measurement in the context of the SDP, and the third section provides the main conclusions obtained from the previous reviews.

1.1 A survey on business processes management suites

This section describes a formal survey on the existing open source Business Process Management Suites (BPMS) (A. Meidan et al. 2016).

This survey aims to investigate to which extent the existing BPMSs support the process lifecycle. Also, provide a guide for the organizations to plan and perform a comparative on the existing BPMSs. Which allow them to discover which BPMS best meet their process management needs in term of defining and modeling the process elements, executing its activities and producing planned outputs, defining measures and indicators to monitor process execution, as well as using the monitoring data to support the process management and improvement.

Through this survey the researchers tried to answer the following research questions:

RQ1. What open source BPMS are available and what do they offer?
RQ2. What areas of improvement are needed for the selected BPMS?

To answer the first question (RQ1), the authors have elaborated a quality assessment (QA) questions and characterization scheme to support the search and selection of the BPMSs; these quality questions are described below:

QA1: Has the BPMS maker provided a new version of the BPMS in the past year (July 1st, 2014 to July 1st, 2015)? (It is necessary to evaluate the BPMSs whose line of development has not died to provide an actual comparative study useful for organizations)

- Yes, they have.
- No, they have not.

QA2: Do the BPMS have an active user community and official support? (Having alive community within the context of open source tools is relevant) (Von Krogh, Spaeth, and Lakhani 2003))

- Yes, it does. BPMS has had a forum with an active community for three months.
- No, it does not.

QA3: Does the BPMS maker provide sufficient documentation (manuals, videos or examples) about the BPMS?

- Yes, they do.
- No, they do not.

In order to answer the **RQ2,** the selected BPMSs were analyzed using a quality model to discover weaknesses and suggest improvements**.**

The survey is based on the proposed quality model which helps organizations and researchers to choose the best BPMS that meets their necessities. It also allows researchers to identify the current characteristics of the existent BPMS, which could guide future investigations.

This quality model has been implemented using a characterization scheme which focuses on the most marked criteria in the literature. These characteristics were grouped in sections according to the process lifecycle stages proposed by Hill's (Hill et al. 2006). Therefore, the researchers have divided the 41 characteristics into these phases: Modeling; Design; Deployment; Execution and Operation; Monitoring and Control; and Analysis. The following section details the quality model used in this survey.

Modeling phase. In this phase, the organization uses a formal language to define and describe the process. This model describes different process perspectives (functional, flow control, information and organizational perspective), which allows stakeholders (e.g., domain experts, process engineers, and end users) to understand the process without ambiguity (W. van der Aalst, ter Hofstede, and Weske 2003).

There are many process modeling languages available, such as BPMN (Business Process Modeling Notation) (Object Management Group 2011), UML (Unified Modeling Language) and its activity diagrams (Seemann, Gudenberg, and Jürgen 1998), and YAWL (Yet Another Workflow Language) (W.M.P. van der Aalst and ter Hofstede 2005). In this sense, interoperability among the different languages is a critical BPMS aspect; interoperability allows the interchange (import/export) of business process models between different BPMSs (Smith and Fingar 2003). Thus, interoperability support is a key criterion in the evaluation of the BPMSs.

Defining business rules is another important aspect of BPMS; through which the business goals restrict the process. These business rules could be embedded in the process definition or defined using Business Rule Management Systems (Sinur and Hill 2010).

The possibility to reuse the process is also another crucial criterion for BPMS. Reuse the process (or part of it) facilitate the business process modeling task in two ways:(i) it

improves the quality of the models through the reuse of established and optimized artifacts, (ii) it reduces the modeling time by avoiding modeling the same business process (BP) or part of it multiple times (Markovic and Pereira 2008).

Moreover, defining Process Performance Indicators (PPIs) is the next important step after describing the process model. PPIs constitute a relevant instrument to evaluate process performance and can be considered the first step to carry out a continuous process improvement (del-Río-Ortega, Resinas, and Ruiz-Cortés 2010). Consequently, a BPMS must allow defining and monitoring PPIs to carry out further analysis. The ability to establish PPIs (and the ease of use for end users) is another important criterion in the evaluation of BPMS.

The ability to generate process documentation is one of the key objectives of BPMS (Giaglis 2001) because it is essential for the communication among stakeholders (Bandara, Gable, and Rosemann 2005). In this sense, it is substantial to evaluate if BPMS allows the generation of the documentation that reflects the different perspectives of the process, such as documentation of activities, roles, and information flow.

Design phase. The design phase is necessary when the organization implements process models using IT (Information Technology) support. For this purpose, it is crucial to design and implement several features such as user interfaces, interactions with other systems and translations to execution models, among others. The characteristics used to evaluate this phase are discussed below.

On the one hand, the end users interact with the process and its data using user interfaces. In this sense, it is essential to assess if BPMS provide mechanisms to define and implement these interfaces in a friendly way (e.g., using separate applications or integrated modules).

BPMS may support various programming languages to encourage designers to create and modify the user interface and implement interfaces to communicate with other organizational services.

Another important criterion is the user management which is necessary to define and control the access privileges of each role in the process execution; this could be done by allowing the definition of the organizational structure within the BPMS or by importing the structure of roles from an existing system like LDAP.

Besides, in the design phase, the organization could link service level agreements with its process performance indicators to facilitate the monitoring and analysis of the process enactment.

On the other hand, implementing exception handling is an essential task in this phase to avoid any unexpected behaviour during the process execution. Exception handling could be implemented at several levels, for instance, at the sequence flow level or the communication with external services. Also, to ensure data integrity and ACID (Atomicity, Consistency, Isolation, and Durability) transaction at the process level(van der Aalst and P. 2013).

The last important design criterion in the evaluation scheme is the translation into executable models (e.g., WS-BPEL (OASIS 2007) and YAWL). This process could be fully automated or in some cases need human and manual intervention. Nevertheless, it must guarantee traceability between the process model and execution model.

Deployment phase. In this phase, the organization deploys the process to be available for end users, as well as to connect and integrate the processes to other internal or external resources and services. In this sense, the organization size and the expected load balance should be analyzed to determine the best way to deploy and integrate process to existing organization systems and the external resource.

Considering these aspects, the BPMS was evaluated to determine to what extent it allows executing the process instance in distributed environments and multiple engines in the same physical server. Also, The BPMS integration technologies were evaluated, these technologies support the interaction with other organization services and resources, such as REST (Representational State Transfer) and WSDL (Web Service Definition Language) technologies.

Execution and Operation Phase. The process execution engine is part of the BPMS and is responsible for instantiating process models. These instances determine the process execution flow according to its data, events and business rules, among other aspects. In addition, at this phase, users interact with process instances through user interfaces to perform tasks, provide data to the process, or work with documents, among other activities.

Besides that, there are many other BPMS features which could support the users to organize and complete their tasks. For instance, calendar and document management, task inbox, email notifications, etc. These aspects could be internally implemented in the BPMS or externally by supporting tools or add-ons from other vendors. .

Moreover, one of the most critical features in the execution phase is the version management by which it is possible to simultaneously execute different versions of the same process and keep track of all running versions. This scenario is interesting when organizations need to evolve their processes without losing information of obsolete instances. This situation is widespread in industrial environments (e.g., aeronautical or automotive environments) in which the processes have a very long life.

Monitoring and Control phase. Monitoring or tracking the organization performance using BPM techniques (van der Aalst and P. 2013) is essential to improve the organization continuously. In those cases where technical monitoring deals with aspects like system response time, system load, server issues, and connection problems, business monitoring focuses on supervising process instances. The objective of this monitoring is to obtain the necessary data to calculate the PPIs.

Considering these aspects, several features were included in the quality model for assessing, controlling and monitoring techniques (e.g., the availability of log files and other historical resources like databases), user notifications in case of failures at infrastructure

levels (e.g., communications or servers) and business levels (such as reaching predefined values of PPI, among others).

Furthermore, the characterization scheme evaluates control features with which BPMS respond to any of the above situations, such as the ability to change the activity´s role (or resource) to balance the workload.

Moreover, an important aspect is to check the status of all running instances and their indicators. This type of control can be carried out using dashboards with which it is possible to improve the prediction of problems and risks during the process execution.

Analysis phase. The analysis is one of the four keys of BPM (model, analyze, enact and manage) because it is the phase in which performance indicators are examined. Once the performance data is analyzed, the organization has the insight to improve their processes. That is the essence of continuous improvement.

There are many techniques for analyzing data (e.g., studying models or event logs (van der Aalst and P. 2013), among others) that could be applied at different phases in process lifecycle. For example, during the modeling phase, the verification analysis could be used to ensure the correctness of the process model against the modeling language and the execution model (Weske, van der Aalst, and Verbeek 2004). Another analysis technique is the simulation/what-if analysis, which is very useful to get ideas on how to reduce costs while improving service levels.

Thus, it is important to have historical data and data mining tools to automate and streamline the application of analytical techniques of process performance.

Other Features. The quality model also includes several other evaluation characteristics grouped into this separate category, such as the maturity of the BPMS, the availability of the support, documentation, and learning materials.

Table II.1 provide a summary of the quality model and the evaluation criteria for each stage in the process lifecycle.

Table II.1. The quality model.

I. Modeling criteria	
	I.1 Supported BP Modeling languages
	I.2. Interoperability and Compatibility
	I.3 Reuse BP models
	I.4 Modeling views
	I.5 Modeling of business rules
	I.6 Modeling of PPIs
	I.7 Generate process documentation
II. Design criteria	
	II.1 Supported programming languages
	II.2 Designing user interface (UI)
	II.3 Way of describing roles

	II.4 Support for importing organizational structure
	II.5 Support for assigning a role to a user
	II.6 Support for SLA
	II.7 Support exception handling and Transaction Control
	II.8 Translation into executable models
	II.9 Supported BP execution languages
III. Deployment Criteria	
	III.1 Support for distributed execution
	III.2 Support for integration into other
IV. Execution and operation	
	IV.1 Version management of BP models
	IV.2 Support for calendar management
	IV.3 Support for informing users
	IV.4 Document management
V. Monitoring and Control	
	V.1 Support for technical monitoring and control
	V.2 Support for business monitoring BAM
	V.3 Change the role or resource
	V.4 Support for changing business rules
	V.5 Support for optimized execution
	V.6 Ability to deal with failures
	V.7 Support for changing the workload balance
	V.8 Support dashboards and reports
	V.9 Support for detail levels
	V.10 Support for different views of monitoring
VI. Analysis Criteria	
	I.1 Support for process verification
	I.2 Support for process simulation
	I.3 Support historical data available for analysis
	I.4 Support for suggestions on improvement
	I.5 Support for BI and Process mining tools
VII. Other criteria	
	II.1 Documentation
	II.2 Training
	II.3 Tool Maturity
	II.4 Commercial support

Evaluation results:

RQ1: What open source BPMS are available and what do they offer?

After carrying out the selection process, the BPMS that do not infringe on any exclusion criteria and comply with all inclusion criteria were: Activiti, Bonita, jBPM, ProcessMaker, uEngine BPM, Camunda, and YAWL.

The results of the evaluation of characteristics supported by each BPMS are shown in Table II.2 and Figure II.1.

Table II.2. Summary of evaluations of each BPMS

	Bonita	ProcessMaker	YAWL	Camunda	Activiti	JBPM	uEngine
I Modeling criteria: PS1 = $\frac{\sum_{i=1}^{7} I_i}{28} \cdot 10$	5.36	5.36	6.79	6.43	5.36	6.79	4.29
I.1 Supported BP Modeling languages	4	4	4	4	4	4	4
I.2. Interoperability and Compatibility	4	0	4	4	4	4	0
I.3 Reuse BP models	0	4	4	0	0	4	0
I.4 Modeling views	4	4	4	4	4	4	4
I.5 Modeling of business rules	3	3	3	3	3	3	4
I.6 Modeling of PPIs	0	0	0	3	0	0	0
I.7 Generate process documentation	0	0	0	0	0	0	0
II Design criteria: PS2 = $\frac{\sum_{i=1}^{9} II_i}{36} \cdot 10$	8.06	8.33	8.61	8.33	8.33	8.61	6.94
II.1 Supported programming languages	4	4	4	4	4	4	4
II.2 Designing user interface (UI)	4	4	4	4	4	4	4
II.3 Way of describing roles	3	4	4	3	3	4	3
II.4 Support for importing organizational structure	4	4	4	4	4	4	0
II.5 Support for assigning a role to a user	3	3	3	3	3	3	3
II.6 Support SLA	0	0	0	0	0	0	0
II.7 Support exception handling and Transaction Control	3	3	4	4	4	4	3
II.8 Translation into executable models	4	4	4	4	4	4	4
II.9 Supported BP execution languages	4	4	4	4	4	4	4
III Deployment criteria: PS3 = $\frac{\sum_{i=1}^{2} III_i}{8} \cdot 10$	5.00	5.00	5.00	10.00	8.75	10.00	5.00
III.1 Support for distributed execution	0	0	0	4	3	4	0
III.2 Support for integration	4	4	4	4	4	4	4
IV Execution & operation criteria: PS4 = $\frac{\sum_{i=1}^{4} VI_i}{16} \cdot 10$	8.13	7.50	9.38	7.50	4.38	6.88	7.50
IV.1 Version management of BP models	4	0	4	4	0	4	4
IV.2 Support for calendar management	1	4	4	2	0	0	0
IV.3 Support for informing users	4	4	3	4	3	3	4
IV.4 Document management	4	4	4	2	4	4	4
V Monitoring & Control criteria: PS5 = $\frac{\sum_{i=1}^{10} V_i}{40} \cdot 10$	4.50	4.25	6.00	6.75	4.75	6.75	7.00
V.1 Support for technical monitoring	4	0	0	0	0	0	0
V.2 Support for business monitoring BAM	3	3	4	4	0	4	4
V.3 Change the role or resources	4	4	4	4	4	4	4
V.4 Support for changing business rules	0	0	3	2	0	0	0
V.5 Support for optimized execution	0	0	3	0	0	0	0

V.6 Ability to deal with failures	3	3	3	3	3	3	4
V.7 Support for adjusting the workload balance	4	4	4	4	4	4	4
V.8 Support dashboards and reports	0	3	0	4	4	4	8
V.9 Support for detail levels	0	0	0	3	0	4	0
V.10 Support for different monitoring views	0	0	3	3	4	4	4
VI Analysis Criteria: PS6 = $\frac{\sum_{i=1}^{5} VI_i}{20} \cdot 10$	8.50	3.50	7.50	5.50	7.50	7.50	4.00
VI.1 Support for process verification	4	0	4	4	4	4	0
VI.2 Support for process simulation	4	0	4	0	4	4	0
VI.3 Support historical data	3	4	4	4	4	4	4
VI.4 Support for suggestions on improvement	3	0	0	0	0	0	0
VI.5 Support for BI and Process mining tools	3	3	3	3	3	3	4
VII Other criteria : PS7 = $\frac{\sum_{i=1}^{4} VII_i}{16} \cdot 10$	10.00	10.00	8.75	10.00	9.38	9.38	8.75
VII.1 Documentation	4	4	3	4	4	4	3
VII.2 Training	4	4	3	4	3	3	3
VII.3 Tool Maturity	4	4	4	4	4	4	4
VII.4 Commercial support	4	4	4	4	4	4	4
FINAL SCORE: FS = $\frac{\sum_{i=1}^{7} PS_i}{70} \cdot 10$	7.08	6.28	7.43	7.79	6.92	7.99	6.21

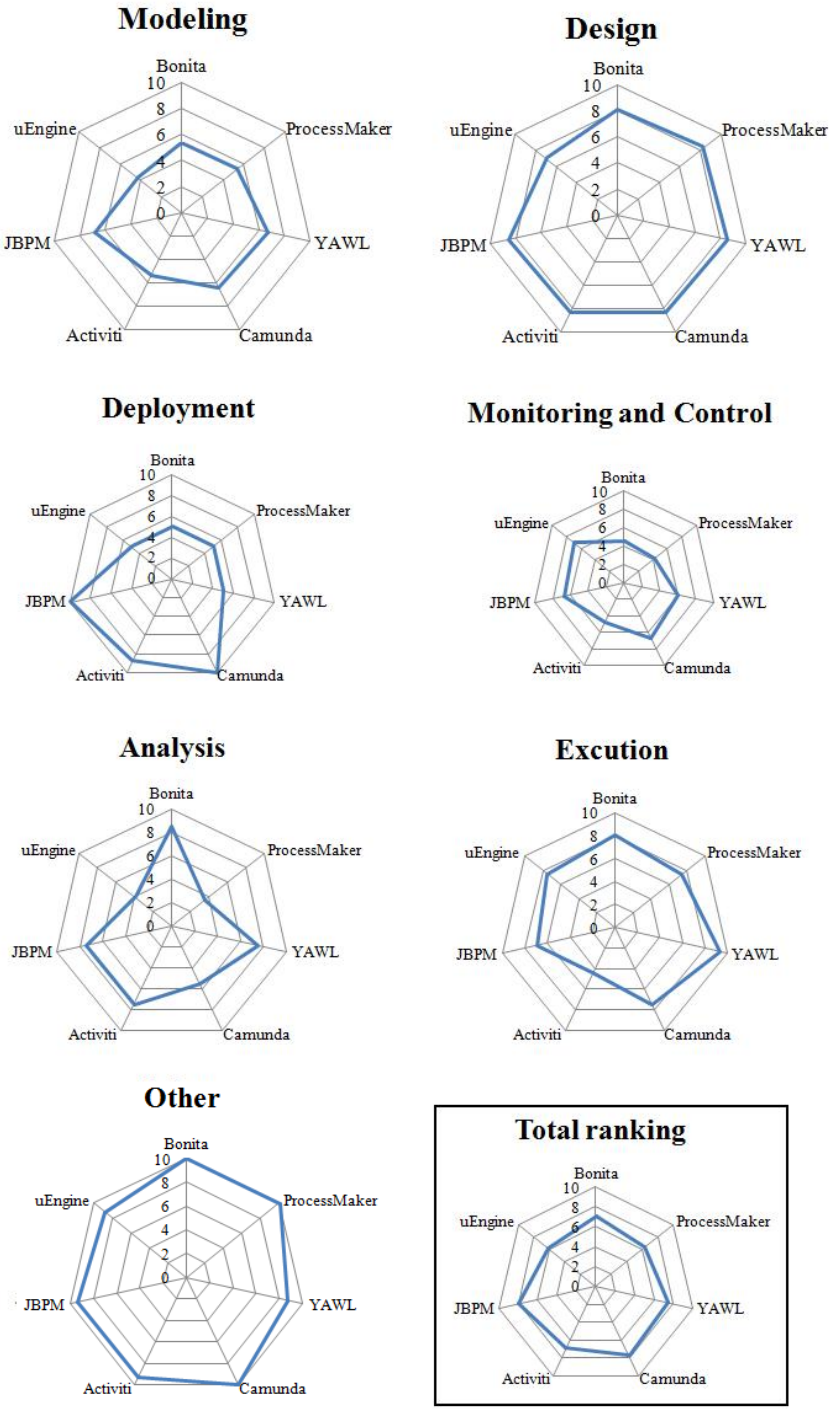

Figure II.1. Summary of the BPMSs evaluation per BPM phase

To answer the second research question, we have analyzed the results of the evaluation to identify issues. The following section describes the main issues and possible improvements:

There is a lack of the definition and integration of the PPIs into the process model, also, in linking the PPIs with the service level agreements. Moreover, there is a need to strengthen native Business Rule Management. Furthermore, the generation of the process documentation needs further support. Also, it is necessary to support the suggestions

when monitoring service discovers an issue which affects the execution of the process (e.g., user overloading, bottlenecks, etc.).

Based on the identified weakness in the existing BPMSs, Authors have included several recommendations for future investigations, such as the necessity to perform further research on how to integrate PPI activities into the process lifecycle. Also, more research is needed on process mining and business intelligence tools..

1.2 Measurement in the context of the software development process – mapping study

The importance of this study arises from the findings and conclusions obtained from the previous survey. These results highlight the need for better support for monitoring and the continuous improvement of the process throughout its lifecycle. Process measurement plays a crucial role in process management and improvement. Thus, efficiently integrating the process measures is essential to understand and predict its performance.

This Systematic Mapping Study (Ayman Meidan et al. 2018) focuses on the measurement of the software development process and its execution projects, mainly to give insight on the measurement related to the "Project" and "Process" entities. These two entities were included - in the systematic literature review carried out by (Gómez Oswaldo and García 2006) - among the most measured entities in the software engineering domain.

This mapping study aims to get a better understanding of measurement in the context of the SDP, to classify and structure the existing research in this area and to identify the quantity, research types, and results available within it.

Next section presents the research questions formulated by the authors to accomplish these goals:

RQ1: How is the research area structured? What are the trends concerning the publication quantity and focus? What are the leading publication channels for this area?

RQ2: What are the principal research types and methods found in the studies?

RQ3: What are the main measured abstractions and attributes and what are the different measurement purposes in the research?

RQ4: What are the main base methods, model, or techniques used in the research?

RQ5: What are the main research contexts in the literature?

We have developed and conducted a research process to answer these questions. We have improved this process gradually in two ways: firstly, by conducting a pilot study after the first database searches to refine the process and the research string. Secondly, by reviewing and improving the search method and tools during the process.

The research process has comprised of the following steps:

(1) **Defining search strategy**:

a) Defining the research string, inclusion and exclusion criteria.
b) Identifying the research source (databases)

(2) **Conducting the research**:

a) Identifying the relevant studies based on keywords, title and abstract.
b) Screening the studies to apply inclusion and exclusion criteria.
c) Designing a classification scheme

(3) **Data extraction**, **updating the classification scheme**, **mapping process and results**.

The following subsections will describe in detail this research process.

1.2.1 Defining Search Strategy

Dybå et al. (Dybå, Dingsøyr, and Hanssen 2007) proposed four steps to define an exhaustive search strategy. Formulating the search string is the first step. The second step is determining the inclusion and exclusion criteria. The third step is applying the inclusion and exclusion criteria to the titles and abstracts to find the relevant studies. And the fourth step consists of retrieving the related articles for a comprehensive evaluation.

The search string used to search in the selected databases is defined as suggested by Kitchenham (B. Kitchenham and Charters 2007), who considers population, intervention, and outcome:

- Population. The population is the software development. The keywords ''software development process'' and ''software process'' are used to search for the population.
- Intervention. Intervention refers to the measurement of the software development process. For the intervention, we use the keywords "measure", "metric", "indicator", "quantitative", "KPI" (key process indicator) and "PPI" (process performance indicator).
- Outcome. The study is not limited to comparative bodies of research or specific outcomes. Therefore, the research strategy does not consider comparison and outcome.

The search string was formulated as the following:

("software development process" OR "software process") AND (measure OR metric OR indicator OR quantitative OR KPI OR PPI)

The inclusion criteria are:

- Topic. The study should provide a subject related to measurement in the context of the software development process.
- The literature must be restricted to peer-reviewed journals and conferences.
- The study should be written in English or Spanish.
- Content. The complete text must be available.

And the exclusion criteria below:

- Studies focused on the measurement of the final product must be discarded.

- Workshop, book, tutorials must also be excluded.

Identify the Research Source (Databases):

We have considered the recommendations of (Dybå, Dingsøyr, and Hanssen 2007) and (B. Kitchenham and Charters 2007), as well as the search experience reported in [S163] in the selection of the search databases. Therefore, after performing the pilot searches, the following databases were selected: ACM Digital Library, IEEE Explore, Science Direct (Elsevier), and SCOPUS and Wiley Inter-science database.

Furthermore, we have applied the backwards snowball sampling method as recommended by Kitchenham and Charters (B. Kitchenham and Charters 2007) to find additional relevant literature not discovered during the database search. The snowballing process involves gathering all relevant references from the studies found using the database search, then applying the inclusion and exclusion criteria for the title first, then on the abstract. This process was applied to each included study in iterative form, and in the third iteration, no new studies were included.

1.2.2 Conducting the Research

Identifying the Relevant Studies Based on Keyword, Title, and Abstract:

The search on the databases (completed on 10 September 2017) has retrieved 13,809 papers, as Figure II.2 shows. After applying the selection criteria to the title, abstract and keyword, 1,686 studies were included as candidates. Once duplicates and unavailable full-text papers were removed, the total amount of full-text reading papers was 1,013, which includes 38 papers identified through other sources, such as references of candidate papers. Finally, 462 studies were included after applying the inclusion and exclusion criteria.

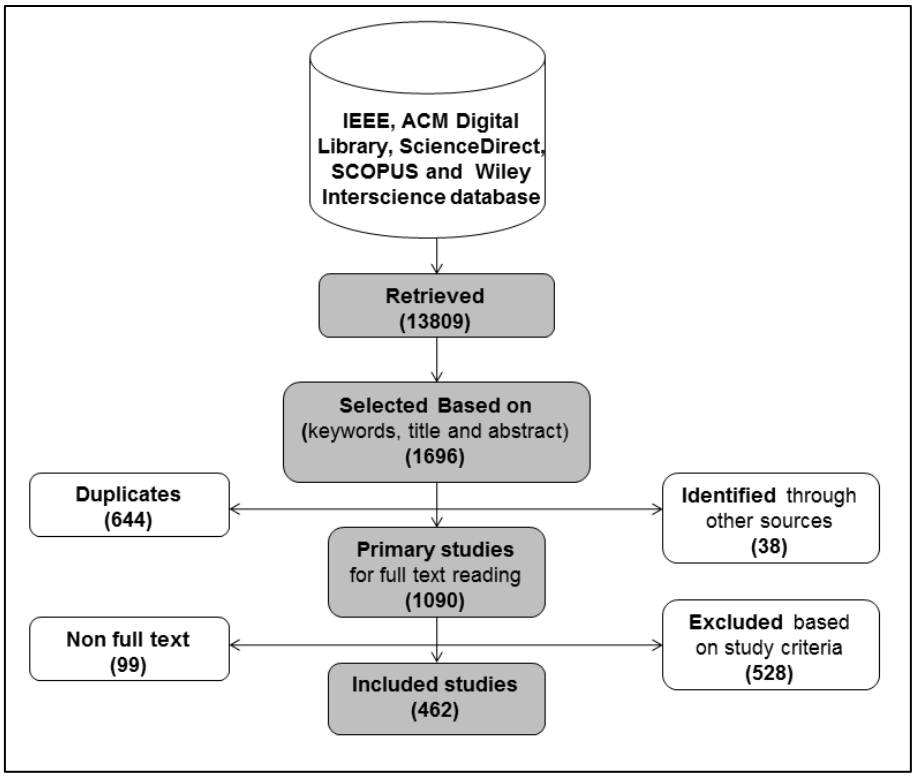

Figure II.2. Selection process.

We have constructed a classification scheme to categorize and structure the included studies. This scheme consists of several aspects such as source type, publication year, research type, contribution type, proposal type, validation type, entity/abstraction and study context.

We have considered the following quality attributes -mentioned in Lough (Lough 2001) - during the development of the classification scheme:

- Orthogonally. Clear definition of the categories, which simplify the classification process.
- Existing literature. Designing the classification/taxonomy on the basis of comprehensive review and analysis of the area.
- Terminology. The classification should use terms inspired by the existing literature.
- Completeness. No absent categories; all existing literature can be mapped to a category.
- Acceptance. The community approves and realizes the classification/ taxonomy.

The classification scheme was designed during the selection process and the full-text reading using a mixed top-down and bottom-up approach. The main topics (i.e., subjects within the measurement domain) and categories are identified based on the research questions.

The following section describes the main aspects of the classification scheme.

- ***Publication Year***: Classifying the studies based on the year of publication helps to group them and to discover trends.
- ***Source Type****:* There are two publication channels identified in the included studies (conference and journal). This classification helps to identify the source of the papers in the research area and the possible target for future publication.
- ***Research Approach****:* The classification of research types provided by (Wieringa et al. 2006) was chosen to classify the included studies. Table II.3 displays the classification summary.

Table II.3. Research types.

Category	Description
Evaluation Research	Evaluate the implementation of a method in a specific context (e.g., industry). Investigates the benefits and drawbacks of implementing the solution in practice.
Validation Research	Investigates the new methods or proposals, which have not been implemented so far (e.g., laboratory works).
Solution Proposal	Propose a solution; this solution could be original or a significant extension of an existing method. The consequences of implementing the solution are shown by applying the proposal in practice or by good arguments.
Philosophical Papers	Provide novel forms to see existing subjects by proposing a new conceptual framework, model or taxonomy.
Opinion Papers	Such papers present personal opinion about a specific method or solution; the opinion could be an assessment of the method or recommendation about it. The authors do not use research methodology or related work to formulate this opinion.
Experience Papers	These papers present a personal experience of the authors about applying a specific method or proposal in the real world.

We have classified the evaluation research as a case study, field study, field experiment, survey, and action research. Moreover, we have categorized the validation studies, which includes laboratory or office works as experiments, simulation, prototyping, mathematical analysis, mathematical proof of properties, reviews, investigations, and comparisons.

Furthermore, we have classified the proposal studies into methods (including frameworks and approaches), models, techniques and tools. These proposals were also classified based on its validation methods.

- ***Abstraction***: The abstraction aspect describes the level at which measurements are conducted. Considering the abstraction is essential because every entity (e.g., project) has different attributes (e.g., progress and velocity). Moreover, each attribute can be measured in several forms. Next, we describe the abstractions found in the literature.

    - *Organization*. Refers to the measurement of the software organization comprehensively (e.g., performance and capability).
    - *Process*. Means that measures are directly concerned with the process properties. The time consumed to complete some tasks or phases and the

numbers of errors encountered at determined periods are examples of process measures.

- *Product*. Means that, the measures are focusing on the attributes of the developed entities during the process.
- *Project*. Refers to an enactment of a process, involving the management process which controls the project execution, and the development process which produces the project outcomes. Typical examples for project measure are the effort, cost, time and quality.
- *Individual*. Refers to the people performing the work (e.g., programmer, designer, tester and project manager).
- *The software process model*. It is a real-world software process defined in a specific language which describes its components (e.g., flow, activities and resources). Some examples of process model measures are complexity and understandability.

- **Measurement Purpose**: M. Lindvall and V. Basili in (Lindvall et al. 2005) classify measurement according to the objective of the measurement process, that is to say, to characterize, understand, evaluate, predict and improve. *Characterizing, understanding and evaluating* are the fundamental purposes of measurement as it involves describing or differentiating software processes, explaining the different relations between the development components, and assessing the achievement of goals or the impact of a technology/process on products. *Prediction* constructs a model based on existing measures and relations to estimate measures at some future point in time. *Improvement* is the most sophisticated purpose of the measurement; it identifies the target and activities of the improvement process.

- ***Base Methods, Models and Techniques***: These studies were classified according to the base method, model or the technique used to develop the solution or to carry out the research. Examples of the base methods found in the included studies are Fuzzy Logic, Goal Question Metric (GQM) method, CMMI model and Process Mining.

- ***The context of the study***: The context defines the setting of the studied object (measurement in this research) and describes the study environment (Petersen and Wohlin 2009) (e.g., Agile software process, object-oriented project, Model Driven Engineering (MDE), and CMMI organization). It also defines in which circumstances the study results are valid. Thus, it is essential to compare the results of the study with other studies and to evaluate the generalizability of the study's conclusions.

- ***Validation Method***: The validity describes the extent to which a measure or measurement instrument characterizes the state of the measured attribute (Balch 1974). Measurement validation is required for pragmatic and theoretical reasons (Barbara Kitchenham, Pfleeger, and Fenton 1995). Fenton and Kitchenham (N. Fenton and Kitchenham 1991) discussed two different validation views: One view investigates the extent to which a measure characterizes the attribute state, and the other analyzes the appropriateness of a measure for predictive purposes. In (Ejiogu 1993) the author proposed five fundamentals for the formal validation of software metrics models: (i)The theoretical examination of the target attribute of software behavior the model measures. (ii)The mathematical satisfiability of the postulates of a

measure (function). (iii)The practical technical experimentation or the metrical vindication of the model beyond the theoretical cross-checking. (iv)The examination of the results of the measure to obtain any beneficial feedback effects for the potential advancement of productivity. And (v) the confirmation of consistency with dimensional analysis.

In this aspect, we classify the studies according to the method used to validate the proposal (e.g., industry experience and case study).

1.2.3 Data Extraction and Mapping Process

We have developed an extraction form to store the relevant data obtained from the included studies (e.g., title, publication year, publication channel, research type and topic). The data have been collected during the selection and full-text reading process. Since the included studies were restricted to peer-reviewed papers, the data (e.g., research type, method, and context) were extracted as mentioned by the original authors, in case the authors do not specify the requested data explicitly, these data were obtained by interpreting the paper content according to Table II.3 and the research methods defined in (Unterkalmsteiner et al. 2012).

The development of the classification scheme and the identification of the topics have been updated and completed during the full-text reading and the data extraction process. When a new topic was discovered, the classification scheme was updated, and the classification process was restarted for the consideration of the updated classification scheme.

The extracted data are not the same for all the topics. The common data extracted from all the included studies are:

- Basic data: Title, authors, publication (year and channel).
- Research type (e.g., evaluation, validation or experience).
- Research method (e.g., case study, review and comparison).
- Abstraction/entity (e.g., process and project).
- Attributes (e.g., capability or performance).
- Base method, model and technique, and the study context.
- Proposal studies (proposal type and validation method).

Next section describes the identified topics and the data extracted for each one:

Measurement concepts and practices: Studies related to this topic focus on the fundamental concepts, principles, and practices of the measurement (i.e., measures, measurement process and the main entities and possible attributes to be measured). This topic tries to find answers to questions such as why, what, and how to measure?

This data was also extracted for this topic:

- Measurement purpose (e.g., characterize and predict)

Assessment and improvement: The main goal of this topic is to characterize an object of interest (e.g., organization, process, project, and individuals) with respect to a chosen model (e.g., CMM, ISO/IEC 15504 and Personal Software Process (PSP) (W. S. Humphrey 1995)) so that it may be evaluated and improved.

This data was also extracted for this topic:

- Objective (e.g., assessment and improvement)

Measurement program: This topic focuses on how to establish a measurement program within an organization. Studies related to this topic describe experiences, methods, and recommendations on establishing a measurement program.

This data was also extracted for this topic:

- Objective (e.g., lesson learned and success factors)

Monitoring, control, and management: This topic focuses on collecting, supervising, analyzing and visualizing data through the execution of the project for management purposes.

The data extracted from the studies of this topic are:

- Objective (e.g., monitoring and management)

The included studies were categorized to topics according to the following process; Figure II.3 demonstrates the classification process:

Step1: Does the study focus on assessment or improvement based on a Model (standard or customized)?

Yes: the study added to the "assessment and improvement" category.

No: **Step2**: Does the study focus on measurement program?

Yes: the study added to the "Measurement program" category.

No: **Step3**: Does the study focus on using the measurements for monitoring, managing or controlling project?

Yes: the study added to the "Monitoring, Control, and Management" category.

No: the study added to the "Measurements concepts and practices" category.

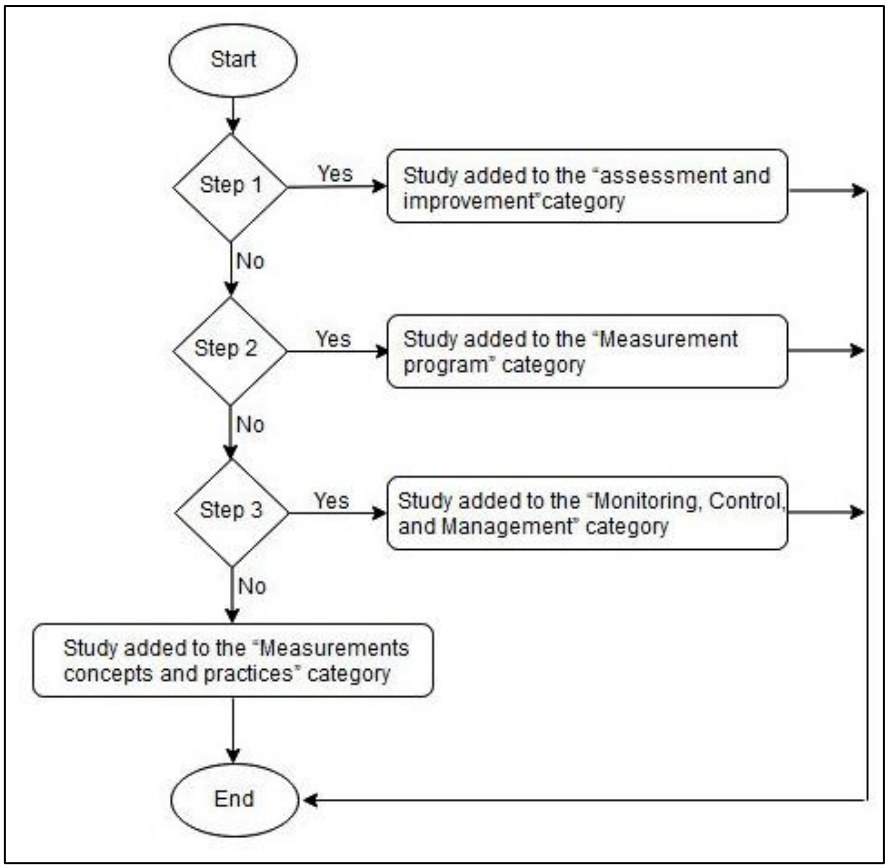

Figure II.3. The classification process.

This study identifies the abstraction levels, measurement purposes, base methods, models, techniques, validation methods and the contexts found in the included literature. These data were identified and extracted as mentioned by the original authors (during the selection and full-text reading process). When the original authors do not indicate these data explicitly, the paper content was interpreted to identify and extract it.

Primary studies were screened to identify the measurement purposes addressed by the original authors, after identifying the purpose of the measurement, it was added to the extraction form, then the paper was mapped to one of the measurement purposes discussed by (Lindvall et al. 2005).

Primary studies were also screened to identify the base methods, models and techniques used by the original authors, after identifying the used method or model it was added to the extraction form.

Moreover, the validation methods used by the original authors to validate the proposals were identified by screening the studies. When the validation method is determined it was added to the extraction form.

Primary studies were also screened to recognize the abstraction level and the attribute addressed by the original authors. The abstraction levels discussed by [S163] and [S306]) were used as a guide to distinguish and classify the abstraction level and attributes addressed in the included studies.

Furthermore, the contexts of the primary studies were identified, extracted, and classified according to the contexts discussed by (Petersen and Wohlin 2009)

1.2.4 Mapping results and conclusions

As shown by Figure II.2 (the selection process) 1090 papers were identified for full-text reading. After applying the inclusion and exclusion criteria, 462 studies were included. This amount of studies reflects the high importance of the subject (measurement in the context of the development process) and the attention it has received from researchers and industry.

Next section briefly reviews the results of this study by answering the research questions:

RQ1: How is the research area structured? What are the trends concerning the publication quantity and focus? What are the leading publication channels for this area?

Four hundred sixty-two studies were included; we have divided those studies into three periods and four topics based on the main focus to facilitate the research and to identify trends. The majority of the identified studies were published in conferences.

Figure II.4 shows the division of the studies into the topics.

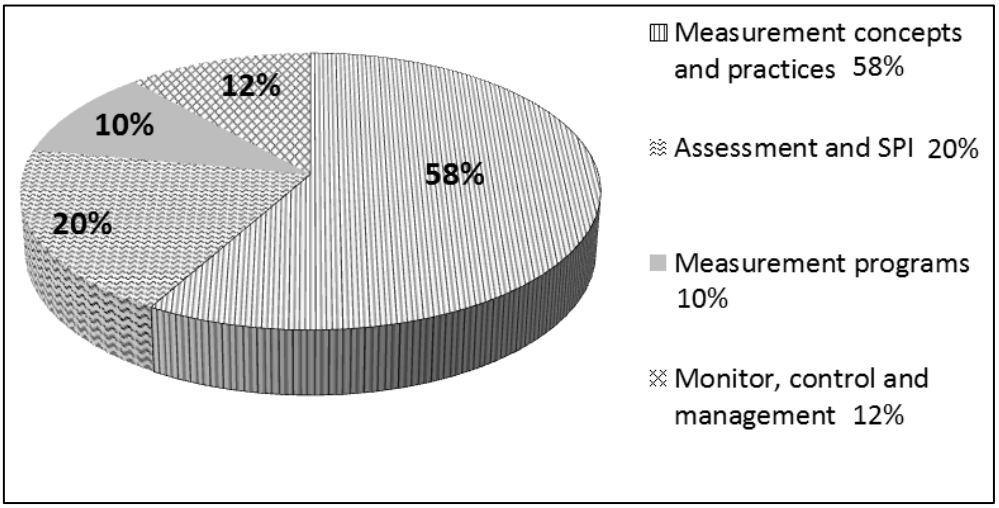

Figure II.4. The percentages of the identified topics.

Figure II.5 shows the number of papers included in each year and period. It also shows that the number of published studies related to the subject has increased over time, which indicates the growing interest in the topic.

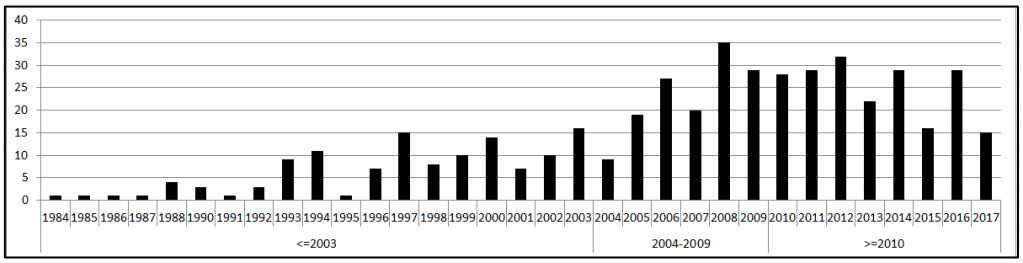

Figure II.5. The studies included throughout the years and publication periods.

Furthermore, Figure II.6 shows the evolution of the topics along the time, which demonstrate the continuous interest in each of the four topics during all the study period, which means that the researchers and the industry still pay constant attention to each of the main indicated topics.

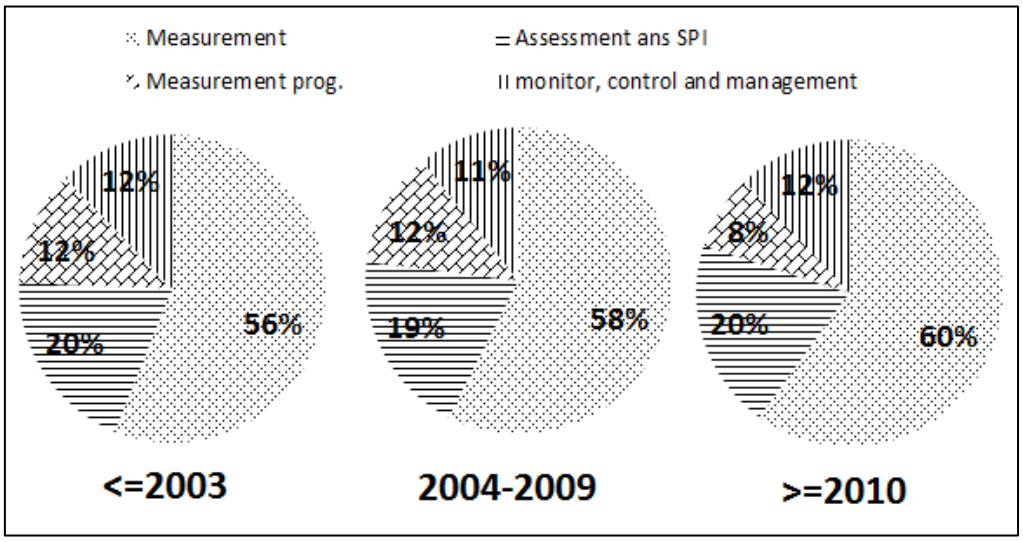

Figure II.6. Distribution of studies included in topics over time.

About 60% of the included studies were published in conferences, whereas 40% were issued in journals. Figure II.7 shows the distribution of forums over the time.

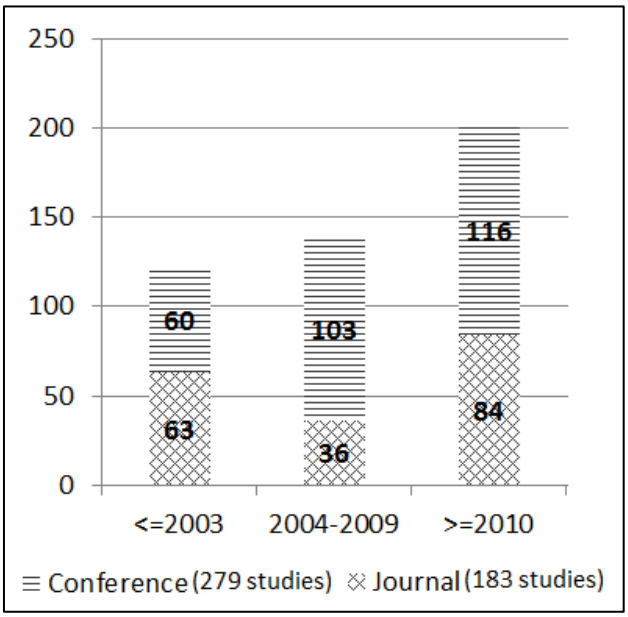

Figure II.7. Studies published in conferences and journals.

The included studies were published in about 220 forums. Due to the large number of forums, Table II.4 lists the top publication forums found in the last years (the period: >=2010).

Table II.4. Top publication forum in the recent years (the period: >=2010)

Rank	Forums type	Forums	No. of papers
1	Conference	Software Process and Product Measurement	20
2	Journal	Software: Evolution And Process	15
3	Journal	Information and Software Technology	8
4	Journal	Systems and Software	6
4	Conference	Software and System Process	6
5	Conference	IEEE/ACM Intel. Conf. on Automated Software Engineering (ASE)	4
5	Journal	Software Maintenance And Evolution: Research And Practice	4
5	Conference	Product-Focused Software Process Improvement (PROFES)	4

RQ2: What are the principal research types and methods found in the studies?

Among the selected papers, almost 70% were solution proposal studies, 15% corresponded to evaluation studies, 8% were experience studies, and 7% concerned validation studies. Neither philosophical nor opinion studies were included. Figure II.8 shows the research types over time and research topics.

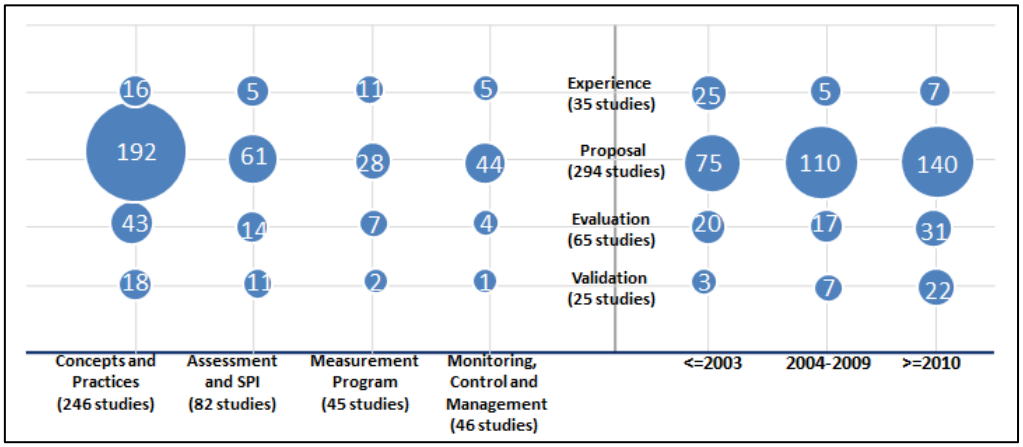

Figure II.8. The distribution of research types.

As mentioned earlier, around 70% of the included studies were solution proposal (325 studies), 74% of the proposals were classified into methods (to increase readability, we include methods, frameworks, approaches, and processes to this category), 19% were models (e.g., conceptual models and mathematical models) and 7% were tools. Thirteen proposal studies(which represented 4% of the total) were based on Model-Driven Paradigm.

Besides, 44% of the proposal studies did not mention any proposal validation, whereas 23% of them used the industrial experience to validate the proposal, 22% used case study, 6% of them used experiment and 6% utilized other methods, such as prototypes. Figure II.9 shows more details about the classification of the included proposals.

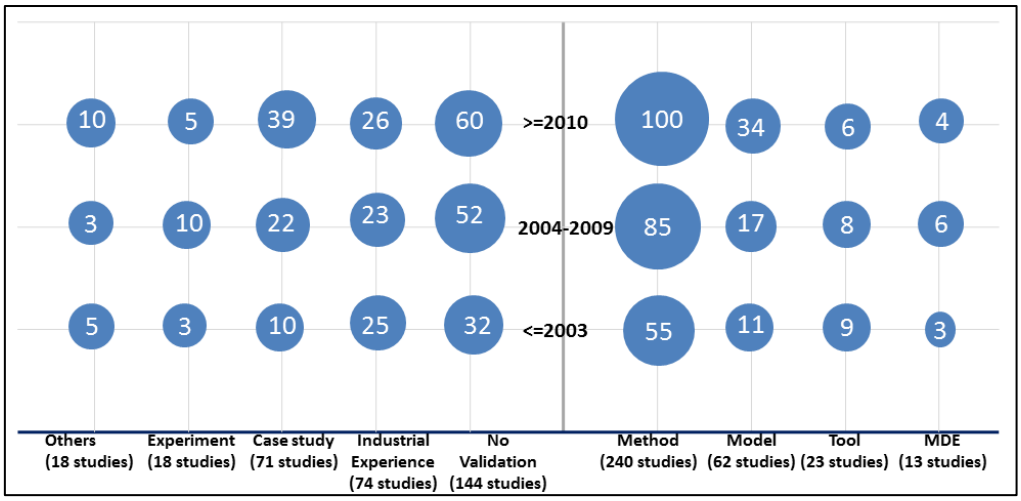

Figure II.9. Proposal types and validation methods based on time periods.

RQ3: What are the main abstractions and attributes measured and what are the frequencies of different purposes of measurement in the research?

The most frequently measured entities were the project (167 studies) and the process (156 studies). For the project, the most measured attributes were: effort, performance, risk, productivity, and progress. While the most measured process attributes were: capability and performance, maturity, quality, productivity, risk, and conformance.

The performance was the most measured attribute for the developer, the quality attribute for the process model and for the organization, the performance and the quality were the most measured attributes.

RQ4: What are the main base methods, model, or techniques used in the research?

Many methods, models, techniques, and standards were used in the studies as a base to build the proposed solution or to carry out the research. The most used methods were Goal Question Metric (GQM), applied to 37 studies, CMMI model used in 13 studies and Statistical Process Control (SPC) carried out in nine studies. Table II.5 displays the main methods according to the research topic. Below, Figure II.10 shows in alphabetical order the main models, methods, and techniques used according to the time periods.

Table II.5. Base models, methods, and techniques.

Type	Model, Method and Technique	Concepts and Practices	Assessment	And SPI	Measurement	Program	Monitoring,	Control and	Management
Models and Standards									
	CMM		4				1		
	CMMI	5	7		1				
	ISO/IEC 15504		4		1				

	ISO/IEC 33000	1			
	Personal Software Process (PSP)		4	1	2
	K-model		2		
	ISO/IEC 15939	2		3	
	ISO/IEC 9126	1			
Methods and Techniques					
	Analogy-based estimation (ABE)	1			
	Balanced Scorecard	3		1	
	Bayesian Network	5			
	Data Envelopment Analysis (DEA)	5			
	Data-mining techniques	2			1
	Earned Value Management (EVM)	2			3
	Function Points	1			
	Fuzzy logic	7			1
	Goal Question Metric (GQM)	23	1	11	2
	Goal-Driven Software Measurement	1			
	Grey Relational Analysis (GRA)	1			
	Neural Networks	3			
	Practical Software Measurement (PSM)	2		1	
	Six Sigma	2	2	1	1
	Socio-Technical Systems theory	1			
	Statistical Process Control (SPC)	1			8
	Use Case Points (UCP) and COSMIC-FFP methods	1			

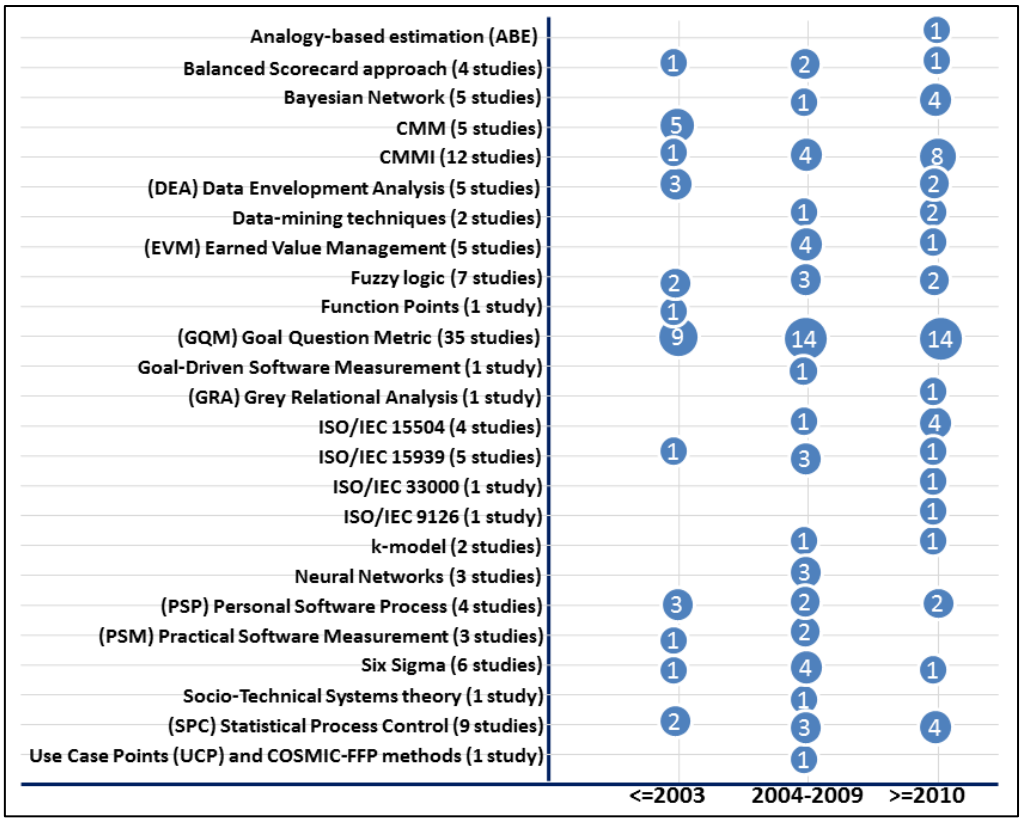

Figure II.10. Base methods, models, and techniques.

RQ5: What are the main research contexts in the literature?

The context of the study defines the environment in which the research is carried out and also determines the validity conditions of the study contribution.

The organization is the most common context dimension found in the primary studies, while the process dimension was the second. Table II.6 categorizes the contexts found in the literature according to the context dimensions mentioned in (Petersen and Wohlin 2009).

Table II.6. Context dimensions.

Dimension	Context
Market	No studies found
Organization (63 studies)	CMM
	CMMI
	ISO/IEC 15504
	Outsourcing
	Small and Medium-Size Enterprises (SME)

People	No studies found
Practices, Tools, Techniques (24 studies)	Application of Metrics in Industry (AMI)
	Goal Question Metric (GQM)
	GQM+ Strategies
	Object-Oriented Development
	Practical Software Measurement (PSM)
	Personal Software Process (PSP)
	Test-Driven Development
Process (45 studies)	Agile and Lean Development
	Incremental and Iterative Software Development
	Open Source Software Development
	Global software development (GSD)
Product (1 study)	Embedded System

Different contexts have been distinguished in the included studies; Agile and Lean development (35 studies) and Small and Medium-Size Enterprise (22 studies) were the most common contexts in the studies. Table II.7 lists the identified contexts according to the topic, and Figure II.11 shows them in alphabetical order according to the time periods.

Table II.7. Identified contexts

Dimension	Context	Concepts And Practices	Assess. And SPI	Measure. program	Monitoring, Control and Management	Total
Organization						
	CMM	2	8			10
	CMMI	9	9	1		19
	ISO/IEC 15504		9			9
	Outsourcing	2			1	3
	Small and Medium-Size	3	15	3	1	22

	Enterprises (SME)					
Practices, Tools, and Techniques						
	Application of Metrics in Industry (AMI)	2				2
	Goal Question Metric (GQM)	3				3
	GQM+ Strategies	2		1		3
	Object-Oriented Development	2				2
	Practical Software Measurement (PSM)	1		1		2
	Personal Software Process (PSP)	3	7			10
	Test-Driven Development	2				2
Process						
	Agile and Lean development	22	7	1	5	35
	Incremental and Iterative Software development	5				5
	Open Source Software Development	1			1	2
	Global software development (GSD)	1	1		1	3
Product						
	Embedded System			1		1

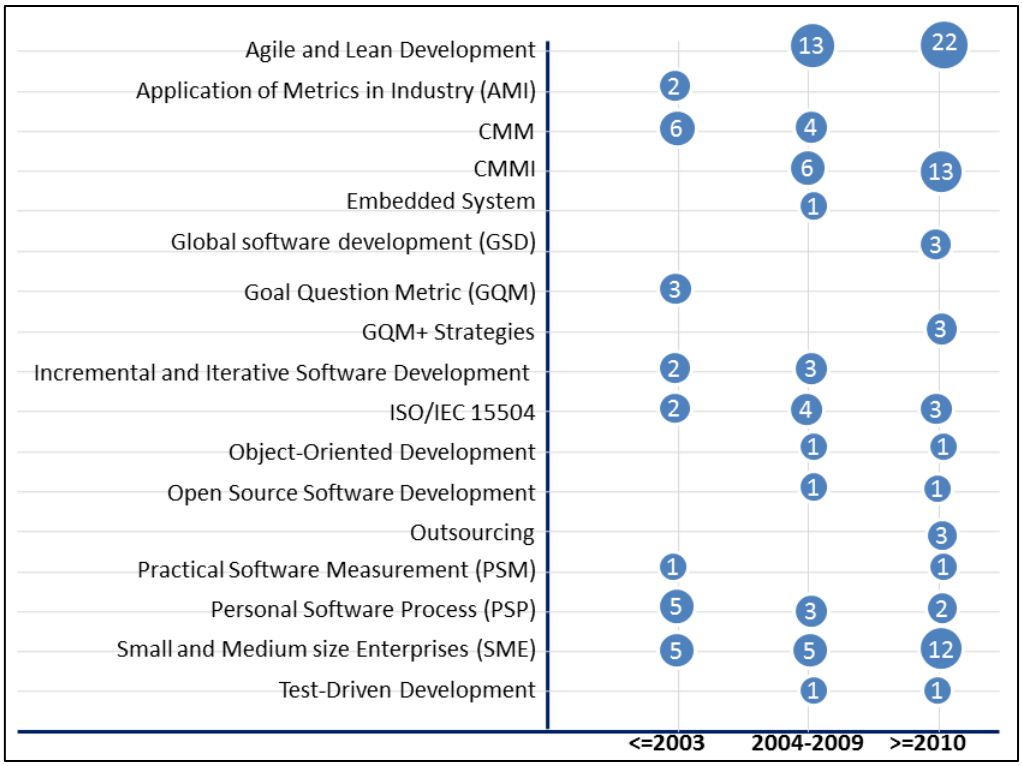

Figure II.11. Identified contexts according to the time periods.

We present below the main conclusions of this study

462 studies were included, those studies were divided into three time periods and four topics based on the main focus to facilitate the research and to identify trends. Five abstractions and 64 attributes were identified, 25 methods and 17 contexts were distinguished.

The majority of the identified studies were published in conferences. The study results demonstrate that the solution proposal was the main research type. About 56% of these proposals were validated. This result highlights the necessity for more focus on validating the proposals. The industrial experience and the case studies were the main methods used to validate the proposals.

Moreover, we found few studies that focus on the process model (6 studies). Furthermore, there is a scarcity of studies that focus on the definition and integration of the measures as an element of the process model. This integration supports the communication and the common understanding of the measurement plan among the participants. Also, few studies use the MDE paradigm (12 proposal studies, 4%) for the definition and integration of the measures in the development process lifecycle. MDE paradigm can support the formal definition and automation of the measure lifecycle.

The process (capability, performance, and maturity) were the most measured process attributes, while the project (effort, performance, and risk) were the most measured project attributes. The results also show the necessity of more focus on the process (agility, security, documentation, sustainability, and risks) and on the project (rework, reuse, communication, progress, and customer satisfaction). Further studies focusing on

the developer are also required, as well as more studies focusing on the prediction and estimation.

GQM, CMMI, and SPC were the main base methods and models used in the studies. Data mining and analysis techniques were the most prominent methods. The results also show limited use of international standards like ISO 15504 and 15939 against industrial standards like CMMI and GQM.

The majority of the studies did not describe the research context (71%), which highlights the need for more emphasis on defining the context of the study by researchers. Agile/lean development and small/ medium-size enterprise were the most frequently identified research contexts.

For researchers and practitioners, this study allows a better understanding of the state of the art of development process measurement, as the first step for further studies. Future research can be oriented towards a specific topic, entity, attribute, method, model or context to cover the necessities mentioned above. Future work could also focus on particular types of research such as surveys to explain the results of this study (e.g., why there is a scarcity of studies related to a specific topic, method or context?). Also, conducting comparisons between the different measures, proposals and methods in specific contexts could support researchers and practitioners to choose between them.

1.3 General conclusions from previous research

Previous studies reveal the lack of support for the measurement in the existing process modeling and management tools. The survey discussed in section 1.1 demonstrates the existing weakness in the definition of the measurements and its integration into the process lifecycle; section "I.6 Modeling of PPIs" in Table II.2 shows that the majority of the investigated tools do not support the definition and integration of the process measurement, this integration promotes the process monitoring and improvement. Therefore, supporting the definition and integration of the process measurements was mentioned in this study as a recommendation for future work.

Furthermore, the mapping study discussed in section 1.2 demonstrates the scarcity of research on defining the measurements in the form that allows its integration into the process lifecycle. This study also reveals that: the definition of measurements completely and operationally as well as considering the measurement issue in all the process stages is essential for strengthening process improvement and project management.

2. Related Proposals

This section discusses the relevant literature to the problem and the solution presented in this thesis. We have identified these works through the mapping study addressed in section 1.2 of this chapter and then were studied extensively in the preparation phase of this thesis.

In this section, we describe the existing research attempts to define and integrate the measurement into the SDP, the main process modeling languages and tools, and also

discuss the main methods to select and define the measures. Furthermore, we outline the main existing process and measurement lifecycles.

2.1 Relevant Research, Modeling languages and Tools

Measurement is essential for the quantitative management and improvement of the SDP, for that it has gained a significant interest for both researchers and practitioners. There are many proposals in the literature related to the measurement definition, modeling, and execution. This section focuses on model-based proposals.

In (Bendraou, Gervais, and Blanc 2006) authors presents a metamodel based proposal for software process modeling. This proposal does not define the measurement as a process element, but authors mention the necessity to measure the different process elements during the process execution for monitoring purpose. They also discuss the need to apply changes to the process elements to support its measurement (e.g., add some attributes to the process elements).

In (Mora et al. 2009; Mora, Garcia, et al. 2008; Mora, Piattini, et al. 2008) the authors propose a measurement framework based on Model Driven Architecture (MDA) (Singh and Sood 2009) to measure any software entity (e.g., database structure, process model, and requirement document) based on the metamodel that represents them. They also present a graphical notation language which allows the users to define software measurement models based on software measurement ontology. This work focuses mainly on measuring model elements based on its metamodel (e.g., count the number of tables in a relational database scheme). Thus, this proposal does not focus on measuring the process execution perspective such as the elapsed time to perform an activity.

In (Larrucea and Iturbe 2010) authors present an approach to combine different metamodels (e.g., SMM (OMG 2009) and SPEM 2.0 (Object Management Group 2002b)) to model the process and the measures to provide control over the execution of processes. This approach allows the definition of measures only for processes and tasks elements but does not allow the process modeler to model the measures in explicit and operational form(Deming 1986) within the process model.

In (Freire et al. 2011) the authors present a model-driven approach for the definition, execution, and monitoring of SDPes, it supports the automatic collection of quantitative measures during project execution. Authors define a metamodel to define the measures. This approach does not define the measures explicitly in the process model, does not consider the manual measures, and also does not measure the process artifacts. Furthermore, the measure definition does not address how the values of the measures will be analyzed and used.

The authors in (Del-Río-Ortega et al. 2013) provides a metamodel and tool for the definition and the design-time analysis of PPIs independently of the language used to model the process. This proposal does not reflect the relation between the information needs, the indicators and the data collected to satisfy this information needs. Moreover, this proposal focuses only on the measures that will be collected automatically; does not support the definition of the manual measurements (e.g., specifies the necessary methods

and tools to perform the measurement activities). Furthermore, the proposal does not allow the definition of context data to be collected with the measurement value.

In the proposal (Garcia-Garcia 2015; García García et al. 2015) the authors present a metamodel to define the development process. This metamodel defines the measure as a process element, the proposal derives the process execution model from the definition model, but the resulting model does not include the measure element defined in the definition model. This proposal could be developed by extending the metamodel to define the measurement concepts (e.g., information needs), and by adding more attributes (e.g., performer role, unit, context, collection method, etc.) the measure element, and also by representing the measure element in the process execution model.

On the other side, the industrial standard BPMN 2.0 (Object Management Group 2011) does not define the measures as a process element. The commercial implementations of this standard (e.g., Bonita BPM (Bonitasoft 2016)) allows the modeler to define an attribute to be measured but does not define the measures as an element to allow the modeler to include it in explicit and operational way. SPEM 2.0 (Object Management Group 2002b) define the measure as a process element but in basic and abstract form, wherefore, the process modeling tools which use SPEM 2.0 metamodel (e.g., EPF(Eclipse Process Framework Project | projects.eclipse.org 2017) and RMC (IBM - Rational Method Composer 2017)) does not support modeling the measures operationally and explicitly within the process model.

These academic and industrial works fail to integrate the measurement into the process lifecycle in such form that allow: (i) the process engineer to define and model the measurement concepts (e.g. information needs, performer, procedure, and context) in operational form during the process definition and modeling phase. (ii) using this definition in the process deployment phase to perform the necessary configurations to collect and store the measurement values. (iii) collecting the measures data during the process execution phase according to the measure's definition. (iv) analyzing the measured data during the process monitoring and analysis phase according to the method indicated in the measure's definition. Furthermore, (v) reporting the measures and its analyses to the indicated role to determine the necessary actions to control, optimize and improve the process.

2.2 Measurements Selection and definition Methods

This section briefly describes the main methods utilized to select and define the measurement concepts:

2.2.1 Goal Question Metric

Goal Question Metric (GQM) (van Solingen et al. 2002) is a top-down and goal-oriented paradigm to define the measures in a hierarchical structure based on a conceptual goal. The first step specifies the measurement objectives or goals. Then, refine the measurement objective into various questions that analyze the principal dimensions of the objective. And finally, the measures which provide the data needed to answer these questions are then determined. This method provides a clear relationship between the defined measures and the organization goals.

GQM method involves the following three levels:

i. Conceptual level: Establish the measurement goals (scope and purpose) aligned with the business needs. Example for measurement objective is: *reduce costs*.
ii. Operational level: Produce a list of questions that can be used to determine the goal accomplishment. For example: *What is the rework effort?*
iii. Quantitative level: Define the necessary measures to answer each question quantitatively. E.g., *Rework effort*

2.2.2 Goal Question Indicator Metric (GQIM)

GQIM (Goethert and Siviy 2004; Park, Goethert, and Florac 1996) is an extension of GQM that does provide an indicator layer to support the analysis and interpretation of the measurement goal.

GQIM steps are organized into three categories of activities (Zubrow 1998):

Goal identification. The measurement should respond to the questions: "What we need to characterize or understand? And what is the decision we need to make?" The main focus of this group of activities is to define the measurement goal and define the indicators that determine if the goal has been achieved. In this step, three types of indicators are distinguished:

- *Success indicators:* These indicators are derived from the established definition of success. This type of indicators is used to clarify if the objectives have been achieved.
- *Progress indicators:* These indicators are used to trace the realization of the defined tasks.
- *Analysis indicators:* These indicators are used to support the analysis of the execution results of the activity.

Indicator identification and data specification. Indicators are instruments to highlight the information contained in the data, specify the characteristics of the data that must be collected and processed, also determine how these data will be analyzed and interpreted and when these activities will occur.

Evaluate the existing measurements status. Existing measurement process, activities and gathered data are analyzed to avoid duplication and to identify missed data. High priority is given to the data needed to generate the indicators.

The following steps summarize the GQ(I)M method as described in Goal-Driven Software Measurement (Park, Goethert, and Florac 1996): (i) Identify business goals, (ii) Refine the goal using Questions, (iii) Decomposing the goals to sub-goals through the perspective, (iv) Identify the entities and attributes related to each sub-goal, (v) Specify the measurement goals, (vi) Determine the quantifiable questions and its related indicators which support the achievement of the measurement goal, (vii) Determine the data elements that will produce the indicators, (viii) Define - in operational form - the measures that will be used, (ix) Determine the necessary procedures to implement the measures, (x) Develop the measures implementation plan.

2.2.3 Practical Software and Systems Measurement

Practical Software and Systems Measurement PSM (McGarry 2002) is a flexible measurement approach; its goal is to provide project and technical managers with the best practices and guidelines in software measurement. PSM focuses on issues in software projects which typically require management and control. It considers the measurement as a process that should be adapted to the characteristics of each project, the purpose of this process is to provide insight into the project issues, risks, and objectives.

PSM organizes the information needs into seven categories; these categories could also be considered as sources of risks and issues:

- Schedule and progress,
- Resources and cost,
- Growth and stability,
- Product quality,
- Development performance,
- Technical adequacy,
- Customer satisfaction.

Furthermore, PSM provides nine principles that define the characteristics of an effective measurement process:

1. The measurement requirements are determined by the project issues, risks, and objectives.
2. The development process determines how the measurement is performed currently.
3. The details level of gathering and analysis the data is adequate to recognize and isolate the problems.
4. Achieve an independent analysis capability.
5. Structured analysis process is used to track how the measures affect the decisions.
6. The measurement results are interpreted in the context of other project data.
7. Software measurement is integrated into the project management process throughout the software lifecycle.
8. Measurement process is used to support objective communication.
9. The measurement process initial focus is on a single project analysis.

2.3 Process Improvement and Lifecycles

The first chapter of this thesis emphasizes the importance of using measurement throughout the process lifecycle for its management and improvement. Therefore, it is essential to consider the measurement issues in all phases of the software process lifecycle.

A lifecycle can be defined as a set of phases- each with a specific focus - performed to achieve specific and integrated tasks for the process continuous improvement. Given the wide range of application areas, different views of the process lifecycle have been proposed over the past decades. The most recent process lifecycles are summarized below.

Authors in (Hill et al. 2006) propose a global process revision cycle with the aim of creating value for organizations. To do this, they contemplate modeling processes as the first step to achieve this goal. In this way, before initiating any design or process review, the organization must decide the scope of its initial activities. The process lifecycle proposed by these authors is based on the following nine phases (see Figure II.12).

Discovery. Establish how the process is/should be performed. In this phase, the organization establishes an agreement between the stakeholders about how to implement the work and how to measure the process objectives.

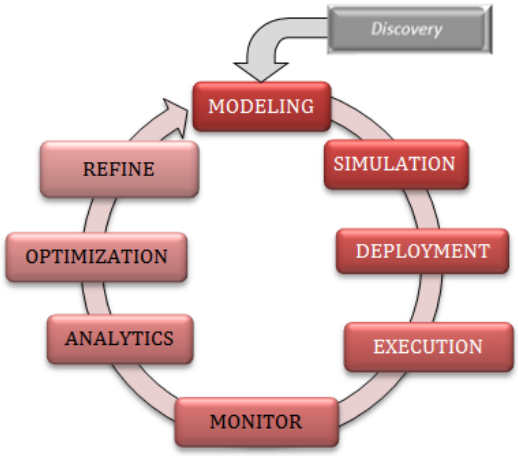

Figure II.12. Hill et al. (Hill et al. 2006) process lifecycle

Modeling. In this phase, the process is defined and described in a formal manner, which support common understanding among stakeholders about the process. Modeling also helps stakeholders to discuss how the suggested improvements could support the achievement of the organization goals.

Simulation. It is the phase in which possible bottlenecks that are not evident during modeling are detected.

Deployment. In this phase, the organization develops the detailed process execution scenarios and performs the necessary modifications to integrate the process into the organization information systems.

Execution. It is the phase in which the processes defined and deployed in the previous phases are instantiated and executed. In this phase, the process activities are performed to deliver the process value.

Monitoring. In this phase, the process execution data are collected to support process management and improvement.

Analytics. The data collected in the previous phase is analyzed in this phase. Also, the process indicators are evaluated to discover deviations from the organizational objectives.

Optimization. In this phase, using the results of the analysis performed in the previous phase, the process is optimized to improve its performance, outputs and reduce the organization risks.

Refine. In this phase, process objectives and activities are reviewed to discover improvements and for better alignment with the organization goals.

On the other hand, in (W.M.P. van der Aalst 2004; Wil M. P. van der Aalst 2004) authors establish a process lifecycle that is much more compact than that presented in the previous proposal. In this case, the lifecycle is based on four phases as shown by Figure II.13. These phases are summarized below.

Process design. During this phase, the process model is defined taking into account their control flow, data flow, user profiles, organization data, and any other operational aspect.

System configuration. It is the phase in which the model of the process defined in the previous phase is deployed and integrated into the organization information system.

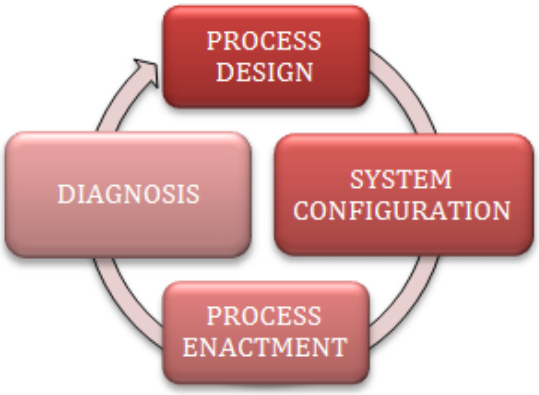

Figure II.13. Process lifecycle (W.M.P. van der Aalst 2004; Wil M. P. van der Aalst 2004).

Process enactment. In this phase, the processes deployed in the organization information system are instantiated and executed.

Diagnosis. is the phase in which the processes in execution are monitored and analyzed to identify problems and find possible improvements.

2.4 Measurement Process Lifecycles

The term measurement lifecycle (see Table II.8) refers to the entire phases of the measurement process (e.g., measurement definition, application, and the exploitation of the measurement result)(Habra et al. 2008).

Table II.8. Measurement process (ISO/IEC/IEEE 12207-2017-International Standard - Systems and software engineering -- Software lifecycle processes 2017, ISO/IEC/IEEE 15288-Systems and software engineering System lifecycle processes 2015)

«The purpose of the Measurement process is to collect, analyzes, and report objective data and information to support effective management and demonstrates the quality of the products, services, and processes. »

Over the last decades, several authors have identified and described phases of the measurement process. The main and most recent proposals are summarized below.

Jacquet et al. (Jacquet and Abran 1997) have decomposed the measurement lifecycle into four successive steps as shown in Figure II.14. We summarize these steps below.

Design of the measurement method. This step includes: defining the measurement objectives, define the measurement object, characterize the measurement concept, and defining the assignment rules.

Measurement method application. In this step, the measurement data are collected, and the measurement methods -defined in the previous step- are performed to produce the measurement results.

Measurement result analysis. The results obtained in the previous step are documented, evaluated, audited and analyzed in this step.

Exploitation of the result. In this step, the measurement results are used in many several forms (e.g., characterizing and predicting purposes).

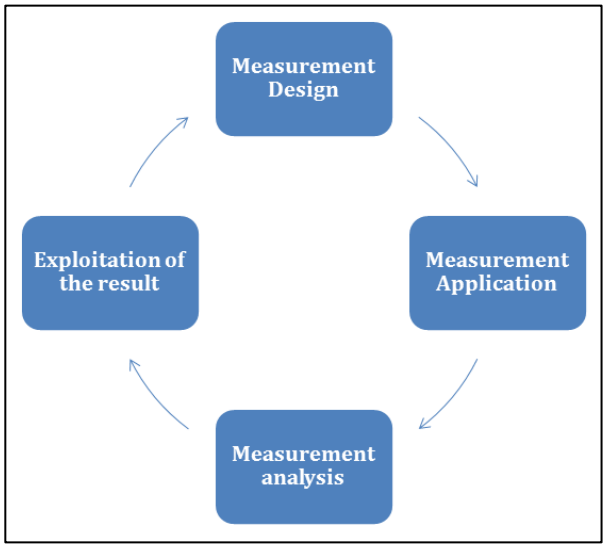

Figure II.14. Measurement Process (Habra et al. 2008; Jacquet and Abran 1997)

In similar form, authors in (Y. Zhang and Sheth 2006) have divided the measurement process into four steps: definition, collection, analysis, and in the last phase, the analysis results are used to control and improve the process.

On the other hand, in (Del-Río-Ortega, Resinas, and Ruiz-Cortés 2009) the authors propose a measurement lifecycle comprise of four phases:

- *Definition*. During this phase, the measures are identified, defined and linked with the process objectives.
- *Measuring*. Where the data is gathered.
- *Analysis*. In this phase, the measured values are compared with the target values, and the causes of any unexpected value are identified.
- *Report*. In this phase, the analysis results are summarized and reported to the users.

Furthermore, the recent version of the standard ISO 15939-2017 (ISO/IEC/IEEE 15939 International Standard - Systems and software engineering--Measurement process 2017) define a measurement process of four phases:

Establish and sustain measurement commitment. In this phase, the measurement requirements and scope are defined, the management commitment is established, and resources are assigned for the measurement activities.

Prepare for measurement. The main activities of this phase are:

- Define the measurement strategy,
- Describe the relevant characteristics to select measures and interpret the information products,
- identify and prioritize the information needs,
- Select and define measures that fulfill the information needs,
- Define procedures for data collection, analysis, access, and reporting,
- Define criteria for evaluating the information items and the measurement process,
- Identify and plan for the enabling systems or services to be used,
- Review, approve and provide resources for measurement tasks,
- Acquire and deploy supporting technologies.

Perform measurement: this phase includes the following activities:

- Integrate procedures for data generation, collection, analysis, and reporting into the relevant processes,
- Collect, store, and verify data,
- Analyze data and develop information items,
- Record results and informs the measurement users.

Evaluate measurement: this phase emphasis on the quality of the measurement process and the information needs. And include the following activities:

- Evaluate information products and the measurement process,
- Identify potential improvements.

3. Chapter summary

In this chapter, we have described the main previous research performed by the author to understand the current state of the art. The results of the first study (survey on the process management suites) demonstrate the necessity to enhance the integration of the measurement into the process management suites to promote the process monitoring and improvement.

To address this subject, we have conducted a mapping study to understand the measurement domain. Also to study how the measurement of the software process is considered in the literature.

This study allowed us to identify the most important entities and attributes related to the software process. It also enabled us to recognize the main measurement proposals and methods, and tools.

We have performed an in-depth study of the literature to explore how the issue of the software process measurement was addressed. The results of this study show that the existing proposals could be improved and completed to provide better support for the integration of the measurement into the software process in the form that promotes its management and improvement.

Furthermore, the results of the mapping study reveal the limited use of MDE technologies in the existing measurement proposals. The measurement proposals discussed earlier, use the metamodel to define the measurement concepts and its relations in a formal way, but they do not use MDE transformation facilities to derive models which support the automation and integration of the measurement throughout the process lifecycle (e.g., deployment and execution phases).

Next chapter describes the problem which will be discussed in this work, and it also outlines the proposed solution to address it. This solution aims to define the measurement concepts in a complete and operational form. And integrate the measurements into the different phases of the software process lifecycle.

Chapter III Problem Statement

This chapter presents the thesis problem and objectives. It also discusses the proposals that influenced this work. Furthermore, it describes the main milestones of the solution proposed to address the thesis problem.

1. Problem definition and motivation

The conclusions obtained from the previous chapter shows the essential role which software measurement plays in understanding and controlling software development process and products; Measuring the software process supports it management, continues improvement, and its capability evaluation (Muketha et al. 2010). Furthermore, measurement provides the necessary data to understand how the process is performing objectively, and how to improve it (Bobkowska 2001).

In the past few years, software engineering community has proposed many methods, standards and techniques (e.g., GQM, PMI, and ISO 15939) to guide the selection and definition of the measurement concepts to optimize the measurement process (Hetzel and Bill 1993). Unfortunately, these methods and processes stop at the point of selecting and identifying the measures and the measurement concepts that satisfy different needs (e.g., monitoring, controlling, estimating, and improving). However, they do not focus on defining the measurement concepts (e.g., indicators, measurement method, and context) in the form that support the measurement process throughout its lifecycle (Barbara A Kitchenham, Hughes, and Linkman 2001).

Defining the measurement concepts in an unambiguous and rigorous (operational) form is essential to support the collection, storing and analysis of the measurement values. Moreover, it promotes the interpretation and reporting of the measurement results in the form that support engineers and managers to adopt quantitative management, make informed decisions, and develop the improvement plan. Furthermore, the operational definition of the measurement concepts motivates and supports the integration of the measurement into the software process (Barcellos, de Almeida Falbo, and Rocha 2013).

After introducing the importance of the measurement process and its impact on the SDP, the following section summarizes the problems addressed in this thesis work:

- The first problem is related to the definition of the measurement concepts in the form that support the measurement throughout its lifecycle (Del-Río-Ortega, Resinas, and Ruiz-Cortés 2009; Habra et al. 2008; Jacquet and Abran 1997). Defining the measurement concepts in such form support the integration of the measurement into the SDP. The research tries to answer several questions to address this problem, among them: (i) What are the essential measurement concepts? And what are the necessary aspects (e.g., unit, scale type, performer, and

context data) to define these concepts operationally? (ii) How to enrich the definition of the measurement concepts in the form that support its integration into the process lifecycle? (iii) How to consolidate the existing measurement selection and definition methods to support the operational definition of measurement concepts.

- The second problem is related to the integration of measurement issues (e.g., concepts, artifacts, and activities) into the process lifecycle. To address this problem, the research focus on (i) Identify the software process and the measurement process lifecycles. (ii) Define the main activities of these lifecycles. (iii) Integrate the measurement lifecycle into the software process lifecycle.

- The third problem is related to the necessity to provide a tool to support the management of both lifecycles (i.e., software process and measurement process) (Bandara, Gable, and Rosemann 2005) in the form that enhances their integration throughout their lifecycles. Resolving this problem requires the following: (i) Study the existing process management tools. (ii) Develop a solution to integrate the management of the measurement process into these tools.

2. Objectives

After defining the scope of the problem, the following section describes the established objectives of this thesis:

1- **Identify** the characteristics that should be satisfied to define the measurement concepts operationally (in the form that supports the measurement throughout its lifecycle).

2- **Integrate** the measurement lifecycle into the process lifecycle by defining three metamodels. The first is the ***Measurement Definition Metamodel (MDMM)*** which allows defining and modeling the measurement concepts in the process modeling phase in the form that integrates these concepts into the process lifecycle. The second is the ***Measurement Execution Metamodel*** which supports the measurement during the process execution phase (e.g., collecting and validating the measures data). The third metamodel is the **monitoring metamodel** which supports the monitoring and reporting of the measurement data. And finally, ***merging*** the measurement metamodels with the process metamodels to complete the integration.

3- **Define** the required ***transformation rules*** to derive the necessary measurement artifacts (e.g., execution and monitoring models, and measurement documentation) from the measurement definition model.

4- **Develop** a ***tool*** to support the practical use of the proposed solution. This tool allows the process engineers to (i) Define and model the process and its related measurement concepts. (ii) Execute the process considering the measurement issues. (ii) Use the measurement data to support process management and improvement.

5- **Validating** the proposal by applying it in a real environment and evaluating the results of this experience.

3. Influences

This section outlines the previous works and the main technological and conceptual aspects that influenced the development of this thesis.

3.1 The conclusions obtained from the previous research.

The results obtained from the survey and the mapping studies (presented in chapter 2.1) have demonstrated the current lack of supporting the process measurement in the existing process management tools. This issue could be divided into two aspects:

On the one hand, the conceptual aspect, this aspect is related to the definition of the measurement concepts in the form that supports the measurement objectives and the alignment with the business needs. Existing process definition languages (e.g., SPEM and BPMN) only allow defining the measures in a simple and generic form (e.g., measure name, description, and value); this definition lack of defining important measurement aspects such as the information needs to be satisfied by this measure, the validation rules, and the context data. For that, it is essential to enrich the measurement definition to comprise all the necessary data to support the measurement configuration, collection, validation, analysis.

On the other hand, the integration aspect, this aspect is related to integrating the measurement into the process lifecycle in the form that use of the measurement definition to (i) Configure the measurements in the process deployment, (ii) Collect the process execution data, and (iii) Analyzing and reporting the measurements data in the process monitoring and improvement phases. Integrating the measurement in such form allows achieving the measurement goals and satisfying the organization needs.

3.2 Model-Driven Engineering

Using model-based approaches (defined in Table III.1) is a growing trend when developing software processes modeling languages (García-Borgoñón et al. 2014). Due to the high-level abstraction and code reuse (or regeneration) that provide, it seems appropriate to consider this approach when developing the solution. Adopting the model-driven approach reduce the development time, enhance the quality of the final code, and also facilitate and improve the process of applying changes or maintenance (Czarnecki and Helsen 2003; Kleppe, Warmer, and Bast 2003).

Therefore, the MDE (Schmidt 2006) paradigm constitute a fundamental element of the proposed solution; since it is the technological sustenance on which the proposed solution is developed. The basic concepts of this paradigm are briefly described below.

Model-Driven Engineering paradigm has emerged to address the complexity of the software systems to to express the concepts of the problem's domain adequately. In this line, the basic principle of MDE is *«Everything is a model»* (Bézivin 2005).

Table III.1. Modeling definition (Rothenberg et al. 1989).

«Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality»

The main idea of MDE is to use a set of models to decrease the level of abstraction (Fondement and Silaghi 2004). Thus, in the early stages of development, models with a high level of abstraction are developed to represent the relevant aspects of the system, and as the development progresses; new models are developed with a higher level of detail and closer to the final implementation of the system.

However, when working with models in this way, arise two main needs that must be met (Fondement and Silaghi 2004): (i) provide a set of common elements so that all models can be developed in the same way; and (ii) define mechanisms that make it possible to obtain models from other different models, in a systematic way and susceptive of partial or total automation.

The Metamodeling technique (Atkinson and Kühne 2002) address the first MDE need by allowing the definition of a class of models. Metamodeling explicitly defines a modeling language by specifying its syntax, and semantics. Using metamodel enables the definition of the main concepts and relationships that constitute the domain of the problem (Schmidt 2006). Thus, a metamodel encompasses the set of constructors that represent the concepts of the metamodel, the valid associations among the constructors, the existing restrictions, and also define the meaning of the represented concepts (Giachetti, Marín, and Pastor 2008). In other terminology, metamodel represents the abstract syntax while the model defines the concrete syntax.

The model transformation (Sendall and Kozaczynski 2003) technique meets the second MDE need by allowing the definition of the relationship between models (Fondement and Silaghi 2004) and deriving model from other models. A transformation between two models represents a relationship between two abstract syntaxes or metamodels. This transformation is defined by a set of relations (expressions or rules) between the elements of the corresponding metamodel (Thiry and Thirion 2009). These rules relate the elements of the input metamodel to the elements of the output metamodel. Furthermore, the transformation expression describes how to construct the elements of the output model from the input model elements.

MDE can be applied in practice in several ways, one of which is defined by the MOF *«Meta-Object Facility»* standard (Object Management Group 2002a). MOF aims to specify and manage metadata on a different level of abstraction; MOF is mainly composed of two structures: MOF class and MOF association, and considered a meta-metamodel which describes an abstract modeling language (based on UML) and could also be used to define the transformations.

In general, Figure III.1 shows how all these concepts are combined to generate new models from other existing models.

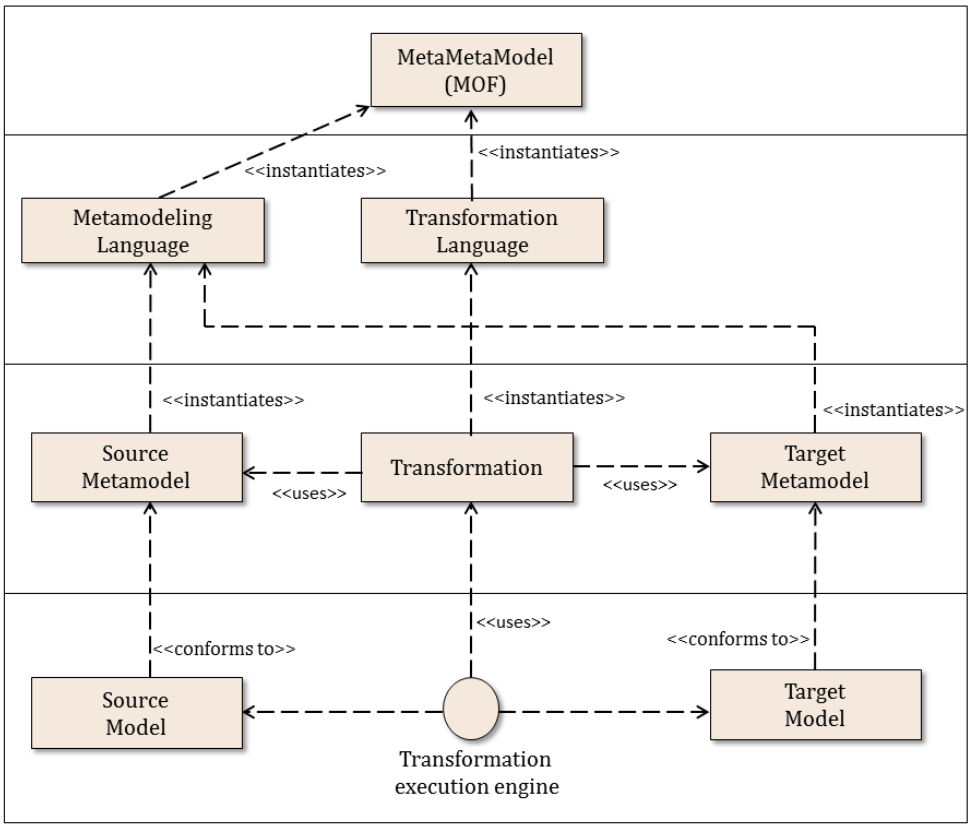

Figure III.1. Transformation process of MDE paradigm

Due to the following reasons, MDE seems appropriate to achieve the objectives of this thesis work:

- The MDE paradigm uses models to represent the information of a given domain. In the context of this thesis, using the models allows the formalization of the measurement information.
- The MDE paradigm applies transformations which are a possible tool to describe, perform, and automate the transformations between the models.
- As mentioned earlier, using model-based approaches is the current trend in the development of the software processes modeling languages.

3.3 Systems and software engineering - Measurement process [ISO/IEC/IEEE 15939:2017]

ISO/IEC/IEEE 15939:2017 (ISO/IEC/IEEE 15939 International Standard - Systems and software engineering--Measurement process 2017) provides a flexible and adaptable measurement process to support the measurement in the system and software engineering domain. The measurement process described in this standard provides a guide to specifying the measurement information needs, implementing the measures, analyzing the measurement results and also defines how the measurement results should be used. Moreover, it supports the selection of the appropriate set of measures to address specific information needs.

Furthermore, ISO/IEC/IEEE 15939:2017 define the main measurement concepts and terms. It also provides a measurement information model (Figure III.2), which describe

the link between the information needs and the measurement concepts. Moreover, it describes how the relevant measures are quantified to address the information needs and supports decision making.

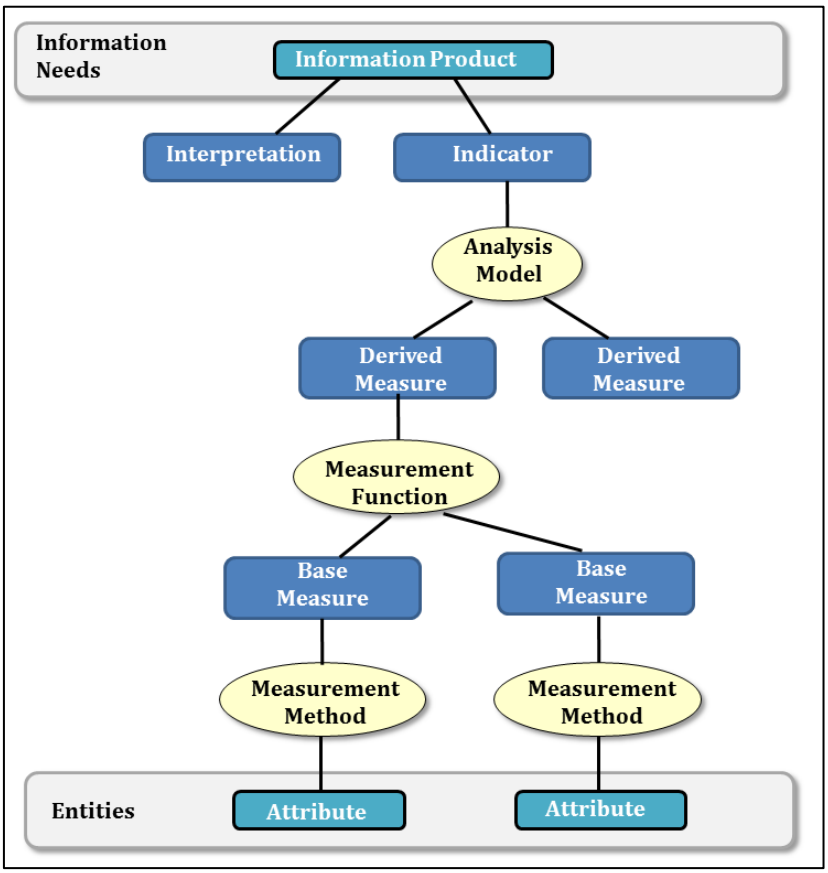

Figure III.2. Key relationships in the measurement information model (ISO/IEC/IEEE 15939 International Standard - Systems and software engineering--Measurement process 2017).

3.4 Product Lifecycle Management for Business-Software (PLM$_4$BS) framework

The PLM$_4$BS is a model driven based framework (García-Borgoñon et al. 2013; Garcia-Garcia et al. 2017) aims to model and manage software processes. It defines metamodels or domain-specific languages to define and executes processes. Furthermore, it establishes systematic protocols to support the necessary transformations between the process models.

PLM$_4$BS is based on a continuous improvement lifecycle. This lifecycle comprises four phases (Figure III.3) as described below.

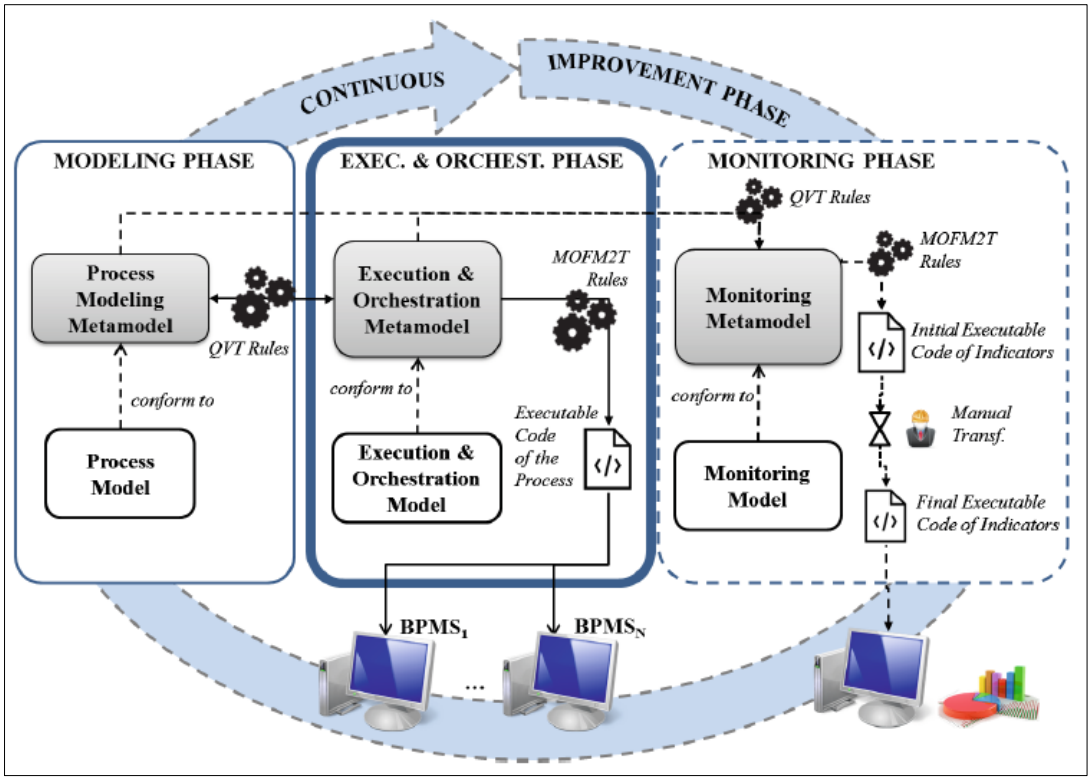

Figure III.3. PLM$_4$BS process lifecycle, based on (Garcia-Garcia et al. 2017)

Modeling phase. PLM$_4$BS provide a MOF-compliant metamodel to support the process modeling in a flexible and structured manner. As shown in Figure III.4, the metamodel defines the process and its main related concepts, such as the activities, work products, and stakeholders. It also provides the necessary elements to represent the process execution sequence (e.g., control elements and business variables). More details about this metamodel and its elements could be found in (Garcia-Garcia 2015).

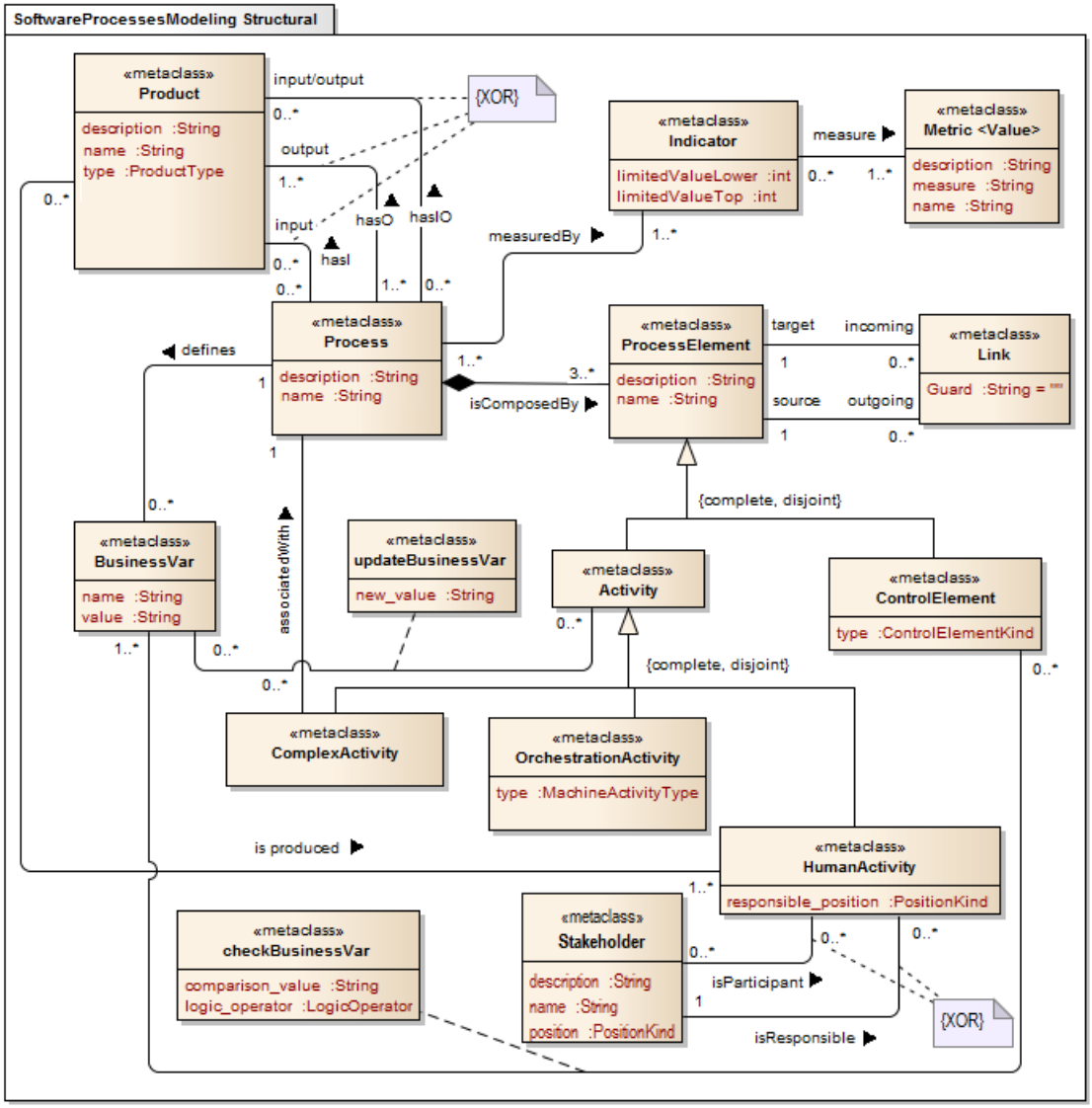

Figure III.4. PLM$_4$BS: Process definition metamodel.

Execution and Orchestration phase. In this phase, PLM$_4$BS supports the execution and orchestration of the process defined in the previous stage by allowing the engineers to specify several parameters (e.g., execution, communication, and integration parameters). Moreover, PLM$_4$BS provides an MDE mechanism to enable the execution of the defined process by BPMSs:

- Defines an execution and orchestration *metamodel* (see Figure III.5) that defines the necessary execution concepts and context to execute the process by the process execution engine. More details about this metamodel and its elements could be found in (Garcia-Garcia 2015).
- Defines a model to model transformation rules to derive the process execution model from the resulting model from the modeling phase.
- Define a model to text transformation protocol to generate executable code from the process execution model.

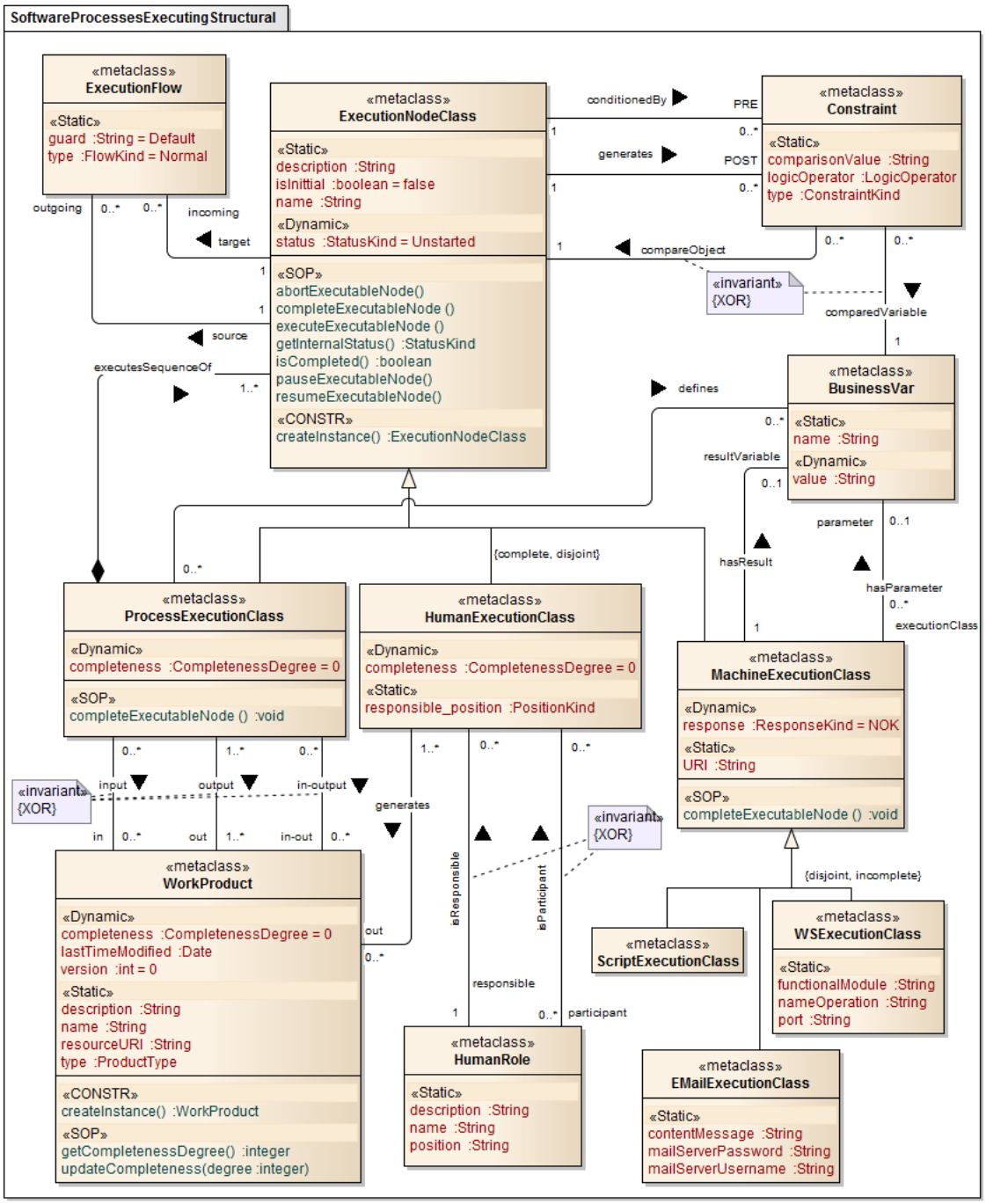

Figure III.5. PLM$_4$BS: Process execution metamodel.

Monitoring phase. PLM$_4$BS supports the evaluation of the process during its execution by using measures and indicators to provide insight into the performance of the process. For this purpose, PLM$_4$BS support the definition of the measures and indicators during the modeling phase. This phase is not implemented currently, but the tool defines the requirements and the MDE architecture to support it in the future.

Continuous Improvement phase. The data provided by the measures and indicators in the previous phase could be used as a base to guide the organization to optimize and improve several aspects of the process , such as the performance, capability, productivity, quality and maturity. This phase is not implemented currently, but the tool defines the requirements and the MDE architecture to support it in the future.

PLM4BS framework has been developed in the same research group in which this thesis work was developed; the Web Engineering and Testing Early (IWT2) research group has started the development of PLM4BS to support the evolution of the NDT (Navigational Development Techniques) (Escalona 2004; Escalona and Aragon 2008) methodology which was developed and used by the group to support the software development lifecycle. For this reason, this thesis work is influenced and motivated by this framework.

As mentioned early, PLM4BS define the process lifecycle in four phases. Currently, the framework provides support for the first two phases (modeling and execution & Orchestration) only. Therefore, the main goal of this work is to support the investigations that aim to cover the rest of the process lifecycle defined by the PLM4BS (Monitoring and improvement). Precisely, this work is part of research that seeks to promote the integration of the measurement issues into the process lifecycle (e.g., design, modeling, execution, and monitoring) in the way that support the process monitoring and improvement.

4. The proposed solution

At the beginning of this chapter, we have identified the problem addressed by this work and established its objectives. In addition, we have described the most important works that influenced this research. In this section, we take a brief look at the proposed solution, which we will explain in detail in the following chapters.

The solution introduced in this study aims to support the measurement throughout the process lifecycle, by proposing the following:

- **Measurement lifecycle.** The proposed lifecycle consists of six phases: measurement selection and Definition, Modeling, Configuration, Collection, Analysis and Reporting, and the last stage is the Evaluation and Improvement. Chapter III.4.1.2 provides more details about the proposed lifecycle and its phases.

- **Integrate the proposed measurement lifecycle into the process lifecycle**. This integration will be described in chapter III.4.1.3.

- **Measurement concepts**. Chapter III.4.2 proposes the necessary measurement concepts to support the measurement lifecycle, its operational definition, and relationships.

- **Criteria to support the selection and definition of the measurement concepts**. During the mapping study described in chapter II.1.2, we found these criteria scattered in the literature. After grouping and organizing them, we recommend its use in an integrated form to support the measurement team in the selection and definition of the measurement concepts, moreover, to control the quality of the defined measurement concepts. These criteria are detailed in chapter III.4.2.1.

- **Metamodels to support the measurement lifecycle and its integration into the process lifecycle:**

- **Measurement definition metamodel (MDMM)**. This metamodel supports the formal definition of the measurement concepts. Chapter IV.2 provides details about this metamodel.
- **Measurement execution and monitoring metamodel**. These metamodels define the measurement issues during the process execution and monitoring phases. These metamodels will be described in chapter IV.3 and chapter IV.4.

In order to evaluate the proposed solution, a supporting tool has been developed and integrated into the process management framework (PLM_4BS) to promote its application in the real environment. This tool was used later to validate the solution by evaluating the results obtained from the real project where the solution was applied. Chapter VI provides more details about the supporting tool and the proposal evaluation.

This section continues by presenting the process and the measurement lifecycles and their integration. Then introduce the proposed measurement concepts and its operational definition.

4.1 Define and integrate the lifecycles

4.1.1 Process lifecycle

Several proposals have been presented to determine and describe the main stages of the process lifecycle; chapter II.2.3 has summarized the most recent lifecycles in the literature. Each of these proposals focuses on a specific perspective according to its context of use. The following section describes the main phases of the process lifecycle:

- ***Discovery* and *Design***: In this phase, the organization applies different techniques to capture, understand, and analyze the existing processes. Moreover, during this phase, the organization defines new processes to address new needs. This phase includes several activities, such as defining and describing the main tasks, products, roles, and workflow.

- ***Modeling and simulation***: In this phase, process engineer uses formal modeling language to describe different perspectives of the process (e.g., functional, data flow, the sequence of the activities and organizational perspective) which allows the stakeholders (e.g., domain experts, developers, process engineers, customers) to understand the process without ambiguity. In this phase, stakeholders can also simulate the execution of the process to detect bottlenecks and even to compare the performance of different designs for the process.

- ***Deployment***: This phase aims to deploy the process to be ready for the execution and available to the users. Thus, in this phase, the necessary configuration is performed to address the process execution requirements and to support its integration with the execution environment and other systems.

- ***Execution***: In this phase, an instance of the process is executed to allow stakeholders to interact with it. During the execution, the process uses the

allocated resources to produce its outcomes. Also, the necessary data to evaluate the process execution is collected.

- ***Monitoring and Analysis:*** Monitoring the process performance is crucial to evaluate its overall performance. The data collected in the previous phase is used to control the process execution, analyze its performance, and for its continuous improvement.

- ***Continuous improvement***: The resulting performance analysis from the previous phase can be used to optimize the process and to improve its competitiveness. This strategy is known as continuous improvement, and it is a highly recommended by different international standards; including the ISO 9001: 2008 standard. Furthermore, the analysis results could be used as a base to support the process re-engineering activities.

4.1.2 Measurement lifecycle

In recent years, many proposals have been presented to define the measurement lifecycle. Chapter II.2.4 has described the main proposals. Below, we propose a more comprehensive measurement lifecycle. As shown in Figure III.6**,** the proposed lifecycle defines all the activities related to the measurement process:

- The first phase of the measurement lifecycle is **measurement Selection and Definition**: in this phase, the measurement objectives and concepts are defined. The measurement selection and definition methods (e.g., GQM, and PSM) are used to choose the optimal set the measurement concepts that satisfy the measurement objectives.

- **Modeling**: In this phase, the resulting measurement definition from the previous phase is represented in a formally and operational form, and also integrated into the process model. The defined measurement concepts and its relationships could be analyzed (e.g., for consistency, correlation, and causality issues) and optimized in this phase (Del-Río-Ortega et al. 2013; Popova and Sharpanskykh 2010).

- **Configuration**: In this phase, the measurement definition established in the previous stage is used to make the necessary configurations to perform the measurement; the process execution environment is prepared to allow the collection, validation and also to store the measurement data during the process execution.

- **Collection**: During the process execution, the defined measures are gathered, validated, prepared, calculated, and stored according to the definition established in the first phase.

- **Analysis and Reporting**: In this phase, measurement data is analyzed and reported according to the measurement definition; the resulting information is generated, formed and communicated to the pre-defined roles as indicated in the measurement definition.

- **Evaluation and improvement**: In this phase the measurement process is evaluated, lesson learned, and feedback about the process is gathered and assessed to discover improvement opportunities. There are many validation and evaluation frameworks (e.g., (N. E. Fenton and Pfleeger 1996; Habra et al. 2008; Barbara Kitchenham, Pfleeger, and Fenton 1995)). Also, the industry standard ISO/IEC/IEEE 15288:2015 (Martin 2015) could be used to validate the measurement from two main perspectives: *Relative verification* which evaluate the design objectives of the measurement, the necessary precision, the maturity of the available knowledge about the attribute, etc. And the *Absolute verification* that focuses on evaluating the measurement principle in itself; to ensure that the process is characterizing what it intended to measure (Habra et al. 2008).

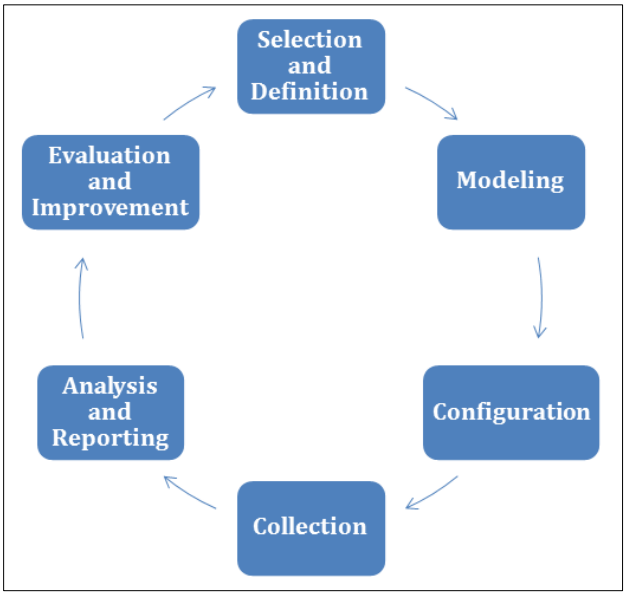

Figure III.6. Measurement process lifecycle.

4.1.3 Integrating the lifecycles

The measurement process is closely related to the software development process since the measurement process provides support for the software process in various phases throughout it lifecycle such as design and simulation, monitoring and improvement(Mora et al. 2009). Therefore, the integrated management of both lifecycles is essential to transform the organization toward quantitative management (management by facts). This integration defines the measurement activities should be carried out at each stage of the development process and also, encourages people to adopt the measurements as part of their work. (Daskalantonakis 1992; Dekkers and McQuaid 2002). The potential benefits of this integration include:

- The integration of the measurement process into the development process establishes a connection (Figure III.7) between the two parts of the development process (that is, the management process and the production process). This connection allows the flow of data from the production process to the management process, which is fundamental for management and decision making.

- Minimize the redundancy of the measurements and improve its consistency in the organization.
- Provide a clear and comprehensive measurement plan at the early stage of the development process. This plan identifies and defines the necessary measurement concepts, activities, and artifacts throughout the process life cycle.
- Promote objective communication between the stakeholders by using common concepts and terminology.
- Defining how the development process (for example, activities, stakeholders, and results) will be measured at an early stage of the development process promotes the achievement of the process objectives in terms of performance, productivity and quality. ("Tell me how you will evaluate me to tell you how I will behave. (Eliyahu and Goldratt 1990)").

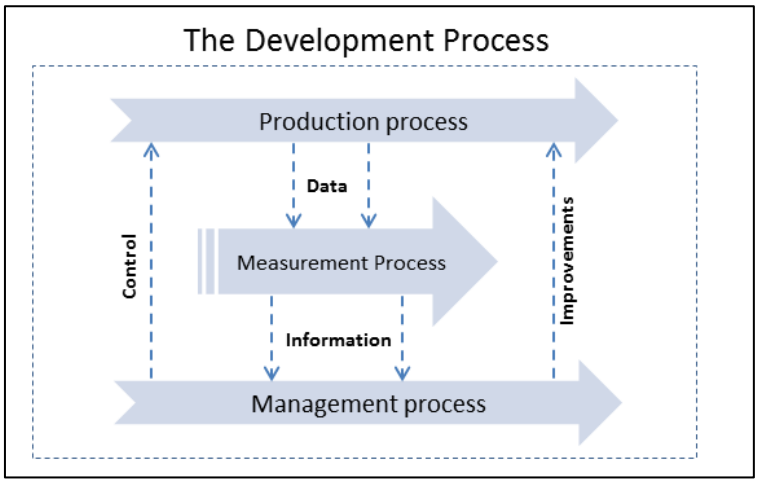

Figure III.7. Measurement process connects the two parts of the SDP.

Figure III.8 shows the relation between the software process lifecycle (defined in section 4.1.1) and the measurement process lifecycle proposed in section 4.1.2.

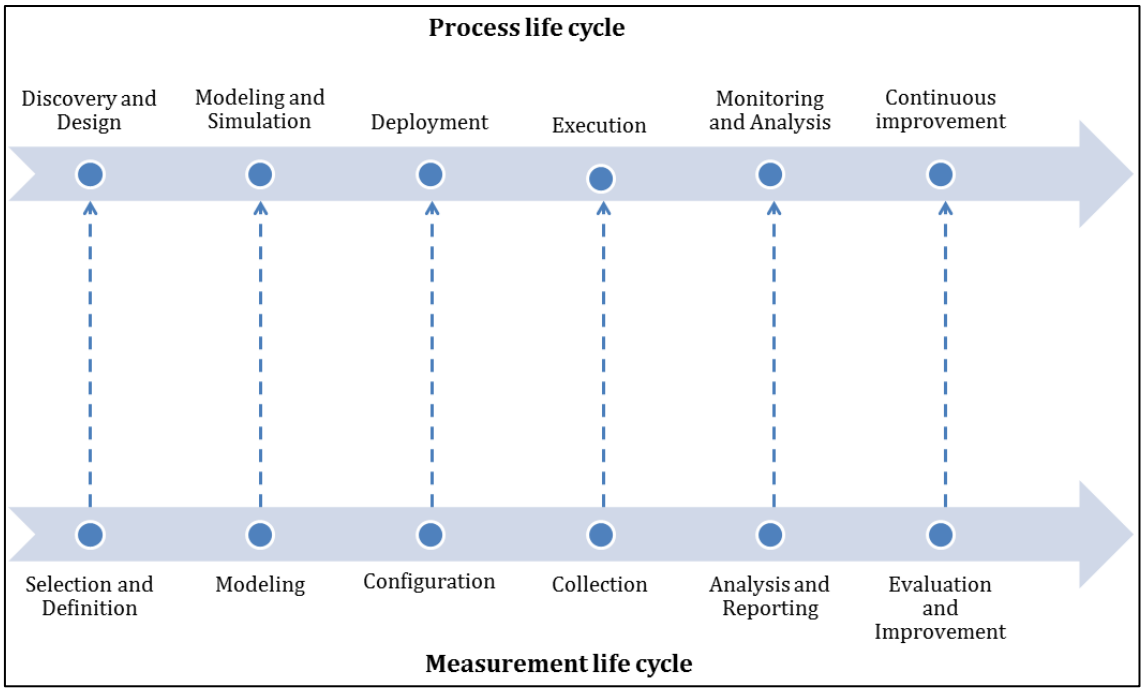

Figure III.8. Integrate the lifecycle of measurement into the lifecycle of the process.

This integration could be done by introducing the measurement activities into its corresponding phases in the software process lifecycle. The integration details are described below:

Process *discovery and design*: Throughout this phase, process engineers define the process main objectives, activities, roles, and outputs. Measurement team - in collaboration with the process engineers- can use these details to (i) Define main measurement goals and concepts. (ii) Derive the indicators and measures from these goals using measurement selection methods (e.g., GQM, GQIM, and PSM).

Taking the measurements into consideration at this stage has several benefits: (i) allow the management team to communicate their information needs, prioritize their objectives, design the format of the reports, defines the expected values, and analysis models, etc. (ii) support the measurement team to understand the measurement requirements and objectives better. (iii) Demonstrate the management's commitment to the measurement processes which is an essential success factor for the measurement process (Tahir, Rasool, and Gencel 2016).

The main output of the measurement activities in this phase is a complete and operational definition of the measurement concepts (detailed in chapter III.4.2). These defined concepts will be used to guide the measurement activities during the next phases of the software process.

Process modeling and simulation: The measurement concepts defined in the previous phase are formally defined and integrated into the process model. This formal definition promotes the success of the measurement process (Briand, Morasca, and Basili 2002; Kasunic 2006). The integration of measurement concepts in the process requires the clarification of the following details (Barcellos, de Almeida Falbo, and Rocha 2013):

- What data should be collected (e.g., entity or process element, and attribute)
- When the data should be obtained (e.g., event or frequency).
- The human role responsible for collecting the measurement data.

Furthermore, it is necessary to establish the link between the measurement concepts (e.g., measure) and the process element (e.g., entity and attribute), as well as define the interrelations between the measurement concepts (for example, the information needs and indicators, the measure and stakeholder) in the form that facilitates its traceability and prioritization.

Moreover, in this phase, the defined measures could be used to support the simulation of the process execution. This simulation evaluates several aspects of the process for different purposes such as possible improvements, changes (Magennis 2015; Sánchez González et al. 2010; H. Zhang et al. 2008), and assessment (Ruiz, Ramos, and Toro 2002). Besides, the defined measurement concepts could be used to build prediction models to estimate process characteristics (such as resources, performance, and time) and product characteristics like (product size and quality).

Process deployment: In this phase, engineers consider the measurement definition to perform the necessary configuration for collecting, validating and storing the process execution data; this configuration include: prepare questionnaires and forms to obtain the data, create connections to services and data sources, create a database to store the measurement data, and also develop the required reports and data visualization needs.

Process execution: In this phase; the data related to the process execution (e.g., resources, performance, and process outputs) is gathered, validated, and stored to be available for monitoring, control, and improvement purposes.

Process monitoring and analysis: The data collected during the process execution is monitored and analyzed to support process management and control. The following activities will be performed according to the measurement concepts defined in the early stages of the measurement process:

(i) Provide the information needs, measures, and indicators for the predetermined audience in a periodic manner.
(ii) Monitor and analyze the measurement data.
(iii) Visualize and communicate the data in the form that support the management and decision-making process.

Furthermore, in this stage, the predefined targets of the indicators are compared with the actual values (Sánchez González et al. 2010), the predefined analysis models and decision criteria are applied to support the management team to analyze the process performance(Perez-Alvarez et al. 2016), the quantitative management (Hikichi et al. 2006; X. Wang, Ren, and Liu 2017), and in-process control.

Process evaluation and improvement: The measurement data could be used in this phase to perform post-mortem analysis and compare the performance and results of the measurement process. Moreover, the measures and indicators can be used in this phase to improve, redesign and re-engineer the process (Kuwaiti and Kay 2000; Nissen 1998).

4.2 Measurement concepts

This section presents the measurement concepts, its operational definition, and also highlights the relationships among these concepts. These measurement concepts are identified and operationally defined in the first phase of the measurement process (**Measurement Selection and Definition**).

In this phase, the Measurement team -with the collaboration of the process engineers- discusses the process objectives to determine the measurement goals. Then the measurement team uses the measurement methods to derive the necessary measurement concepts to achieve these goals. These measurement concepts will be used as a guide for the measurement process throughout the rest of its lifecycle. Therefore, it is essential to select and define these measurement concepts in complete and operational form as described in Table III.2.

Table III.2. Operational definition (Park, Goethert, and Florac 1996).

« Operational definitions must satisfy two important criteria:
- *Communication: Will others know what has been measured, how it was measured, and what has been included and excluded?*
- *Repeatability: Could others, armed with the definition, repeat the measurements and get essentially the same results? »*

4.2.1 Quality criteria to support the selection and definition of the measurement concepts

The measurement selection methods (described in chapter II.2.2) provide a guide to select and define the optimal set of the measurement concepts that realize the measurement objectives. Given the importance of the outputs of the Measurement Selection and Definition phase, we propose the use of the following criteria to support the measurement team in the selection and definition of the measurement concepts, and also to enhance the quality of the output of this phase.

These criteria already exist (dispersed) in the literature (identified and grouped during the literature review described in chapter II.1.2). In this section, we propose the use of the below criteria in an integrated form through the first phase of the measurement lifecycle.

Authors in (Daskalantonakis 1992) suggest considering the following dimensions when defining the measurement concepts:

- ***Usefulness and Utility*** (simple, objective, cost-effective, and informative)
- ***Types*** (process, product, and project measures)
- ***Audiences or Users*** (e.g., Process Engineers and Software Quality Assurance)
- ***User Needs*** (obtain acceptance of the users, training users to implement and use the measures, and automating the data collection, analysis, and feedback process)
- ***Application Levels*** (e.g., organization and project)

Furthermore, according to Kitchenham (B. A. Kitchenham and A. 1996) and Winchell (Winchell 1996), the measurement concepts should comply with the following requirement criteria:

- ***Quantifiability***: If the measurement concept is not quantitative by nature, it should be transformed.
- ***Sensitivity***: Reflect how much the measurable concept should change before the change can be detected.
- ***Linearity***: Indicates the degree to which the changes in measurable concept correspond to the change in the value measure of the measure.
- ***Reliability:*** A reliable measurement is free of measurement errors. This means that the measurement value should not depend on the subjective evaluation of the individual.
- ***Efficiency***: Measurement effort should be worth from a cost/benefit point of view.
- ***Improvement -oriented***: Measurement concepts should be oriented to highlight improvement rather than alignment with instructions.

Moreover, Mendonca and Basili (Mendonca and Basili 2000) defined the following properties of the proper measurement:

- ***Sound*** (when the measures and measurement models are valid in the environment in which they are used)
- ***Complete*** (when it measures everything the users need to achieve the goals)
- ***Lean*** (when the measures are cost-effective)
- ***Consistent*** (when the measures are consistent with the user goals)

Additionally, Shahin proposes the criteria (SMART) that measurement should meet (Shahin and Mahbod 2007):

- ***Specific*** (well-defined, accurate, and explicit).
- ***Measurable*** (possible to measure its current value)
- ***Attainable*** (achievable)
- ***Relevant*** (aligned with the measurement objective)
- ***Time-bound*** (specify the measurement time period)

On the other side, the authors (Staron et al. 2016) propose a measurement quality model. This model constitutes good practices to define the measurement concepts, improve the quality of the specified measures, and also reduce the number of the measures. This quality model aims to define effective measures and reduce the data collection cost by reducing the number of collected measures.

This model is organized into five quality dimensions. Each dimension includes quality attributes related to several measurement concepts:

- ***Data analysis***: e.g., information product, interpretation, indicator and analysis model
- ***Data preparation***: quality attributes associated with the necessary mathematical operations to transform base measures into derived measures.
- ***Data collection***: describe the quality attributes related to the process of mapping the measured attribute to a value (base measure, measurement method, and attribute).

- ***Organizational reference context****:* describes the quality of the definition and reference to the organizational context (reference values for the context and the decision criteria)
- ***Standard reference model***: describes the quality of the links of the measurement concepts to the standard reference model (generic model of information Needs, the algorithm criteria, and the quality of the relation of the indicators towards the measures which are used to calculate it)

4.2.2 Measurement Information Model

After describing the quality criteria that should be considered when defining the measurement concepts, this section continues by presenting the proposed information model. This information model is based on the information model proposed in the ISO standard 15939 (ISO/IEC/IEEE 15939 International Standard - Systems and software engineering--Measurement process 2017).

The presented information model defines the measurement concepts and also describe the relationships between the information needs (measurement goals) and the necessary objective data (measures) to be collected to satisfy these needs (Card and Jones 2003).

The Measurement Information Model (MIM) shown in Figure III.9**,** demonstrates the proposed measurement concepts and their relationships from the high level 'information need' down to the measurable attributes.

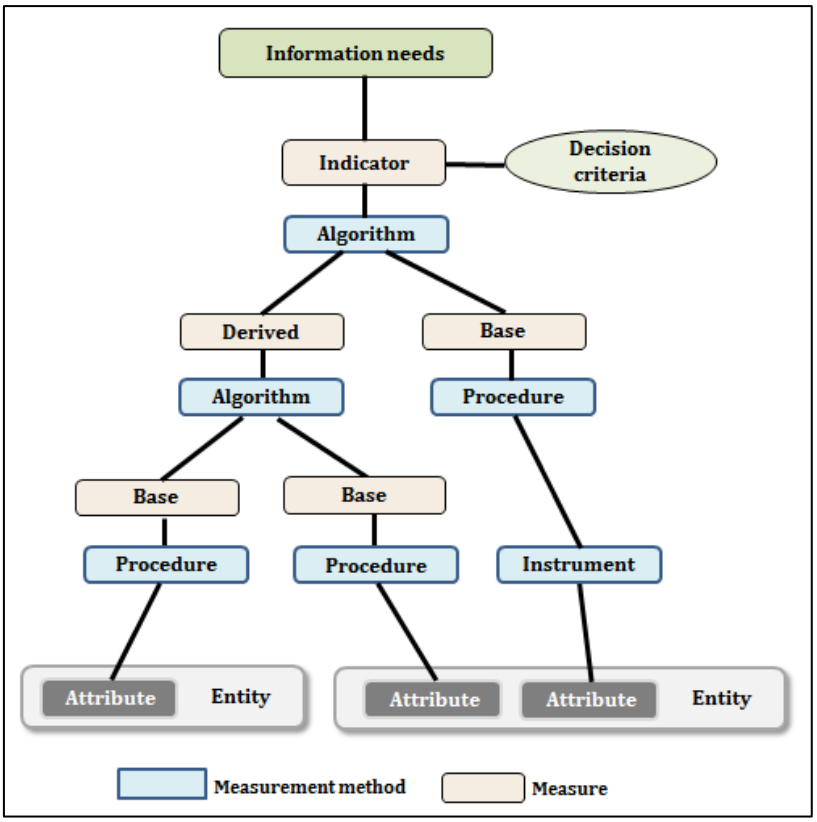

Figure III.9. Measurement Information model.

The main measurement concepts of this MIM are described below:

- ***Information needs***: Represents the required information to track an objective (e.g., improvement or performance target) or constraint (e.g., schedule, effort, or budget).

- ***Measure***: It is a value (number or symbol) assigned by mapping rules to characterize some attribute of an entity. Measures could be classified into three types or levels (Staron et al. 2016), first one (base) is used to obtain the data, second level (derived)is used to prepare the data to the analysis, and third level (indicator) is used in the analysis that satisfies the measurement requirements or needs:

    - *Base measure:* characterize and quantify the extent to which the entity possesses a certain attribute (Ordonez and Haddad 2008), defined procedures are used to determine this degree (e.g., counting the number of defects detected in a specific process phase).
    - *Derived measure:* represents a relationship or algorithm/function between multiple measures (Y. Wang et al. 2002)(e.g., productivity= size/duration).
    - *Indicator:* is a measure that provides an estimation or evaluation (using a model and decision criteria) to support the management in the analysis and decision making (ISO/IEC/IEEE 15939 International Standard - Systems and software engineering--Measurement process 2017). It apply the evaluation/estimation models (calculations or algorithm) to the measures, then, display and communicate the results to the stakeholders. The *decision criteria* (e.g., patterns, thresholds, or target values (Staron et al. 2014)) provide support for interpreting the indicator value and also to suggest actions based on the indicator results.
- ***Measurement method:*** Provide an operational description of how the measurement value will be obtained (counting rules) by describing the measurement procedure and instrument for the base measures and the algorithm for the derived measures and the indicators, and it involves:

        - *Measurement procedure:* Define the steps that should be followed to quantify an attribute. E.g., counting defects or lines of code.
        - *Measurement instrument:* Define how the measurement method is implemented to obtain the measurement value (Staron et al. 2011). *Examples*:
            - Software program to count the line of code.
            - Person or program who get data from a data source (web page, excel, database, etc.).
            - Questionnaire.
            - Checklist.
        - *Measurement algorithm:* Define the required operations to obtain the measurement value. E.g., Formula.
- ***Attribute***: is a property or characteristic of an entity, such as the size of a program, the size of requirement list, the productivity of a team, and the time required to achieve a milestone.
- ***Entity***: is an object or event (e.g., process, resource, project, or product) its attributes should be measured to achieve the measurement objectives.

4.2.3 The operational definition of the measurement concepts

This section introduces the operational definition (Table III.2) of the measurement concepts described in the previous section.

Next, we describe the proposed aspects that define these concepts in an operational form.

Information needs

- **ID:** Unique identifier,
- **Title:** Define the subject of the item,
- **Description:** Provide details to support the understandability and describe the necessity of this item,
- **Author:** Refer to the role or unit that proposed and following the item,
- **Priority:** Define the priority of the item (Berander and Jönsson 2006),
- **Accessibility:** Define who can access the item,
- **Version**: Provide traceability information about the item.

Indicator

- **ID:** Unique identifier,
- **Title:** Define the subject of the item,
- **Description:** Provide details to support the understandability and describe the necessity of this item,
- **Information needs ID**: Refer to the information need satisfied by this indicator,
- **Objective:** Define the indicator in natural language (e.g., describe relations).

  Examples:
  - Display Earned value over time
  - Show the Defect density over time
  - Show Schedule deviation rate for each phase.
  - Display downtime for each release
  - Show the mean and standard deviation of all projects productivity values
  - Display process center and limits using defect density values over time
  - Show the performed activities concerning the planned activities.
- **Measurement method:**

  Define mathematical operations and expressions to be used (if necessary) to obtain the indicator results. *Examples:*
  - Indicator = measure1
  - Indicator = average (measure_1, measure_2…, measure_n),
  - Indicator = Effort_prod1+ Effort_prod2
  - Indicator = actual cost/planned cost
- **Analysis and interpretation guide:**

  Provide the necessary details to support and guide the analysis and interpretation of the indicator results. This could include:
  - Thresholds (upper limit, center limit, low limit)
  - Color scale with the traffic light metaphor(Pandazo et al. 2010; Staron et al. 2014)
- **Decision criteria:** Define actions to be taken based on specific indicator results,
- **Interpretation:**

Provide support to interpret and understand the indicator results (e.g., if there are two consecutive points out of the low or upper limit, then this is a deviation trend, and management actions is needed to investigate this deviation.),

- **Analyst:** Assign responsibility (role or unit) for analyzing the indicator results,
- **Responsible:** Assign responsibility (e.g., project manager, product manager) for monitoring the indicator results (the audiences),
- **Accessibility:** define the role or unit which can access the indicator results (Dekkers and McQuaid 2002),
- **Priority:** Define the priority of the indicator (Berander and Jönsson 2006),
- **Scheduling:** Define when the indicator is evaluated, analyzed and reported,
- **Presentation guide:** Provide a guide to visualize and communicate the indicator results (e.g., XmR chart (Montgomery 2009) is recommended to represent data over time (e.g., daily, weekly, or monthly).

Derived measure

- **ID:** Unique identifier,
- **Title:** Define the subject of the item,
- **Description:** Provide details to support the understandability and describe the necessity of this item,
- **Measurement method**:

  Define how the measurement value is calculated. Use an algorithm to combine other measures (based and derived measures). E.g., value= (base_m1 + derived_m3)* base_m7.

Base measure

- **ID:** Unique identifier,
- **Title:** Define the subject of the item,
- **Description:** Provide details to support the understandability and describe the necessity of this item,
- **Entity:** Define the measured entity (e.g., phase, activity, work product, or team),
- **Attribute:** Define the measured attribute (e.g., cost, effort, size, or progress),
- **Scale-type:**

  The scale-type determine the type of operations and transformations that could be applied to the measured value (Habra et al. 2008). The most common scale types (ISO/IEC/IEEE 15939 International Standard - Systems and software engineering--Measurement process 2017) are:

  - *Nominal*: In this scale, the value represents labels or categories, no mathematics or order operations accepted, it could be compared only by "=" and "≠". It represents discrete value (could be mapped to integers) (e.g., defect type, defect phase, or fault source),
    - *Ordinal*: The value that represents this scale could be ordered, it represent discrete value (could be mapped to integers) (e.g., severity levels, application complexity, or defect priority),
    - *Interval*: In this scale, the values have equal distances corresponding to equal quantities of the attribute; the value could be continuous or discrete (e.g., Cyclomatic complexity has the minimum value of one, but each increment represents an additional path. The addition

(mathematical operation) has meaning in this scale. The value of zero is not allowed in this scale,

- *Ratio*: This scale is similar to the interval scale but also includes the value of (e.g., one can calculate the measurement averages in the ratio scale type) (e.g., the size in terms of the number of requirements is a ratio scale because the value of zero corresponds to no requirements and each additional requirement defined represents an equal incremental quantity),

- **Scale:**
  Define the type of measurement value (e.g., Integers from zero to infinity, positive number, decimals, or label such as experienced, not experienced),
- **Unit:**
  A measurement unit determines how the attribute is measured (Barbara Kitchenham, Pfleeger, and Fenton 1995). *Examples*:
  - The size could be measured by the units: number of line of code, function point, implemented functions/requirements or number of implemented classes,
  - Program correctness or test case could be measured by the unit: Fault rate.
  - The effort could be measured using the unit: work hours.
- **Performer:**
  Assign responsibility to role or unit for obtaining the measurement value,
- **Scheduling:**
  Define when the measurement value is obtained (collection interval) (e.g., (every week), or when an event occurs (e.g., activity complete)),
- **Measurement Method:**
  Describe how the measurement value is collected or obtained; by defining the measurement procedure and instrument,
- **Context data:**
  The context data includes the necessary information to verify, interpret, or evaluate the measurement value (Daskalantonakis 1992; ISO/IEC/IEEE 15939 International Standard - Systems and software engineering--Measurement process 2017). Examples of the context data and its categories:
  - The measured entity/ attribute: E.g., when measuring the program size (LOC) it is essential to indicate the programing language used to implement the file.
  - The measurement performer: E.g., name, and role.
  - The environment: E.g., measurement date and time, data source.
- **Validation guide:**
  Define how the collected data could be validated for correctness and consistency (e.g., describes the valid data, the range of possible data, or expected values).

Measurement Method

- **ID:** Unique identifier,
- **Title:** Define the subject of the item,

- **Description:** Provide details to support the understandability and describe the necessity of this item,
- **Measurement procedure:** Define steps to obtain the measurement value,
- **Instrument:** Define how the procedure is implemented,
- **Algorithm:** Define a formula to calculate the measurement value.

4.2.4 Example of using the proposed measurement concepts in the practice

The following scenario –based on (Staron and Meding 2009)- illustrates how to use the proposed concepts in practice.

Project management needs to know the cost situation of the project (e.g., the ratio between allocated and used budgets).

In this case, **the information needs** is understand the cost situation of the project; this need is fulfilled by **the indicator** "cost situation" which informs the management about the cost situation. This indicator defines the following analysis model and decision criteria to satisfy the management requirements.

The analysis guide or model defines three levels for the cost situation indicator. The indicator could have a "**Red-unacceptable**" level defined when the cost of the project exceeds the budget and a "**Green-acceptable**" level when the cost is up to 90% of the budget, leaving the 10% remaining to be the "**Yellow-warning**" level of the indicator.

The decision criteria associated with the indicator define the actions that must be taken when a specific criterion occurs:

- "**Red-unacceptable**". Call for meeting with project management.
- "**Green-acceptable**". Inform management
- "**Yellow-warning**". Inform management and call for meeting with the project team.

This indicator uses a **derived measure** to evaluate the cost situation by applying the **Algorithm** (e.g., calculation) **measurement method** which divides the **base measure** "current cost" by another **base measure** "planned cost". While the values of the base measures (the current cost and the planned cost) are obtained using defined **procedures** and **instruments.**

5. Chapter summary

This chapter has presented the research problem and the objectives of this thesis. Furthermore, it highlighted the previous works which influenced this proposal. Moreover, this chapter has outlined the main aspects of the proposed solution and introduced its first parts.

Next chapter will introduce the metamodels that define the proposed measurement concepts in a formal form.

Chapter IV Defining the Metamodels

The previous chapter has identified the problem addressed by this work and summarized the research objectives. Furthermore, we have outlined the main features of the proposed solution; we have introduced the process and the measurement lifecycles and described the main measurement concepts and its operational definition.

In this chapter, we describe the necessary metamodels to support the measurement lifecycle and promote its integration into the development process. Those metamodels are formally defined and represented by UML class diagrams.

This chapter is structured as follows: Section 1 introduces the proposed metamodels, describes the role of each one and explains the relationship between them. The MDMM is defined in section 2, and section 3 presents the measurement execution metamodel. In section 4 we describe the monitoring metamodel, and finally, section 5 presents the chapter summary.

1. Introduction

During the previous chapter, we have defined the process lifecycle and the measurement lifecycle, and also described the main measurement concepts, its operational definition, and its relationships. Furthermore, the previous chapter has discussed the integration of the measurement lifecycle into the process lifecycle.

This integration implies merging the measurement activities and concepts throughout the process lifecycle (e.g., discovery and design, modeling and simulation, etc.)

Therefore, this chapter supports this integration by defining the metamodels which allow defining and integrating the measurement issues into the process lifecycle (e.g., modeling and execution).

- The first metamodel is the **measurement definition** metamodel (MDMM). This metamodel support the formal definition of the measurement concepts during the measurement modeling phase. This metamodel will be defined in section 2 of this chapter.

- The second metamodel (The **measurement execution** metamodel) presents the necessary concepts to allows the integration of the measurement issues into the process execution. This metamodel provides the essential information to perform the measurement activities throughout the process execution phase.

- The third is the **monitoring metamodel**. This metamodel supports the analysis and reporting phase of the measurement lifecycle.

As shown in **Figure** IV.1, the models created with conformance to the first metamodel (i.e., the MDMM**)** will be used to derive the execution and monitoring models and the measurement documentation. We describe below how these artifacts will be generated (The transformation protocols, languages and the resulting artifacts will be explained in details in chapter V).

- The measurement **execution model**. This model uses the measurement specifications -defined in the Measurement Definition Model (MDM) - to identify the necessary measurement concepts that achieve the measurement goals (established in the MDM) during the process execution. This model supports the measurement collection phase by defining the required elements to collect, obtain, validate and store the measurement concepts specified in the measurement definition model.
- **The monitoring model**. This model is derived from the MDM to define the necessary concepts to monitor the measurement goals. This model uses the dashboard concept as a container for all the measurement goals (i.e., the information needs specified in the MDM) and also, preserves the relationship between these goals and its related measurement data.
- **Measurement documentation**. These documents provide the specifications of each measurement concept defined in the MDM. These documents will be derived from the MDM using model-to-text transformation (M2T).

The model-to-model transformation rules (M2M) will be used to derive the measurement execution and monitoring models from the MDM. The resulting models will be used to generate the necessary initial code and setting automatically to support the measurement collection and monitoring tasks. The proposed transformation process (chapter V.2) allows engineers to apply minor changes and customizations to the initial code to obtain the final measurement code.

These metamodels are formally defined and represented by UML class diagrams. Figure IV.1 demonstrate these measurement artifacts and their relationships.

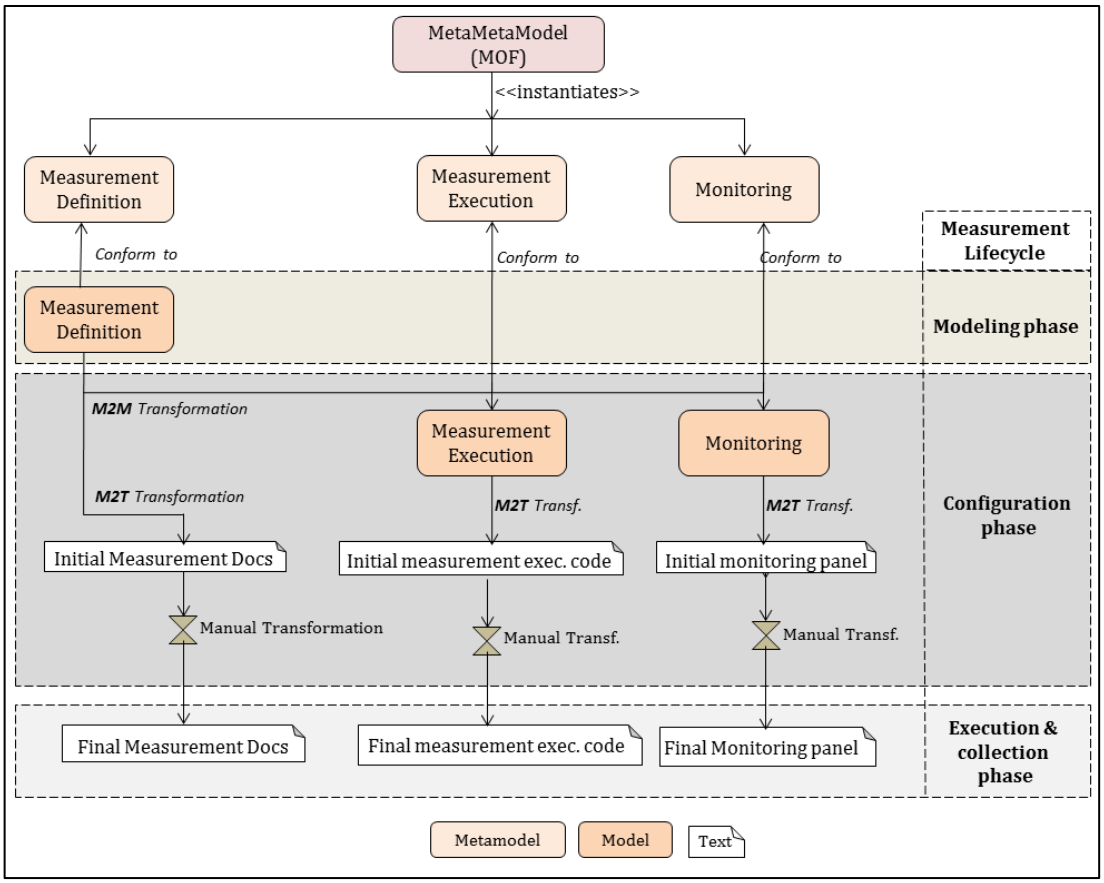

Figure IV.1. Measurement models and their relationships

During the definition of these metamodels, the priority was given to its simplicity, but without losing the horizon of practical applicability. This decision has been taken for two reasons:

1. The definition of simple languages allows its application and validation within different contexts, Thereby promoting its improvement. In this sense, the research group Web Engineering and Early Testing (IWT2) of the University of Seville - in which this thesis is developed - has extensive experience in R + D projects and technology transfer. These opportunities enable the application and validation of the results of this thesis with the aim of obtaining valuable information for its evaluation improvement.
2. The definition of simple languages also facilitates the application of MDE derivation protocols and also reduces the cognitive overload of the user when using these languages (Moody 2009; Moody and van Hillegersberg 2009).

Following the objectives of the chapter, the subsequent sections define the proposed metamodels.

2. Measurement definition metamodel (MDMM)

The MDMM is proposed to support the measurement modeling phase. This metamodel allows the engineers to define the necessary measurement concepts to achieve the

measurement goals established in the previous phase of the measurement process (Selection and Definition).

The measurement languages introduced in this section is described in the form of a MOF metamodel and presented by the UML class diagram notation. Moreover, we have defined the necessary semantic constraints -as recommended by the Object Management Group (OMG)- using the OCL language ISO/IEC 19507 (OMG 2012).

The elements of the MDMM (and the other metamodels defined in this chapter) are described in details following the style and structure template of the documents used by the OMG for its standards (e.g., the UML specification (OMG 2017)).

This style begins with a brief description of the metamodel element, its generalization (if it exists), associations, and its related restrictions. Then describes the attributes of the element, and finally, define the associations and the restrictions related to the attributes (if it exists).

As shown in Figure IV.2, the main metaclasses of the measurement definition metamodel are «*AbstractMeasure*», «*InformationNeeds*», «*MeasurementMethode*», and «*stakeholder*».

The metaclass *«AbstractMeasure»* represents a generalization of the three types of measures (base measure, derived measure, and the indicator). This metaclass defines the common attributes and relations of these metaclasses.

The *«InformationNeeds»* metaclass represents the measurement requirements or objectives. The *«Indicator»* metaclass provides the data that evaluates the measurement goals.

The metaclass *«MeasurementMethode»* define how the measure will be obtained; the base measures use this metaclass to define «*procedures*» and «*instruments*», while the derived measure and the indicator use the «*algorithm*» metaclass to obtain its values. The «*stakeholder*» metaclass represents the author, responsible, performer, or the analyzer roles in the metamodel.

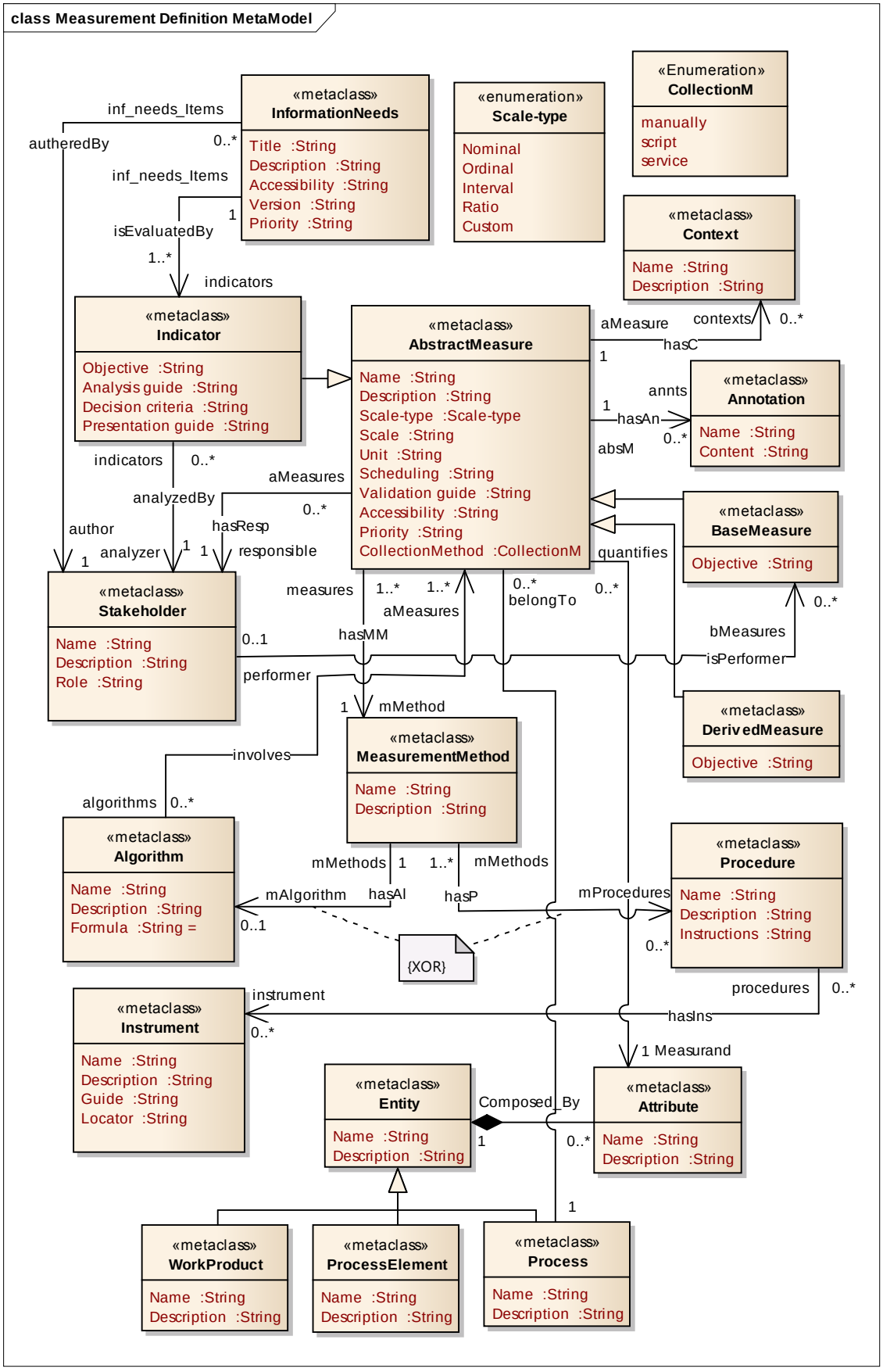

Figure IV.2. The measurement definition metamodel (MDMM).

The following section provides a detailed description of the MDMM elements.

2.1 The definition of the metamodel

In this section, we describe the elements, associations, attributes, restrictions, and operations of the MDMM.

2.1.1 Metaclass «AbstractMeasure»

Description: this metaclass is the abstract parent metaclass for all measure metaclasses (Indicator, base measure, and derived measure).
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.
- *Scale-type*: Scale-type
  Specify the scale type associated with the item. The possible values of this attribute are defined in the enumeration *«Scale-type»* section IV.3.17.
- *Scale*: string
  Specify the type of measurement value, e.g., integer, real number.
- *Unit*: string
  Specifies the unit associated with the item.
- *Scheduling*: string
  Specify when the measurement value will be obtained, analyzed or reported.
- *Validation guide*: string
  Specify how the obtained data could be validated for correctness and consistency.
- *Accessibility*: string
  Specify who can access the obtained value
- *Priority*: string
  Specifies the priority of the item
- *CollectionMethod*: CollectionM
  This attribute allows the modeler to specify the method of collecting the measurement data. The collection method enumeration *«CollectionM»* control the possible values of this attribute, which are: *«manually»*, *«script»* and *«service»,* this enumeration is described in section 2.1.19.

Operations: None.
Associations:

- *responsible*: Stakeholder [1]
  Specify the responsible for the measure.
- *contexts*: Context [0..*]
  Specify the context elements related to the measure.
- *annts*: Annotation [0..*]
  Allow the modeler to add custom attributes and notes to the measure.
- *mMethod*: Measurement Method [1]

Specify the measurement method associated with the measure.

- *algorithms*: Algorithm [0..1]
  Establish a relationship with the algorithm, where the algorithm uses the AbstractMeasure.
- *measurand*: Attribute [1]
  Specify the relation between the measure and the quantified attribute.
- *belongTo*: process [1]
  Specify a relationship between the measure and the process to which it belongs.

Restrictions: None.

2.1.2 Metaclass «BaseMeasure»

Description: This element allows quantifying a specific attribute of an entity.
Generalization: Metaclass *«AbstractMeasure»*
Attributes:

- *Objective:* string
  Define the measure in natural language.

Operations: None.
Associations:

- *performer*: Stakeholder [0..1]
  Specify the role responsible for performing the measurement.

Restrictions:

The *«MeasurementMethod»* associated with the *«BaseMeasure»* should define at least one *«mProcedures» metaclass.* The OCL expression which implements this restriction is:

```
context BaseMeasure
inv measureHasProcedure : self.mMethod.mProcedures->size()>=1
```

2.1.3 Metaclass «DerivedMeasure»

Description: This metaclass represents a relationship or algorithm/function between multiple measures (i.e., base measures or derived measures).
Generalization: Metaclass «*AbstractMeasure*»
Attributes:

- *Objective*: string
  Define the measure in natural language.

Operations: None.
Associations: None.
Restrictions:

The metaclass *«MeasurementMethod»* associated with the metaclass *«DerivedMeasure»* should define a *«mAlgorithm»*, the OCL expression which implements this restriction is:

```
context DerivedMeasure
inv measureHasAlgorithm : self.mMethod.mAlgorithm->size()>=1
```

2. 1.4 Metaclass «Indicator»

Description: This element allows the evaluation of the measurement objective based on defined analysis rules and suggests actions based on decision criteria.
Generalization: Metaclass *«AbstractMeasure»*

Attributes:

- *Objective*: string
  Define the indicator in natural language.
- *Analysis guide*: string
  Define the necessary details to support and guide the analysis and interpretation of the indicator results.
- *Decision criteria*: string
  Specify actions to be taken based on specific indicator results.
- *Presentation guide*: string
  Provide a guide about how to visualize and report the indicator results.

Operations: None.
Associations:

- *analyzer*: Stakeholder [1]
  Specify the role responsible for analyzing the item.
- *inf_needs_Items*: InformationNeeds [1]
  Establish a relationship between the indicator and the information needs which evaluate.

Restrictions: None.

2. 1.5 Metaclass «InformationNeeds»

Description: This element specifies the required information to track a goal or constraint.
Generalization: None.
Attributes:

- *Title*: string
  Specify short and concise description allow the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.
- *Accessibility*: string
  Specify who can access the item.
- *Version*: string
  Specify the item version, which provides traceability information about the item.
- *Priority*: string
  Specify the priority of the item.

Operations: None.
Associations:

- Indicators: Indicator [1...*]
  Specify the indicators that evaluate this element.
- author: Stakeholder [1]
  Specify the author of this item.

Restrictions: None.

2.1.6 Metaclass «MeasurementMethod»

Description: This element defines how the measurement value is obtained.

Generalization: None.

Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None.

Associations:

- *measures*: AbstractMeasure [1...*]
  Specify the AbstractMeasure associated with the measurement method.
- *mAlgorithm*: Algorithm [0..1]
  Specify the algorithm associated with the measurement method (if it exists).
- *mProcedure*: Procedure [0...*]
  Specify the procedure associated with the measurement method (if it exists).

Restrictions:

- It is not allowed to associate procedures and algorithm with the same Metaclass *«MeasurementMethod»*, this restriction is defined in the metamodel using the UML logical operator *«XOR»* associated with the roles: *«mProcedures»* and *«mAlgorithm»*.
- The metaclass *«MeasurementMethod»* should be associated with at least one *«mProcedures»* or one *«mAlgorithm»*, this restriction is implemented using the following OCL expression:

```
context MeasurementMethod
inv hasProcedureOrAlgorithm :
(self.mProcedures->size()>=1) or (self.mAlgorithm->size()>=1)
```

2.1.7 Metaclass «Procedure»

Description: This element specifies how the attribute of the entity is characterized.

Generalization: None.

Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.
- *Instructions*: string
  Specify steps to obtain the measurement value.

Operations: None.

Associations:

- mMethods: MeasurementMethod [1...*]
  Specify the Measurement Method associated with the procedure.
- *instrument*: Instrument [0...*]
  Specifies the instruments which implement the procedure (if it exists).

Restrictions: None.

2.1.8 Metaclass «Algorithm»

Description: This element defines a relation between measures to calculate the derived measure.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.
- *Formula*: string
  Specify the necessary mathematical operations to calculate the measurement value.

Operations: None.
Associations:

- mMethods: MeasurementMethod [1]
  Specify the Measurement Method associated with the algorithm.
- aMeasures: AbstractMeasure [1...*]
  Specify the measures associated with the algorithm.

Restrictions: None.

2.1.9 Metaclass «Stakeholder»

Description: This metaclass represents the human roles involved in the measurement activities.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details about the stakeholder, e.g., competencies and skills.
- *Role*: string
  Specify the stakeholder position and profile within the organization structure.

Operations: None.
Associations:

- *inf_needs_Items*: InformationNeeds [0…*]
  Specify the information needs authored by the stakeholder.
- *Indicators*: Indicator [0…*]
  Specify the indicators which the stakeholder is assigned as an analyst.
- *aMeasures*: AbstractMeasure [0...*]
  Specify the measures which the stakeholder is assigned as a responsible
- bMeasures: BaseMeasure [0...*]
  Specify the measures which the stakeholder is assigned as a performer.

Restrictions: None.

2.1.10 Metaclass «Instrument»

Description: This element represents the necessary instruments to obtain the measurement value.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.
- *Guide*: string
  Specify how the instrument will be used to obtain the measurement value.
- *Locator*: string
  Specify the instrument location, e.g., URI.

Operations: None.
Associations:

- *procedures*: Procedure [0...*]
  Represent the procedures which use the instrument.

Restrictions: None.

2.1.11 Metaclass «Context»

Description: This metaclass represents the necessary information to verify, interpret, or evaluate the measurement value.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None.
Associations:

- *aMeasure*: AbstractMeasure[1]
  Specify the AbstractMeasure element related to the context element.

Restrictions: None.

2.1.12 Metaclass «Annotation»

Description: This element allows the user to add more notes or attributes to define the measure item.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Content*: string
  Represent the annotation content.

Operations: None.
Associations:

- *absM*: AbstractMeasure [1]
  Specify the AbstractMeasure element related to the annotation elements.

Restrictions: None.

2.1.13 Metaclass «Entity»

Description: This item represents the entity that is being measured.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None.
Associations:

- *Composed_By*: Attribute[0...*] (*composition association*)
  Establish the relationship between the entity and the attribute(s).

Restrictions: None.

2.1.14 Metaclass «Attribute»

Description: This item represents the attribute that is being measured.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None.
Associations:

- *Quantifies*: AbstractMeasure [0...*]
  Specify the measures that quantify the attribute.
- *Composed_By*: Entity[1] (*composition association*)
  Establish a relationship between the attribute and the entity.

Restrictions: None.

2.1.15 Metaclass «Process»

Description: This item represents the process that is being measured (or one of its elements).
Generalization: Metaclass *«Entity»*.
Attributes:

- *Name*: string

Specify a brief and concise description that allows the clear and unambiguous identification of the element.

- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None.
Associations:

- *belongTo*: AbstractMeasure [0...*]
  Specify the measures that belong to the process.

Restrictions: None

2.1.16 Metaclass «ProcessElement»

Description: This element represents any process's element (e.g., activity, resource, or stakeholder).
Generalization: Metaclass *«Entity»*.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None.
Associations: None.
Restrictions: None.

2.1.17 Metaclass «WorkProduct»

Description: This element represents any work product involved in the process.
Generalization: Metaclass *«Entity»*.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None.
Associations: None.
Restrictions: None.

2.1.18 Enumeration «Scale-type»

Description: This enumeration classifies the measurement scale type into four levels: nominal, ordinal, interval, or ratio. Furthermore, it allows the engineer to define more measurement scale types as necessary using custom value.

2.1.19 Enumeration « CollectionM »

Description: This enumeration defines the possible values of the collection methods, which are: *«manually»* to specify that the measurement data will be collected manually,

«script» which means that the data will be automatically collected using a script, and *«service»* in this case, the data will be collected automatically by invoking a service.

3. Measurement Execution metamodel

This section describes the proposed Measurement Execution Metamodel. This metamodel supports the measurement subjects during the process execution.

The Metamodel defines how the measurement activities will be carried out through the process execution. It supports the integration of the measurement issues into the process execution by providing the necessary concepts and specifications to achieve the measurement goals (e.g., collecting, validating and storing the measurement data) during the process execution.

This metamodel uses the measurement specifications and goals defined in the MDM to (i) derive the necessary activities to achieve these goals. (ii) provide the essential information and configurations to accomplish these activities.

These measurement activities could be integrated into the rest of the process activities to be performed together through the process execution phase.

Furthermore, it provides necessary information (e.g., user guides and instructions) to realize the manual measurement activities and the configurations required to achieve the measurement machine activities (non-human activities, such as scripts and services).

Figure IV.3 shows the elements that represent the measurement subject through the process execution together with the relationships that exist between them. These elements are described in the form of a MOF metamodel and presented by UML class diagram notation.

The main metaclasses of this metamodel are the «*Measurement Activity*» metaclass; this metaclass represents the necessary measurement activity to achieve a measurement goal. This metaclass is specialized in two types of activities («*MeasurementHumanActivity*» and «*MeasurementMachineActivities*») based in the form of performing the activity.

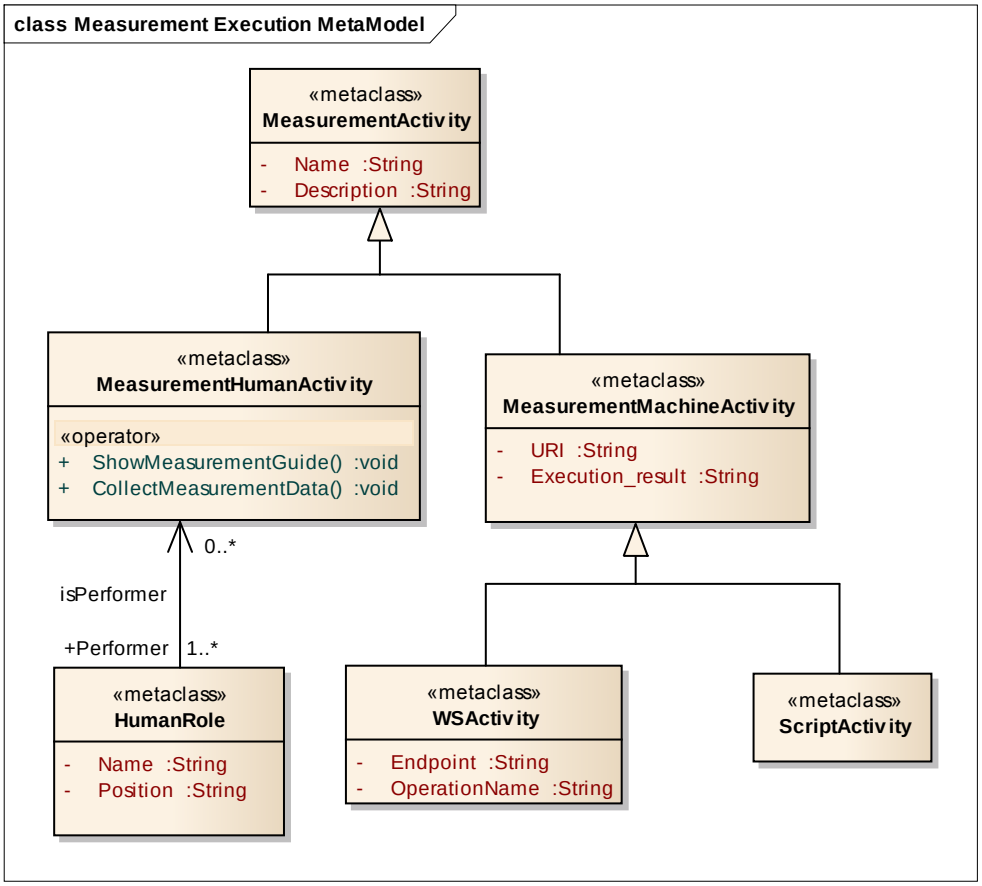

Figure IV.3. Measurement execution Metamodel

The following section provides a detailed description of the measurement execution metamodel elements.

3.1 The definition of the metamodel

3.1.1 Metaclass «MeasurementActivity»

Description: This metaclass represents the abstract parent class for the different types of measurement activities metaclasses (Human Activities and Machine Activities). This element represents the measurement tasks to be performed to achieve the measurement goals during the process execution.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None.
Associations: None.
Restrictions: None.

3.1.2 Metaclass « MeasurementHumanActivity»

Description: This metaclass represents the measurement activities that are performed by human roles.
Generalization: Metaclass *« MeasurementActivity».*
Attributes: None.
Operations:

- *ShowMeasurementGuide: void*
  This operation allows the performer of the activity to check the measurement definition and the measurement method related to the measurement tasks. This documentation is derived from the MDM as HTML files (see chapter V.4.3). This operation could be implemented using the programming language supported by the execution environment, for example in the case of using Java, the following Java statement open an HTML file in the default system browser:
  ```
  Desktop.getDesktop().browse(elementURL);
  ```
- *CollectMeasurementData: void*
  This operation allows the activity performer to set the measurement data. This operation could be implemented to open predefined forms to allow introducing and validating the measurement data then store this data using SQL "*insert"* statement.

Associations:

- *Performer*: *HumanRole* [1...*]
  The human agent(s) who perform the activity.

Restrictions: None.

3.1.3 Metaclass « MeasurementMachineActivity»

Description: This metaclass represents the abstract parent class for the different types of the measurement machine activities metaclasses (i.e., web service activity and script activity). This element represents the measurement tasks which do not need human intervention.
Generalization: Metaclass *«MeasurementActivity».*
Attributes:

- *URI:* string
  Establish the location of the resource or service accessed during the activity execution. This data is known as Uniform Resource Identifier or more commonly by its acronym URI.
- *Execution_result*: string
  Represent the state of the measurement activity after carrying out its task.

Operations: None.
Associations: None.
Restrictions: None.

3.1.4 Metaclass «WSActivity»

Description: This metaclass represents, in a specific way, the execution element which defines the requirements to invoke a web service.
Generalization: Metaclass *«MeasurementMachineActivity»*
Attributes:

- *Endpoint:* string
  Define the address or connection point (e.g., URL string) to the Web Service. This data is necessary to execute the web service.
- *OperationName*: string
  Define the method or function which will be executed.

Operations: None.
Associations: None.
Restrictions: None.

3.1.5 Metaclass « ScriptActivity»

Description: This metaclass represents, in a specific way, the execution element which defines the requirements to invoke a script. The location of this script is provided through the URI of its super metaclass.
Generalization: Metaclass *«MeasurementMachineActivity»*
Attributes: None.
Operations: None.
Associations: None.
Restrictions: None.

3.1.6 Metaclass « HumanRole»

Description: This metaclass represents the human role that performs the measurement human activities.
Generalization: None.
Attributes:

- *Name*: string
  Provide a short and concise identification of the human agent.
- *Position*: string
  Represent the human role within the organization structure.

Operations: None.
Associations:

- *isPerformer*: *MeasurementHumanActivity* [0...*]
  Represent a set of measurement activities carried out by the human role.

Restrictions: None.

4. The Monitoring metamodel

This metamodel supports the monitoring of the measurement goals (information needs) during the measurement analysis and reporting phase. The main activities of this phase are visualizing, analyzing and reporting the measurement data.

The proposed metamodel to support this phase provides a global view of the established measurement goals and its related measurement data by using the dashboards (control panel). The dashboard is the tool that organizes and visualize the information in a simple and structured form to facilitates its reading, understanding, and interpretation (Chowdhary et al. 2006). It allows the analyst to view the measurement goals and the indicators that satisfy these goals, also support the analyst by providing the necessary

analysis guide and decision criteria to understand, interpret and act based on the obtained conclusions.

This metamodel extracts the necessary measurement concepts and relationships from the MDM.

Furthermore, it uses the data defined in the MDM to build a data model. This model relates every information needs (measurement goal) to its related indicators. This solution links the measurement goals with its related measurement data and also supports displaying the measurement goals and data in a hierarchical form, which allow navigating the goals and their related measurement data.

Once the monitoring model is defined, it will be transformed automatically into an XML representation. This transformation is defined in the form that preserves the information of all the elements and relations of the model.

The XML representation of the monitoring model could be used - by code generators - to generate the necessary code to visualize the monitoring data.

Figure IV.4 shows the elements that represent the monitoring metamodel; these elements are described in the form of a MOF metamodel and presented by UML class diagram notation.

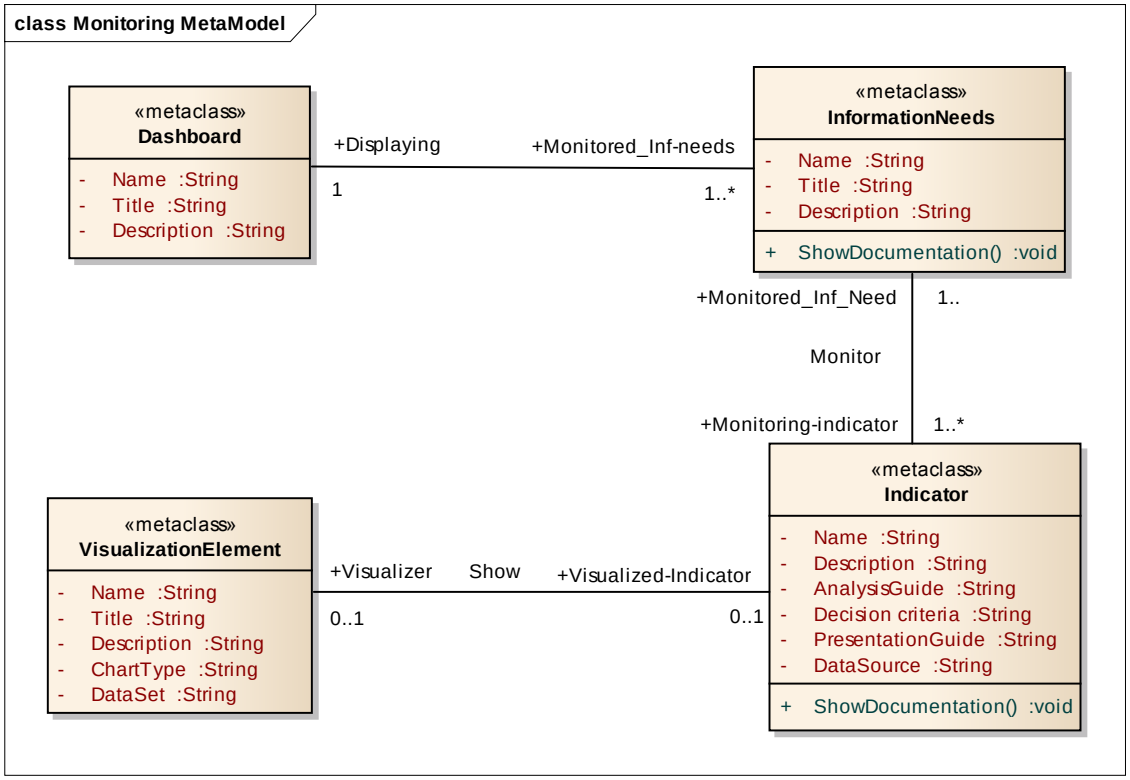

Figure IV.4. The Monitoring Metamodel.

The following section provides a detailed description of the measurement execution metamodel elements.

4.1 The definition of the monitoring metamodel

4.1.1 Metaclass «Dashboard»

Description: This metaclass is the central element in the metamodel, represents the panel which group the measurement goals and its related data. This element is responsible for organizing and visualizing the information to be analyzed.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Title*: string
  Specify a brief and concise explanation appended to the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations: None
Associations:

- *Monitored_Inf-needs*: InformationNeeds [1...*]
  Specify the Information needs monitored by the Dashboard.

Restrictions: None.

4.1.2 Metaclass « InformationNeeds»

Description: This metaclass represents the measurement goals which will be monitored and analyzed by the dashboard.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Title*: string
  Specify a brief and concise explanation appended to the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.

Operations:

- *ShowDocumentation: void*
  Open the documentation that shows the specification of the element. This documentation is derived from the MDM as HTML files (see chapter V.4.3). This operation could be implemented using the programming language supported by the execution environment. For example in the case of using Java, the following Java statement show the HTML file in the default system browser:

```
Desktop.getDesktop().browse(elementURL);
```

Associations:

- *Displaying*: Dashboard [1]
  Specify the Dashboard which monitors the information needs.
- *Monitoring-indicator*: Indicator [1...*]
  Specify the indicator elements which monitor this element.

Restrictions: None.

4.1.3 Metaclass « Indicator»

Description: This metaclass represents the indicators which monitor the defined measurement goals.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Description*: string
  Provide details to support the understandability and the necessity of the item.
- *Analysis guide*: string
  Define the necessary information to assist and guide the analysis and interpretation of the indicator results.
- *Decision criteria*: string
  Specify actions to be taken based on specific indicator results.
- *Presentation guide*: string
  Provide a guide about how to visualize and report the indicator results.
- *DataSource*: string
  SQL and /or connection statement to return the measurement data related to the element.

Operations:

- *ShowDocumentation: void*
  Open the documentation that shows the specification of the element. This documentation is derived from the MDM as HTML files (see chapter V.4.3). This operation could be implemented using the programming language supported by the execution environment. For example in the case of using Java, the following Java statement show the HTML file in the default system browser:
  ```
  Desktop.getDesktop().browse(elementURL);
  ```

Associations:

- *Monitored_Inf_Need*: InformationNeed*s* [1]
  Specify the information needs elements monitored by this indicator element.
- *Visualizer:* VisualizationElement [0...1]
  Specify the visualization element which is defined to represents the indicator data.

Restrictions: None.

4.1.5 Metaclass «VisualizationElement»

Description: This metaclass represents the element which visualizes the indicator data.
Generalization: None.
Attributes:

- *Name*: string
  Specify a brief and concise description that allows the clear and unambiguous identification of the element.
- *Title*: string
  Specify a short and concise explanation of the element.

- *Description*: string
  Provide details to support the understandability and the necessity of the item.
- *ChartType*: string
  Specify how the indicator data will be presented.
- *DataSet*: string
  Specify the data to be displayed.

Operations: None

Associations:

- *Visualized-Indicator:* Indicator [0...1]
  Specify the indicator elements presented using this element.

Restrictions: None.

5. Chapter summary

Throughout this chapter, we have introduced the necessary metamodels to define the measurement concepts and relationships in a formal manner. These metamodels were formally defined and represented by UML class diagrams.

This formal definition of the measurement concepts allows establishing derivation relationships between the defined metamodels. These derivation relationships can automate the integration of measurement concepts, artifacts, and activities into the process life cycle. These derivation relationships will be defined in the following chapter.

Chapter V Derivations between Models

Previous chapters have described the main phases of the proposed measurement process lifecycle and the main concepts that support this lifecycle. Besides, they have introduced the UML metamodels that support the operational and formal definition of these measurement concepts.

The implementation of the measurement process and its integration into the process lifecycle require several artifacts, such as execution code and measurement documentation.

This chapter aims to describe how the required artifacts to support the measurement process throughout its life cycle are derived from the models that conform to the metamodels defined in the previous chapter. In addition, this chapter describes the required transformations or derivation rules to obtain these artifacts.

For this purpose, the chapter is organized as follows. The first section presents a brief introduction to the main concepts of the MDE transformations, and it also provides an overview of the transformation languages used in this work. Next section offers a general vision about the relation between the different models and artifacts, as well as the necessary transformations to generate the target models and artifacts. The third section formalizes the necessary model-to-model transformations using QVT. The fourth section formalizes the necessary model-to-text transformations using MOFM2T. And finally, we summarize the most relevant conclusions.

1. Introduction

This section presents the main MDE transformation concepts. It also reviews the fundamental transformation approaches. Moreover, it provides a brief description of the transformation languages used in this work.

1.1 MDE transformation concepts and languages

As mentioned in chapter III.3.2, model transformation or derivation is one of the fundamental components of the MDE paradigm because it allows the derivation of different artifacts from the defined models.

Kleppe et al. (Kleppe, Warmer, and Bast 2003) define model transformation as the automatic generation of a target model(s) from a source model(s) according to predefined procedures. These procedures include the transformation rules which describe how one or more construct(s) of a source model(s) can be converted into one or more construct(s) in the destination model(s). Model transformations support different activities in the software and system development, such as abstraction, querying, translation, analysis, refactoring, optimization, merging, and debugging (Bagherzadeh, Hili, and Dingel 2017;

Lúcio et al. 2016). Model transformation could be classified into three main types (Kahani et al. 2018):

1.1.1 Model-to-Model (M2M) transformations

This type of transformations uses one (or more) source model to generate different kinds of model(s) in different languages and on different levels of abstraction. M2M transformation approaches can be divided into three paradigms based on their focus (Kahani et al. 2018; Prakash et al. 2006):

- Relational/Declarative Approaches. The languages and tools that use this paradigm focus on *which* elements of the source models should be transformed into its corresponding elements in the target model and define a relationship between these two elements, without specifying directly *how* this transformation should be performed. Languages and tools implement this paradigm could be logical or functional; **Logical transformation languages** implement the characteristics which support the relational approach such as searching, constraint propagation, and backtracking. **In functional languages**, the transformation from input(s) to output(s) models is described as a set of functions which maps the elements of the input(s) model(s) to the corresponding elements in the output(s) model(s). *Query/View/Transformation* Relational (QVTr) (Object Management Group 2008b) is an example of a relational model transformation language, its provide a textual and graphical concrete syntax. In QVTr, a relation is specified using two or more domains, with a pair of when and where clauses. Each domain represents a (partial) model in the transformation. The when clause specifies the conditions under which the transformation relationship exists, and the where clause defines the conditions that must be met by all model elements to apply the transformation. Patterns are used to define the domains, which can be marked as either check-only or enforced. In the check-only mode, the consistency of the output model elements is checked. In the case of a false result, the rule is enforced by modifying the elements of the output model to make the output consistent with the input model.

- Imperative/Operational Approaches. The languages of this paradigm focus on how and when the transformation should be performed, without focusing on the relations that must exist between the source and target elements. The imperative languages specify transformations as a sequentially executed list of actions or rules. QVT Operational (QVTo) (Object Management Group 2008b) is an example of the imperative transformation language, In QVTo, transformations are specified as a set of mapping functions, each mapping transforms one (or more) input model element(s) to one (or more) output model element(s). QVTo mappings, similarly to QVTr relations, can have when and where clauses to limit their application.

- Graph-based Approaches. The transformation languages that follow the graph-based paradigm represent the input and output models using variations of typed and attributed graphs. A graph transformation consists of a set of graph transformation rules (also called rewriting rules or production rules), which are applied to an input (host) graph to produce an output graph. GROOVE (Kastenberg and Rensink 2006) is an example of the tools that belong to this category.

- Hybrid Approaches. Approaches of this type allow developers to combine and mix different transformation approaches (e.g., imperative, operational, graph-based, etc.) when developing model transformations. For example, ATL (Jouault et al. 2008) combine the relational and imperative approaches. Enterprise Architect (Sparx Systems 2018) is an example of the tools that support hybrid approaches for building model transformations.

1.1.2 Model-to-Text (M2T) transformations.

This type allows the transformation of one (or more) model to textual artifacts such as code, reports, or documentation. M2T transformation tools could be classified based on its implementation approach into (Kahani et al. 2018) :

- Visitor-based Approaches. These approaches generate the text output by traversing a tree-based internal representation of the input model. Kermeta2 (Fleurey et al. 2006) and Melange (Degueule et al. 2015) are examples of the tools that apply these approaches.

- Template-based Approaches. Approaches in this category use templates to specify the output text resulting from the transformation of the source model(s). The template consists of static text (i.e., target text that will be generated in common for any given input model) and placeholders for data to be extracted from the input model. MOF Model to Text Transformation Language (MOFM2T) (Object Management Group 2008a) is an example of these approaches; it facilitates template composition and modular organization to handle complex M2T transformations. Enterprise Architect is an example of the tools that support these approaches.

- Hybrid Approaches. Approaches belong to this category combine the above two approaches. Actifsource (Actifsource 2018) and MetaEdit+ (Kelly, Lyytinen, and Rossi 1996) are examples of the tools that apply hybrid M2T transformation approaches.

1.1.3 Text-to-Model (T2M) transformations.

This type of transformation allows the generation of models based on textual artifacts. These kinds of transformations are usually used in reverse engineering and are mostly based on compiler/parser tools. This kind of transformation is out of the scope of this thesis because it is not used in this work.

Previous classifications demonstrate that there are a high number of different transformation approaches, languages, and tools that can be used. Therefore, the process of choosing between them is not an easy task; the appropriate language or tool depends mainly on the complexity and the type of the required transformations. It also depends on the background of who will use this language or tool.

We have chosen the QVT and MOFM2T languages to implement the necessary M2M and M2T transformations for this work. These languages were selected because they are the standard languages proposed by the Object Management Group, also because the related tools (which we plan to merge and integrate our proposal with) use these standard languages. The PLM4BS described in chapter III.3.4 is an example of these tools. Therefore,

using these transformation languages in this proposal can promote the compatibility between these proposals.

Next sections provide a brief introduction to these languages (QVT and MOFM2T).

1.2 Query/View/Transformation (QVT): Model-to-Model transformation language

Query/View/Transformation (QVT) is a language to define the rules which specify how an output model is derived from an input model.

As Figure V.1 shows, QVT is not a single language but its specification defines three different languages, which establish a hybrid (declarative/imperative) approach: The *relational and declarative language* support complex object pattern matching which allow the definition of patterns to be matched with the model elements during the execution. The *core language*, also defined as a declarative language to supports the pattern matching. And finally, the *operational mappings language*, which is an extension and complement of the relational and core languages to incorporate imperative constructions such as conditional, loops, etc.

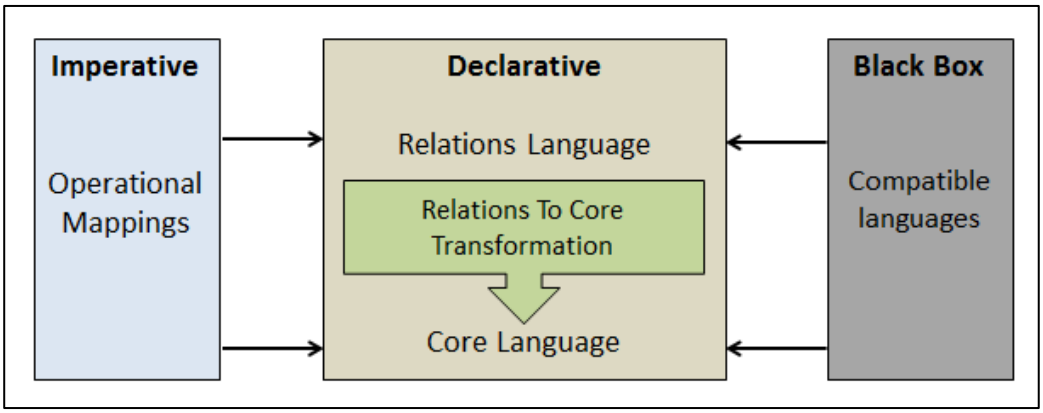

Figure V.1. QVT architecture and relations

Furthermore, QVT integrates the OCL language which allows a more procedural style and a concrete syntax that looks familiar to imperative programmers. In addition, it introduces the Black Box concept which enables the use of an external programming language for the implementation of complex algorithms and the reuse of external libraries, such as XQuery or Java.

The definition of the QVT transformation rules is based on the elements of the input metamodel. Then, these rules are applied to the instances of this metamodel to derive the output metamodel elements.

Regarding its definition, a QVT transformation rule contains a header and the specification of all necessary keys, auxiliary mappings, and functions. The header indicates which are the input and output metamodels. These metamodels allow the specification of which model type the transformation can be applied.

QVT language also allows the definition of auxiliary functions that can be referenced from any mapping. There are two types of auxiliary functions: helpers and queries. The

main difference between them is that a helper function does not define the return of any value while a query function does.

1.3 MOFM2T: Model-to-Text transformation language

MOF Model to Text Transformation Language (MOFM2T) is a standard language proposed by the Object Management Group. This standard addresses the transformation of a model into textual artifacts such as code, documents, etc.

For this purpose, MOFM2T define the transformation rule as a template. The template specifies the text which will be generated from the input model as a text with placeholders for data to be extracted from the input model.

In addition to the transformation templates, the MOFM2T language also allows the definition of auxiliary functions that can be referenced from any template.

2. The relationships between the metamodels and the transformation process

This section describes the relationships between the metamodels defined in the previous chapter. Moreover, it outlines the necessary transformation processes to obtain the output artifacts.

As shown in Figure IV.1 in the previous chapter, there are semantic relationships between the defined metamodels. Once the process engineer specifies the MDM, this model will be used to generate the necessary artifacts to support the measurement process lifecycle.

The transformation process consists of two phases; in the first phase, the transformation rules (M2M/M2T) are applied on the source model to obtain the basic artifact (e.g., model, documentation, code, etc.). The resulting artifact from this phase needs to be reviewed and modified by the process engineers to ensure that it is adapted, in the best possible way, to the execution environment. Thus the second phase of the transformation process allows the experts to make the appropriate manual modifications on the basic artifact to obtain what can be called the final artifact.

As mentioned above, the MDM will be used to generate the following basic artifacts:

- The (basic) measurement execution metamodel (M2M transformation)
- The (basic) monitoring model (M2M transformation)
- The (basic) documentation of the measurement concepts (M2T transformation)

After applying the second phase of the transformation process to obtain the final artifacts, the resulting models (i.e., final execution model and final monitoring model) will be transformed -as shown by Figure V.2- using M2T transformation rules –applying the two phases of the transformation process- to obtain the final execution code and monitoring panel.

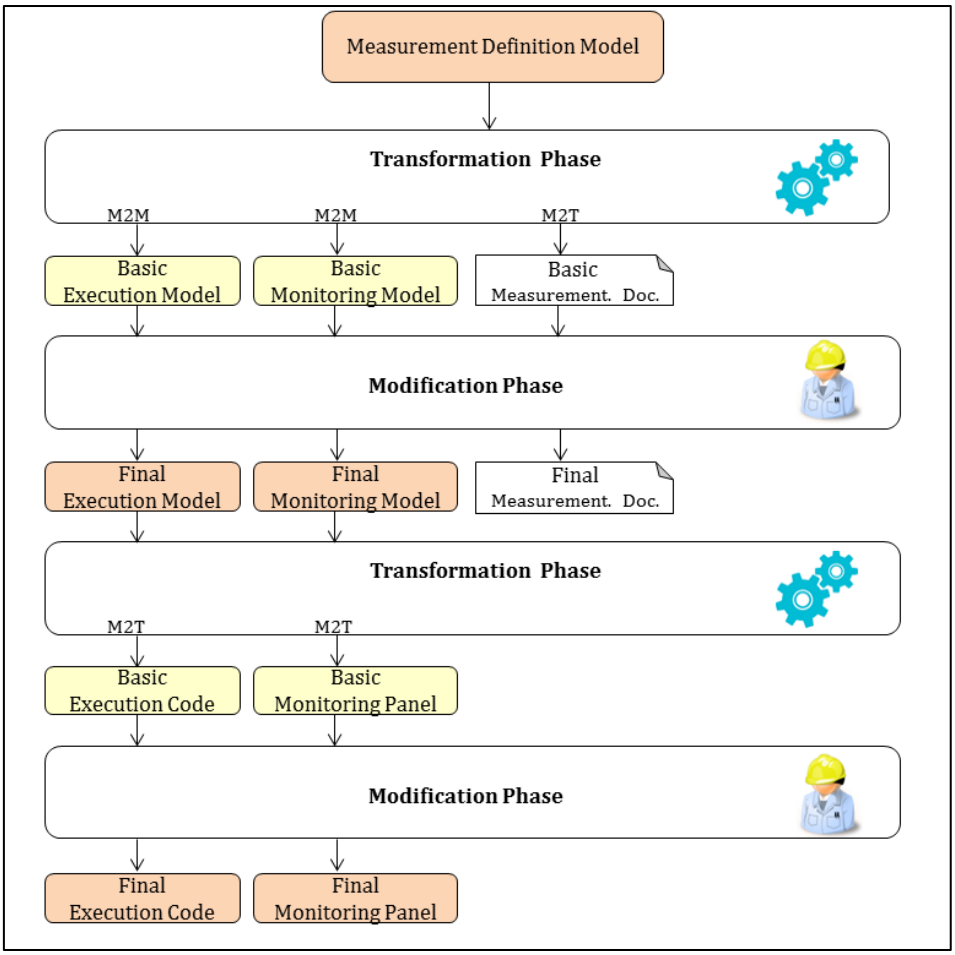

Figure V.2. Transformations map to obtain the final artifacts

Next sections describe the required M2M and M2T transformation rules to obtain the target artifacts.

3. Model-to-Model transformations

This section describes how to apply the transformation process defined in the previous section to derive the measurement execution and monitoring models. Moreover, it defines the required M2M transformation rules, it also outlines the necessary modifications to obtain the final measurement models.

Sections 3.1 and 3.2 specifies the necessary QVT transformation rules to derive the basic execution and monitoring models from the MDM. The resulting models from applying these rules need to be reviewed and modified -in the second phase of the transformation process- to obtain the final version of these models. These reviews and modification activities are outlined in section 3.3.

3.1 Measurement definition model to Measurement execution Model

As a previous step to the design and the planning of the required QVT transformation rules to derive the measurement execution model from the MDM, it is convenient to define the relationships between the elements of both metamodels. Table V.1 highlights these relationships.

Table V.1. The relationships between the definition model elements and the elements of the execution model.

Definition Metamodel Elements	Execution Metamodel Elements	Comments
«AbstractMeasure»	--	
«BaseMeasure»	*«MeasurementHumanActivity»* *«ScriptActivity»* *«WSActivity»*	This relationship is established based on the value of the property "CollectionMethod" of the metaclass *«AbstractMeasure»*. In case the defined value is *«CollectionType=manually»* then the measurement data is collected manually using human activity. In case the defined value is *«CollectionType =script»* then the measurement data is collected automatically using a script. In case the defined value is *«CollectionType=service»* then the measurement data is collected automatically using service.
«DerivedMeasure»	*«ScriptActivity»* *«WSActivity»*	This relationship is established based on the value of the property "CollectionMethod" of the metaclass *«AbstractMeasure»*. *«CollectionType =script»* then the measurement data is collected automatically using a script. In case the defined value is *«CollectionType=service»* then the measurement data is collected automatically using service.
«Indicator»	*«ScriptActivity»* *«WSActivity»*	*«CollectionType =script»* then the measurement data is collected automatically using a script. In case the defined value is *«CollectionType=service»* then the measurement data is collected automatically using service.
«Stakeholder»	*«HumanRole»*	Both metaclasses represent human role.
«InformationNeeds»	--	These elements of the definition metamodel are not used in the execution model derivation; The information described by these elements will be used in subsequent derivations (e.g., the derivation of the monitoring model) or by the process engineers in upcoming tasks such as generating the necessary documentation to guide the collection of the measurement value and the context data. Moreover, the data of the «Algorithm» metaclass will be used to implements the scripts and the services which obtain or calculate the derived measures and the indicators values.
«MeasurementMethod»	--	
«Procedure»	--	
«Algorithm»	--	
«Instrument »		
«Context»	--	
«Annotation»	--	

Once the relationships between both models are presented, the following sections specify the main QVT rules to derive the measurement execution model from the MDM.

3.1.1 Specify the Model types and the Transformation declaration

This section specifies the main QVT rule to derive the measurement execution model from the measurement definition model. Figure V.3 Shows the *«modeltype»* keyword to define the metamodels used in this transformation and the *«transformation»* keyword to

specifies the input and output models. Furthermore, it demonstrates the main transformation function which starts by using the *«key»* keyword to define locators or identifiers of the objects. These keys will be used to locate a matching object in the model. After defining the keys, the main function starts calling the necessary mapping operations which transform the specified objects of the source model into the corresponding objects in the target model. Next sections define the logic of these mapping operations.

```
modeltype DefMetaM "strict" uses DefMetaM ('https://DefMetaM.com');
modeltype ExecMetaM "strict" uses ExecMetaM ('https://ExecMetaM.com');

transformation Def_M2MTransf_Exec (in source:DefMetaM,out target:ExecMetaM );
main() {

 /* Define the set of properties which act as identifiers of the new objects */
        key ScriptActivity (id);
        key WSActivity (id);
        key HumanRole (id);

/* Calling the mapping functions */
-- Map source stakeholder to target Humanrole
source.objectsOfType(Stakeholder)-> map toHumanRole();

-- Map base measures to measurement activity based on its collection method
source.objectsOfType(BaseMeasure)-> map toMeasurementActivity();

-- Map Derived measures to measurement activity based on its collection method
source.objectsOfType(DerivedMeasure)-> map toMachineActivity();

-- Map Indicator measures to measurement activity based on its collection method
source.objectsOfType(Indicator)-> map toMachineActivity();

}
```

Figure V.3. QVT rule to derive the execution model from the definition model

3.1.2 Mapping the «Stakeholder» objects

As shown by Figure V.4, this mapping operation creates an object of the type *«HumanRole»* in the output model for each *«Stakeholder»* object in the source model. After creating the target object, the mapping operation uses the value of the attributes of the source objects to initialize the attributes of the constructed objects.

```
mapping Stakeholder :: toHumanRole(): HumanRole {
        -- initialize the attributes of the transformed Humanrole.
        Name:=self.name;
        Position:=self.position;
}
```

Figure V.4. Mapping operation to transform the *«Stakeholder»* objects

3.1.3 Mapping the «BaseMeasure» objects

The mapping operation *«toMeasurementActivity()»* shown in Figure V.5 transform the *«BaseMeasure»* objects of the source model into a measurement activity depending on the value of the attribute *«CollectionMethod»* of the source object; for this, the mapping operation check the value of the attribute *«CollectionMethod»*, the possible values of this attributes is defined in the enumeration *«CollectionM»* defined in chapter IV.2.1.19.

```
mapping BaseMeasure :: toMeasurementActivity():MeasurementActivity {
switch { -- transform the base measure according to its collection method
        case (self.ColectionMethod.toString() = " manually") {
        -- If it is manually collected then it is mapped to Measurement Human Activity
          result:= object MeasurementHumanActivity {
                    Name:=self.Name; Description:=self.Description;
                    humanrole += self.stakeholder.
                    oclAsType(DefMetaM::Stakeholder).resolve(HumanRole); }
          };
        case (self.ColectionMethod.toString() = "script") {
        -- If it is script collected then it is mapped to Measurement Script Activity
          result:= object ScriptActivity {
                    URI:= ""; Execution_result:= ""; }
          };
        else {
        -- If it is service collected then it is mapped to Measurement service Activity
          result:= object WSActivity {
                    URI:=""; Execution_result:= "";
                    EndPoint:=""; OperationName:=""; }
            };
        };
}
```

Figure V.5. Mapping operation to transform the *«BaseMeasure»* objects

This mapping operation transforms the source object based on the value of the *«CollectionMethod»* attribute as follow:

- *«manually»*: In this case, the mapping operation creates an object of the type *«MeasurementHumanActivity»* and initializes its attribute using the values of the attributes of the source object. Also, use the resolution construct «resolve»

to set the *«HumanRole»* object which acts as the performer of this human measurement activity.

- *«script»*: In this case, the mapping operation creates an object of the type *«ScriptActivity»* and initializes its attributes with empty text, these attributes could be specified manually by the process engineers in the second phase of the transformation process.

- *«service»*: In this case, the mapping operation creates an object of the type *«WSActivity»* and initializes its attributes with empty text, these attributes could be specified manually by the process engineers in the second phase of the transformation process.

3.1.4 Mapping the «DerivedMeasure» and «Indicator» objects

The mapping operation *«toMachineActivity()»* transforms the *«DerivedMeasure»* (Figure V.6) and the *«Indicator»* (Figure V.7) objects of the source model into a measurement machine activity depending on the value of the attribute *«CollectionMethod»* of the source object; for this, the mapping operation checks the value of the attribute *«CollectionMethod»*.

```
mapping DerivedMeasure :: toMachineActivity(): MeasurementMachineActivity{
switch {
        case (self.ColectionMethod.toString() = "script") {
        -- If it is script collected then it is mapped to Measurement Script Activity
                        result:= object ScriptActivity {
                        URI:=""; Execution_result:= ""; }
                        };
        else {
       -- If it is service collected then it is mapped to Measurement WSActivity Activity
                        result:= object WSActivity {
                        URI:=""; Execution_result:= "";
                       EndPoint:=""; OperationName:=""; }
                };
          };
}
```

Figure V.6. Mapping operation to transform the *«DerivedMeasure»* objects

This mapping operation transforms the source object based on the value of the *«CollectionMethod»* attribute as follow:

- *«script»*: In this case the mapping operation creates an object of the type *«ScriptActivity»* and initializes its attributes with empty text; these attributes could be specified manually by the process engineers in the second phase of the transformation process.

- *«service»*: In this case the mapping operation creates an object of the type *«WSActivity»* and initializes its attributes with empty text; these attributes could be specified manually by the process engineers in the second phase of the transformation process.

```
mapping Indicator :: toMachineActivity(): MeasurementMachineActivity{
switch {
        case (self.ColectionMethod.toString() = "script") {
        -- If it is script collected then it is mapped to Measurement Script Activity
          result:= object ScriptActivity {
                                    URI:=""; Execution_result:= ""; }
              };
        else {
       -- If it is service collected then it is mapped to Measurement WSActivity Activity
                    result:= object WSActivity {
                                    URI:=""; Execution_result:= "";
                                    EndPoint:=""; OperationName:=""; }
              };
         };
}
```

Figure V.7. Mapping operation to transform the *«Indicator»* objects

3.2 Measurement definition model to Monitoring model

This section describes the required transformation rules to create the necessary elements to monitor the measurement objectives and its related data. In the beginning, it is important to clarify the relationships between the elements of both models. Table V.2 highlights these relationships.

Table V.2. Relationships between the measurement definition and monitoring models.

Definition Metamodel Elements	Monitoring Metamodel Elements
--	*«Dashboard»*
--	*«VisualizationElement»*
«Indicator»	*«Indicator»*
«InformationNeeds»	*«InformationNeeds»*
«AbstractMeasure»	--
«BaseMeasure»	--
«DerivedMeasure»	--
«MeasurementMethod»	--
«Procedure»	--
«Algorithm»	--
«Stakeholder»	--
«Instrument »	--
«Context»	--
«Annotation»	--

Some elements of the definition metamodel are not used in the monitoring model derivation, such as *«MeasurementMethod» and «Context»*. The information provided by

these elements is used in other transformations (e.g., execution model and measurement documentation).

Once the relationships between both models are presented, the following sections specify the main QVT rules to derive the measurement monitoring model from the MDM.

3.2.1 Specify the Model types and the Transformation declaration

This section specifies the main QVT rule to derive the measurement monitoring model from the measurement definition model. Figure V.8 Shows the use of the *«modeltype»* keyword to define the metamodels used in this transformation, and the *«transformation»* keyword to specifies the input and output models. Furthermore, it demonstrates the main transformation function. This function starts by using the *«key»* keyword to define locators or identifiers of the objects. These keys will be used to locate a matching object in the model. After defining the keys, the main function calls the necessary mapping operations which transform the specified objects of the source model into the corresponding objects in the target model. Finally, it creates the *«Dashboard»* object because it has no corresponding object in the source model. Next sections define the logic of these mapping operations.

```
modeltype DefMetaM "strict" uses DefMetaM ('https://DefMetaM.com');
modeltype MonitorMetaM "strict" uses ExecMetaM ('https://MonitorMetaM.com');

transformation Def_M2MTransf_Monit (in source:DefMetaM,out target:MonitMetaM );
main() {
 /* Define the set of properties which act as identifiers of the new objects */
        key Dashboard (id);
        key InformationNeeds (id);
        key Indicator (id);
        key Measure (id);
        key VisualizationElement (id);

/* Calling the mapping functions */
        -- Map source InformationNeeds to the target InformationNeeds
        source.objectsOfType(InformationNeeds)-> map toInfoNeeds ();

        -- Map the source Indicators to the target indicators
        source.objectsOfType(Indicator)-> map toIndicator ();

--Create the Dashboard object.
        object Dashboard {
                        -- Initialize the "Dashboard" Attributes.
                        title:=""; name:="";
                        description:=""; };
}
```

Figure V.8. QVT Algorithm to derive the monitoring model from the definition model

3.2.2 Mapping the «InformationNeeds» objects

The mapping operation *«toInfoNeeds() »* shown in Figure V.9 transform the *«InformationNeeds»* objects of the source model into *«InformationNeeds»* objects of the target model. The (title and description) attributes of the new object are initialized using the values of the same attributes of the source object, but the attribute (name) is initialized with empty text because it has no corresponding attribute in the source model. This attributes could be specified manually by the process engineers in the second phase of the transformation process. Moreover, this operation establishes the relation between the new *«InformationNeeds»* object and the previously created *«Dashboard»* object.

```
mapping source::InformatioNeeds:: toInfoNeeds ():target::InformatioNeeds {
        -- Initialize the values the attributes of "InformationNeeds" objects.
                name:="";
                title := self.title;
                description := self.description;

        --Establish the association with "Dashboard" (Displaying)
                Displaying := Dashboard;
}
```

Figure V.9. Mapping operation to transform the *« InformationNeeds »* objects

3.2.3 Mapping the «Indicator» objects

The mapping operation *«toIndicator () »* shown in Figure V.10 transform the *«Indicator»* objects of the source model into *«Indicator»* objects of the target model. The attributes (name, description, analysis Guide, decisionCriteria, and presentationGuide) of the new object are initialized using the values of the same attributes of the source object, but the attribute (dataSource) is initialized with empty text because it has no corresponding attribute in the source model. This attributes could be specified manually by the process engineers in the second phase of the transformation process.

After initializing the attributes of the new *«Indicator»* objects, the mapping operation establishes the relation between the new *«Indicator»* object and the previously created *«InformationNeeds»* objects. Also creates a new *«VisualizationElement »* object, initialize its attributes with empty texts and finally establish the association between the indicator and the new *«VisualizationElement »* object.

```
mapping source::Indicator :: toIndicator ():target::Indicator {
 -- Initialize the values the attributes of "Indicator" objects.

        name := self.name;  description := self.description;
        analysisGuide:=self.analysisGuide;  decisionCriteria:=self.decisionCriteria;
        presentationGuide:=self.presentationGuide;  dataSource:="";

--Establish the association with "InformatioNeeds" (Monitored_Inf_Need)

       Monitored_Inf_Need := self. inf_needs_items➔forAll (inf_need : InformatioNeeds |
       inf_need.resolveone(target::InformatioNeeds);

--Establish the association with "VisualizationElement" (Visualizer)
/*  Create the visualization element which will represent this indicator
 and assign the new object to the visualizer attribute.*/

       Visualizer:=object VisualizationElement {
                -- Initialize the values the attributes of "VisualizationElement".
                    title:="";  name:="";  description:="";
                    chartType:="";        dataset:="";  };
}
```

Figure V.10. Mapping operation to transform the *« Indicator »* objects

3.3 Generating the final models

As mentioned early in this chapter, the models obtained by executing the transformation rules described in the previous sections needs to be verified and modified to obtain the final version of these models. This section describes the main activities which should be carried out by the domain experts and/or the process engineers to obtain the final models.

These activities aim to verify and ensure the validity of the structure, relations, and attributes of the basic models -generated during the first phase of the transformation process- and also to adapt these models to the execution environment.

Therefore, we recommend carrying out the following activities in the second phase of the transformation process (verification and modification phase).

3.3.1 Review the structure and the completeness of the basic models

This activity aims to verify and perform the required modifications which ensure that: the structure of the basic model is appropriate for the execution environment, the existence and the correctness of all the necessary classes, attributes, and associations. This include reviewing the class names, attributes names and types, the source and target of the associations.

3.3.2 Specify the values of the empty attributes

In this activity, the attributes which were initialized as an empty text during the transformation process should be specified with the execution environment data, such as setting the URIs of the scripts, the documentation files. And for the web services, it is necessary to specify also the attributes *«endpoint»* and *«operation name»*. Besides, it is necessary to assign values of the model elements which have no corresponding element in the source models such as the *«title, name, and description»* of the *« Dashboard»* element, and the *«title, name, description, chartType, and dataSet»* of the visualization elements.

The validation and modification activities performed during the second phase of the transformation process produce the final measurement execution and monitoring models. These models are ready to be transformed -using the model-to-text transformations- to generate the necessary artifacts to support the measurement process and its integration into the software process. These transformations will be described in the next section.

4. Model-to-Text transformations

In chapter IV, we have established the necessary languages (metamodels) to define and represent the measurement concepts throughout the software process lifecycle. Moreover, during the previous section (section V.3), we have specified the required transformation rules to derive the necessary models to support the measurement process throughout its lifecycle.

As mentioned before, we have used the UML modeling language to define the measurement metamodels. UML provides compressibility and expressiveness, both based on its graphical representation and its high-level constructors for the extension of the UML language itself.

Furthermore, UML provides concepts with execution semantics such as its metaclasses *«Activity»* and *«Action»*. However, there is currently a technological gap that makes it impossible to execute UML-based languages (such as the one defined in this thesis) because there are no virtual machines, interpreters or UML compilers.

Therefore, to transfer the theoretical solution presented in this thesis towards real and production environments, it is necessary to search for alternative mechanisms that allow the transformation of the measurement models to the artifacts that support the measurement process in the execution environment (e.g., execution code and documentation) and its integration into the process execution phase.

This section aims to define the mechanism that supports the integration of the specified measurement concepts and artifacts into the process execution as well as to establish the necessary derivation rules to generate the required artifacts for this integration.

Next sections describe the required model-to-text transformations (see Figure V.2) to derive the execution code (i.e., WS-BPEL and BPEL4People) which represents the measurement activities defined in the measurement execution model, as well to obtain

the XML presentation of the monitoring model, and finally, generate the measurement documentation HTML files from the measurement definition model.

4.1 Measurement execution model to WS-BPEL and BPEL4People extension

As mentioned in the introduction of this chapter (section 1.1), we have chosen the MOFM2T transformation language to specify the model-to-text transformation. In this section, we will define the required transformations to derive the execution code that represents the measurement activities defined in the measurement execution model. The resulting code can be combined with the process execution code, which allows the integration and execution of measurement tasks with the rest of the process tasks.

Currently, several execution languages describe the processes to allow its execution in the BPMS tools. The most important standards in this area are: BPML (Business Process Modeling Language) (Havey 2005) and WS-BPEL (OASIS 2007).

BPML was proposed by the BPMI initiative ("Business Process Management Initiative"). It is an XML based language that describes the structural representation of a business process and the semantics of its execution, based on the concept of the transactional finite state machine.

Although BPML is a complete and formal standard language for the execution of business processes, the BPMI has dropped the support of this language due to the lack of market acceptance. This abandonment was in favor of WS-BPEL standardized by the "Organization for the Advancement of Structured Information Standards (OASIS)".

WS-BPEL currently is the most important standard in the market and the de facto standard in the software industry for the execution of processes through the use of web services combined with process activities and flow controls (Mateo, Valero, and Díaz 2012; Zeng et al. 2004). This XML based language allows the specification of processes using WSDL (Web Service Definition Language) in such a way that enables constructing the process from the invocations of existing web services and the interaction of the process with other external participants.

However, despite this power, the WS-BPEL language has a problem: it does not support human interactions during the process. This interaction is important aspect of many real-world business processes. Particularly for software organizations, this is a significant limitation since the human factor is essential in the execution of software processes. For example, the human role is crucial for decision making in project management, in the technical review tasks carried out by members of quality offices and in coding tasks, among others.

To resolve this issue, the BPEL4People ("WS-BPEL Extension for People") (Organization for the Advancement of Structured Information Standards (OASIS) 2010) extends the WS-BPEL specification with the WS-HumanTask specification (Organization for the Advancement of Structured Information Standards(OASIS) 2012; Russell et al. 2007) to support a broad range of human interaction patterns.

Among all the business process execution languages, WS-BPEL (with its BPEL4People extension) is the most supported language by most process execution engines (e.g., IBM Websphere, Oracle BPM Suite, jBPM, BonitaOS, and Activiti). Besides, WS-BPEL is widely recognized and used in the business sector.

For this reason, we have chosen to transform the measurement activities defined in the measurement execution model to WS-BPEL (with its BPEL4People extension) execution language. Another reason for this choice is to ensure the highest degree of compatibility with the related works (which we plan to merge and integrate our proposal with). These works (e.g., PLM4BS described in chapter III.3.4) use BPEL4People as a process execution language. The resulting measurement activities from the transformations of this section should be merged with the process activities to build the complete process (normal activities with the measurement activities). Therefore, the choice of the target elements and the implementation of the transformation rules are influenced by the derivation operations defined in the PLM4BS.

Before specifying the necessary transformations to derive the selected execution language (WS-BPEL and BPEL4People) from the measurement execution model, it is important to determine the relationships between the elements of execution metamodel and the elements of the execution language. Table V.3 highlights these relationships.

Table V.3. The relationships between the elements execution metamodel and the execution language.

Measurement Execution Metamodel Elements	BPEL4People and WS-BPEL XML Tags	Comments
«ScriptActivity»	*«bpel:empty»*	The WS-BPEL language does not provide its own mechanisms for executing scripts. However, there are some technical solutions provided to resolve this situation. For example, Oracle introduces the "bpelx: exec" (Oracle 2014) tag on its BPEL process execution engine. As a temporary solution, this thesis adopts the solution proposed by PLM4BS, which propose the introduction of a marker in the WS-BPEL code based on the label "empty". Using the extension mechanism provided by WS-BPEL to create a new type of generic activity with which to map the execution of a script.
«WSActivity»	*«bpel:partnerLink»* & *«bpel:invoke»*	The tag*«bpel:partnerLink»* allows specifying the interaction of the process with other web services. The manual intervention of process engineer or analyst will be necessary to complete all the information of this web service (URL, operation, port).
«MeasurementHumanActivity»	*«b4p:peopleActivity»*	The extension "b4p:peopleActivity" that defines BPEL4People on WS-BPEL will be used to perform this mapping.

«HumanRole»	*«htd:logicalPeopleGroup»* *«b4p:humanInteractions»* *«b4p:peopleAssignments»*	The label "htd:logicalPeopleGroup" comes from the namespace of the specification WS-HumanTask is used by the BPEL4People specification itself. This tag is used in combination with the tag "b4p:humanInteractions" of the BPEL4People language and specific logical role groups and are used in human activities. In addition, the process roles are transformed within the "b4p:peopleAssignments" tag to define what each role group can do about processes: from starting it to managing it.

After presenting the relationships between the elements of the model and the execution language, the following sections specify the required MOFM2T templates to derive the WS-BPEL and BPEL4People code that represents the measurement activities defined in the measurement execution model.

4.1.1 Specifying the transformation «ExecToBPEL_m2t»

This section specifies the main transformation template *«ExecToBPEL_m2t»*, which generates the executable BPEL code from the measurement execution model. This transformation takes as input an instance of the Measurement Execution metamodel and transforms all its elements to WS-BPEL and BPEL4People language.

Table V.4 specifies the structure of this transformation in the MOFM2T language. This structure is based on the MOFM2T "template" keyword which defines a text template with placeholders for data to be extracted from models. These placeholders are specified using OCL expressions to select and extract data from the model elements.

Table V.4. MOFM2T transformation of *«Measurement execution model»* to (WS-BPEL and BPEL4People) language.

```
[module generate('MeasurementExecutionMetaModel')/]
[template public ExecToBPEL_m2t (m : MeasurementExecutionModel) ]
 [comment @main/]
<?xml version="1.0" encoding="UTF-8"?>
<MeasurementActivities name= "Measurement Activities" targetNamespace= "www.iwt2.org" [print_xml_NS()/]>
[comment Add extensions for human activities/]
<!-- Add extensions for human activities-->
<bpel:extensions>
        < bpel:extension
              namespace ="http://docs.oasis-open.org/ns/bpel4people/bpel4people/200803"
                                                    mustUnderstand ="yes"/>
   <bpel:extension
              namespace="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
                                  mustUnderstand="yes"/>
  </bpel:extensions>

[comment Transform the HumanRoles and Human Activities/]
```

```
    [humanRolesAndActivitiesToBPEL4People(m)/]
[comment Transform WS_Activities/]
    [ws_ActivitiesToBPEL (m)/]
<sequence>
[printBPEL_ActivitiesStructure(m)/]
</sequence>
</MeasurementActivities>
 [/template]
```

A relevant aspect of MOFM2T is that it allows invoking *«template»* within others, allowing in this way to modularize the transformation rules. The *«ExecToBPEL_m2t»* template – specified in Table V.4- invokes several auxiliary templates to achieve its task and also to support the readability of the template code.

The *«ExecToBPEL_m2t»* template starts by printing the header of the XML output file, then continue by printing the main XML element (<MeasurementActivities>) which define the attribute *«targetNamespace»*, the auxiliary template *«print_xml_NS()»* is invoked to print the necessary *«namespaces»*. Then the main template adds the required extensions for processing the human tasks (<bpel: extensions>).

Consequently, the template *«humanRolesAndActivitiesToBPEL4People() »* is invoked by the main template to transform the human roles and activities. Then the template *«ws_ActivitiesToBPEL ()»* is invoked to transform the web service activities, and finally, the main template invokes the auxiliary template *«printBPEL_ActivitiesStructure()»* which define and print the sequence of the measurement activities. We will specify these auxiliary templates in the following sections.

4.1.2 The template «print_xml_NS()»

The specification of the namespaces provides a semantic context (as in the case of the WS-BPEL code) for the elements and attributes used in the XML document by associating them with a namespace identified by URI references. Defining the namespaces of the XML document is one of the recommendations of the W3C to assign a unique name to the elements and attributes used in the same XML document and to resolve the ambiguity between them.

The template *«print_xml_NS()»* shown in Table V.5 adds the necessary namespaces for the WS-BPEL, BPEL4People, WS-HumanTask, and the XML Schema.

Table V.5. The MOFM2T template *«print_xml_NS()»*

```
[template public print_xml_NS() ]
xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
xmlns:ty="http://www.example.com/types"
xmlns:b4p="http://docs.oasis-open.org/ns/bpel4people/bpel4people/200803"
xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ns/bpel4people/bpel4people/200803 ../../xml/bpel4people.xsd
http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803 ../../xml/ws-humantask.xsd
```

```
http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803 ../../xml/ws-humantask-types.xsd"
[/template]
```

4.1.3 The template «humanRolesAndActivitiesToBPEL4People()»

This template generates the necessary text to represents the *«HumanRoles»* and *«MeasurementHumanActivity» metaclasses* of the measurement execution model; As shown in Table V.6, the template add the «*b4p:humanInteractions»* element of the BPEL4People and define <htd:logicalPeopleGroup> for each *«HumanRoles»* metaclass. This template define also a <htd:tasks> for each *«MeasurementHumanActivity»* metaclass.

Table V.6. The MOFM2T template *«humanRolesAndActivitiesToBPEL4People()»*

```
[template public humanRolesAndActivitiesToBPEL4People(m : MeasurementExecutionModel)]
<!-- Add HumanRoles-->
<b4p:humanInteractions>
        [for (role : HumanRole | m.eAllContents(HumanRole))]
            <htd:logicalPeopleGroup name="[role.Name/]_LPG">
                <htd:documentationxml:lang="en-US"> [role.Position /] </htd:documentation>
                <htd:parametername="region"type="xsd:string"/>
            </htd:logicalPeopleGroup>
        [/for]

<!-- Add Human activities-->
        <htd:tasks>
            [for (activity : MeasurementActivity| m.eAllContents(MeasurementActivity))]
                [let hactivity : MeasurementHumanActivity = activity]
                    <htd:task name="[ hactivity.Name /]_TSK" >
                        <htd:documentation xml:lang="en-US"> [hactivity.Performer /]
                            </htd:documentation>
                            <htd:peopleAssignments>
                                <htd:potentialOwners>
                                    [for (role : HumanRole | hactivity.Performer)]
                                        <htd:from  logicalPeopleGroup="[role.Name /]_LPG">
                                        </htd:from>
                                    [/for]
                                <htd:potentialOwners>
                                <htd:peopleAssignments>
                    </htd:task>
                [/let]
            [/for]
        </htd:tasks>
</b4p:humanInteractions>
[/template]
```

4.1.4 The template «ws_ActivitiesToBPEL ()»

This template implements the measurement web service activities by adding the necessary WS-BPEL references to the BPEL code; As shown in Table V.7 the template prints the "bpel: partnerLink" element for each *«WS_Activities»* metaclass to describes the interfaces used for the interaction with the web services.

Table V.7. The MOFM2T template *«ws_ActivitiesToBPEL ()»*

```
[template public WS_ActivitiesToBPEL (m : MeasurementExecutionModel)]
<!-- Add Measurement Web Service Activity-->
 <partnerLinks>
        [for (activity : MeasurementActivity | m.eAllContents(MeasurementActivity))]
                [let wsactivity : WSActivity = activity]
                        <partnerLink name="[ wsactivity.Name /]_PL" partnerLinkType="
                            lns:WSExecutionClass" />
                [/let]
        [/for]
</partnerLink>
 [/template]
```

4.2.5 The template « printBPEL_ActivitiesStructure()»

This template prints the final code with the structure and sequence of WS-BPEL activities that represents the execution model. As shown in Table V.8, this template refers to elements defined by the MOFM2T templates specified in previous sections.

Table V.8. The MOFM2T template *« printBPEL_ActivitiesStructure()»*

```
[template public printBPEL_ActivitiesStructure (m : MeasurementExecutionModel)]
[for (for (activity : MeasurementActivity | m.eAllContents(MeasurementActivity))]
        [let hactivity : MeasurementHumanActivity = activity]
                <!-- Add Human Activity -->
                <extensionActivity>
                        <b4p:peopleActivity name="[ hactivity.Name /]" inputVariable="ToBeSpecified"
                                outputVariable="ToBeSpecified">
                                <b4p:localTask reference="[ hactivity.Name /]_TSK" />
                        </b4p:peopleActivity>
                </extensionActivity>
        [/let]

        [let wsactivity : WSActivity = activity]
                <!-- Add Service Activity -->
                <invoke name="[ wsactivity.Name /]" partnerLink="[ wsactivity.Name /]_PL"
                        portType="[ wsactivity.EndPoint/]"
                        operation="[ wsactivity.OperationName/]" inputVariable="ToBeSpecified"
                        outputVariable="[ wsactivity.ExcResult /]" createInstance="yes" />
        [/let]

        [let sctactivity : ScriptActivity = activity]
                <!-- Add Script Activity -->
                <empty name="[ sctactivity.Name /]" description="[ sctactivity.Description /]"
                                uri="[ scriptactivity.URI /]" result="[ sctactivity.ExcResult /]"/>
        [/let]
[/for]
[/template]
```

As discussed in Section 4.1 of this chapter, the WS-BPEL language has some limitations to cover all the semantics of the execution metamodel described in chapter IV. These limitations include, for example, representing the metaclass "ScriptActivity".

Due to the existence of these WS-BPEL limitations, the BPEL code generated by the previous templates represents the basic version of the BPEL code.

To obtain the final WS-BPEL execution code, the process engineer or the analyst need to complete all required technical parameters manually. This task is performed in the second phase of the transformation process (verification and modification phase). All the characteristics that need to be specified manually were identified by the text "ToBeSpecified" in the code.

4.2 Measurement monitoring model to XML file

This section specifies the necessary MOFM2T transformations to derive an XML representation of the monitoring model. This representation is independent of the tool which could be used to build the monitoring tool (e.g., Dashboard). As shown in (Chowdhary et al. 2006), the XML representation could be used by a code generator to produce the necessary software components (such as SQL script, data sources, application components, etc.) to deploy and execute the monitoring control panel.

Table V.9 specify the MOFM2T template *«generateMonitoringXML ()»* which perform this transformation. The template takes an instance of the monitoring metamodel as input, then prints the XML elements and attributes which represents the metaclasses of the model.

Table V.9. The MOFM2T template *«generateMonitoringXML ()»*

```
[template public generateMonitoringXML (m : MeasurementMonitoringModel)]
[comment @main/]
[file ('Monitoring_XML_file', false, 'UTF-8')]
<?xml version="1.0" encoding="UTF-8"?>
[for (dashboard : Dashboard | m.dashboard)]
  <DashBoard name="[dashboard.name/]" title="[dashboard.title/]"
                    description="[dashboard.description/]">
      [for (infNeeds : InformationNeeds | dashboard.MonitoredInforNeeds)]
          <InformationNeeds name="[infNeeds.name/]" title="[infNeeds.title/]"
                        description="[infNeeds.description/]">
           [for (indicator : Indicator | infNeeds.Monitoring_indicator)]
                <Indicator name="[indicator.name/]" description="[indicator.description/]"
                              analysisGuide="[indicator.analysisGuide/]"
                              decisionCriteria="[indicator.decisionCriteria/]"
                              presentationGuide="[indicator.presentationGuide/]"
                              dataSource="[indicator.dataSource/]">
                 [for (vElement : VisualizationElement | indicator.visualizer)]
                      <visualizationElement name="[vElement.Name/]"
                                    description="[vElement.description/]"
                                    title="[vElement.title/]"
                                    chartType="[vElement.chartType/]"
                                    dataSet="[vElement.dataSet/]">
                      </visualizationElement>
                 [/for]
                </Indicator>
           [/for]
          </InformationNeeds>
```

```
    [/for]
  </DashBoard>
[/for]
[/file]
[/template]
```

The output of this template represents the basic XML representation of the monitoring model. To obtain the final representation, the process engineer needs to validate this basic representation. Also to perform the appropriate modifications such as checking the values of the attributes and complete the data attributes, e.g., "dataset". These manual tasks are performed in the second phase of the transformation process (verification and modification phase).

4.3 Measurement definition model to HTML documentation

This section specifies the necessary MOFM2T templates to derive the documentation of the measurement concepts defined in the measurement definition model. These templates generate HTML documentation file for each measurement concepts.

The main template is *«generateHTMLDocumentation()»*, as shown in Table V.10, it takes an instance of the MDMM as a parameter, then invoke the appropriate template to generate the HTML documentation files of each model element type.

Table V.10. The MOFM2T template *«generateHTMLDocumentation()»*

```
[template public generateHTMLDocumentation(m : MDefinitionModel)]
[comment @main/]

    [for ( m.eAllContents())]
            [let mElement : BaseM = self]
                [ generateDocs_baseM(mElement)/]
            [/let]
            [let mElement : InformationNeeds = self]
                [ generateDocs_Inf_needs(mElement)/]
            [/let]
            [let mElement : Stakeholder = self]
                [ generateDocs_Stakeholder(mElement)/]
            [/let]
            [let mElement : Indicator = self]
                [ generateDocs_Indicator(mElement)/]
            [/let]
            [let mElement : DerivedM = self]
                [ generateDocs_DerivedM(mElement)/]
            [/let]
            [let mElement : Mmethod = self]
                [ generateDocs_MMethod(mElement)/]
            [/let]
            [let mElement : Algorithm = self]
                [ generateDocs_Algorithm(mElement)/]
            [/let]
            [let mElement : Procedure = self]
                [ generateDocs_procedure(mElement)/]
```

```
                [/let]
                [let mElement : Instrument = self]
                        [ generateDocs_Instrument(mElement)/]
                [/let]
    [/for]
  [/template]
```

The main template invokes several auxiliary templates to create the documentation of the MDM elements. These auxiliary templates receive a measurement element as a parameter then generate the HTML documentation file of this element. This file includes the key data that define this element such as:

- The element name,
- The element type,
- The attributes which define the element,
- Links to the main associations of the element.

Next sections specify these sub-templates.

4.3.1 The template «generateDocs_baseM()»

This template generates the HTML documentation file of the Base Measure elements. As shown in Table V.11, this template takes an instance of *«BaseMeasure»* metaclass as a parameter.

Table V.11. The MOFM2T template *«generateDocs_baseM()»*

```
[template public generateDocs_baseM (basem : BaseMeasure)]
[file (basem.Name + '.html', false, 'UTF-8')]
<html>
<Head>
  <title>Measurement Element: [basem.Name/]</title>
 </Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Measurement element:</span></h2>
 [basem.Name/]
<h2><span style="color: #993300;">Element type:</span></h2>
Base Measure
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
<tr> <td style="height: 21px;">Name</td> <td style="height: 21px;">[basem.Name/]</td></tr>
<tr><td style="height: 21px;">Description</td> <td style="height: 21px;">[basem.description/]</td></tr>
<tr><td style="height: 21px;">Scale-Type</td> <td style="height: 21px;"> [basem.scale-type/] </td></tr>
<tr><td style="height: 21px;">Unit</td> <td style="height: 21px;"> [basem.unit/] </td></tr>
<tr><td style="height: 21px;">Scheduling </td> <td style="height: 21px;"> [basem.scheduling /] </td></tr>
<tr><td style="height: 21px;">Validation Guide</td> <td style="height: 21px;"> [basem.validationGuide /] </td></tr>
<tr><td style="height: 21px;">Accessibility</td> <td style="height: 21px;"> [basem.accessibility /] </td></tr>
```

```
<tr><td style="height: 21px;">Priority</td> <td style="height: 21px;"> [basem.priority /] </td></tr>
<tr><td style="height: 21px;">Collection Method</td> <td style="height: 21px;"> [basem.collectionMethod /]
</td></tr>
<tr><td style="height: 21px;">Objective</td> <td style="height: 21px;">[basem.Objective/]</td></tr>
<tr><td style="height: 21px;">Quantified Attribute</td> <td style="height: 21px;">[basem.quantifies.name/]</td></tr>
<tr><td style="height: 21px;">Entity</td> <td style="height: 21px;">[basem.quantifies.entity.name/]</td></tr>
</tbody>
</table>
<h2><span style="color: #993300;">Measurement Method:</span></h2>
<p><a href=[basem.mmethod.name/].html>[basem.mmethod.name/]</a> </p>
<h2><span style="color: #993300;">Context Data:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Context Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
[for (c : Context | basem._context)]
        <tr> <td style="height: 21px;">[c.name/]</td> <td style="height: 21px;">[c.description/]</td></tr>
[/for]
</tbody>
</table>
<h2><span style="color: #993300;">Annotations:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Annotation Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
[for (ann : Annotation | basem.annotation)]
        <tr> <td style="height: 21px;">[ann.name/]</td> <td style="height: 21px;">[ann.content/]</td></tr>
[/for]
</tbody>
</table>
<h2><span style="color: #993300;">Performers:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong> Details </strong></td>
</tr>
</thead>
<tbody>
[for (role : Stakeholder | basem.performer)]
        <tr> <td style="height: 21px;">[role.name/]</td> <td style="height: 21px;"><p><a
                                href=[role.name/].html>Detailes</a></td></tr>
[/for]
</tbody>
```

```
</table>
[/file]
[/template]
```

4.3.2 The template «generateDocs_DerivedM()»

This template generates the HTML documentation file of the Derived Measure elements. As shown in Table V.12, this template takes an instance of *«DerivedMeasure»* metaclass as parameter.

Table V.12. The MOFM2T template *«generateDocs_DerivedM()»*

```
[template public generateDocs_DerivedM (derivedm : DerivedM)]
[file (derivedm.Name + '.html', false, 'UTF-8')]
<html>
<Head>
<title>Measurement Element: [derivedm.Name/]</title>
</Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Measurement element:</span></h2>
[derivedm.Name/]
<h2><span style="color: #993300;">Element type:</span></h2>
Derived Measure
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
<tr> <td style="height: 21px;">Name</td> <td style="height: 21px;">[basem.Name/]</td></tr>
<tr><td style="height: 21px;">Description</td> <td style="height: 21px;">[basem.description/]</td></tr>
<tr><td style="height: 21px;">Scale-Type</td> <td style="height: 21px;"> [basem.scale-type/] </td></tr>
<tr><td style="height: 21px;">Unit</td> <td style="height: 21px;"> [basem.unit/] </td></tr>
<tr><td style="height: 21px;">Scheduling </td> <td style="height: 21px;"> [basem.scheduling /] </td></tr>
<tr><td style="height: 21px;">Validation Guide</td> <td style="height: 21px;"> [basem.validationGuide /] </td></tr>
<tr><td style="height: 21px;">Accessibility</td> <td style="height: 21px;"> [basem.accessibility /] </td></tr>
<tr><td style="height: 21px;">Priority</td> <td style="height: 21px;"> [basem.priority /] </td></tr>
<tr><td style="height: 21px;">Collection Method</td> <td style="height: 21px;"> [basem.collectionMethod /]
</td></tr>
<tr><td style="height: 21px;">Objective</td> <td style="height: 21px;">[basem.Objective/]</td></tr>
<tr><td style="height: 21px;">Quantified Attribute</td> <td style="height: 21px;">[basem.quantifies.name/]</td></tr>
<tr><td style="height: 21px;">Entity</td> <td style="height: 21px;">[basem.quantifies.entity.name/]</td></tr>
</tbody>
</table>
<h2><span style="color: #993300;">Measurement Method:</span></h2>
<p><a href=[derivedm.mmethod.name/].html>[derivedm.mmethod.name/]</a> </p>
<h2><span style="color: #993300;">Context Data:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
```

```
<td style="height: 21px; text-align: center;"><strong>Context Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
[for (c : Context | derivedm._context)]
        <tr> <td style="height: 21px;">[c.name/]</td> <td style="height: 21px;">[c.description/]</td></tr>
[/for]
</tbody>
</table>
<h2><span style="color: #993300;">Annotations:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Annotation Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
[for (ann : Annotation | derivedm.annotation)]
        <tr> <td style="height: 21px;">[ann.name/]</td> <td style="height: 21px;">[ann.content/]</td></tr>
[/for]
</tbody>
</table>
[/file]
[/template]
```

4.3.3 The template « generateDocs_Indicator()»

This template generates the HTML documentation file of the Indicator elements. As shown in Table V.13, this template take an instance of *«Indicator»* metaclass as a parameter.

Table V.13. The MOFM2T template *«generateDocs_Indicator()»*

```
[template public generateDocs_Indicator (ind : Indicator)]
[file (ind.Name + '.html', false, 'UTF-8')]
<html>
<Head>
<title>Measurement Element: [ind.Name/]</title>
</Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Measurement element:</span></h2>
[ind.Name/]
<h2><span style="color: #993300;">Element type:</span></h2>
Indicator
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
```

```
</thead>
<tbody>
<tr> <td style="height: 21px;">Name</td> <td style="height: 21px;">[ ind.Name/]</td></tr>
<tr><td style="height: 21px;">Description</td> <td style="height: 21px;">[ ind.description/]</td></tr>
<tr><td style="height: 21px;">Scale-Type</td> <td style="height: 21px;"> [ind.scale-type/] </td></tr>
<tr><td style="height: 21px;">Unit</td> <td style="height: 21px;"> [ind.unit/] </td></tr>
<tr><td style="height: 21px;">Scheduling </td> <td style="height: 21px;"> [ind.scheduling /] </td></tr>
<tr><td style="height: 21px;">Validation Guide</td> <td style="height: 21px;"> [ind.validationGuide /] </td></tr>
<tr><td style="height: 21px;">Accessibility</td> <td style="height: 21px;"> [ind.accessibility /] </td></tr>
<tr><td style="height: 21px;">Priority</td> <td style="height: 21px;"> [ind.priority /] </td></tr>
<tr><td style="height: 21px;">Collection Method</td> <td style="height: 21px;"> [ind.collectionMethod /] </td></tr>
<tr><td style="height: 21px;">Objective</td> <td style="height: 21px;">[ ind.Objective/]</td></tr>
<tr><td style="height: 21px;">Quantified Attribute</td> <td style="height: 21px;">[ind.quantifies.name/]</td></tr>
<tr><td style="height: 21px;">Entity</td> <td style="height: 21px;">[ ind.quantifies.entity.name/]</td></tr>
<tr><td style="height: 21px;">Analysis Guide</td> <td style="height: 21px;"> [ind.analysisGuide/]</td></tr>
<tr><td style="height: 21px;">Decision Criteria</td> <td style="height: 21px;"> [ind.decisionCriteria /]</td></tr>
<tr><td style="height: 21px;">Presentation Guide</td> <td style="height: 21px;"> [ind.presentationGuide /]</td></tr>
</tbody>
</table>
<h2><span style="color: #993300;">Measurement Method:</span></h2>
<p><a href=[ind.mmethod.name/].html>[ind.mmethod.name/]</a> </p>
<h2><span style="color: #993300;">Context Data:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Context Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
[for (c : Context | ind._context)]
        <tr> <td style="height: 21px;">[c.name/]</td> <td style="height: 21px;">[c.description/]</td></tr>
[/for]
</tbody>
</table>
<h2><span style="color: #993300;">Annotations:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Annotation Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
[for (ann : Annotation | ind.annotation)]
        <tr> <td style="height: 21px;">[ann.name/]</td> <td style="height: 21px;">[ann.content/]</td></tr>
[/for]
</tbody>
</table>
<h2><span style="color: #993300;">Analayzer:</span></h2>
<p><a href=[ind.Analyzer.name/].html>[ind.Analyzer.name/]</a> </p>
</tbody>
```

```
</table>
[/file]
[/template]
```

4.3.4 The template « generateDocs_Inf_needs()»

This template generates the HTML documentation file of the Information Needs elements. As shown in Table V.14, this template take an instance of *«InformationNeeds»* metaclass as a parameter.

Table V.14. The MOFM2T template *« generateDocs_Inf_needs()»*

```
[template public generateDocs_Inf_needs (inf_nds : InformationNeeds)]
[file (inf_nds.title + '.html', false, 'UTF-8')]
<html>
<Head>
<title>Measurement Element: [inf_nds.title/]</title>
 </Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Measurement element:</span></h2>
[inf_nds.title/]
<h2><span style="color: #993300;">Element type:</span></h2>
Information Nedds
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
<tr><td style="height: 21px;">Title</td> <td style="height: 21px;">[inf_nds.title/]</td></tr>
<tr><td style="height: 21px;">Description</td> <td style="height: 21px;">[inf_nds.description/]</td></tr>
<tr><td style="height: 21px;">Version</td> <td style="height: 21px;"> [inf_nds.version/]</td></tr>
<tr><td style="height: 21px;">Accessibility</td> <td style="height: 21px;"> [inf_nds.accessibility /]</td></tr>
<tr><td style="height: 21px;">Priority</td> <td style="height: 21px;"> [inf_nds.priority /]</td></tr>
</tbody>
</table>
<h2><span style="color: #993300;">Related Indicators:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Indicator Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Details</strong></td>
</tr>
</thead>
<tbody>
[for (ind : Indicator | inf_nds.indicator)]
<tr><td style="height: 21px;">[ind.Name/]</td> <td style="height: 21px;"><p><a
href=[ind.Name/].html>Detailes</a></p></td></tr>
[/for]
</tbody>
```

```
</table>
<h2><span style="color: #993300;">The Author:</span></h2>
<p><a href=[inf_nds.author.name/].html>[inf_nds.author.name/]</a> </p>
[/file]
[/template]
```

4.3.5 The template «generateDocs_MMethod()»

This template generates the HTML documentation file of the Measurement method elements. As shown Table V.15, this template take an instance of *«MeasurementMethod»* metaclass as a parameter.

Table V.15. The MOFM2T template *«generateDocs_MMethod()»*

```
[template public generateDocs_MMethod (mm : Mmethod)]
[file (mm.name + '.html', false, 'UTF-8')]
<html>
<Head>
<title>Measurement Element: [mm.name/]</title>
</Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Measurement element:</span></h2>
[mm.name/]
<h2><span style="color: #993300;">Element type:</span></h2>
Measurement Method
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
<tr> <td style="height: 21px;">Name</td> <td style="height: 21px;">[mm.name/]</td></tr>
<tr><td style="height: 21px;">Description</td> <td style="height: 21px;">[mm.description/]</td></tr>
</tbody>
</table>
[if (mm.algorithm.oclIsUndefined())]
<h2><span style="color: #993300;">Measurement Procedure:</span></h2>
<p><a href=[mm.procedure.name/].html>[mm.procedure.name/]</a> </p>
[/if]
[if (mm.procedure.oclIsUndefined())]
        <h2><span style="color: #993300;">Measurement Algorithm:</span></h2>
            <p><a href=[mm.algorithm.name/].html>[mm.algorithm.name/]</a> </p>
[/if]
[/file]
[/template]
```

4.3.6 The template « generateDocs_Stakeholder()»

This template generates the HTML documentation file of the Stakeholder elements. As shown in Table V.16, this template take an instance of *«Stakeholder»* metaclass as a parameter.

Table V.16. The MOFM2T template *«generateDocs_Stakeholder()»*

```
[template public generateDocs_Stakeholder (role : Stakeholder)]
[file (role.name + '.html', false, 'UTF-8')]
<html>
<Head>
<title>: [role.name/]</title>
  </Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Stakeholder name:</span></h2>
 [role.name/]
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
<tr><td style="height: 21px;">Name</td> <td style="height: 21px;">[role.name/]</td></tr>
<tr><td style="height: 21px;">Description</td> <td style="height: 21px;">[role.description/]</td></tr>
<tr><td style="height: 21px;">Role</td> <td style="height: 21px;">[role.role/]</td></tr>
</tbody>
</table>
[/file]
[/template]
```

4.3.7 The template «generateDocs_procedure()»

This template generates the HTML documentation file of the Procedure elements. As shown in Table V.17, this template take an instance of *«Procedure»* metaclass as a parameter.

Table V.17. The MOFM2T template *«generateDocs_procedure()»*

```
[template public generateDocs_procedure (pro : Procedure)]
[file (pro.name + '.html', false, 'UTF-8')]
<html>
<Head>
<title>Measurement Element: [pro.name/]</title>
</Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Measurement element:</span></h2>
[pro.name/]
<h2><span style="color: #993300;">Element type:</span></h2>
Measurement Procedure
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
```

```
</thead>
<tbody>
<tr> <td style="height: 21px;">Name</td> <td style="height: 21px;">[pro.name/]</td></tr>
<tr><td style="height: 21px;">Description</td> <td style="height: 21px;">[pro.description/]</td></tr>
<tr><td style="height: 21px;">Instructions</td> <td style="height: 21px;">[pro.instructions /]</td></tr>
</tbody>
</table>
<h2><span style="color: #993300;">Related instruments:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong> Details </strong></td>
</tr>
</thead>
<tbody>
[for (inst : Instrument | pro.instrument)]
        <tr> <td style="height: 21px;">[inst.name/]</td> <td style="height: 21px;"><p><a
                                        href=[inst.name/].html>Detailes</a></td></tr>
[/for]
</tbody>
</table>
[/file]
[/template]
```

4.3.8 The template «generateDocs_Algorithm()»

This template generates the HTML documentation file of the Algorithm elements. As shown in Table V.18, this template take an instance of *«Algorithm»* metaclass as a parameter.

Table V.18. The MOFM2T template *«generateDocs_Algorithm()»*

```
[template public generateDocs_Algorithm (alg : Algorithm)]
[file (alg.name + '.html', false, 'UTF-8')]
<html>
<Head>
<title>Measurement Element: [alg.name/]</title>
</Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Measurement element:</span></h2>
[alg.name/]
<h2><span style="color: #993300;">Element type:</span></h2>
Measurement Algorithm
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
```

```
<tr> <td style="height: 21px;">Name</td> <td style="height: 21px;">[alg.name/]</td></tr>
<tr><td style="height: 21px;">Description</td> <td style="height: 21px;">[alg.description/]</td></tr>
<tr><td style="height: 21px;">Formula</td> <td style="height: 21px;">[alg.formula/]</td></tr>
</tbody>
</table>
[/file]
[/template]
```

4.3.9 The template « generateDocs_Instrument()»

This template generates the HTML documentation file of the Instrument elements. As shown in Table V.19, this template take an instance of *«Instrument»* metaclass as a parameter.

Table V.19. The MOFM2T template *«generateDocs_Instrument()»*

```
[template public generateDocs_Instrument (inst : Instrument)]
[file (inst.name + '.html', false, 'UTF-8')]
<html>
<Head>
<title>Measurement Element: [inst.name/]</title>
</Head>
<h1 style="text-align: center;"> <span style="color: #993300;">Measurement documentation </span></h1>
<h2><span style="color: #993300;">Measurement element:</span></h2>
[inst.name/]
<h2><span style="color: #993300;">Element type:</span></h2>
Measurement Instrument
<h2><span style="color: #993300;">Element attributes:</span></h2>
<table style="height: 42px;" cellspacing="1" cellpadding="1" border="1">
<thead>
<tr style="height: 21px;">
<td style="height: 21px; text-align: center;"><strong>Name</strong></td>
<td style="height: 21px; text-align: center;"><strong>Value</strong></td>
</tr>
</thead>
<tbody>
<tr> <td style="height: 21px;">Name</td> <td style="height: 21px;">[inst.name/]</td></tr>
<tr> <td style="height: 21px;">Description</td> <td style="height: 21px;">[inst.description /]</td></tr>
<tr> <td style="height: 21px;">User Guide</td> <td style="height: 21px;">[inst.userGuide /]</td></tr>
<tr><td style="height: 21px;">Locator</td> <td style="height: 21px;">[inst.uri/]</td></tr>
</tbody>
</table>
[/file]
[/template]
```

To obtain the final HTML documentation files, the process engineer or the analyst need to review and modify the generated files to ensure its completeness and consistency with the measurement definition model. These manual tasks are performed in the second phase of the transformation process (verification and modification phase).

5. Chapter summary

In this chapter, we have addressed the transformations between the metamodels defined in the previous chapter and analyzed the relationships between them. These transformations were specified using the QVT and MOFM2T languages and tested using the Eclipse framework.

The transformations presented in this chapter use the MDM to derive the necessary activities and artifacts to achieve the measurement objectives throughout the measurement lifecycle, also to support the integration of these activities and artifacts into the process lifecycle.

The transformation process consists of two phases; in the first phase, the transformation rules are applied to derive the target artifacts (e.g., models). These artifacts need to be reviewed and modified by experts to obtain the final artifacts which meet the execution environment. These reviews and modifications are performed in the second phase of the transformation process.

QVT rules were defined to implements the necessary model-to-model transformations; the measurement execution and monitoring models were obtained from the measurement concepts and objectives established in the definition model at the early phases of the measurement process.

Furthermore, MOFM2T transformation rules were defined to obtain the WS-BPEL and BPEL4People code which allows executing the measurement activities. Moreover, it used to derive the necessary XML artifacts to generate the monitoring dashboard and also to create the measurement HTML documentation files.

Chapter VI Application and Evaluation of the Proposal

Previous chapters have described the proposed measurement lifecycle and the main metamodels which support this lifecycle. The necessary transformations were also defined to derive the required models and artifacts to support the integration of the measurement concepts and activities into the process lifecycle.

After discussing the thesis problem and presenting the proposed solution, this chapter fulfills the last two objectives of this thesis work (see chapter III.2); these objectives aims to develop the theoretical solution proposed in the previous chapters in the form that allow its application and evaluation in the real world.

Therefore, the investigation focused on how to develop it to allow its practical use by engineers and organizations.

This chapter is structured as follows: The first section introduces the practical case and the project through which the proposal was evaluated. The second section describes the challenges faced to develop the proposal in the form that allows its use in real environments and how we have overcome them. Section three demonstrates how the proposal was applied and the evaluation of this experience. Finally, the last section presents conclusions related to the experience of implementing the proposal in the real environment.

1. Overview of the influences and the validation project

After presenting the thesis problem and the proposed solution to address this problem, arises the need to verify the ability of the proposed solution to solve this problem. Therefore, it is important to put the solution into practice for its evaluation and to receive feedback about its effectiveness.

As mentioned in chapter III, sections 3.4 and 4, this work has been developed within the research group Web Engineering and Early Testing (IWT2). This group has acquired, since its inception, extensive experience in the transfer of research results to the business sector through its implementation and cooperation in various R&D projects and technology transfer projects in very diverse domains. These projects have been carried out with different public and private organizations, which allowed the group to build continuous partnerships with numerous parties.

Thanks to these experiences and the established relationships, IWT2 group has been able to provide cases, and situations in organizations in which the problem addressed in this thesis work is a reality. These cases allowed the validation of the proposed solution and ensured the immediate transfer of the results of this thesis to real environments.

In this context, and as mentioned early in this thesis, the current work has been influenced from the beginning by the strategic research line of IWT2 group which investigates on how to successfully combine the model-driven paradigm with the business processes management to solve specific needs. One of the most prominent fruits of this research is the development of PLM$_4$BS framework (discussed in chapter III section 3.4) which define a process lifecycle and provide a process modeling language for describing software processes and business processes in general.

The primary objective of this work is to allow the PLM$_4$BS framework to provide the necessary support for the continuous improvements of the software process. To achieve this goal, we have incorporated the measurement issues into the PLM$_4$BS process lifecycle (e.g., modeling, execution, and monitoring) in a way that supports the quantitative management, evaluation and continuous improvement of the software processes.

For this reason, the proposed solution must be integrated with the PLM$_4$BS framework and validated within one of the projects in which this framework is involved. Therefore, we have selected the IDE$_4$ICDS project to evaluate the proposed solution. Next section presents a summary of this project.

1.1 Background of the IDE$_4$ICDS project

This section describes the main features of the validation project and its objectives.

Project name:

IDE$_4$ICDS: Integrated Development Environment for Improving Clinical Decision Support based on Clinical Guides

Project definition:

This project is subsidized by the Ministry of Economy and Competitiveness and co-financed by FEDER funds, within the Challenges-Collaboration convocation of the State Program of Research, Development, and Innovation Oriented to the Challenges of Society, within the framework of the State Plan for Scientific and Technical Research and Innovation 2013-2016. (File: RTC-2016-5824-1). The project has an approved budget of € 824,881.60.

Project Description:

In recent years, organizations have developed conceptual models for specific pathologies through the definition of clinical guides. A clinical guide is a set of systematic statements to support health professionals and patients in decisions about appropriate health care for specific circumstances (Audet, Greenfield, and Field 1990).

The definition of a clinical guide includes the description of clinical processes, set of information, recommendations and rules to support clinical decisions. By using clinical guides, health professionals obtain the following advantages:

- The application of evidence-based medicine.
- Normalize and improve the quality of patient care.

- Reduce unjustified variations in clinical practice.
- Reduce healthcare costs.

Clinical guides define guidelines in narrative text, which implicitly implies a certain level of ambiguity. In addition, there are no formal methodologies and software tools that support the representation, implementation, and monitoring of these guidelines. For this reason, the translation of these guidelines into a computer platform that supports all its aspects (modeling, execution, monitoring, decision support, etc.) is complex and, in many cases, is developed in an ad hoc manner for each health organization that decides to implement these guidelines. This means that the same clinical guide can be developed (and evolved) in different ways in different health organizations.

In practice, the application of effective management of clinical guides requires support tools that allow improving and guaranteeing the quality and applicability of this type of process in real environments. In this sense, this management needs information systems and execution engines that offer practical and efficient support to this management, but the design and development of these systems are a complex and expensive task.

This project aims to cover this need by the definition of a formalized work methodology and the development of a software platform that will support and provide a centralized, comprehensive and collaborative management of clinical guides. This platform is called IDE$_4$ICDS and is characterized by:

- Centralized, the data of all the components (processes, simple elements, and decision rules) that compose the clinical guide are stored, and their traceability is maintained through a single information repository.
- Integral, in such a way that a single platform provides modules to define, execute, monitor and interoperate the clinical guide with the other systems of the health organization and the physical devices of the patient (sensors, smartphones, etc.).
- Collaborative, in such a way that if a healthcare professional, based on his experience, considers it necessary to improve or evolve a clinical guide, he can do it in an intuitive and friendly way, another health organization can use that modification.

The management of clinical guides includes critical and complex tasks (such as definition, maintenance, evolution and execution of clinical guides). These tasks involve both engineers and technicians ICT (Information and Communications Technology) as health professionals, who must work collaboratively to define, capture and validate the requirements of this type of systems. For this purpose, software engineering techniques offer viable alternatives (Escalona and Aragon 2008; Pohl 2010) to define these requirements.

In this context, it is necessary to conduct research, propose and define a mechanism to ensure that the implementation of the software systems to support the management of clinical guides is consistent with their definition and allows the reduction in the variation of the application of clinical guides by medical teams on the different patients.

The IDE4ICDS platform offers an approach for a specific problem: the need to establish a real working philosophy oriented to clinical guides, together with effective, systematic and automatic mechanisms within the health sector organizations. This approach allows the representation, maintenance, execution of the clinical guides, also, capturing feedback about its use to improve the quality of the health care received by the patient; all using software tools that enable these tasks as well as the interoperability between systems with the purpose of transferring and sharing clinical knowledge.

Concerning the work methodology, it could be defined at a high level using the Model-Driven paradigm (MDE) through the following steps (Figure VI.1):

- Modeling of the clinical guide (together with its elements: processes, simple elements, and decision rules) according to a generic metamodel for the definition of clinical guides.
- Systematic and automatic generation of an executable version of the clinical guide that can be deployed in a process execution engine.
- Automatic and systematic generation of a view in the form of a dashboard (based on clinical indicators) to monitor the execution of the clinical guides.

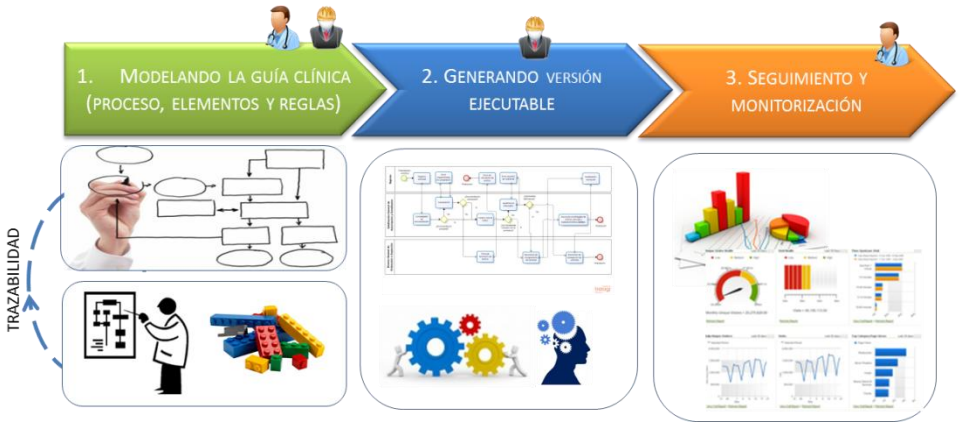

Figure VI.1. Work methodology (IDE4ICDS 2017)

The IDE4ICDS platform can be deployed in health organizations interested in (i) improve the health outcomes in patients by promoting appropriate procedures and reducing unjustified variability in the choice of treatment, (ii) provide guidance to health professionals based on the best biomedical research findings and references, (iii) improve the resources efficiency and control costs without threatening the quality of assistance.

Project objectives:

1. Define the lifecycle and work methodology of the IDE4ICDS platform.
2. Deploy, execute and monitor the IDE4ICDS platform. The IDE4ICDS platform will be supported by the MDE paradigm to achieve this objective. To realize this objective, it is necessary to:
   - 2.1 Define a set of systematic mechanisms to transform models, which allows generating an executable and interoperable model based on the clinical guide definition metamodel.
   - 2.2 Design and develop an applicable and general version of the metamodel and the set of transformation mechanisms mentioned

above. This objective entails: (i) defining domain-specific languages needed to allow the instantiation of the metamodels defined in the project; (ii) implement a CASE (Computer-Aided Software Engineering) tool that supports the use of these languages within an agile, friendly and flexible environment within the IDE$_4$ICDS platform.

2.3 Design and develop a module within the platform that acts as a repository of simple elements (recommendations, processes, information set, rules, etc.). These simple elements can be reused in different clinical guides.

2.4 Design and develop a module to monitor the status of the clinical guides, based on an initial catalog of metrics and indicators supported on the platform. This initial catalog may be extensible. Besides, this objective contemplates the generation of reports on the results of the compliance with the mentioned clinical guides.

2.5 Design and develop a module that acts as an interoperability layer between existing clinical systems and the IDE$_4$ICDS platform through an architecture based on SOA services prioritizing the communication based on standards.

3. Validate the clinical guide management platform
4. Strategic brand positioning.

The previous description of the project and its objectives highlight the role that the proposal presented in this work can play in achieving these objectives. The proposal can contribute to the realization of the second project goal, precisely, the sub-objective 2.4, which aim to develop a module to monitor the status of the clinical guides based on measures and indicators.

The lifecycle proposed in this work could support the activities related to the clinical guides monitoring, such as (the definition, documentation, collection, and analysis) of the required measures and indicators.

Furthermore, the use of MDE paradigm in the implementation of the project as in the proposal presented in this work allows the proposal to participate in the achievement of the project's objectives by several aspects.

From one side, the metamodels defined in this work could support the formal definition of the measures and indicators also support the derivation and integration of the measurement activities throughout the clinical guides' lifecycle.

From the other side, the transformations (M2M and M2T) defined in this work could support the derivation of the necessary dashboards (based on the defined measurement needs and indicators) to monitor and report the status of the clinical guides.

To allow the use of the solution proposed in this thesis work within the PLM$_4$BS framework and the CASE tool used in this project. It is necessary to develop the proposed solution to enable its integration into them. The main tasks to allow this integration are: (i) develop a specific language (UML profiles) to allow the practical use of the metamodels defined in this proposal, and (ii) integrate this specific language into the CASE tool used in the project. The following section describes how these tasks have been accomplished.

2. The integration of the proposal of this thesis into the clinical guide definition module (PLM$_4$BS framework and the CASE tool)

The metamodels and transformations proposed in this thesis need to be integrated into a modeling tool to allow its practical use by engineers and organizations.

Although the proposal presented in this thesis is mainly intended to support the modeling of software process measurements, however, we have taken into consideration, since the beginning, developing the solution in a way that allows its use in different sectors and contexts. This facilitated its integration into the framework and the tool used to support the clinical guide lifecycle in the IDE4ICDS project.

The integration process involved two phases. The first phase aimed to develop a domain-specific modeling notation (i.e., UML profile) which allows the practical use of the MDMM -defined in this thesis- with the PLM$_4$BS process modeling language. The second phase aimed to integrate this modeling notation and the necessary MDE transformations into the clinical guide definition module of the IDE$_4$ICDS project. Following sections describe the integration activities carried out during these phases.

2.1 Developing a UML Profile for the Measurement Definition Metamodel (MDMM)

UML profile provides a usable, expressive and flexible mechanism to adapt a theoretically defined metamodel with specific constructs for a particular domain (Object Management Group 2015). The profile allows the instantiation of the MDMM using a visual notation that can be used by CASE tools (e.g., NDT-Tool (Escalona et al. 2003), IBM Rational Software Architect Designer (IBM 2018), and Enterprise Architect (Sparx Systems 2018)).

The UML extension protocol is based on three basic mechanisms:

1- «*Stereotype*», which allow the definition of the specific domain elements that must, in turn, extend specific UML metaclasses,
2- «*Tagged value*», which allow adding particular properties to the stereotyped element defined within the profile,
3- «*Constraint*», which allow the definition of semantic restrictions on the stereotyped elements to condition the instantiation of the metamodel in order to build well-defined models.

The UML profile that implements the MDMM defines a stereotype for each element of the MDMM and, in addition, includes the required tagged values in each stereotype according to the corresponding element in the metamodel. It also adapts the semantic constraints of the metamodel to restrict the behavior of the UML metaclass used.

Figure VI.2 shows the UML profile developed for the MDMM using UML notation. The stereotypes represent the metaclasses of the MDMM, whereas the tagged values represent the attributes of the metamodel metaclasses.

Each stereotype in the UML profile extends a UML metaclass; Table VI.1 shows the stereotypes of the measurement definition profile, as well as the metaclass used to implement it.

Table VI.1. The stereotypes of the measurement definition profile.

UML metaclass	*Stereotype*	Comment
«Class»	*«AbstractMeasure»*	The UML metaclass *«Class»* specifies a classification of objects and define their properties (attributes, operations, associations, etc.) that characterize the structure and context of these objects.
«Class»	*«BaseMeasure»*	
«Class»	*«DerivedMeasure»*	
«Class»	*«Indicator»*	
«Class»	*«InformationNeeds»*	
«Class»	*«MeasurementMethod»*	
«Class»	*«Procedure»*	
«Class»	*«Algorithm»*	
«Class»	*«Instrument »*	
«Class»	*«Context»*	
«Class»	*«Annotation»*	
«Class»	*«Attribute»*	
«Actor»	*«Stakeholder»*	The UML metaclass *«Actor»* specifies a role or agent (human or not) that interacts with the system.
«Enumeration»	*«Scale-type»*	The UML metaclass *«Enumeration»* is a data type whose values are enumerated in the model.
	«CollectionM»	
«Association»	*«isPerformer», «hasAnn», «hasC», «hasP», «hasAl», «hasIns», «isComposedBy»*	The UML "Association" metaclass has been chosen to extend the indicated stereotypes because this metaclass specifies a semantic relationship that can occur between two instances and declares that there may be links between the instances of the associated types.
«Dependency»	*«isEvaluatedBy», «hasMM», «autheredBy», «belongTo», «analyzedBy», «hasResp»,«quantifies», «involves»*	The UML «Dependency» metaclass has been chosen to extend the indicated stereotypes because this metaclass establishes a semantic relationship between the linked objects in such a way that the semantics of the source object is not complete without the target object.

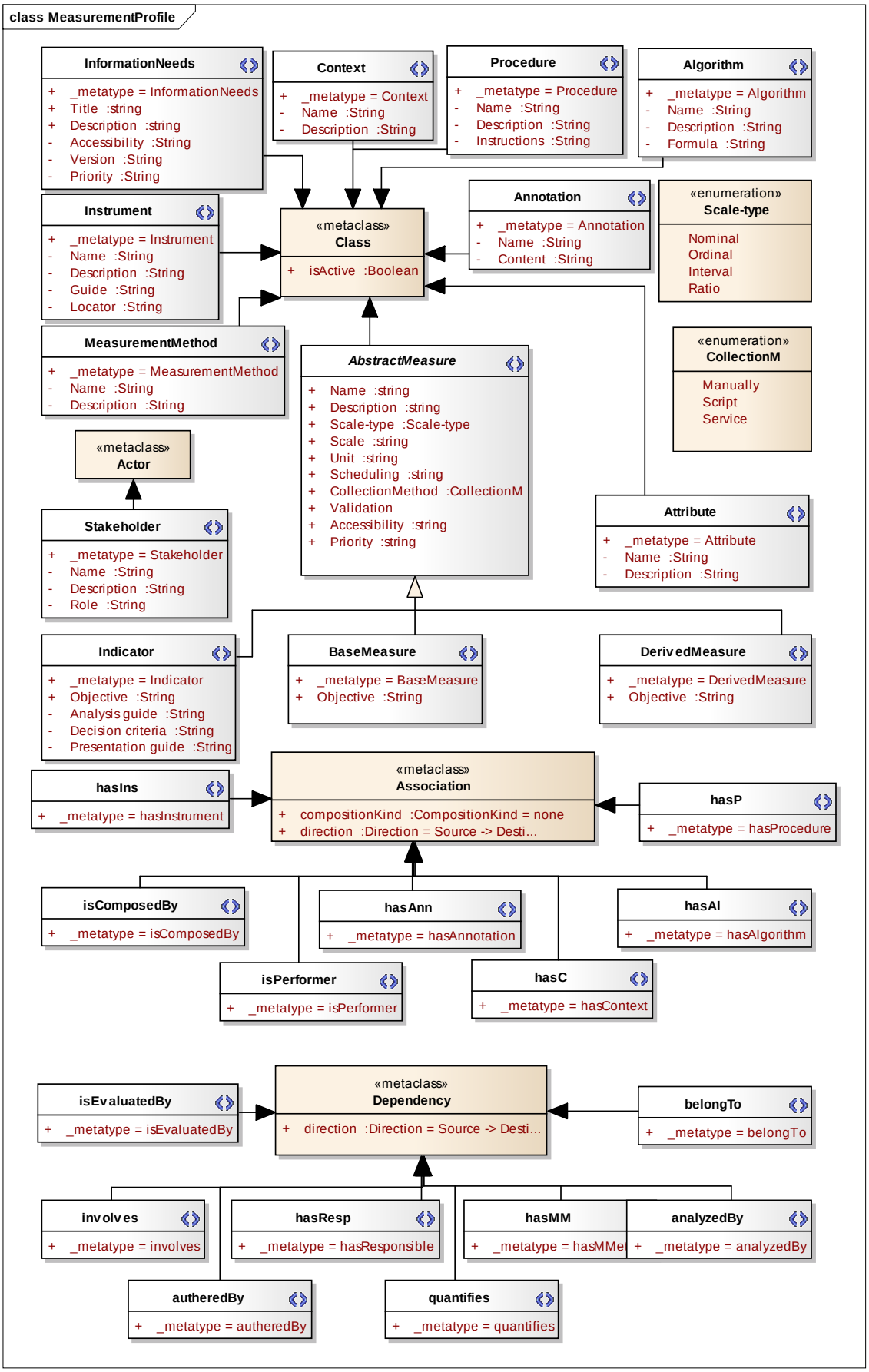

Figure VI.2. Measurement definition UML profile

2.2 Integrating the measurement definition profile into the clinical guides definition module of the IDE$_4$ICDS project

To allow the practical use of the solution proposed in this thesis, we need to integrate the proposed metamodels into the clinical guide definition module used in the IDE$_4$ICDS project. This module is based on the Enterprise Architect (EA) CASE tool; this tool supports the creation of visual instances of the metamodels that describe the clinical guides. To perform this integration, we need to add the UML profile developed in the previous step to the EA.

One of the advantages that EA provides is its extension capacity, which allows the development of Add-Ins. For this purpose, EA provides a Model Driven Generation (MDG) Technology, which allows the development of Add-Ins and deploys them in the EA project, providing a solution tailored to specific domains or environments.

We have used the MDG technology to develop an Add-In to allow the instantiation of the MDMM proposed in this work. The following sections describe the steps we have taken to build this Add-In.

2.2.1 Create MDG technology project in Enterprise Architect

The first step is creating the "MDG Technology" project using the EA wizard (Figure VI.3).

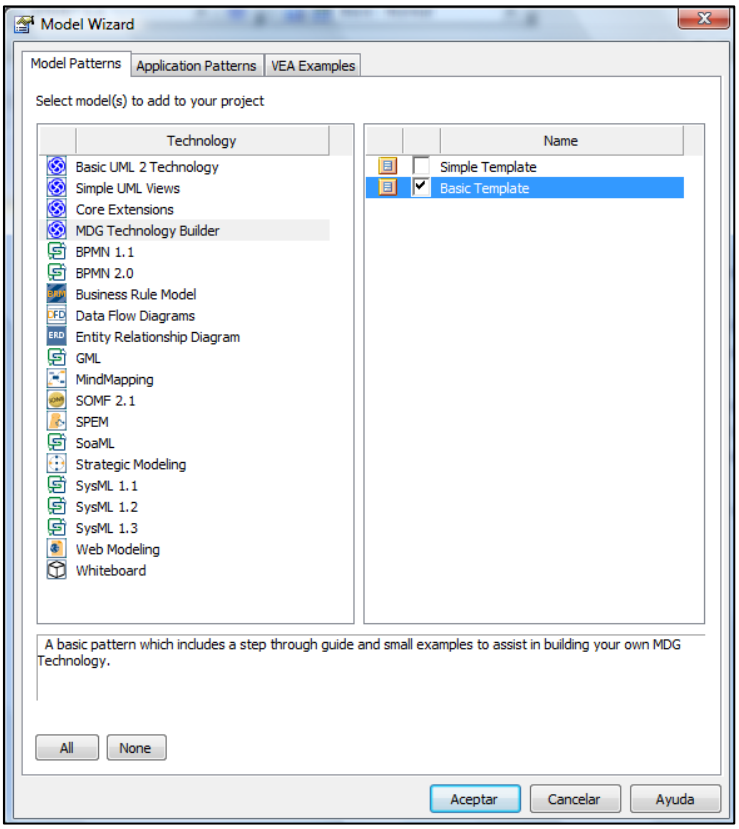

Figure VI.3. The creation of the "MDG Technology" project.

The MDG project is divided into three main parts (Figure VI.4); these parts are described below.

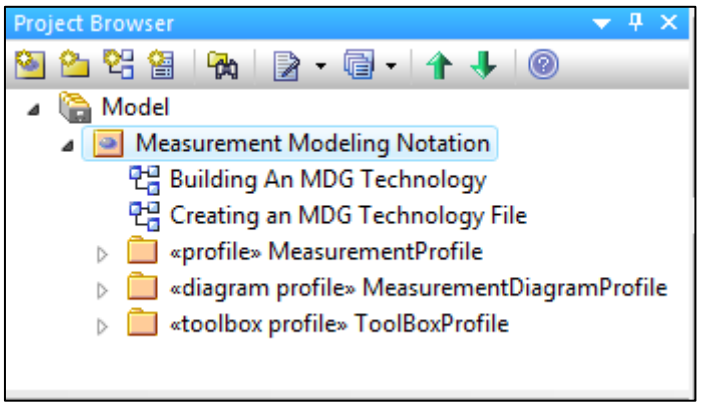

Figure VI.4. Main parts of the MDG project.

Package «*profile*». In this package, the metaclasses of the MDMM are represented as stereotypes. Each stereotype has a set of corresponding tagged values, which are the attributes of the metaclasses. In addition, each stereotype extends the appropriate UML metaclass. Figure VI.2 show the measurement definition UML profile.

Package «*diagram profile*». This package defines all the required artifacts to define diagrams according to the stereotypes of the profile.

Package « *toolbox profile*». The toolbox profile contains the necessary artifacts to create a customized toolbox following the defined profile. Figure VI.5 shows the measurement toolbox defined in this package.

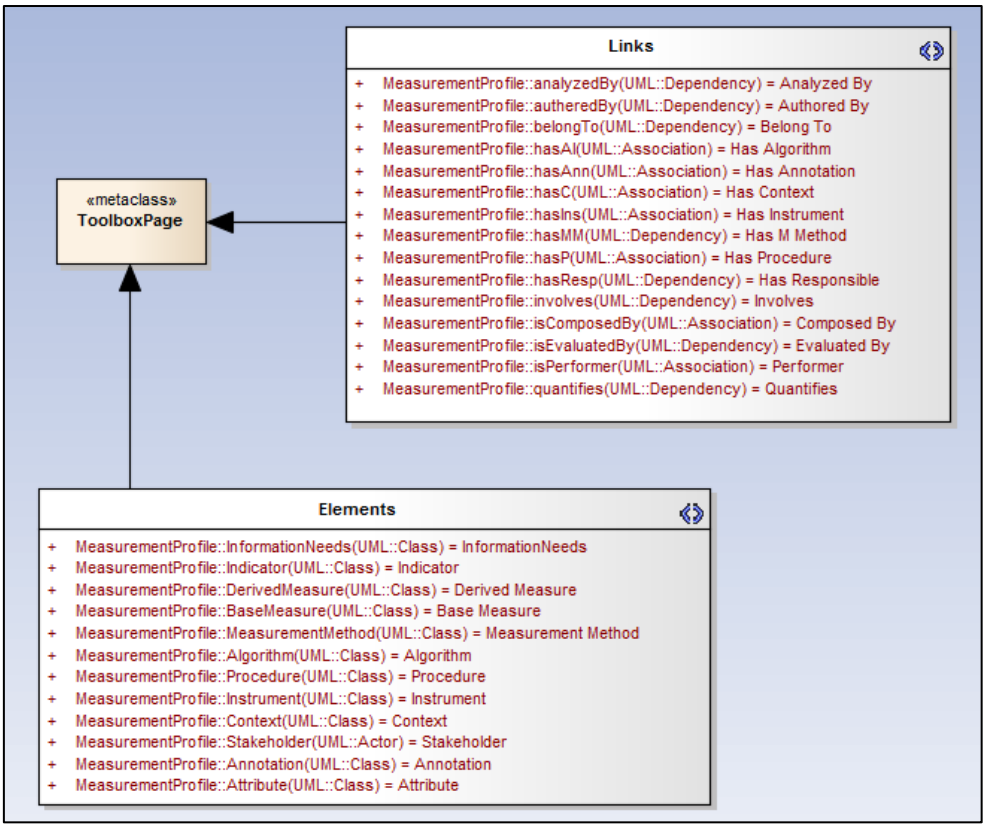

Figure VI.5. The defined measurement toolbox.

2.2.2 Saving the profiles

Saving the UML profile and the toolbox profile created in the previous steps generates XML files (one for each profile); these files will be used in the next step to build the Add-In file.

2.2.3 Generating the MDG Technology file

The generation of the MDG technology file creates the file which contains instructions that allows EA to import the MDG Technology Add-In (Figure VI.6).

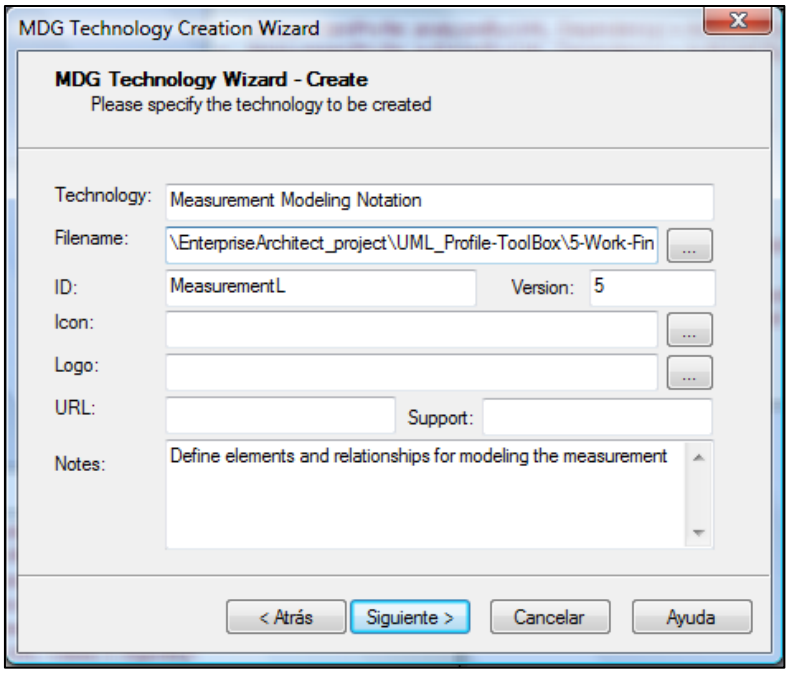

Figure VI.6. Generating of the MDG technology file.

2.2.4 Import the MDG technology file (the UML profile and toolbox) to EA

To make the MDG Technology file accessible to EA models, we need to add it to the EA as a new MDG Technologies file (see Figure VI.7 and Figure VI.8).

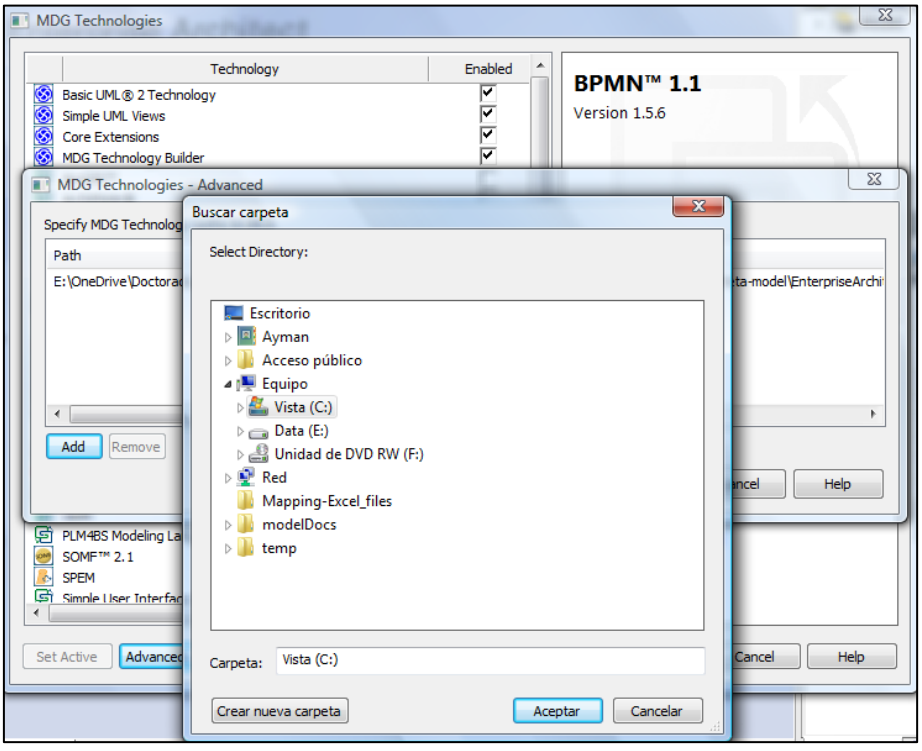

Figure VI.7. Importing MDG Technology file (1).

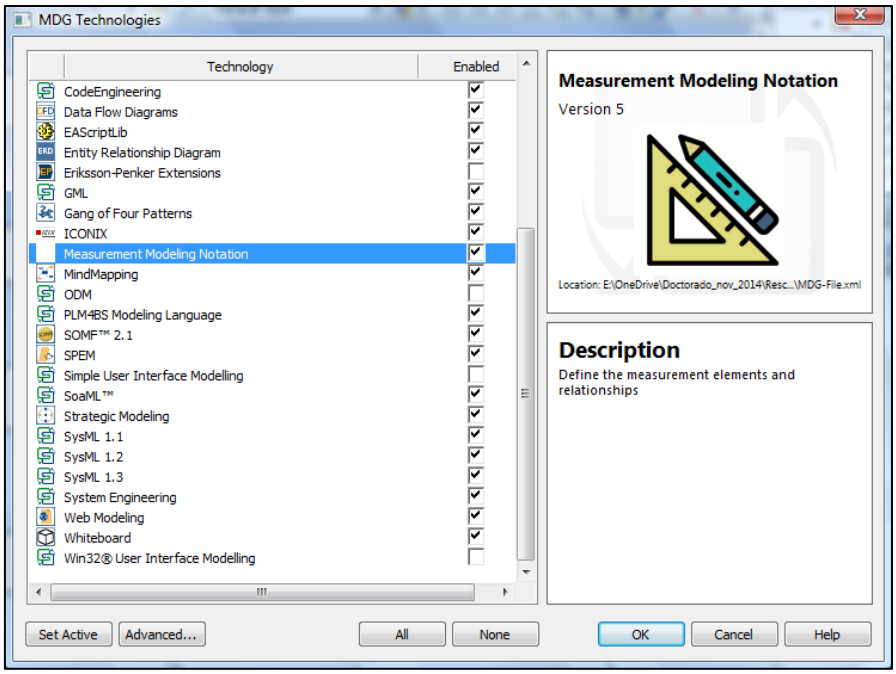

Figure VI.8. Importing MDG Technology file (2).

2.2.5 Using the UML profile and toolbox to instantiate the measurement metamodel elements

As a result, the developed Add-In allow the instantiation of the MDMM –proposed in this work- using the stereotypes defined in the UML profile and the created toolbox, Figure VI.9, Figure VI.10, and Figure VI.11 show the toolbox and part of measurement models as examples of the practical use of the proposed metamodels and UML profiles.

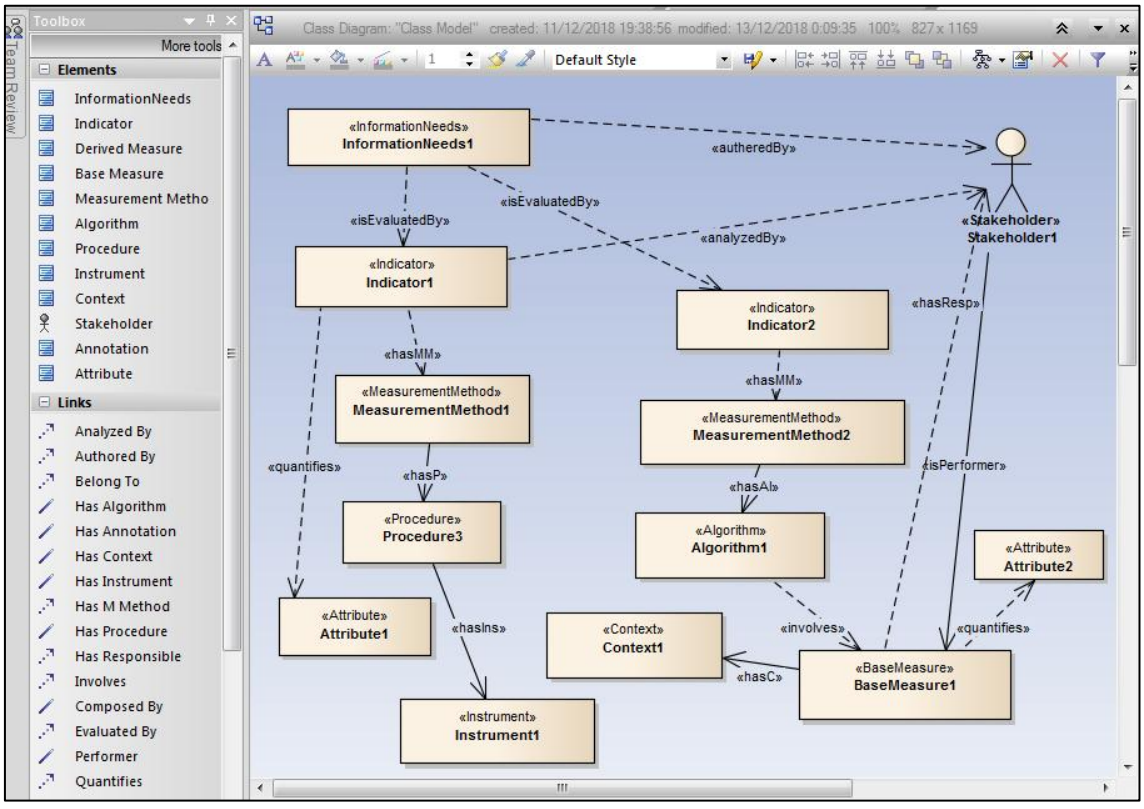

Figure VI.9. Measurement Modeling Notation toolbox.

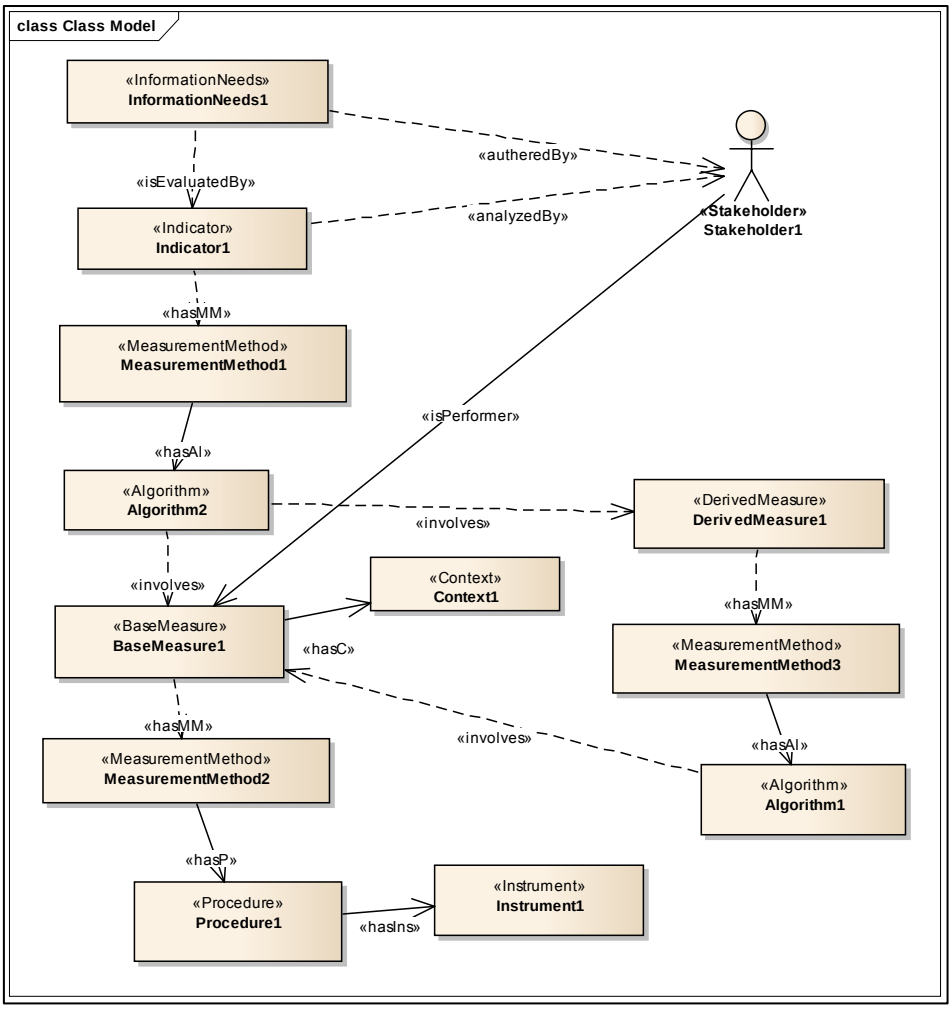

Figure VI.10. Part of a measurement model.

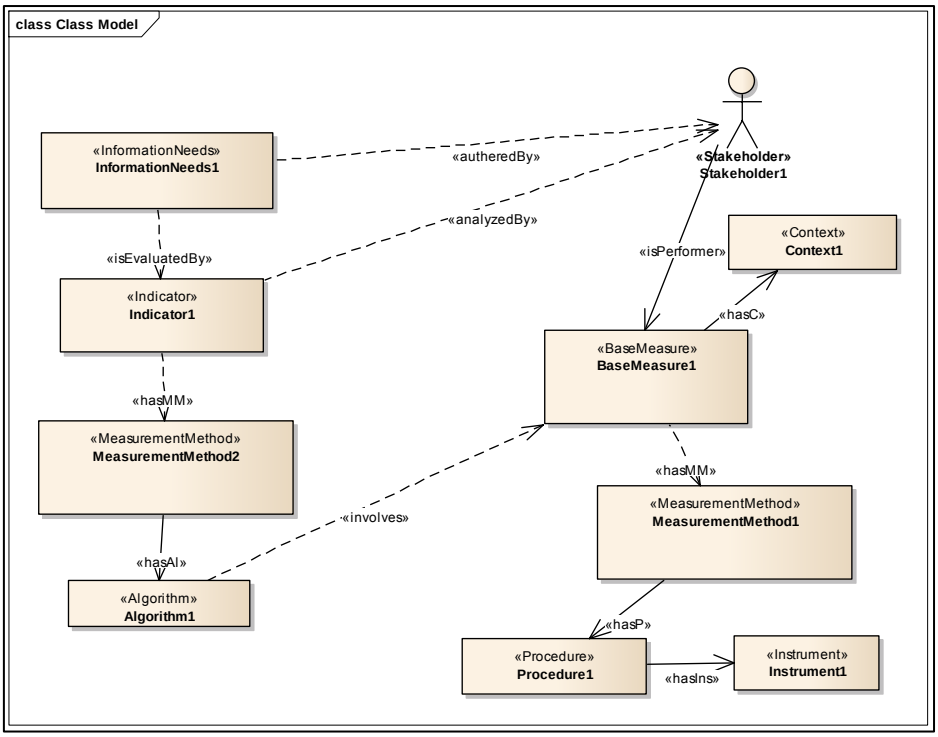

Figure VI.11. Part of a measurement model.

2.2.6 Integrating the MDE transformations

The CASE tool used in this project (Enterprise Architect) does not support the MDE transformation languages (e.g., QVT); therefore, the project team has developed an Add-In to implement the required transformations into the IDE$_4$ICDS platform. Thus, the necessary transformations to derive the clinical guide artifacts (e.g., clinical guides model, code, and documentation) were developed in C # according to the approach published in (Borgoñón 2016; Garcia-Garcia 2015; García García et al. 2015).

The project team is still working on this add-in (i.e., development and testing phase). For a better understanding of this Add-In, and to facilitate the readability of this dissertation, we include a sample of the C# code used in the Add-In.

Table VI.2 presents the QVT algorithm (see Figure V.3) defined to derive the measurement execution model from the measurement definition model.

Table VI.2. QVT Algorithm to derive the execution model from the definition model.

```
modeltype DefMetaM "strict" uses DefMetaM ('https://DefMetaM.com');
modeltype ExecMetaM "strict" uses ExecMetaM ('https://ExecMetaM.com');


transformation Def_M2MTransf_Exec (in source:DefMetaM,out target:ExecMetaM );
main()
{
 / *  Define the set of properties which act as identifiers of the new objects  */
            key ScriptActivity (id);
            key WSActivity (id);
            key HumanRole (id);

/ *  Calling the mapping functions */
-- Map source stakeholder to target HumanRole
source.objectsOfType(Stakeholder)-> map toHumanRole();

-- Map base measures to measurement activity based on its collection method
source.objectsOfType(BaseMeasure)-> map toMeasurementActivity();

-- Map Derived measures to measurement activity based on its collection method
source.objectsOfType(DerivedMeasure)-> map toMachineActivity();

-- Map Indicator measures to measurement activity based on its collection method
source.objectsOfType(Indicator)-> map toMachineActivity();
}
```

And Table VI.3 demonstrates the C# implementation of this QVT algorithm.

Table VI.3. The C# implementation of the QVT algorithm which derive the execution model from the definition model.

```
class DefMM2ExecMM
{
  Hashtable _idMaps = new Hashtable( ); // input/outputModelElements !
  public EA.Package transform (EA.Repository Repository, EA.Package inputModel,
                               EA.Package outputModel)
```

```
{
        _idMaps.Clear( );
        addNewRoot ( Repository , outputModel);
        foreach (EA.Element e in inputModel.Elements )
        {
                switch(e.Type)
                                {
                                        case "Stakeholder":
                                                        toHumanRole(e) ;
                                                        break;
                                        case "BaseMeasure":
                                                        toMeasurementActivity(e);
                                                        break;
                                        case "DerivedMeasure":
                                                toMachineActivity(e);
                                                        break;
                                        case "Indicator":
                                                toMachineActivity(e);
                                                        break;
                                }
        }
        _package.Update ( ) ;
        Repository.RefreshModel(_package.PackageID);
        return _package ;
}
```

3. The application of the proposed solution

The previous section has described how the proposed solution was developed and integrated into the EA tool to allow its use in the IDE4ICDS project.

This section begins by describing the status of measurement activities and issues in the IDE$_4$ICDS project before the application of the proposed solution. Then explains how the proposed solution was applied. And finally, provide a summary of the results of this experience.

3.1 The status of the IDE$_4$ICDS project in terms of the measurement goals and activities

In this project, the measurement activities (e.g., selecting and defining the measurement goals and concepts of the clinical guides) were not described or planned.

Most measurement activities were performed in an ad hoc manner, by any member of the teamwork (health professional, technical or process engineer) and at any phase of the clinical guide lifecycle (usually during its execution). In addition, since the measurement tasks were not planned, there were no resources allocated for their implementation.

Furthermore, there was no agreement among the stakeholders about the vocabulary that describes the measurement concepts.

Besides, no tool or language allows the stakeholders to describe the measurement concepts (e.g., measurement method) formally and operationally.

The stakeholders did not have sufficient knowledge about the measurement concepts and its selection and definition methods.

Moreover, they did not have a methodology to guide and assist them in planning and implementing the measurement activities (e.g., what should be done? How? And when?).

3.2 Applying the proposed solution

The proposed solution defines the main phases and activities of the measurement life cycle. This lifecycle provides a comprehensive guide to establish the measurement objectives, identify the measurement concepts. Also to obtain, analyze and report the measurement data.

Moreover, this solution integrates this lifecycle into the process lifecycle (in this case, the clinical guide lifecycle) in a way that allows the measurement activities to be carried out with the rest of the clinical guide lifecycle activities.

Furthermore, it provides a measurement modeling language that allows the stakeholders to define the measurement concepts operationally and formally. Below, we describe how the proposed solution was applied during the clinical guide lifecycle.

- *Design phase*: in this phase, the stakeholders (e.g., health professionals and process engineers) work on defining the objectives and requirements of the clinical guides, in the sense of identifying the biomedical best practices and references, the technical requirements to execute the process, etc.

  The integration of the measurement *selection and definition* activities (defined in the first phase of the measurement lifecycle, see chapter III.4.1.2) have allowed the stakeholder to define the measurement objectives (based on the clinical guide goals), supported them in identifying the main measurement concepts that satisfy these objectives. Furthermore, these activities allowed them to apply measurement methods to select and define these concepts in an operational manner. Moreover, the quality criteria proposed in this measurement phase has assisted the stakeholders in controlling and ensuring the quality of the defined measurement concepts.

- *Modeling phase*: in this phase, stakeholders describe the different perspectives of the clinical guide in a formal language; the objective is ensuring a common understanding of the clinical guide perspectives between the various stakeholders. The MDMM and the UML-profile proposed in this work allow stakeholders to describe formally the measurement objectives and concepts defined in the previous step.

- *Deployment phase*: the necessary configuration to execute the clinical guide and to ensure its interaction with the execution environment and the other systems are performed in this phase. Also, the settings required to collect, validate, and store the measurement data are performed in this phase. A relational database was created in this project to store the measurement data; it consists of four tables: three tables to

store the collected data of the different measurement types (base measures, derived measures and indicators) and one table to store the collected context data. Furthermore, the defined model-to-model and model-to-text transformation are applied to the MDM defined in the previous phase to generate the necessary artifacts to support the subsequent measurements activities (e.g., measurement documentation, measurement execution activities, and monitoring model).

- *Execution phase*: the clinical guide is executed in this phase; the stakeholders interact with the clinical guide (e.g., health professionals, patients, and other systems). While the clinical guide is performed, the measurements task are carried out to *collect, validate, prepare and store* the measurement data.

- *Monitoring and Analysis*: in this phase, the measurement data is analyzed and reported to evaluate and communicate the performance of the clinical guide, these activities are achieved according to the analysis and reporting guides defined in the measurement selection and definition phase.

- *Continuous improvement*: in this phase, the evaluation of the measurement data (performed in the previous phase) is used to improve the clinical guide performance; its highlight the improvement opportunities to optimize the clinical guide efficiency and effectiveness, thereby increasing stakeholder satisfaction.

Figure VI.12 shows part of a clinical guide modeled using the PLM$_4$BS framework, also show the defined measures and indicators integrated into the clinical guide model.

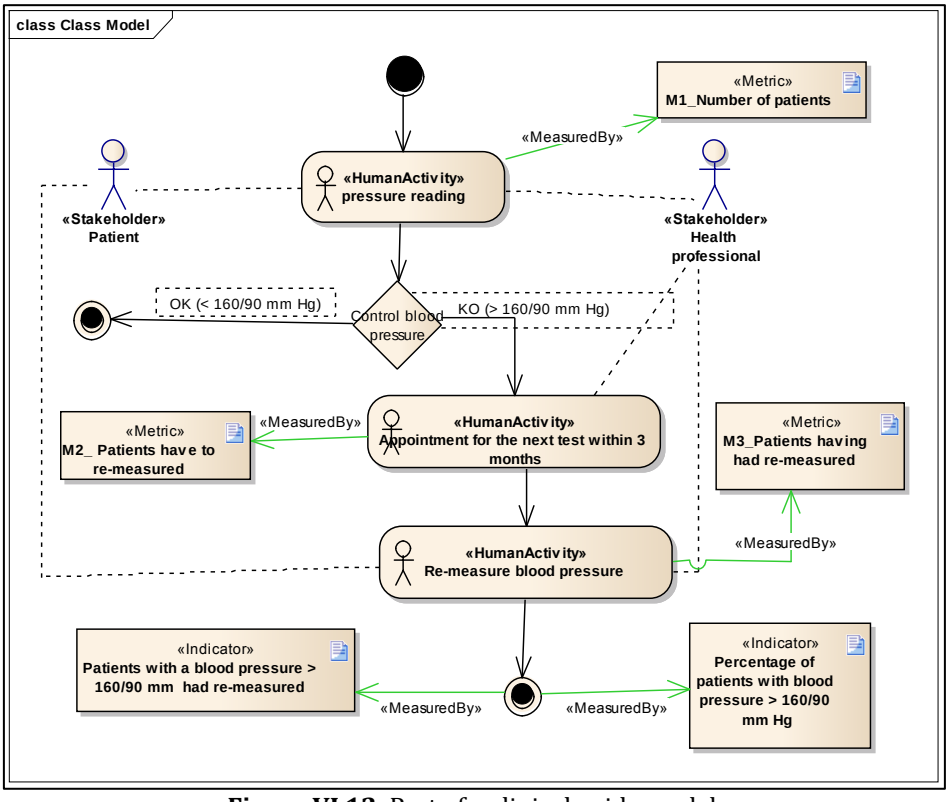

Figure VI.12. Part of a clinical guide model.

And next figures (Figure VI.13 and Figure VI.14) demonstrate parts of the formal description of the measurement concepts defined for this clinical guide.

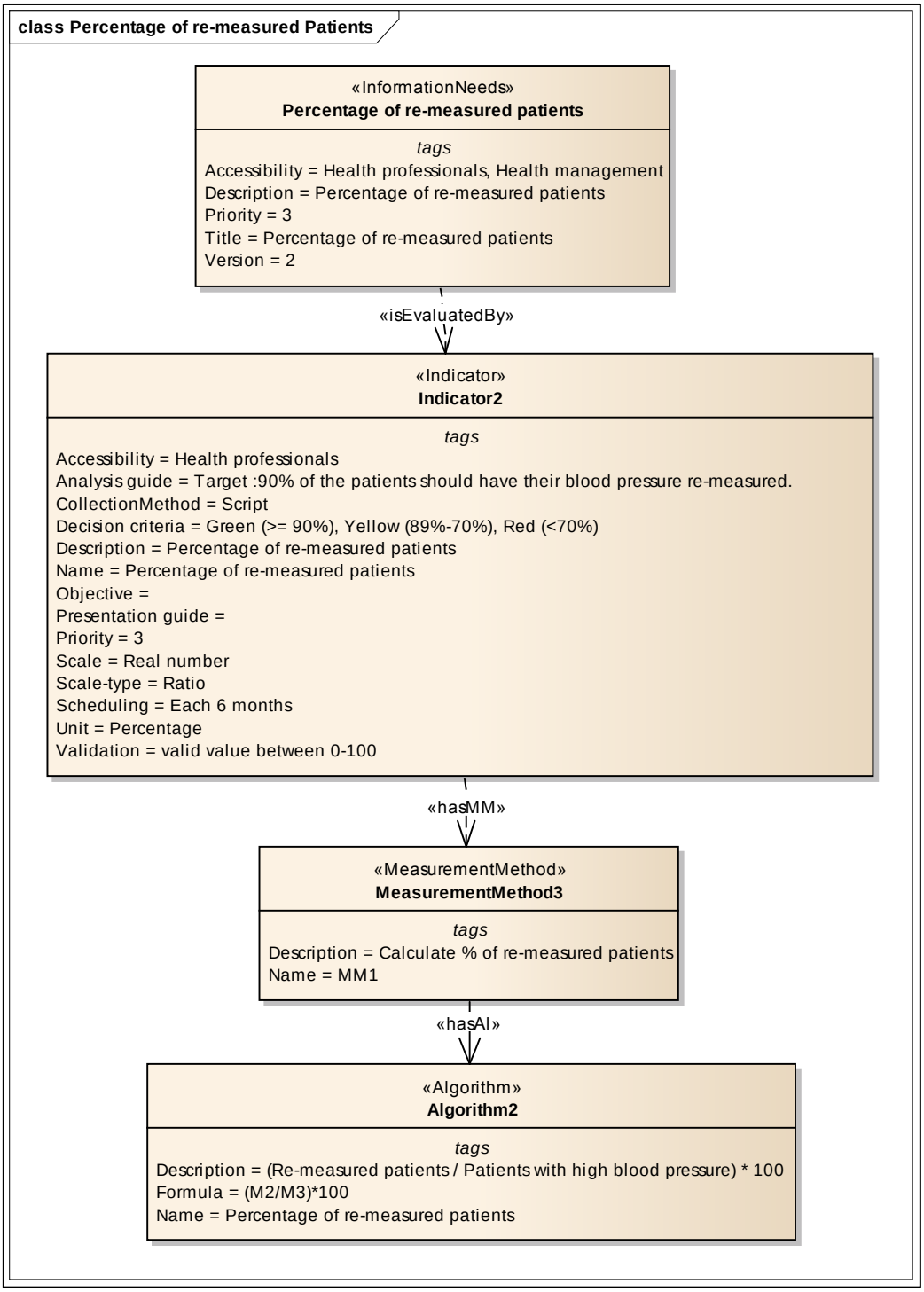

Figure VI.13. Part of the definition model of the measurement concepts (1)

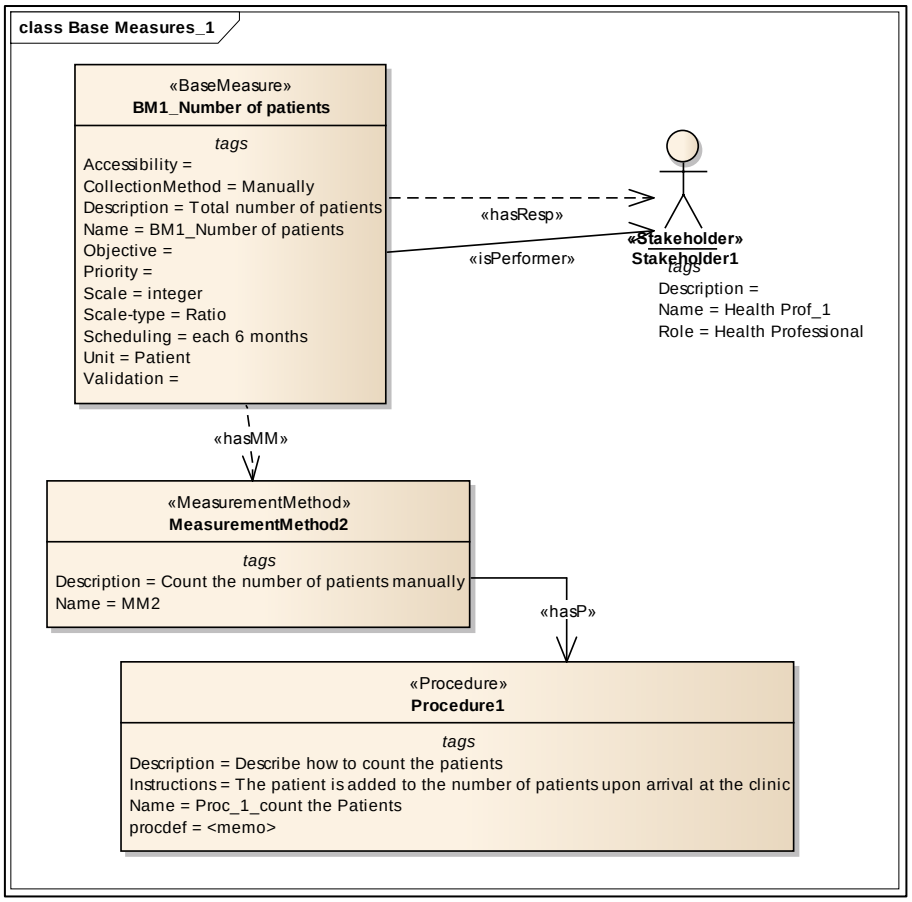

Figure VI.14. Part of the definition model of the measurement concepts (2).

And, finally, Figure VI.15 demonstrates part of a clinical guide dashboard.

Figure VI.15. Part of a clinical guide dashboard

4. The results of applying the proposed solution.

The proposed solution has provided the support needed by the project team to carry out the measurement activities during the project; the proposed measurement lifecycle has been used to plan and identify the required measurement activities, and the proposed

measurement modeling language (MDMM) has been used to describe the measurement objectives and concepts formally and operationally. Some examples of the benefits of applying the solution are described below.

The proposed measurement concepts and information model have contributed to the unification of the measurement vocabulary used in the project and connected them coherently, also ensured the traceability between these concepts.

Using the proposed measurement concepts and information model has promoted the clear and common understanding of the measurement goals and concepts and its relationships among stakeholders, which has supported communication between them.

Moreover, the proposed measurement language has supported the operational definition of the measurement concepts.

The proposed measurement lifecycle has provided a clear and comprehensive guide to the project team; it has defined and consolidated the measurement activities which should be performed during the clinical guide lifecycle.

Furthermore, this lifecycle has supported the project team in planning and performing the measurement activities by defining why? When? And how? These activities should be conducted. And also by establishing the criteria that control the quality of the measurement methods outputs.

As well as, the formal definition of the measurement concepts using the proposed MDMM has supported the communication between the different roles in the project and reduced the errors, time and the costs.

Also, the M2M and M2T transformations were used to automate the generation of the necessary measurement activities and artifacts that support the measurement throughout the process lifecycle.

Besides this, since the proposed solution addresses the international standards (e.g., ISO 1593-2017) and the best practices available in the literature, it has supported the project team to follow and comply with the measurement standards.

5. Chapter Summary

This chapter has described the influences which affected and motivated the development of this thesis work. Also explained how these influences had affected the choice of the validation project. Furthermore, it has outlined the validation project and its context. Besides, this chapter has illustrated the necessary developments that allowed the practical use of the theoretical solution presented in this work. And finally, it has described how the project team used the proposal? How has it affected project implementation? And the benefits obtained from its application.

The measurement lifecycle was applied by the project team to plan and perform the measurement activities throughout the clinical guide lifecycle.

Furthermore, a domain-specific language –based on the proposed MDMM- was developed and integrated into the CASE tool- used in the project- to allow the project team to define the measurement concepts of the clinical guides.

Besides, the M2M and M2T transformations were applied to derive the necessary measurement activities and artifacts that support the measurement throughout the process lifecycle. These transformations have automated the generation of these artifacts and activities which reduced the time, errors and the cost.

As a conclusion to this chapter, the results obtained from applying the solution proposed in this work and the feedback received from the project team, allow us to claim that the proposal presented in this thesis work is of real interest, and the results of its application can lead to good outcomes in different contexts..

Chapter VII Contribution, Future Work and Conclusions

Previous chapters have addressed the main aspects of the software process measurement. We have carried out a survey and a mapping study to study the current situation of the process measurement issues in the process management suites and the literature. The results obtained from these studies have highlighted the problem of this thesis and motivated the development of the proposed solution to support measurement issues and integrate them into the software lifecycle.

We have adopted the model-driven approach to developing the solution presented in this dissertation. The proposed solution consists of three main elements: a measurement lifecycle which defines and organized the required measurement activities and artifacts throughout the process lifecycle, a modeling language to describe the measurement concepts formally and operationally. And, finally, the necessary transformations to support the measurement lifecycle and promote the integration of the measurement concepts and artifacts into the process lifecycle.

Besides, we have developed a specific language (UML profile) and integrated it into a CASE tool to allow the practical use of the proposed solution in real environments, which allowed its validation in a real project.

This chapter begins with a description of the research framework in which this thesis was developed. Next, presents a summary of the main contributions of this work. Furthermore, it highlights the main lines for future research. Then, outline the main relationships that have been established during this work with other research groups and organizations. And, finally, presents the main conclusions of this work.

7.1 Research framework in which this work was carried out

From the beginning and throughout this dissertation, we have highlighted the important role of the IWT2 research group in the development of this work. It was not a theoretical role only but has also contributed significantly to the identification of the problems and needs faced by different working environments. Thanks to the extensive experience acquired by the group through the implementation of R & D transfer projects with the public sector (for example, the Ministry of Education, Health) and private organizations (such as Airbus Military, Tecnocom, and Fujitsu).

As mentioned before, the strategic research line of IWT2 group is investigating how to successfully combine the model-driven paradigm with the management of software/business processes to solve specific needs.

This strategic line contemplates; in the first place, the definition of the necessary theoretical frameworks to supports the comprehensive management of the software/business processes. And later, develop proposals that allow organizations to manage, know-how, and control its processes, through a series of CASE tools that support the practical use of these theoretical frameworks.

The main objectives that the group aims to achieve include, on the one hand, modernize organizations and improve their competitiveness, and on the other hand, support the continuous improvement of the quality of their processes through its effective and efficient management.

The IWT2 group has adopted the MDE paradigm as a core technology to achieve these objectives. Below, we provide an overview of the main research goals of the strategic research line of IWT2 group and the work carried out in each one.

- Provide organizations with a model based environment to orchestrate and execute their processes. This solution can reduce and even eliminate the existing gap between the defined processes and what is executed in reality. A process modeling language was developed to achieve this goal. This modeling language describes processes in a formal manner. Also, a series of protocols were proposed to transform the process definition model into its execution and orchestration model. Furthermore, these transformations derive the deployable and executable version of the process model to allow its execution in a process execution engine.

- Provide a framework to facilitate interoperability and maintainability of the software/business processes. This approach ensures the independence of the process model against the used modeling language, which supports the common understanding and the collaboration between the stakeholders in the inter-organizational projects.

- Provide organizations with the ability to modernize through the dynamic extraction of the global view of their processes. This solution gathers and handles the data stored in relational databases such as the table structures, constraints, and business rules. In this way, the essential elements involved in the organization can be obtained (e.g., the composition of processes and metrics, documents, work team structures, etc.).

- Support the requirements formalization process in the agile environment. This solution contributes to discovering the services in the early stages of agile SDPs. The requirements formalization is carried out by agile techniques like the Product Backlog or User Stories within the NDT-Agile framework. The formalized requirement can be mapped to a catalog of existing services to evaluate the existence of a service that already covers the requirement. A metamodel is defined to achieve this goal. This metamodel supports the requirement formalization. Mapping requirements to an existing catalog of services is an important activity because it promotes the reusability of software components.

- Provide a framework for modeling the Agile Requirements Engineering (RE). Organizations can use this solution to improve agile RE systematically. It can be used in terms of value delivery, and also, to enables improvements in terms of collaboration. In this context, the framework gives guidance for choosing appropriate agile techniques, which meet the needs of the people working in an organizational environment. Moreover, it allows improving hybrid process models for agile RE in a systematic manner without being restricted to one specific agile methodology. The

framework is based primarily on an agile RE metamodel which allows reducing complexity, as it provides an overview and clearness about the influencing parameters in an agile environment. In this context, A UML profile was developed to allow creating visual instances of the agile RE metamodel and facilitates the creation of domain-specific models. Such models show how the agile RE metamodel can be utilized in industry as well as how agile RE process models can be optimized through a problem-solving approach.

- Provide a framework to support organizations in monitoring and the continuous improvement of their processes. This solution facilitates the integration of the measurement concepts and artifacts into the software process lifecycle. This is the solution addressed in this thesis work. This proposal ensures the inclusion of the monitoring and continuous improvement activities from the early phases of the process lifecycle. It also links measurement concepts, activities, and artifacts with the process and organization objectives. This solution is based on a comprehensive measurement lifecycle, measurement metamodels, and the transformation rules that derive the measurement activities and artifacts required to support the measurement issues throughout the process lifecycle.

Besides, the IWT2 group is involved in several projects within the scope of clinical process management (e.g., the IDE$_4$ICDS project). These activities form part of the group's endeavor to transfer and harness knowledge to serve society.

7.2 Contributions of this thesis

As we have explained in the previous chapters, this thesis was motivated by ideas that came of earlier works.

On the one hand, the related works discussed in Chapter II, these works include proposals, methods, lifecycles, and process modeling languages.

On the other hand, the previous works of the IWT2 research group, these works are discussed in chapter III. The accumulated experience of the research group has motivated and supported the work under the model-driven paradigm and the necessary tools to work with this perspective, mainly UML and QVT.

Given all these influences and effort, this thesis culminates in a series of contributions that propose an original solution to support the organization in its endeavor to manage and improve its processes by integrating the measurement activities into its processes life cycle. We highlight below the main contributions of this work.

7.2.1 A survey business process management suites

A survey (see chapter II.1.1) on the existing open source BPMSs was performed to investigate to which extend the current tools support the process lifecycle and also to highlight the existing gaps and possible improvements. Furthermore, this survey provides a guide for the organizations to plan and perform a comparative on the existing BPMSs formally.

We have developed a characterization scheme to allow the evaluation of the different aspects of all the process lifecycle phases. After applying the inclusions and exclusions criteria, seven BPMSs were included in this study.

Several results have been obtained from this study, one of the main findings of this survey is demonstrating the existing weakness in the definition of the measurements and its integration into the process lifecycle; the majority of the investigated tools do not support the definition and integration of the process measurement. Therefore, the study mention "the necessity to perform further investigations on the definition and integration of the process measurements" as a recommendation for future work.

7.2.2 Mapping study on the measurement of the software process

The results and conclusions obtained from the previous survey led to conducting a mapping study (see chapter II.1.2) on the measurement of the SDP and its execution projects. The primary objective of this mapping study is to get a better understanding of the measurement in the context of the SDP, to classify and structure the existing research in this area.

After applying the selection criteria, 462 papers were included in this study. These papers were classified into four topics based on their focus and into three groups according to the publishing date. Five abstractions and 64 attributes were identified, 25 methods/models and 17 contexts were distinguished.

The main findings of the study include: the capability and performance were the most measured process attributes, while effort and performance were the most measured project attributes. Besides, Goal Question Metric and Capability Maturity Model Integration were the main methods and models used in the studies, whereas, agile/lean development and small/medium-size enterprise were the most frequently identified research contexts.

Furthermore, the study illustrates scarcity of research focusing on the definition of measurements in the form that allows its integration into the process lifecycle. The results also reveal that the complete and operational definition of measurements, as well as considering the measurement issue in all the process stages is essential for promoting the project management and process improvement.

The results obtained from the previous studies (i.e., the survey and the mapping study) revealed the need for further studies on the software process measurement (i.e., objectives, activities, concepts, and artifacts) and its integration into the software process.

These results represent the main motivation for conducting the research presented in this thesis. The prime objective of this work is to study the measurement issues in all stages of the software process, also to support the integration of measurement issues into the process lifecycle.

One of the main contributions of this thesis is a theoretical solution to support the measurement issues during the software process lifecycle; this solution is based on three fundamental pillars: (i) a measurement lifecycle, (ii) metamodels, and (ii) transformations.

7.2.3 Measurement lifecycle

The measurement lifecycle proposed in chapter III. 4.1.2 define the measurement activities throughout the software process lifecycle. Also, organize these activities in six phases.

The first phase includes the necessary tasks to select and specify the measurement goals and concepts in an operational manner, this definition includes all the information required to support the measurement in the rest of the lifecycle. The second phase comprises the description of the measurement concepts and goals in a formal language and integrates this description into the process model. In the next stage, the process execution environment is prepared to allow the collection, validation and storing measurement data during the process execution. The activities of the fourth phase include gathering, validating, preparing, and storing the measurement data during the process execution. The fifth phase covers the necessary tasks to analyze and report the measurement data. And in the final stage of the lifecycle, the measurement process is evaluated and assessed to discover improvement opportunities.

This lifecycle is more comprehensive and detailed than the existing measurement lifecycles (described in chapter II.2.4); the proposed lifecycle distinguishes and defines six phases, while the others define four stages.

Besides, the proposed lifecycle is the only lifecycle that addresses the operational and formal description (modeling) of measurement concepts.

7.2.4 Measurement concepts and information model

In Chapter III.4.2, we define the necessary measurement concepts to support the measurement process. Besides, we describe the relationships between these concepts in the proposed measurement information model.

7.2.5 Measurement metamodels

The metamodels described in chapter IV support and automate the measurement lifecycle; the first metamodel is the MDMM, this metamodel supports the complete and formal definition of the measurement goals and concepts. The second is the measurement execution metamodel, which comprises the measurement aspects during the process execution, and finally, the monitoring metamodel which support the fifth phase of the measurement lifecycle (Analysis and Reporting).

7.2.6 Transformation process and rules

We have defined a systematic transformation process of two phases (see chapter V). This process uses the relationships between the measurement metamodels to derive the artifacts that support the measurement process throughout its lifecycle.

This process specifies the necessary QVT rules to derive the measurement execution and monitoring models from the measurement definition model.

Furthermore, specify the required MOFM2T transformations to allow the automatic generation of the measurement documentation, the necessary WS-BPEL and BPEL4People

codes to execute the measurement activities, and the needed xml artifacts to generate the monitoring dashboard.

These transformation rules support the measurement lifecycle by automating the measurement activities and reducing necessary human intervention.

Moreover, we have developed a domain-specific modeling language. This language allows the formal definition of the measurement goals and concepts. Besides, we have integrated this modeling language into the CASE tool "Enterprise Architect" to support the practical use of the proposed solution.

Once the proposed solution is developed and ready for practical use, it has been validated through a real project to evaluate its usefulness and ensure that it meets the expectations. This experience not only served to validate the proposed solution but also has opened new investigation lines to improve it.

7.3 Future Work

After summarizing the main contributions of this work, it is convenient to explain that this doctoral thesis is not the end of the research work. The results and the experiences obtained from this work have opened new lines for future research. Below, we point some possible directions for future research.

Research on supporting the analysis of the measurement model to verify its integrity and ensure its syntax and semantic validity could support the measurement definition and modeling phase.

Furthermore, the traceability of the relationships between the measurement objectives and concepts (i.e., indicators and measures) highlight the role and weight of each measurement element, this contributes to understanding the importance of each measurement element. Improving the possibility of tracking these relationships is one of the areas for future research. Moreover, the investigation could focus on creating a reusable measurement catalog and patterns for different domains (e.g., software development, business process, and clinical guides).

Another investigation line is related to the exploitation of the collected measurement data; in order to learn from previous experiences and to improve the ability to foresee the future performance, investigations may focus on applying data science methods to the measurement data (e.g., data mining, fuzzy logic, statistical process control, and data envelopment analysis).

Many CASE tools (e.g., Enterprise Architect) do not support transformation languages such as QVT and MOFM2T. Therefore, the research on promoting the practical use of these transformation languages is another trend for future work.

In the last decades, software development has evolved to meet the market needs and to keep abreast of modern technologies and infrastructures. Thus, the measurement process needs to keep up with this evolution. Therefore, adapting the measurement

process to keep abreast of modern development processes (e.g., DevOps, distributed development, and outsourcing) is another line for future research.

7.4 Relations with other research groups and the business environment

The participation of the doctoral student in several R & D and transfer projects has given rise to very close relationships with other research groups, universities, and organizations from the business sector.

On the one hand, many relationships were established with other research groups and universities during this thesis. We seek to strengthen them in the future to exchange ideas and experiences. Below, we highlight some of these relationships.

Technology Innovation Group (GiT). As detailed in chapter VI, the results of this thesis have been validated in the project IDE$_4$ICDS to support the development of a module to monitor the status of the clinical guides.

In this project, the IWT2 research group - in which this thesis is developed - has collaborated closely with the Technological Innovation Group (GiT). GiT is an R & D & I research group, work under the coordination of the Head of R & D & I and Manager of the Andalusian Public Foundation for Health Research Management in Seville (FISEVI). And in close collaboration with the Clinical Management Units (UGC) and the Non-Assistance Units of the University Hospitals: Virgen Macarena and Virgen del Rocío, especially in the Information Technology Service.

Maximilians-University Munich, Institute for Informatics (Germany). The collaboration between the doctoral student and Dr. Nora Koch were essential to capture the requirements of the measurement frameworks, identify relevant works in the literature, and to define the main objectives of the solution proposed in this thesis work. This collaboration has contributed to the achievement of the research: "A model-based proposal for integrating the measures lifecycle within the process lifecycle" (A Meidan et al. 2017).

University of Applied Sciences Emden/Leer, (Germany). The doctoral student has collaborated with Dr. Eva-Maria Schön to prepare and perform the mapping study (chapter II.1.2). This collaboration has contributed to the research methodology, selecting the research databases, and formulating the research string of this study.

On the other hand, the experience of evaluating and validating the proposed solution in a real project has served to establish many relationships with several public and private organizations. These relationships open new work lines and increase the possibilities of applying the results obtained from this thesis in real environments. Here are examples of these organizations.

Soltel IT Software, **Fujitsu** and **Serviguide Consultoria**, the IWT2 group maintains close collaboration with these companies in several projects regarding different business areas (for example, in the field of health and the management of

clinical care processes in various medical areas, among others). The doctoral student has collaborated with members of these companies to achieve works related to business process design & modeling and model-driven engineering tasks. Furthermore, these companies are partners in the realization of the project IDE$_4$ICDS, which allowed the doctoral student to exchange experiences, knowledge, and ideas with members of these companies.

7.5 Conclusions

This dissertation presents a detailed description of all phases of this thesis work, starting from the definition of the problem based on the results obtained from the preliminary studies to proposing the theoretical solution, its implementation, its development to allow its practical use, and finally, to its evaluation in a real project.

The first chapter introduces the main concepts related to the domain of this thesis and presents the structure of this document.

The second chapter presents the research conducted to understand the current state of the art and to discover the existing gaps in the domain. This chapter comprises a summary of a survey and mapping study and outlines its main results. Furthermore, it discusses the main related proposals..

The third chapter presents the thesis problem, the objectives of this work, proposals which influenced this work. Also describes the main milestones of the proposed solution. Furthermore, this chapter introduces the proposed measurement lifecycle and concepts.

The proposed measurement metamodels are described in chapter four; the first metamodel supports the operational and formal definition of the measurement concepts, the second metamodel (i.e., the execution metamodel) represents the measurement issues during the process execution, and finally, the monitoring metamodel which supports the process monitoring, management and improvement.

Next chapter presents a systematic transformation process to automate the derivation of the necessary artifacts to support the measurement lifecycle and its integration into the process lifecycle. Through this chapter, QVT rules were defined to derive the required model to model transformations (e.g., the measurement execution model from the measurement definition model). Moreover, MOFM2T transformation templates have been defined to perform the required model to text transformations (e.g., generate XML, WS-BPEL and BPEL4People code).

The sixth chapter describes how the theoretical solution proposed in this thesis was developed in the form that allows its practical use. Also explains how a specific language (UML profiles) was developed to enable the practical use of the metamodels defined in this thesis. This language has been integrated into the CASE tool used in the validation project. Subsequently, the chapter outlines how the proposal was applied in the project, the results, and the main conclusions of this experience.

The last chapter outlines the context, strategic framework, and the related works that significantly influenced this work. Also, it highlights the relationship between this work and the investigation line of the IWT2 research group. Furthermore, the main contributions and future investigations are summarized in this chapter. Moreover, it describes the main relationships established between the doctoral student and the other research groups, universities, and business organizations.

References

Aalst, van der, and Wil M. P. 2013. "Business Process Management: A Comprehensive Survey." *ISRN Software Engineering* 2013: 1–37. https://www.hindawi.com/archive/2013/507984/.

van der Aalst, W.M.P. 2004. "Business Process Management: A Personal View." *Business Process Management Journal* 10(2): bpmj.2004.15710baa.001. http://www.emeraldinsight.com/doi/10.1108/bpmj.2004.15710baa.001.

van der Aalst, W.M.P., and A.H.M. ter Hofstede. 2005. "YAWL: Yet Another Workflow Language." *Information Systems* 30(4): 245–75. http://linkinghub.elsevier.com/retrieve/pii/S0306437904000304 (February 10, 2018).

van der Aalst, Wil M. P. 2004. "Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management." In Springer, Berlin, Heidelberg, 1–65. http://link.springer.com/10.1007/978-3-540-27755-2_1.

van der Aalst, Wil, A. H. M. ter Hofstede, and M. Weske. 2003. "Business Process Management: A Survey." In *Lecture Notes in Computer Science*, , 1–12. http://www.springerlink.com/index/9YH5WYAWLWV20UAE.pdf.

Abreu Fernando Brito, and Rogério Carapuça. 1994. "Object-Oriented Software Engineering: Measuring and Controlling the Development Process." In *Proceedings of the 4th International Conference on Software Quality*, , 1–8.

Actifsource. 2018. "Actifsource." http://www.actifsource.com/.

Ahern, Dennis M, Aaron Clouse, and Richard Turner. 2004. *CMMI Distilled: A Practical Introduction to Integrated Process Improvement*. Addison-Wesley Professional.

Alegría, Julio Ariel Hurtado, Alejandro Lagos, Alexandre Bergel, and María Cecilia Bastarrica. 2010. "Software Process Model Blueprints." In Springer, Berlin, Heidelberg, 273–84. http://link.springer.com/10.1007/978-3-642-14347-2_24.

Atkinson, Colin, and Thomas Kühne. 2002. "The Role of Metamodeling in MDA." In *Proc. UML 2002 Workshop on Software Model Engineering*, , 67–70.

Audet, Anne-Marie, Sheldon Greenfield, and Marilyn Field. 1990. "Medical Practice Guidelines: Current Activities and Future Directions." *Annals of Internal Medicine* 113(9): 709–14.

Avison, David E, and Guy Fitzgerald. 1999. "Information Systems Development." *Rethinking management information systems*: 250–78.

Bagherzadeh, Mojtaba, Nicolas Hili, and Juergen Dingel. 2017. "Model-Level, Platform-

Independent Debugging in the Context of the Model-Driven Development of Real-Time Systems." In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, New York, New York, USA: ACM Press, 419–30. http://dl.acm.org/citation.cfm?doid=3106237.3106278 (October 30, 2018).

Balch, George I. 1974. "Multiple Indicators in Survey Research: The Concept' sense of Political Efficacy.'" *Political Methodology*: 1–43.

Bandara, Wasana, Guy G Gable, and Michael Rosemann. 2005. "Factors and Measures of Business Process Modelling: Model Building through a Multiple Case Study." *European Journal of Information Systems* 14(4): 347–60. http://link.springer.com/10.1057/palgrave.ejis.3000546 (January 6, 2018).

Bang, Soon K, Sam Chung, Young Choh, and Marc Dupuis. 2013. "A Grounded Theory Analysis of Modern Web Applications: Knowledge, Skills, and Abilities for DevOps." In *Proceedings of the 2nd Annual Conference on Research in Information Technology*, , 61–62.

Barcellos, Monalessa Perini, Ricardo de Almeida Falbo, and Ana Regina Rocha. 2013. "A Strategy for Preparing Software Organizations for Statistical Process Control." *Journal of the Brazilian Computer Society* 19(4): 445–73. http://www.journal-bcs.com/content/19/4/ (January 27, 2018).

Bellini, Carlo Gabriel Porto, Rita De Cássia De Faria Pereira, and João Luiz Becker. 2008. "Measurement in Software Engineering: From the Roadmap To the Crossroads." *International Journal of Software Engineering and Knowledge Engineering* 18(1): 37–64.

Bendraou, Reda, Benoit Combemale, Xavier Crégut, and Marie-Pierre Gervais. 2007. "Definition of an Executable SPEM 2.0." In *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, , 390–97.

Bendraou, Reda, Marie-Pierre Gervais, and Xavier Blanc. 2006. "UML4SPM: An Executable Software Process Modeling Language Providing High-Level Abstractions." In *Enterprise Distributed Object Computing Conference, 2006. EDOC'06. 10th IEEE International*, , 297–306.

Berander, Patrik, and Per Jönsson. 2006. "A Goal Question Metric Based Approach for Efficient Measurement Framework Definition." *Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering - ISESE '06*: 316–25. http://portal.acm.org/citation.cfm?doid=1159733.1159781.

Bézivin, Jean. 2005. "On the Unification Power of Models." *Software & Systems Modeling* 4(2): 171–88.

Bhide, Sandhiprakash. 1990. "Generalized Software Process-Integrated Metrics Framework." *Journal of Systems and Software* 12(3): 249–54.

Bobkowska, Anna. 2001. "Quantitative and Qualitative Methods in Process Improvement and Product Quality Assessment." In *Proc. of the ESCOM*, , 347–56.

Boehm, Barry. 2006. "A View of 20th and 21st Century Software Engineering." In *Proceedings of the 28th International Conference on Software Engineering*, , 12–29.

Bonitasoft. 2016. "Bonitasoft." http://www.bonitasoft.com/.

Borgoñón, Laura García. 2016. “A Framework to Facilitate the Interoperability and Maintainability of Software Process Models - Thesis.” https://documat.unirioja.es/servlet/autor?codigo=3895499 (February 14, 2017).

Bourgault, Mario et al. 2002. “Discussion of Metrics for Distributed Project Management: Preliminary Findings.” In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, , 10--pp.

Bourque, P. et al. 1999. “The Guide to the Software Engineering Body of Knowledge.” *IEEE Software* 16(6): 35–44. http://ieeexplore.ieee.org/document/805471/ (February 19, 2018).

Briand, L.C., S. Morasca, and V.R. Basili. 2002. “An Operational Process for Goal-Driven Definition of Measures.” *IEEE Transactions on Software Engineering* 28(12): 1106–25. http://ieeexplore.ieee.org/document/1158285/ (January 27, 2018).

Card, D.N., and C.L. Jones. 2003. “Status Report: Practical Software Measurement.” In *Third International Conference on Quality Software, 2003. Proceedings.*, IEEE, 315–20. http://ieeexplore.ieee.org/document/1319116/ (February 6, 2018).

Chou, Shih-Chien. 2002. “A Process Modeling Language Consisting of High Level UML-Based Diagrams and Low Level Process Language.” *Journal of Object Technology* 1(4): 137–63.

Chowdhary, Pawan et al. 2006. “Model-Driven Dashboards for Business Performance Reporting.” In *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC’06)*, IEEE, 374–86. http://ieeexplore.ieee.org/document/4031224/ (July 7, 2018).

Chrissis, Mary Beth, Michael Konrad, and Sandra Shrum. 2011. *CMMI for Development: Guidelines for Process Integration and Product Improvement*. Third. Addison-Wesley Professional.

Cugola, G, and C Ghezzi. 1998. “Software Processes: A Retrospective and a Path to the Future.” *Software Process: Improvement and Practice*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.2499&rep=rep1&type=pdf.

Curtis, Bill, Marc I Kellner, and Jim Over. 1992. “Process Modeling.” *Commun. ACM* 35(9): 75–90. http://doi.acm.org/10.1145/130994.130998.

Czarnecki, Krzysztof, and Simon Helsen. 2003. “Classification of Model Transformation Approaches.” *2nd OOPSLA’03 Workshop on Generative Techniques in the Context of MDA*: 1–17. http://www.softmetaware.com/oopsla2003/czarnecki.pdf.

Daskalantonakis, M.K. 1992. “A Practical View of Software Measurement and Implementation Experiences within Motorola.” *IEEE Transactions on Software Engineering* 18(11): 998–1010.

Degueule, Thomas et al. 2015. “Melange: A Meta-Language for Modular and Reusable Development of DSLs.” *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*: 25–36. https://dl.acm.org/citation.cfm?id=2814252 (November 1, 2018).

Dekkers, C.A., and P.A. McQuaid. 2002. “The Dangers of Using Software Metrics to (Mis)manage.” *IT Professional* 4(2): 24–30. http://ieeexplore.ieee.org/document/1000457/ (January 23, 2018).

Del-Río-Ortega, Adela, Manuel Resinas, Cristina Cabanillas, and Antonio Ruiz-Cortés. 2013. "On the Definition and Design-Time Analysis of Process Performance Indicators." *Information Systems* 38(4): 470–90.

Del-Río-Ortega, Adela, Manuel Resinas, and A Ruiz-Cortés. 2009. "Towards Modelling and Tracing Key Performance Indicators in Business Processes." *II Taller sobre Procesos de Negocio e ingeniería de Servicios, PNIS*.

del-Río-Ortega, Adela, Manuel Resinas, and Antonio Ruiz-Cortés. 2010. "Defining Process Performance Indicators: An Ontological Approach." In Springer, Berlin, Heidelberg, 555–72. http://link.springer.com/10.1007/978-3-642-16934-2_41 (February 10, 2018).

Deming, W Edwards. 1986. "Out of the Crisis, Massachusetts Institute of Technology." *Center for advanced engineering study, Cambridge, MA* 510.

Dybå, Tore, Torgeir Dingsøyr, and Geir Kjetil Hanssen. 2007. "Applying Systematic Reviews to Diverse Study Types: An Experience Report." In *ESEM*, , 225–34.

Ebert, Christof, Reiner Dumke, Manfred Bundschuh, and Andreas Schmietendorf. 2007. Best Practices in Software Measurement: How to use Metrics to Improve Project and Process Performance *Best Practices in Software Measurement: How to Use Metrics to Improve Project and Process Performance*. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84891983644&partnerID=40&md5=bbf0be8edefdd1b27a1071bf2ec21f55.

"Eclipse Process Framework Project | Projects.eclipse.org." 2017. https://projects.eclipse.org/projects/technology.epf.

Ejiogu, Lem O. 1993. "Five Principles for the Formal Validation of Models of Software Metrics." *ACM SIGPLAN Notices* 28(8): 67–76.

Eliyahu, and M Goldratt. 1990. *The Haystack Syndrome: Sifting Information out of the Data Ocean*. North River Press.

Emam, Khaled El, Walcelio Melo, and Jean-Normand Drouin. 1997. *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*. IEEE Computer Society Press.

Escalona, M.J. 2004. "Modelos Y Técnicas Para La Especificación Y El Análisis de La Navegación En Sistemas Software." University of Seville.

Escalona, M.J, and G. Aragon. 2008. "NDT. A Model-Driven Approach for Web Requirements." *IEEE Transactions on Software Engineering* 34(3): 377–90. http://ieeexplore.ieee.org/document/4497213/ (January 17, 2018).

Escalona, M J, J Torres, M Mejías, and A Reina. 2003. "NDT-Tool: A Case Tool to Deal with Requirements in Web Information Systems." In *Web Engineering*, eds. Juan Manuel Cueva Lovelle et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 212–13.

Feigenbaum, A.V. 2001. "How To Manage Quality in Today's Economy." *Quality Progress* 34(5): 26–27.

Fenton, Norman E. 1991. *Software Metrics : A Rigorous Approach*. Chapman & Hall.

Fenton, Norman E., and Shari Lawrence Pfleeger. 1996. *Software Metrics: A Rigorous and Practical Approach*. Second Edi. London: International Thomson Computer Press.

Fenton, Norman, and Barbara Kitchenham. 1991. “Validating Software Measures.” *Software Testing, Verification and Reliability* 1(2): 27–42.

Fleurey, Franck et al. 2006. “Kermeta Language, Reference Manual.” *Internet: http://www. kermeta. org/docs/KerMeta-Manual. pdf. IRISA*.

Florac, WA, and AD Carleton. 1999. *Measuring the Software Process: Statistical Process Control for Software Process Improvement*. https://books.google.es/books?hl=en&lr=&id=gcA-S8bK9YoC&oi=fnd&pg=PR9&dq=W.A.+Florac+and+A.D.+Carleton,+Measuring+the+Software+Process&ots=wbJ7ffMpAS&sig=Qtg7edID1-9gPXhhxtdSc7kgaVQ.

Fondement, Frédéric, and Raul Silaghi. 2004. “Defining Model Driven Engineering Processes.” In *Third International Workshop in Software Model Engineering (WiSME), Held at the 7th International Conference on the Unified Modeling Language (UML)*,.

Freire, Marília Aranha et al. 2011. “Automatic Deployment and Monitoring of Software Processes: A Model-Driven Approach.” In *SEKE*, , 42–47.

Fuggetta, Alfonso. 2000. “Software Process: A Roadmap.” In *Proceedings of the Conference on the Future of Software Engineering*, , 25–34.

García-Borgoñón, L. et al. 2014. “Software Process Modeling Languages: A Systematic Literature Review.” *Information and Software Technology* 56(2): 103–16.

García-Borgoñon, L., J. A. García-García, M. Alba, and M. J. Escalona. 2013. “Software Process Management: A Model-Based Approach.” In *Building Sustainable Information Systems*, Boston, MA: Springer US, 167–78.

Garcia-Garcia, Julián Alberto. 2015. “Una Propuesta Para El Uso Del Paradigma Guiado Por Modelos (MDE) Para La Definición Y Ejecución de Procesos de Negocios.” Sevilla. https://idus.us.es/xmlui/handle/11441/26740.

Garcia-Garcia, Julián Alberto, Ayman Meidan, Antonio Vázquez Carreño, and Manuel Mejias Risoto. 2017. “A Model-Driven Proposal to Execute and Orchestrate Processes: PLM4BS.” In Springer, Cham, 211–25.

García, Félix et al. 2006. “Towards a Consistent Terminology for Software Measurement.” *Information and Software Technology* 48(8): 631–44.

García García, Julián Alberto et al. 2015. “Clinical Process Management: A Model-Driven & Tool-Based Proposal.”

Gardner, Robert A. 2001. “Resolving the Process Paradox.” *Quality progress* 34(3): 51.

Giachetti, Giovanni, Beatriz Marín, and Oscar Pastor. 2008. “Perfiles UML Y Desarrollo Dirigido Por Modelos: Desafíos Y Soluciones Para Utilizar UML Como Lenguaje de Modelado Específico de Dominio.” *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos* 2(3).

Giaglis, George M. 2001. “A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques.” *International Journal of Flexible Manufacturing Systems* 13(2): 209–28. http://link.springer.com/10.1023/A:1011139719773.

Gill, P J. 1999. “Application Development: Business Snapshot-Business Modeling Tools Help Companies Align Their Business and Technology Goals.” *Information week* (April).

Goethert, Wolfhart, and Jeannine Siviy. 2004. Software Engineering Institute *Applications of the Indicator Template for Measurement and Analysis*. http://repository.cmu.edu/sei/490.

Gómez Oswaldo, Hanna Oktaba Mario Piattini, and Félix García. 2006. "A Systematic Review Measurement in Software Engineering: State-of-the-Art in Measures." In *International Conference on Software and Data Technologies*, , 165–76.

Habra, Naji, Alain Abran, Miguel Lopez, and Asma Sellami. 2008. "A Framework for the Design and Verification of Software Measurement Methods." *Journal of Systems and Software* 81(5): 633–48.

Havey, Michael. 2005. *Essential Business Process Modeling*. O'Reilly. https://dl.acm.org/citation.cfm?id=1203641.

Hetzel, William C., and Bill. 1993. *Making Software Measurement Work : Building an Effective Measurement Program*. QED Pub. Group. https://dl.acm.org/citation.cfm?id=529480 (January 3, 2018).

Hikichi, Kazumasa, Kyohei Fushida, Hajimu Iida, and Ken'ichi Matsumoto. 2006. "A Software Process Tailoring System Focusing to Quantitative Management Plans." In Springer, Berlin, Heidelberg, 441–46.

Hill, Janelle B, Jim Sinur, David Flint, and Michael James Melenovsky. 2006. "Gartner's Position on Business Process Management." *Gartner Research G* 136533.

Humphrey, W. S. 1995. *A Discipline for Sofware Engineering*. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Humphrey, Watts S. 1988. "Characterizing the Software Process: A Maturity Framework." *IEEE software* 5(2): 73–79.

Humphrey, Watts S. 1989. 16 OMEGAINTERNATIONAL JOURNAL OF MANAGEMENT SCIENCE *Managing the Software Process*. Addison-Wesley.

IBM. 2018. "IBM Rational Software Architect Designer." https://www.ibm.com/us-en/marketplace/rational-software-architect-designer/details?mhq=Rational Software Modeler&mhsrc=ibmsearch_p.

"IBM - Rational Method Composer." 2017. http://www-03.ibm.com/software/products/en/rmc.

IDE4ICDS. 2017. "IDE4ICDS." https://www.serviguide.com/portfolio-posts/ide4icds/.

ISO/IEC/IEEE 12207-2017-International Standard - Systems and Software Engineering -- Software Life Cycle Processes. 2017. https://www.iso.org/standard/63712.html.

ISO/IEC/IEEE 15288-Systems and Software Engineering System Life Cycle Processes. 2015. 15288 www.iso.org.

"ISO/IEC/IEEE 15939 International Standard - Systems and Software Engineering--Measurement Process." 2017. *ISO/IEC/IEEE 15939:2017(E)*: 1–49.

"ISO/IEC 15504-5:2006, Information Technology - Process Assessment - Part 1-5:" 2006.

Jacquet, J.-P., and A. Abran. 1997. "From Software Metrics to Software Measurement Methods: A Process Model." In *Proceedings of IEEE International Symposium on Software Engineering Standards*, IEEE Comput. Soc, 128–35.

Jethani, Kanhaiya. 2013. “Software Metrics for Effective Project Management.” *International Journal of System Assurance Engineering and Management* 4(4): 335–40.

Jouault, Frédéric, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. 2008. “ATL: A Model Transformation Tool.” *Science of Computer Programming* 72(1–2): 31–39. https://www.sciencedirect.com/science/article/pii/S0167642308000439 (October 30, 2018).

Kahani, Nafiseh et al. 2018. “Survey and Classification of Model Transformation Tools.” *Software & Systems Modeling*. http://link.springer.com/10.1007/s10270-018-0665-6 (October 30, 2018).

Kastenberg, Harmen, and Arend Rensink. 2006. “Model Checking Dynamic States in GROOVE.” In Springer, Berlin, Heidelberg, 299–305. http://link.springer.com/10.1007/11691617_19 (October 30, 2018).

Kasunic, Mark. 2006. *The State of Software Measurement Practice: Results of 2006 Survey*. Pittsburgh, PA. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8095.

Kelly, Steven, Kalle Lyytinen, and Matti Rossi. 1996. “MetaEdit+ A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment.” In Springer, Berlin, Heidelberg, 1–21. http://link.springer.com/10.1007/3-540-61292-0_1 (November 1, 2018).

Kitchenham, B. A., and Barbara A. 1996. *Software Metrics : Measurement for Software Process Improvement*. NCC Blackwell. https://dl.acm.org/citation.cfm?id=525161.

Kitchenham, B., and S Charters. 2007. “Guidelines for Performing Systematic Literature Reviews in Software Engineering.” In *Technical Report, Ver. 2.3 EBSE Technical Report. EBSE*,.

Kitchenham, Barbara A, Robert T Hughes, and Stephen G Linkman. 2001. “Modeling Software Measurement Data.” *IEEE Transactions on Software Engineering* 27(9): 788–804.

Kitchenham, Barbara, and Shari Lawrence Pfleeger. 1996. “Software Quality: The Elusive Target.” *IEEE software* 13(1): 12.

Kitchenham, Barbara, Shari Lawrence Pfleeger, and Norman Fenton. 1995. “Towards a Framework for Software Measurement Validation.” *IEEE Transactions on software Engineering* 21(12): 929–44.

Kleppe, Anneke, Jos Warmer, and Wim Bast. 2003. 83 AddisonWesley Professional *MDA Explained: The Model Driven Architecture: Practice and Promise*. http://www.amazon.com/MDA-Explained-Architecture-Practice-Promise/dp/032119442X.

Von Krogh, Georg, Sebastian Spaeth, and Karim R. Lakhani. 2003. “Community, Joining, and Specialization in Open Source Software Innovation: A Case Study.” *Research Policy* 32(7): 1217–41.

Kuwaiti, M.E., and John M. Kay. 2000. “The Role of Performance Measurement in Business Process Re-engineering.” *International Journal of Operations & Production Management* 20(12): 1411–26. http://www.emeraldinsight.com/doi/10.1108/01443570010353086 (January 29, 2018).

Larrucea, Xabier, and Eider Iturbe. 2010. “A Metamodel Integration for Metrics and

Processes Correlation." In *ICSOFT (1)*, , 63–68.

Lennselius, B, C Wohlin, and C Vrana. 1987. "Software Metrics: Fault Content Estimation and Software Process Control." *Microprocessors and Microsystems*. http://www.sciencedirect.com/science/article/pii/0141933187905242 (July 12, 2016).

Lindvall, Mikael, Paolo Donzelli, Sima Asgari, and Vic Basili. 2005. "Towards Reusable Measurement Patterns." In *Proceedings of the 11th IEEE International Software Metrics Symposium*, METRICS '05, Washington, DC, USA: IEEE Computer Society, 21--. http://dx.doi.org/10.1109/METRICS.2005.49.

Lough, Daniel Lowry. 2001. "A Taxonomy of Computer Attacks with Applications to Wireless Networks." Virginia Polytechnic Institute and State University.

Lúcio, Levi et al. 2016. "Model Transformation Intents and Their Properties." *Software & Systems Modeling* 15(3): 647–84. http://link.springer.com/10.1007/s10270-014-0429-x (October 30, 2018).

Magennis, Troy. 2015. "The Economic Impact of Software Development Process Choice -- Cycle-Time Analysis and Monte Carlo Simulation Results." In *2015 48th Hawaii International Conference on System Sciences*, IEEE, 5055–64.

Markovic, Ivan, and Alessandro Costa Pereira. 2008. "Towards a Formal Framework for Reuse in Business Process Modeling." In Springer, Berlin, Heidelberg, 484–95. http://link.springer.com/10.1007/978-3-540-78238-4_49 (February 10, 2018).

Martin, James N. 2015. "Architecture Definition -- A New Process in the ISO International Systems Engineering Standard." *INCOSE International Symposium* 25(1): 463–72.

Mateo, José Antonio, Valentín Valero, and Gregorio Díaz. 2012. "An Operational Semantics of BPEL Orchestrations Integrating Web Services Resource Framework." In Springer, Berlin, Heidelberg, 79–94.

McGarry, John. 2002. *Practical Software Measurement: Objective Information for Decision Makers*. Addison-Wesley Professional.

Meidan, A et al. 2017. "A Model-Based Proposal for Integrating the Measures Lifecycle within the Process Lifecycle."

Meidan, A., J.A. García-García, M.J. Escalona, and I. Ramos. 2016. "A Survey on Business Processes Management Suites." *Computer Standards & Interfaces*.

Meidan, Ayman, Julián A García-García, Isabel Ramos, and María José Escalona. 2018. "Measuring Software Process: A Systematic Mapping Study." *ACM Comput. Surv.* 51(3): 58:1--58:32. http://doi.acm.org/10.1145/3186888.

Mendonca, M.G., and V.R. Basili. 2000. "Validation of an Approach for Improving Existing Measurement Frameworks." *IEEE Transactions on Software Engineering* 26(6): 484–99.

Montgomery, Dc. 2009. John Wiley & Sons Inc. *Introduction to Statistical Quality Control*. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Introduction+to+Statistical+Quality+Control#0.

Moody, Daniel. 2009. "The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering." *IEEE Transactions on Software*

Engineering 35(6): 756–79.

Moody, Daniel, and Jos van Hillegersberg. 2009. "Evaluating the Visual Syntax of UML: An Analysis of the Cognitive Effectiveness of the UML Family of Diagrams." In *Software Language Engineering*, eds. Dragan Gašević, Ralf Lämmel, and Eric Van Wyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 16–34.

Mora, Beatriz, Felix Garcia, et al. 2008. "Software Generic Measurement Framework Based on MDA." *IEEE Latin America Transactions* 6(4): 363–70.

Mora, Beatriz, Felix Garcia, Francisco Ruiz, and Mario Piattini. 2009. "Model-Driven Software Measurement Framework: A Case Study." In *Quality Software, 2009. QSIC'09. 9th International Conference on*, , 239–48.

Mora, Beatriz, Mario Piattini, Francisco Ruiz, and Felix Garcia. 2008. "Smml: Software Measurement Modeling Language." In *Proceedings of the 8th Workshop on Domain-Specific Modeling (DSM'2008)*,.

Muketha, G M, A A A Ghani, M H Selamat, and R Atan. 2010. "A Survey of Business Process Complexity Metrics." *Information Technology Journal* 9(7): 1336–44.

Nissen, Mark E. 1998. "Redesigning Reengineering through Measurement-Driven Inference." *MIS Quarterly* 22(4): 509. http://www.jstor.org/stable/249553?origin=crossref.

Nitto, Elisabetta Di et al. 2002. "Deriving Executable Process Descriptions from UML." In *Proceedings of the 24th International Conference on Software Engineering*, , 155–65.

OASIS. 2007. "Web Services Business Process Execution Language." http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

Object Management Group. 2002a. "Meta Object Facility Specification v1." *4, April* 4(April): http//wwwomgorg/docs/formal/02-04-03. http://www.omg.org/spec/MOF/.

———. 2002b. "SPEM 2.0 Software & Systems Process Engineering Metamodel Specification." http://www.omg.org/spec/SPEM/.

———. 2008a. "The MOF Model to Text Transformation Language Specification Version 1.0." https://www.omg.org/spec/MOFM2T.

———. 2008b. "The MOF Query/View/Transformation Specification Version 1.0." https://www.omg.org/spec/QVT/1.0/.

———. 2011. "Business Process Model and Notation (BPMN) Version 2.0." *Business* 50(January): 170. http://books.google.com/books?id=GjmLqXNYFS4C&pgis=1.

———. 2015. "MDA." http://www.omg.org/mda/specs.htm.

OMG. 2009. "Software Metrics Metamodel." http://www.omg.org/spec/SMM/1.0/Beta1/PDF/.

———. 2012. *ISO/IEC 19507:2012.Information Technology - Object Constraint Language (OCL)*. https://www.iso.org/standard/57306.html.

———. 2017. "Unified Modeling Language Specification Version 2.5.1." http://www.omg.org/spec/UML/2.5.1/.

Oracle. 2014. "Oracle® BPEL Process Manager Developer's Guide."

https://docs.oracle.com/cd/B14099_19/integrate.1012/b14448/java.htm.

Ordonez, Mauricio J., and Hisham M. Haddad. 2008. "The State of Metrics in Software Industry." In *Fifth International Conference on Information Technology: New Generations (Itng 2008)*, IEEE, 453–58. http://ieeexplore.ieee.org/document/4492521/.

Organization for the Advancement of Structured Information Standards(OASIS). 2012. "OASIS Metadata Document." http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.html.

Organization for the Advancement of Structured Information Standards (OASIS). 2010. "WS-BPEL Extension for People (BPEL4People)." http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html.

Pandazo, Kosta, Arisa Shollo, Miroslaw Staron, and Wilhelm Meding. 2010. "Presenting Software Metrics Indicators: A Case Study." In *Proceedings of the 20th International Conference on Software Product and Process Measurement (MENSURA)*,.

Park, Robert E, Wolfhart B Goethert, and William A Florac. 1996. *Goal-Driven Software Measurement. A Guidebook.*

Paulish, DJ, and AD Carleton. 1994. "Case Studies of Software-Process-Improvement Measurement." *Computer*. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=312039.

Paulk, M C, C Weber, B Curtis, and M B Chrisses. 1994. "The Capability Maturity Model: Guidelines for Improving the Software Process." *Addisot Wesley*.

Perez-Alvarez, Jose Miguel, Maria Teresa Gomez-Lopez, Luisa Parody, and Rafael M. Gasca. 2016. "Process Instance Query Language to Include Process Performance Indicators in DMN." In *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*, IEEE, 1–8.

Petersen, K, and C Wohlin. 2009. "Context in Industrial Software Engineering Research." In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, , 401–4.

Pohl, Klaus. 2010. *Requirements Engineering: Fundamentals, Principles, and Techniques*. 1st ed. Springer Publishing Company, Incorporated.

Popova, Viara, and Alexei Sharpanskykh. 2010. "Modeling Organizational Performance Indicators." *Information Systems* 35(4): 505–27.

Prakash, Naveen, Sangeeta Srivastava, Sangeeta Sabharwal, and Sangeeta Sabharwal. 2006. "The Classification Framework for Model Transformation." *Journal of Computer Science* 2(2): 166–70. http://www.thescipub.com/abstract/?doi=jcssp.2006.166.170 (October 30, 2018).

Rothenberg, Jeff, Lawrence E Widman, Kenneth A Loparo, and Norman R Nielsen. 1989. 3027 *The Nature of Modeling*. Rand.

Ruiz-gonzález, Francisco, and Gerardo Canfora. 2004. "Software Process : Characteristics , Technology and Environments." *October* V(5): 5–10.

Ruiz, Mercedes, Isabel Ramos, and Miguel Toro. 2002. "A Dynamic Integrated Framework for Software Process Improvement." *Software Quality Journal* 10(2): 181–94.

Russell, N, WMP van der Aalst - Bpm center Report, Department Of, and Undefined 2007. 2007. “Evaluation of the BPEL4People and WS-HumanTask Extensions to WS-BPEL 2.0 Using the Workflow Resource Patterns.” *Bpm center report, Department of Technology Management, Eindhoven University of Technology GPO Box*.

Sánchez González, Laura, Félix García Rubio, Francisco Ruiz González, and Mario Piattini Velthuis. 2010. “Measurement in Business Processes: A Systematic Review.” *Business Process Management Journal* 16(1): 114–34.

Schmidt, Douglas C. 2006. “Model-Driven Engineering.” *COMPUTER-IEEE COMPUTER SOCIETY-* 39(2): 25.

Seemann, Jochen, Von Gudenberg, and Wolff Jürgen. 1998. “UML- Unified Modeling Language.” *Informatik-Spektrum* 21(2): 89–90. http://link.springer.com/10.1007/s002870050092.

Sendall, S., and W. Kozaczynski. 2003. “Model Transformation: The Heart and Soul of Model-Driven Software Development.” *IEEE Software* 20(5): 42–45.

Shahin, Arash, and M Ali Mahbod. 2007. “Prioritization of Key Performance Indicators: An Integration of Analytical Hierarchy Process and Goal Setting.” *International Journal of Productivity and Performance Management* 56(3): 226–40.

Shepperd, Martin. 1995. *Foundations of Software Measurement*. Prentice Hall.

Singh, Y, and M Sood. 2009. “Model Driven Architecture: A Perspective.” *Advance Computing Conference, 2009. IACC 2009. IEEE International* (March): 1644–52.

Sinur, J, and J Hill. 2010. “Magic Quadrant for Business Process Management Suites, Gartner RAS Core Research Note G00205212.”

Smith, Howard, and Peter Fingar. 2003. *Business Process Management : The Third Wave*. Meghan-Kiffer Press. https://dl.acm.org/citation.cfm?id=599722.

van Solingen, Rini et al. 2002. “Goal Question Metric (GQM) Approach.” In *Encyclopedia of Software Engineering*, Hoboken, NJ, USA: John Wiley & Sons, Inc.

Sommerville, Ian. 2004. *Software Engineering (7th Edition)*. Pearson Addison Wesley.

Sparx Systems. 2018. “Full Lifecycle Modeling for Business, Software and Systems.” https://sparxsystems.com/products/ea/.

Staron, Miroslaw et al. 2014. “Dashboards for Continuous Monitoring of Quality for Software Product under Development.” In *Relating System Quality and Software Architecture*, Elsevier, 209–29. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84942241263&partnerID=40&md5=407348be1d5fec35685f99f5202e81e5.

Staron, Miroslaw, and Wilhelm Meding. 2009. “Using Models to Develop Measurement Systems: A Method and Its Industrial Use.” *Software Process and Product Measurement*: 212–26. http://link.springer.com/chapter/10.1007/978-3-642-05415-0_16.

Staron, Miroslaw, Wilhelm Meding, Göran Karlsson, and Christer Nilsson. 2011. “Developing Measurement Systems: An Industrial Case Study.” *Journal of Software Maintenance and Evolution: Research and Practice* 23(2): 89–107. http://doi.wiley.com/10.1002/smr.470.

Staron, Miroslaw, Wilhelm Meding, Kent Niesel, and Alain Abran. 2016. "A Key Performance Indicator Quality Model and Its Industrial Evaluation." In *2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, IEEE, 170–79. http://ieeexplore.ieee.org/document/7809605/.

Tahir, Touseef, Ghulam Rasool, and Cigdem Gencel. 2016. "A Systematic Literature Review on Software Measurement Programs." *Information and Software Technology* 73: 101–21.

Team, CMMI Product. 2002. "Capability Maturity Model{®} Integration (CMMI), Version 1.1--Staged Representation."

Thiry, Laurent, and Bernard Thirion. 2009. "Functional Metamodels for Systems and Software." *Journal of Systems and Software* 82(7): 1125–36.

Tihinen, Maarit et al. 2012. "Metrics and Measurements in Global Software Development." *International Journal on Advances in Software* 5(3): 278–92.

Unterkalmsteiner, M. et al. 2012. "Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review." *IEEE Transactions on Software Engineering* 38(2): 398–424. http://ieeexplore.ieee.org/document/5728832/ (June 16, 2018).

Wang, Xin, Aihua Ren, and Xiangshang Liu. 2017. "Researching on Quantitative Project Management Plan and Implementation Method." In , 20176. http://aip.scitation.org/doi/abs/10.1063/1.4992993 (January 29, 2018).

Wang, Yingxu et al. 2002. "Product and Process Metrics: A Software Engineering Measurement Expert System." In Springer, Berlin, Heidelberg, 337–50. http://link.springer.com/10.1007/3-540-36209-6_29 (February 6, 2018).

Weske, M., W.M.P. van der Aalst, and H.M.W. Verbeek. 2004. "Advances in Business Process Management." *Data & Knowledge Engineering* 50(1): 1–8.

Wieringa, Roel, Neil Maiden, Nancy Mead, and Colette Rolland. 2006. "Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and a Discussion." *Requir. Eng.* 11(1): 102–7. http://dx.doi.org/10.1007/s00766-005-0021-6.

Winchell, William. 1996. *Inspection and Measurement in Manufacturing: Keys to Process Planning and Improvement*. Society of Manufacturing Engineers.

Zeng, Liangzhao et al. 2004. "QoS-Aware Middleware for Web Services Composition." *IEEE Transactions on Software Engineering* 30(5): 311–27.

Zhang, He, Jacky Keung, Barbara Kitchenham, and Ross Jeffery. 2008. "Semi-Quantitative Modeling for Managing Software Development Processes." In *19th Australian Conference on Software Engineering (Aswec 2008)*, IEEE, 66–75.

Zhang, Yuefeng, and Dhaval Sheth. 2006. "Mining Software Repositories for Model-Driven Development." *IEEE Software* 23(1): 82–90.

Zubrow, David. 1998. "Measurement With a Focus: Goal-Driven Software Measurement?" *CrossTalk* 11(9): 24–26.

Appendix A Glossary of Terms

Term	Abbreviation	Description
Attributes	-	A measurable characteristic of an entity.
Business Process Management Suite	BPMS	Software that allows modeling, implementing and executing sets of interrelated activities.
Computer Aided Software Engineering	CASE	Computer tools aim to increase the productivity of software development by reducing the effort, time and cost.
Domain	-	Specific area or subject for which the product is being developed.
Entity	-	An object, event or activity.
Indicator	-	It is an instrument to highlight the information contained in the data, specify the characteristics of the data that must be collected and processed, also determine how these data will be analyzed, interpreted, and when these activities will occur.
Lifecycle	-	Could be defined as a set of phases- each with a specific focus - performed to achieve specific and integrated tasks.
MDE Transformation	-	Refers to the specification of how to generate a model in accordance with a metamodel, taking as input another model according to another metamodel.
Measure	-	A number or symbol assigned to describe some attribute of an entity by the measurement process.
Measurement	-	The process which assigns values in accordance with certain rules to describe properties of entities.
Measurement Definition Metamodel	MDMM	The definition of the syntax and structure that allow describing the elements and relations of the measurement domain.
Measurement Definition Model	MDM	Is a representation of the measurement concepts, relations and constraints, written according to specific measurement language (metamodel).
Measurement lifecycle	-	Refers to the entire phases of the measurement process
Metamodel	-	A metamodel is the definition of the syntax and structure that allow describing elements and relations of a specific domain.
Meta-Object Facility	MOF	Provides an open and platform-independent metadata management framework and associated set of metadata services to enable the development and interoperability of model and metadata driven systems.
Method	-	The procedure that is used to achieve an objective
Methodology	-	A system of methods used in a particular area

		of study or activity
Model	-	Is a representation of a system, written according to a specific language (metamodel).
Model-Driven Engineering	MDE	A paradigm uses models to represent the information of a given domain.
MOF Model to Text Transformation Language	MOFM2T	A model to text transformation language.
Navigational Development Techniques	NDT	Model-driven approach to deal with requirements in Web systems
Object Constraint Language	OCL	A language for the formal description of expressions in UML models. Its main role is to complete the different UML notation artifacts with formally expressed requirements.
Query/View/Transform	QVT	A model to model transformation language.
Role	-	Description of tasks and responsibilities that are related to a specific person
Software development lifecycle	-	Refers to the time period that begins with the decision to develop a software product and covers all the phases of this product (e.g., analyses, design, implementation, testing, delivery, execution, maintenance, etc.)
Software development process	SDP	Refers to the set of resources, tools, activities, methods, and practices used to produce a software artifact.
Software measurement	-	Covers the software engineering activities that deal with defining data that is needed, implementing tools and procedures to collect and analyze these data.
Software Process	-	Software engineering process is a means to support, guide and control the activities of software projects.
Software project	-	Refers to an enactment of a process, involving the management process which controls the project execution, and the development process which produces the project outcomes.
Stakeholder	-	Individual or organization having a right, share, claim or interest in a system or in the characteristics that meet their needs and expectations (e.g., management, sales, marketing, or customer)
UML Profile	-	A UML profile is a formal extension of the UML modeling language with the aim of defining new concepts from existing UML constructors
Unified Modeling Language	UML	A standard graphic modeling language, used mainly to define the metamodels and models presented in this document.

Appendix B Publications

– Journal Papers –

A survey on business processes management suites
Ayman Meidan, Julián A. García-García, María José Escalona, Isabel Ramos
Over the last decade, processes have become an important asset for daily life in organizations because an adequate Business Processes Management (BPM) of an organization (e.g., software development companies) can help achieve organizational objectives. Especially, it is important to efficiently manage these processes vital for the organizational performance in order to continually improve, therefore increasing productivity and competitiveness within the organization (e.g., software processes in software companies). This management is associated with the process lifecycle and, at present, there are many tools (Business Process Management Suites, BPMS) for managing this lifecycle. However, all BPMSs do not provide full support for this lifecycle what makes it more difficult to choose the right BPMS (according to the needs of the organization). This paper presents a survey on BPMS highlighting each phase of the process lifecycle what enables organizations to compare specific BPMS according to their own organizational objectives. This survey has been carried out using a methodology that combines systematic literature review with quality models. This methodology has been used successfully in other contexts. Finally, this paper also describes how this survey has been instantiated on specific open source BPMSs.
Computer Standards & Interfaces , Volume 51, March 2017, Pages 71-86
Readers: 892
Citations:24

Measuring Software Process: A Systematic Mapping Study
Ayman Meidan, Julián A García-García, Isabel Ramos, María José Escalona
Context: Measurement is essential to reach predictable performance and high capability processes. It provides support for better understanding, evaluation, management, and control of the development process and project, as well as the resulting product. It also enables organizations to improve and predict its process's performance, which places organizations in better positions to make appropriate decisions. Objective: This study aims to understand the measurement of the software development process, to identify studies, create a classification scheme based on the identified studies, and then to map such studies into the scheme to answer the research questions. Method: Systematic mapping is the selected research methodology for this study. Results: A total of 462 studies are included and classified into four topics with respect to their focus and into three groups based on the publishing date. Five abstractions and 64 attributes were identified, 25 methods/models and 17 contexts were distinguished. Conclusion: capability and performance were the most measured process attributes, while effort and performance were the most measured project attributes. Goal Question Metric and Capability Maturity Model Integration were the main methods and models used in the studies, whereas agile/lean development and small/medium-size enterprise were the most frequently identified research contexts.
ACM Computing Surveys (CSUR), Volume 51, Issue 3, July 2018
Readers: 291
Citations: 0

– Conference Papers –

A model-based proposal for integrating the measures lifecycle within the process lifecycle
Ayman Meidan, Julián Alberto Garcia-Garcia, I Ramos, María José Escalona, Carlos Arévalo
Software development process (SDP) is a complex and long endeavor, the quality and management of this process affect the quality of its results. Measuring the SDP is essential to gain insight on its performance and to discover improvements. This work proposes to use Model-Driven Engineering (MDE) paradigm to integrate the measures lifecycle within the process lifecycle in order to explicitly and operationally model measures during the process modeling. Also defines transformation rules to derive executable code to run these measures into enterprise tools in order to support the measures lifecycle.
Jornadas de Ingeniería del Software y Bases de Datos (JISBD), 2017
Readers: N/A
Citations:1

A Model-Driven Proposal to Execute and Orchestrate Processes: PLM4BS
Julián Alberto Garcia-Garcia, Ayman Meidan, Antonio Vázquez Carreño, Manuel Mejías Risoto
Business Processes Management (BPM) is a widely consolidated business strategy to improve and optimize the internal operation of any company. However, BPM is not usually simple to apply in software organizations because Software Processes (SPs) involve high degree of creativity, abstraction and rework, among other aspects. This situation provokes that these companies usually focus on modeling their processes but later, the orchestration and execution are manually and/or unilaterally performed by each involved role. This situation makes each SP difficult to maintain, monitor, evolve and measure. At present, there are model-based proposals to model SPs, but most of them fail to define the execution context of the process. This paper presents PLM4BS, a model-driven framework to support modeling, execution and orchestration of SPs. It has been successfully validated in different real environments, what has returned us valuable feedback to improve PLM4BS in the near future.
International Conference on Software Process Improvement and Capability Determination, 2017, volume 770, p 211-225
Downloads: 780
Citations:2