SOFTWARE MEASUREMENT BY USING QVT TRANSFORMATIONS IN AN MDA CONTEXT

Beatriz Mora, Félix García, Francisco Ruiz, Mario Piattini

Department of Information Technologies & Systems, University of Castilla-La Mancha, Ciudad Real, Spain

Artur Boronat, Abel Gómez, José Á. Carsí, Isidro Ramos

Department of Information Systems and Computation Polytechnic University of Valencia, Valencia, Spain

Keywords: Measurement, Model, FMESP, MDA, QVT, SMF.

Abstract: At present the objective of obtaining quality software products has led to the necessity of carrying out good software processes management in which measurement is a fundamental factor. Due to the great diversity of entities involved in software measurement, a consistent framework is necessary to integrate the different entities in the measurement process. In this work a Software Measurement Framework (SMF) is presented to measure any type of software entity. In this framework, any software entity in any domain could be measured with a common Software Measurement metamodel and QVT transformations. This work explains the three fundamental elements of the Software Measurement Framework (conceptual architecture, technological aspects and method). These elements have all been adapted to the MDE paradigm and to MDA technology, taking advantage of their benefits within the field of software measurement. Furthermore an example which illustrates the framework's application to a concrete domain is furthermore shown.

1 INTRODUCTION

The current necessity of the software industry to improve its competitiveness forces continuous process improvement. This must be obtained through successful process management (Florac, et al., 2000). Measurement is an important factor in the process life cycle due to the fact that it controls issues and lacks during software maintenance and development. In fact, measurement has become a fundamental aspect of Software Engineering (Fenton, et al., 1997).

Software Processes constitute the work base in a software organization. Companies therefore wish to carry out an effective and consistent software measurement to facilitate and promote continuous process improvement. To do this, a discipline for data analysis and measurement (Brown, et al., 2004), and measure definition, compilation and analysis in the process, projects and software products, is needed.

The great diversity in the kinds of entities which are candidates for measurement in the context of the software processes points to the importance of providing the means through which to define measurement models in companies in an integrated and consistent way. This involves providing companies with a suitable and consistent reference for the definition of their software measurement models along with the necessary technological support to integrate the measurement of the different kinds of entities.

With the objective of satisfying the exposed necessities, it is highly interesting to consider the MDE (Model-Driven Engineering) paradigm (Bézivin, et al., 2005) in which software measurement models (SMM) are the principal elements of the measurement process. Its main goal is to ensure that the core artifacts in software engineering processes will be models rather than code, so that designs are expressed and managed in the manner of models with a much higher level of abstraction than the code. MDA (Model-Driven Architecture) is the OMG proposal by which to carry out the MDE Paradigm. The core of MDA is a set of standards (MOF, QVT, OCL and XMI).

According to the QVT standard, the software development process is a set of model

transformations, from an abstract to a specific level. The requirements are in the more abstract level and the code is in the more specific level.

Software measurement can benefit from the MDE paradigm, providing integration and support to carry out an automatic software measurement of any software type. This implies that: a) the definition of measurement models conform to a Software Measurement metamodel; b) the definition of generic measurement methods are applicable to any model-based software artifact; and c) support for computing measures, for storing results and for enhancing decision making.

These aspects constitute the main interest of this paper, in which the application of MDA principles, standards and tools are used in software measurement. The goal of this proposal is to develop a generic framework to define measurement models which conform to a common measurement metamodel, and to measure any software entity with regard to a domain metamodel. In order to develop this proposal, MOMENT environment has been used, which supports the automatic model management MDA compliant.

Publications of García, Bertoa, et al. (2006); García, Piattini, et al. (2006); and García, et al. (2007) were used as a starting point for this work. These works present FMESP, which consists of a framework based on MOF Architecture. This includes a software measurement ontology and metamodel, and the GenMETRIC tool which is used to define software measurement models, and to calculate defined measures for these models. The ontology permits the identification of all the concepts, proportions exact definitions for all the terms and clarifies the relationship between them. This paper presents an adaptation of FMESP to MDA, which is described in detail in following sections.

The remainder of the paper is organized as follows. Section 2 provides an overview of related works and Section 3 describes the Software Measurement Framework (SMF), including conceptual architecture, technological aspects, and method. In Section 4 the use of the framework is illustrated with an example. Finally, conclusions and future works are outlined in Section 5.

2 RELATED WORKS

We have found numerous publications which deal with tools that have important success factors in software measurement efforts (Komi-Sirviö, et al., 2001), which supply work environments and general approximations (Kempkens, et al., 2000), or which give architectures more specific solutions (Jokikyyny, et al., 1999). Brown, et al. (2004) include a list of tools which support the creation, control and analysis of software measurements. Auer, et al. (2003) furthermore examine various software measurement tools, such as MetricFlame, MetricCenter, Estimate Professional, CostXPert and ProjectConsole, in heterogenic environments.

It is also possible to find certain proposals through which to tackle software measurement which are more integrated and less specific than in the aforementioned cases. Palza, et al. (2003) propose the MMR tool which is based on the CMMI model for the evolution of software processes, and it is possible to consult similar tools of Harrison, (2004); Lavazza, et al. (2005); and Scotto, et al. (2004). These proposals are, however, restricted to concrete domains or to evaluation models of specific quality.

Vépa, et al. (2006) present a metamodel which allows the storage of measurement data, and a set of transformations through which to carry out the measurement of models based on a metamodel is presented. This paper focuses upon the technological aspects needed to implement the software measurement with ATL technology, by offering the user a variety of graphic representations of the measurement results obtained.

This final proposal and that which is presented here are complementary as they both focus upon two key support elements of generic measurement: the conceptual base, which is the main contribution of FMESP, and technological implementation. Some differences from technological point of view exist.

The measurements which are applied in the work of Vépa, et al. (2006) are previously defined in the ATL transformation archives. The measurable entities are typical of the metamodels presented in this work (KM3 and UML2). For example, the measurable entities for a model which is expressed in km3 might be package, class, attribute, reference etc.

The measurements in the proposal presented here are defined by the user, i.e. the model transformation needed to carry out the measurement it is not a model previously defined, but this model is defined according to the users needs. The measurement definition is possible thanks to the software measurement model, which contains all that is relative to the measurement to be carried out in each case. Moreover, the measurable entities are those which are defined in their corresponding domain and measurement metamodel (expressed in ecore). A further difference is that SMF uses QVT.

3 SOFTWARE MEASUREMENT FRAMEWORK

In order to carry out this proposal it was considered of interest to adapt FMESP to the MDE paradigm. The objective of this was to exploit the benefits that the paradigm could contribute to software measurement by, on one hand adopting the software measurement metamodel defined in FMESP, and on the other by evolving GenMETRIC to an environment which would allow the definition of software measurement models and the computation of the models defined. All this would take place within the context of models and model transformations of the MDA architecture. The Software Measurement Framework (SMF) is the evolution of the FMESP, but is adapted to the MDE paradigm and uses MDA technology.

The following subsections explain the conceptual, technological and methodological elements which are part of SMF.

3.1 Conceptual Architecture

Due to the necessity of having a generic and homogeneous environment for software measurement (García, Bertoa, et al., 2006; García, Piattini, et al., 2006; García, et al., 2007), a conceptual architecture and a tool with which to integrate the software measurement are proposed. In the following section, the main characteristics of this proposal are described. In García, et al. (2007) a more detailed description can be found.
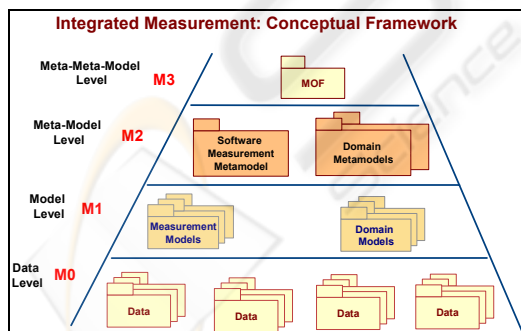

Figure 1: Conceptual framework with which to manage software measurement.

The proposed software measurement described in this paper is part of the FMESP framework (García, Piattini, et al., 2006). The FMESP framework permits representing and managing software processes from the perspectives of modeling and measurement. We focus on the measurement support of the framework whose elements are detailed according to the three layers of abstraction of metadata that they belong to, according to the MOF standard. In Figure 1, the conceptual architecture for integrated measurement is represented.

As can be observed in Figure 1, the architecture has been organized into the following conceptual levels of metadata:

- Meta-MetaModel Level (M3). At this level, an abstract language for the definition of metamodels, is found. This is the MOF language.
- Metamodel Level (M2). In the M2 level, two generic metamodels which conform with this framework are required. These are: the Measurement Metamodel, to define specific measurement models; and Domain Metamodels, to represent the kinds of entities which are candidates for measurement in the context of the evaluation of the software processes, such as, UML and Process metamodels.
- Model Level (M1). Specific models are included at this level. These models may be of two types: Measurement Models, which are examples of the measurement metamodel in the M2 level and which are defined in such a way as to satisfy some of the company's information needs; and Domain Models, which are defined according to their corresponding domain metamodels.

In order to establish and clarify the concepts and relationships that are involved in the software measurement domain before designing the metamodel, an ontology for software measurement was developed (García, Bertoa, et al., 2006). The measurement metamodel was derived by using the concepts and relationships stated in the ontology as a base. The Software Measurement metamodel (which is integrated in SMF) is organized around four main packages (for greater detail see the work of García, Bertoa, et al. (2006)):

- **Software Measurement Characterization and Objectives**, which includes the constructors required to establish the scope and objectives of the software measurement process.
- **Software Measures**, which aim at establishing and clarifying the key elements in the definition of a software measure.
- **Measurement Approaches**. This package introduces the element of measurement approach to generalize the different

approaches used by the three kinds of measures to obtain their respective measurement results. A base measure applies a measurement method. A derived measure uses a measurement function. Finally, an indicator uses an analysis model to obtain a measurement result that satisfies an information need.

- **Measurement Action**. This establishes the constructs related to the act of measuring software. A measurement (which is an action) is a set of measurement results, for a given attribute of an entity, using a measurement approach. Measurement results are obtained as the result of performing measurements (actions).

3.2 Technological Aspects

In this section the technological aspects of SMF are explained.

3.2.1 Adaptation to MDA

In Figure 2 the necessary elements for the FMESP adaptation to MDA are presented according to MOF levels.
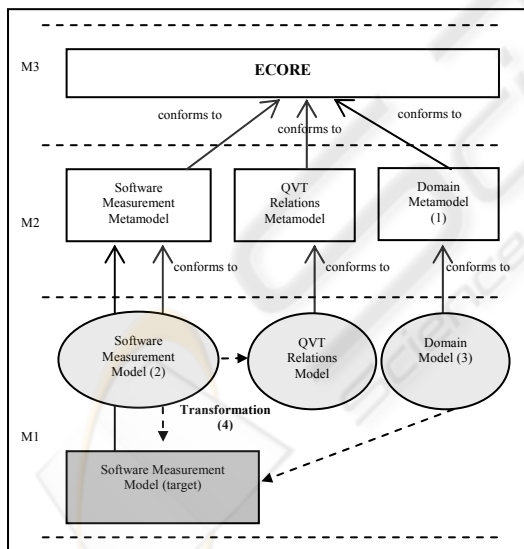

Figure 2: Elements of the FMESP adaptation in a MDA context.

As can be observed in Figure 2, two new elements, namely the QVT Relations model and metamodel, have been added to adapt the conceptual architecture illustrated Figure 1 to MDA. The QVT Relations Model (which is described in greater detail

in Section 3.2.2) is obtained automatically through a transformation from a Measurement model. It contains all the information necessary to carry out the transformation of the SMF proposal. Ecore language has been selected because it is a common modeling language based on EMOF. EMOF is the part of the MOF 2.0 specification that is used for defining simple metamodels using UML-like concepts.

3.2.2 QVT Relations Transformation

The QVT Relations model is the transformation needed to perform the measurement. In this transformation two source models are involved: a Software Measurement model and a domain model; the target model is the Software Measurement Model with the measurement results (see Figure 2). Due to the fact that the proposal is about generic measurement, it is very important that the QVT model is obtained in a generic way. The MDE paradigm and MDA technology are applied for this reason.

This transformation is obtained automatically from the previous QVT transformation shown in Figure 3. The QVT Relations model, called the extended or final QVT Relations model, is obtained from a QVT transformation, where there are two source models: the basic or initial QVT Relations model (which conforms to the QVT Relations metamodel) and the Software Measurement model (previously defined).
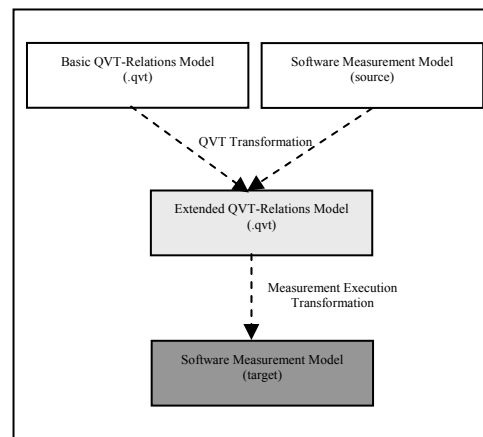

Figure 3: QVT Relations transformation model.

The extended QVT Relations model extends the basic QVT Relations model with the following aspects:

- **Transformation Model:** to obtain the extended QVT Relations model, the source

model specification is needed. In this case, there are two source models: the Software Measurement model and the domain model. Due to the fact that the Software Measurement model is always the same, this model is already defined in the basic QVT Relations model. Therefore, only the domain model needs to be defined. This information is taken from the Software Measurement model which contains all the measurement information.

- **Relation Domain:** in order to perform the transformation, it is necessary to define the *checkonly domain* rules. In this case there are two, one for each source model: the domain model and the Software Measurement model. It is only necessary to define *checkonly domain* of the domain model, because checkonly domain of the measurement model is already defined in the basic QVT Relations model.

- **Function:** this contains the necessary OCL queries to carry out the measurement. These OCL queries are the implementations of the "*Measurement Action*" package defined in the Software Measurement Metamodel.

These elements are empty in the basic QVT Relations model, and they are extended to obtain the extended QVT Relations model, the transformation model necessary to carry out the measurement. In the Figure 4 all the Software Measurement process is shown.
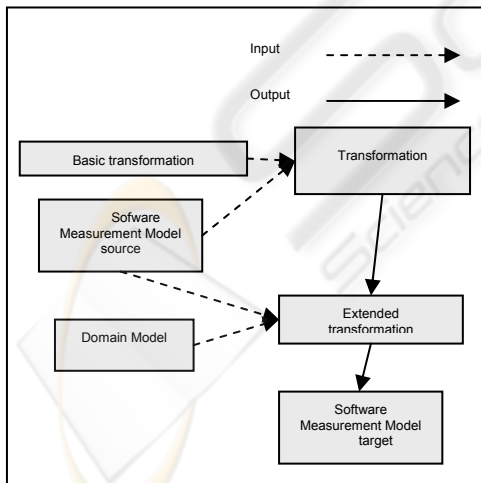

Figure 4. Software Measurement process.

3.2.3 Technological Environment

In this paper, the tool selected has been the model management environment called MOMENT

(MOment manageMENT)(Boronat, et al., 2007). This framework is integrated in the Eclipse platform. It provides a set of generic operators to deal with models through the Eclipse Modeling Framework (EMF)("Eclipse Modelling Framework (EMF) Main Page," 2007). The underlying formalism of the model management approach is the algebraic language Maude ("The Maude System," 2007).

From a functional point of view, MOMENT has two components: OCL query execution (MOMENT-OCL) and QVT Transformations (MOMENT-QVT).

MOMENT-OCL (Boronat, et al., 2006) has implemented an editor integrated in the Eclipse platform to check OCL invariants and to execute OCL queries over instances of ecore models. It uses Ecore models in the entire software development process to store the OCL expressions by following the Model Driven Engineering approach. One advantage of this is the persistence mechanisms in XMI that EMF provides automatically. In this work, it has been used in order to check and validate the OCL queries used in the QVT transformations. The results have been shown by screen.

On the other hand, the MOMENT-QVT tool (Queralt, et al., 2006) is a model transformation engine that provides partial support for the QVT Relations language. It implements the metamodel definition QVT, given in the QVT standard, and provides an editor for the QVT Relations language, which permits the definition of model transformations between EMF metamodels.

In order to carry out a QVT transformation in MOMENT, a transformation textual specification (coded by the Textual QVT Editor and stored in a .qvtext) or, its equivalent QVT Relations model (stored in a .qvt) can be used. This model conforms to the QVT Relations metamodel and it is possible to obtain it by parsing the textual specification.

3.3 Method

The necessary steps to carry out the software measurement by using the SMF are explained below (see Figure 2):

1. **Incorporation of Domain Metamodel:** the measurement is made in a specific domain. This domain must be defined according to its metamodel (it is situated in the M2 level and it conforms to the Ecore meta-metamodel).

2. **Creation of Measurement Model:** the measurement model is created according to the Software Measurement metamodel which is integrated in SMF. This first model is the source model, so the results are therefore still not

defined, i.e. the "*Measurement Action*" package from the Software Measurement metamodel is still not instantiated.

3. Creation of Domain Model: which is defined according to its corresponding domain metamodel (created in the first step). The domain models are the entities whose attributes are measured by calculating the measurements defined in the corresponding measurement models. Examples of domain models are: the UML models (use cases, class diagrams, etc.), or the E/R models.

4. Measurement Execution: the measurement execution is carried out through QVT transformation, in which, the measurement model is obtained by starting from the two source models (the measurement model and the domain model) where the results are defined, i.e. the "*Measurement Action*" package is instantiated. The target measurement model is the extension of the source measurement model. The measurement results are calculated by running OCL queries on the domain model.

An example of the method application is shown in the following section.

4 EXAMPLE

To illustrate the benefits of the proposal, consider the example of relational database measurement. For greater simplicity, only the following elements are shown in Figure 5: Measurement *Method, Entity* (to which the measurement method is applied) and *Measurement result* (the result is obtained by executing the measurement method on the entity).
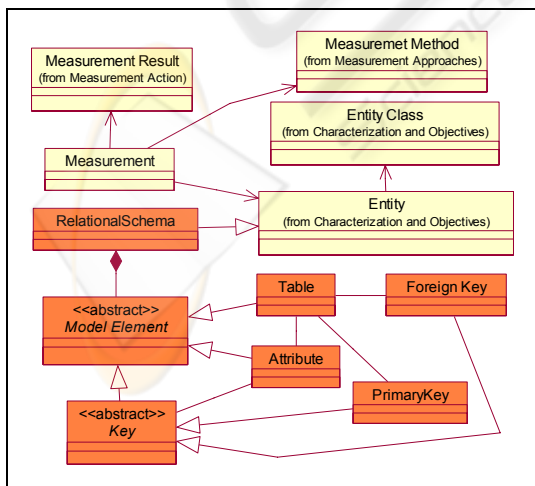

Figure 5: Relationship between Relational Database (domain) Metamodel and SMM.

Furthermore, it is necessary for the domain metamodel, in this case Relational Databases domain, to have been previously chosen. Both metamodels are independent (Figure 5), although they are logically related. In Figure 5 the measurement and domain metamodels have been represented in a clear and a dark colour, respectively.

In this example, the chosen measurement method has been "COUNT elements of type TABLE", which is an instantiation of the abstract method "COUNT elements of type X".

In order to carry out the measurement, the following steps (four steps) must take place:

1. Incorporation of Relational Databases metamodel (represented in a dark colour in Figure 6).
2. Creation of measurement model conforms to Software Measurement metamodel. For the measurement method "COUNT elements of type TABLE", the values of *Entity* and *Measurement Method* are Table and Count, respectively. The *Measurement Result* is not still defined.
3. Creation of model conforms to the Relation Database metamodel. In this case, the model (relational schema) is a university domain composed of five tables with their corresponding primary keys (bold and shaded), foreign keys (underlined and italic), and attributes (see Figure 6).
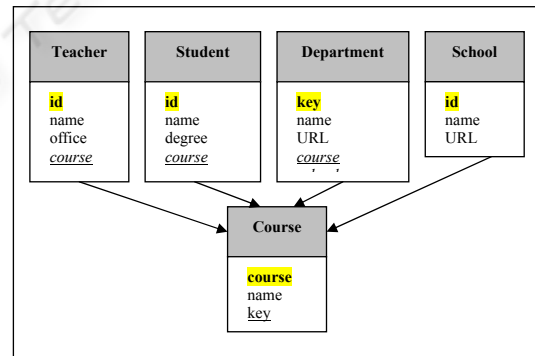

Figure 6: Relational Database model (relational schema).

The extended QVT Relations model was needed to carry out the fourth step. This transformation is obtained automatically (see section 3.2.2). The extended elements are detailed below:

- Transformation Model: the target model is the relational databases domain model.
- Relation Domain: the checkonly domain of the relational schema domain is indicated (see Figure 7).

```
checkonly domain relationalDomain
 srcRelationalSchema : RelationalSchema {
    name = myRelationalSchema
};
checkonly domain measurementDomainSrc
 srcMeasurementModel : MeasurementModel {

    modelName = myModelName,

    measurements = dstMeasurement1 : Measurement {
        name = myMeasurementName,
        method = dstMethod : MeasurementMethod{
            nameMethod = myMethod
        },
        entity = dstEntity : Entity{
            nameEntity = myEntity
        },
        result = dstResult : MeasurementResult{}
    } //result not defined
};
```

Figure 7: "Relation Domain" elements from extended QVT Relations model.

- Function: this contains the OCL queries with which to perform the measurement, in this case, the queries necessary to implement the "count element of type X" measurement method where X is Table (see Figure 8).

```
    enforce domain measurementDomainDst
     dstMeasurementModel : MeasurementModel {
        modelName = myModelName,

        measurements = dstMeasurement1 : Measurement {
            name = myMeasurementName,
            method = dstMethod : MeasurementMethod{
                nameMethod = myMethod
            },
            entity = dstEntity : Entity{
                nameEntity = myEntity
            },
            result = measurementAction(srcRelationalSchema,
             myMethod, myEntity)
        }
    };
}//TOP RELATION
function measurementAction(relationalSchema: RelationalSchema,
    method: String, entity: String) : Integer
{
    relationalSchema.modelElements->select(m:ModelElement
    |m.oclIsTypeOf(Table))->size()
}
```

Figure 8: "Function" elements from extended QVT Relations model.

4. The source models used to carry out the measurement are: the measurement model (2nd step), the domain model (3rd step) and the extended QVT Relations model. The target model obtained is the measurement model with defined *Measurement Result* (see Figure 9). In this example the value of *Measurement Result* is 5 (number of tables).

```
<?xml version="1.0" encoding="ASCII"?>
<measurement:MeasurementModel xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:measurement="http://bmora/metamodels/measurement"
    modelName="ER_MEASUREMENT">
  <measurements name="RELATIONAL SCHEMA MEASUREMENT">
    <method nameMethod="COUNT"/>
    <entity nameEntity="TABLE"/>
    <result result="5"/>
  </measurements>
</measurement:MeasurementModel>
```

Figure 9: Measurement result.

In the same way as is illustrated with Relational Databases, the method can be applied to any other domains, such as for example, UML models, Project Management or Business Processes, etc.

5 CONCLUSIONS AND FUTURE WORK

In this paper, a generic framework for the definition of measurement models based on a common metamodel has been presented. The framework allows the integrated management and measurement of a great diversity of entities.

Following the MDA approach and starting from a (universal) measurement metamodel, it is possible to carry out the measurement of any domain by means of QVT transformation, and this process is completely transparent to the user.

With SMF, it is possible to measure any software entity. The user task consists in selecting the domain metamodel (the domain to be measured) and defining the source models. The software metamodel is integrated in the framework.

At the present time, a Software Measurement Modeling Language (SMML) is being developed to supply measurement engineers with the definition of software measurement models according to the proposed metamodel; this language will be integrated in SMF.

Among related future works, one important work is the realization of a plug-in based on Eclipse which will supply the user with the data introduction and the measurement process. This plug-in will enable users to instantiate measurement models in an easy and intuitive way. Other future work will be to align our metamodel with the Software Metrics Meta-Model (SMM) OMG proposal (OMG, 2007), which is at present in its development phase. Finally, we shall apply SMF to real environments to obtain further refinements and validation.

ACKNOWLEDGEMENTS

REFERENCES

Auer, M., Graser, B., & Biffl, S. (2003). *A Survey on the Fitness of Commercial Software Metric Tools for Service in Heterogeneous Environments: Common Pitfalls* Paper presented at the Ninth International Software Metrics Symposium. (Metrics '03).

Bézivin, J., Jouault, F., & Touzet, D. (2005). *Principles, standards and tools for model engineering.* Paper presented at the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'2005).

Boronat, A., & Meseguer, J. (2007). *Algebraic Semantics of EMOF/OCL Metamodels* (No. Technical Report UIUCDCS-R-2007-2904): CS Dept., University of Illinois at Urbana-Champaign.

Boronat, A., Ramos, I., & Carsí, J. Á. (2006, Marzo 2003). *Definition of OCL 2.0 Operational Semantics by means of a Parameterized Algebraic Specification.* Paper presented at the WAFOCA'06. First International Workshop on algebraic foundations for OCL and Applications.

Brown, M., & Dennis, G. (2004). Measurement and Analysis: What Can and Does Go Wrong? *10th IEEE International Symposium on Software Metrics (METRICS'04)*, 131-138.

. Eclipse Modelling Framework (EMF) Main Page (2007), from http://www.eclipse.org/emf

Fenton, N., & Pfleeger, S. L. (1997). *Software Metrics: A Rigorous & Practical Approach, Second Edition*: PWS Publishing Company.

Florac, W. A., Carleton, A. D., & Barnard, J. (2000). Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process. *IEEE Software, 17 (4)*(4).

García, F., Bertoa, M. F., Calero, C., Vallecillo, A., Ruíz, F., Piattini, M., et al. (2006). Towards a consistent terminology for software measurement. *Information and Software Technology 48*(8), 631-644

García, F., Piattini, M., Ruiz, F., Canfora, G., & Visaggio, C. A. (2006). FMESP: Framework for the modeling and evaluation of software processes. *Journal of Systems Architecture - Agile Methodologies for Software Production 52*(11), 627-639

García, F., Serrano, M., Cruz-Lemus, J., Ruiz, F., & Piattini, M. (2007). Managing Software Process Measurement: A Metamodel-Based Approach. *Information Sciences*.

Harrison, W. (2004). A flexible method for maintaining software metrics data: a universal metrics repository. *Journal of Systems and Software 72*(2), 225-234

Jokikyyny, T., & Lassenius, C. (1999). *Using the internet to comunicate software metrics in a large organization.* Paper presented at the Proceedings of GlobeCom'99.

Kempkens, R., Rösch, P., Scott, L., & Zettel, J. (2000). *Instrumenting Measurement Programs with Tools.* Paper presented at the PROFES 2000, Oulu, Finland.

Komi-Sirviö, S., Parviainen, P., & Ronkainen, J. (2001). *Measurement Automation: Methodological Background and Practical Solutions-A Multiple Case Study.* Paper presented at the Seventh International Software Metrics Symposium (METRICS'01), London.

Lavazza, L., & Agostini, A. (2005). Automated Measurement of UML Models: an open toolset approach. *Object Technology, 4(4)*, 115-134.

. The Maude System (2007), from http://maude.cs.uiuc.edu/

OMG (2007). *Architecture-Driven Modernization (ADM): Software Metrics Meta-Model (SMM). OMG Document: dmtf/2007-03-02*: Object Management Group.

Palza, E., Fuhrman, C., & Abran, A. (2003). *Establishing a Generic and Multidimensional Measurement Repository in CMMI context* Paper presented at the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), Greenbelt (Maryland, USA).

Queralt, P., Hoyos, L., Boronat, A., Carsí, J. Á., & Ramos, I. (2006, October 2006). *Un motor de transformación de modelos con soporte para el lenguaje QVT relations.* Paper presented at the Desarrollo de Software Dirigido por Modelos - DSDM'06 (Junto a JISBD'06), Sitges, Spain.

Scotto, M., Sillitti, A., Succi, G., & Vernazza, T. (2004). *A relational approach to software metrics.* Paper presented at the Proceedings of the 2004 ACM symposium on Applied computing (SAC'2004), Nicosia, Cyprus.

Vépa, É., Bézivin, J., Brunelière, H., & Jouault, F. (2006). *Measuring Model Repositories.* Paper presented at the Model Size Metrics Workshop at the MoDELS/UML 2006 conference, Genoava, Italy.