

SUPPORTING SOFTWARE PROCESS MEASUREMENT BY USING METAMODELS

A DSL and a Framework

Beatriz Mora, Felix Garcia, Francisco Ruiz and Mario Piattini

*Alarcos Research Group, Department of Computer Science, University of Castilla-La Mancha
Paseo de la Universidad 4, Ciudad Real, Spain*

Keywords: Measurement, MDA, SMML.

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. Besides, we present a Software Measurement Modelling Language (SMML) in order to define the measurement models with take part in the measurement process. 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, Carleton 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 and Pfleeger, 1997). Software Processes constitute the work base in a software organization. Companies therefore wish to carry out an effective and consistent software measurement process to facilitate and promote continuous process improvement. To do this, a discipline for data analysis and measurement (Brown and Dennis, 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, Jouault 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. 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.

The availability of a language which allows us to represent those elements which must be taken into account in the measurement processes might, therefore, be important in decision making and in process improvement. It is thus of interest to consider the use of Domain Specific Languages (DSLs) such as the Software Modelling Measurement Language (SMML)(Mora, Ruiz et al., 2008).

These aspects constitute the main interest of this paper: in which the application of MDA principles, standards and tools are used in software measurement. We present the Software Measurement Framework (SMF), 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. MOMENT environment has been used, which supports the automatic model management MDA compliant. These measurement models involved in the framework can be defined by using SMML. SMML is integrated in SMF and permits software measurement models to be created in a simple and intuitive manner. This language has been done by using the Software Measurement Metamodel (SMM)(García, Serrano et al., 2007), (Mora, Ruiz et al., 2008) as the Domain Definition Metamodel (DDMM). The task of the SMML is to facilitate the definition of software measurement models, which is the starting point in the generic software measurement process.

The remainder of the paper is organized as follows. Section 2 provides an overview of related works. Section 4 presents the SMML and Section 3 describes the Software Measurement Framework (SMF), including conceptual architecture, technological aspects, and method. In Section 5 the use of the framework and SMML is illustrated with an example. Finally, conclusions and future works are outlined in Section 6.

2 RELATED WORKS

We have found numerous publications which deal with tools that have important success factors in software measurement efforts (Komi-Sirviö, Parviainen et al., 2001), which supply work

environments and general approximations (Kempkens, Rösch et al., 2000), or which give architectures more specific solutions (Jokikyyny and Lassenius, 1999), (Brown and Dennis, 2004) includes a list of tools which support the creation, control and analysis of software measurements. (Auer, Graser et al., 2003) furthermore examines 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, Fuhrman et al., 2003) proposes the MMR tool which is based on the CMMI model for the evolution of software processes, and it is possible to consult similar tools in (Harrison 2004), (Scotto, Sillitti et al., 2004), (Lavazza and Agostini, 2005). These proposals are, however, restricted to concrete domains or to evaluation models of specific quality.


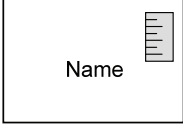

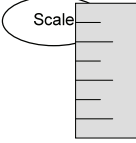
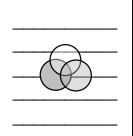

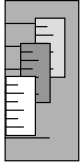
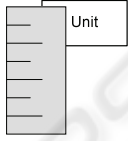
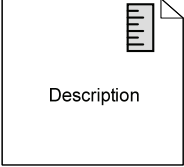
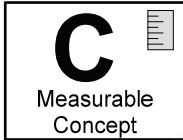
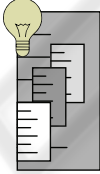
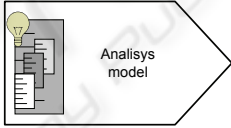
(Vépa, Bézivin et al., 2006) presents a metamodel which allows the storage of measurement data, and a set of transformations through which to carry out the measurement of models based on a metamodel is presented. This paper focuses upon the technological aspects needed to implement the software measurement 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, Bézivin et al., 2006) are previously defined in the ATL transformation archives. The measurable entities are typical of the metamodels presented in this work (KM3 (Jouault and Bézivin, 2006) 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

Table 1: A selection of the SMML elements and icons.

Information need	Entity	Base Measure	Scale
 Information need	 Name		
Quality Model	Attribute	Derived Measure	Unit
 Quality Model	 Attribute		
Description	Measurable Concept	Indicator	Analysis model
 Description	 Measurable Concept		 Analysis model

which are defined in their corresponding domain and measurement metamodel (expressed in ecore). A further difference is that SMF uses QVT.

Finally, it is important to mention a method for specifying models of software data sets in order to capture the definitions and relationships among software measures presented in (Kitchenham, Hughes et al., 2001).

3 SOFTWARE MEASUREMENT MODELLING LANGUAGE (SMML)

SMML (Mora, Ruiz et al., 2008) is a language which permits software measurement models to be built in a simple and intuitive manner. The Software Measurement Metamodel (SMM) which is derived from the Software Measurement Ontology (SMO)(García, Bertoa et al., 2006) defines the abstract syntax of SMML. The Software Measurement Metamodel supports the graphical language to represent in an intuitive way software measurement models.

With regard to expected requirements (Kolovos, Paige et al., 2006), we shall now show the requirements which are valid in our Language: a)

Conform: the language constructs correspond to important domain concepts; b) Orthogonal: Each language construct is used to represent exactly one distinct concept (*Attribute*, *Base Measure*, etc.) in the domain; c) Supportable: The SMML language is supported by tools such as MS/DSL Tools or GMF (Eclipse, 2007); d) Simple: the DSL is simple in order to express the domain concepts and to support its users; e) Usable: DSL constructs are expressive and easy to understand. Table 1 shows some of the most representative graphical elements of the SMML (for a greater detail see (Mora, Ruiz et al., 2008)).

4 SOFTWARE MEASUREMENT FRAMEWORK

The Software Measurement Framework (SMF) (for greater detail see (Mora, García et al., 2008)) permits us to measure any type of software entity. In this framework, any kind of software entity represented by its corresponding metamodel in any domain can be measured with a common Software Measurement metamodel and QVT transformations. SMF has three fundamental elements: 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. The Software Measurement Framework (SMF) is the evolution of the FMESP (García, Piattini et al., 2006), but is adapted to the MDE paradigm and uses MDA technology.

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

4.1 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, Serrano 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, Serrano et al. 2007) a more detailed description can be found.

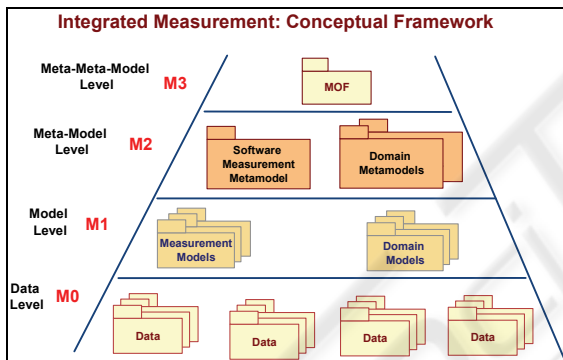


Figure 1: Conceptual framework with which to manage software measurement.

SMF is part of the FMESP framework (García, Piattini et al., 2006). The FMESP framework permits representing and managing software processes from the perspectives of modelling 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. As can be observed in Figure 1, the architecture has been organized into the following conceptual levels of metadata: Meta-Model Level (M3), Metamodel Level (M2) and Model Level (M1).

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, Software Measures, Measurement Approaches and Measurement Action.

4.2 Technological Aspects

In this section the technological aspects of SMF are explained.

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

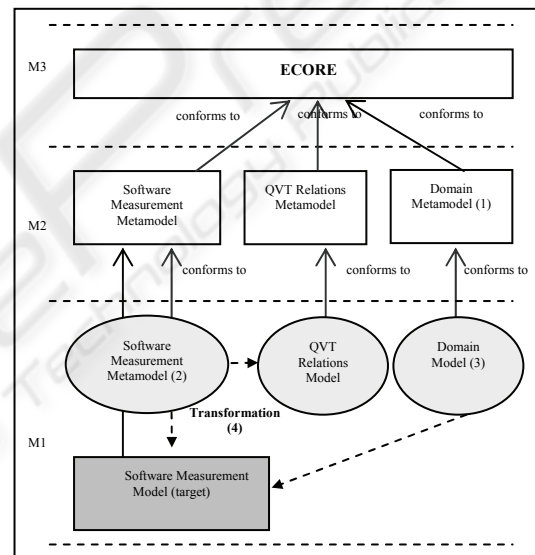


Figure 2: Elements of the FMESP adaptation in a MDA context.

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 in Figure 1 to MDA. The QVT Relations Model is obtained automatically through a transformation from a Measurement model. It contains all the information necessary to carry out the QVT transformation of the SMF proposal. Ecore language has been selected because it is an implementation of EMOF. EMOF is the part of the MOF 2.0 specification that is used for defining simple metamodels using simple concepts.

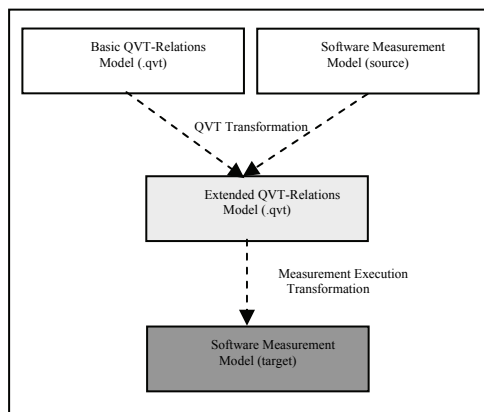


Figure 3: QVT Relations transformation model.

- 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 Fig. 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 Fig. 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). (for a greater detail see (Mora, García et al., 2008)).
- Technological Environment.** In this work, the tool selected has been the model management environment called MOMENT (MOment manageMENT)(Boronat and Meseguer, 2007). This framework is integrated in the Eclipse platform. It provides a set of generic operators to deal with models through the Eclipse Modelling Framework (EMF)(2007). The underlying formalism of the model management approach is the algebraic language Maude.

4.3 Method

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

- Incorporation of Domain Metamodel:** the measurement is made in a specific domain. This domain must be defined according to its metamodel.
- 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. In order to facilitate the measurement model definition, SMML can be used.
- 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.
- 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.

5 CASE STUDY

To illustrate the benefits of the proposal, consider the example of relational database measurement. For greater simplicity, only the following elements are shown in Fig. 4: *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).

Furthermore, it is necessary for the domain metamodel, in this case Relational Databases domain, to have been previously chosen. Both metamodels are independent (see fig 4), although they are logically related. In Fig. 4 the measurement and domain metamodels have been represented in a clear and a dark colour, respectively.

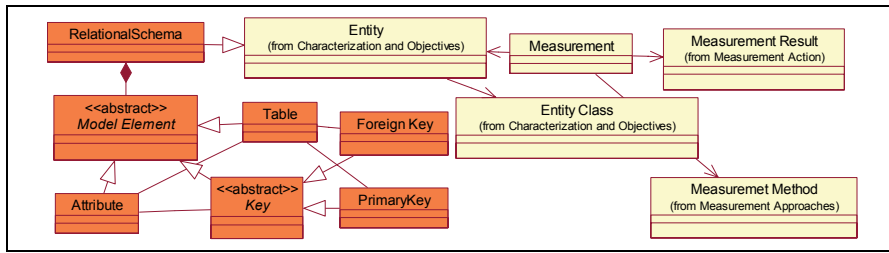


Figure 4: Relationship between Relational Database (domain) Metamodel and SMM.

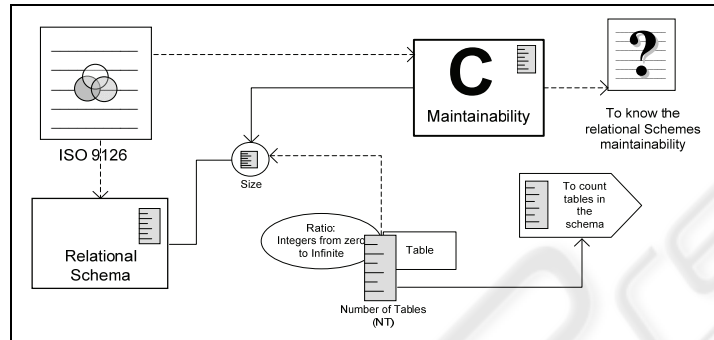


Figure 5: Measurement Model with SMML.

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 Fig. 4).
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. In this case SMML can be used to define the measurement model (see Fig. 5).
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, foreign keys, and attributes. The extended QVT Relations model was needed to carry out the fourth step. This transformation is obtained automatically (see Fig. 6).
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* (Fig. 7). In this example the

value of *Measurement Result* is 5 (number of tables).

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.

```

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.ocIsTypeOf(Table))->size()
}
    
```

Figure 6: “Function” elements from extended QVT Relations model.

```

<?xml version="1.0" encoding="ASCII"?>
<measurement:MeasurementModel xmlns:measurement="http://www.omg.org/XMI"
  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 7: Measurement result.

6 CONCLUSIONS AND FUTURE WORK

In this paper, a generic framework for the definition of measurement models based on a common metamodel has been outline, and we have explained how to work with it. 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 (QVT transformation) 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.

In order to facilitate the measurement models definition, a Software Measurement Modelling Language (SMML) is used to supply measurement engineers with the definition of software measurement models according to the proposed metamodel.

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). Finally, we shall apply SMF to real environments to obtain further refinements and validation.

ACKNOWLEDGEMENTS

This work has been partially financed by the following projects: INGENIO (Universidad Pólitecnica de Valencia, PAC08-0154-9262) and ESFINGE (Ministerio de Educación y Ciencia, TIN2006-15175-C05-05).

REFERENCES

- (2007). "Eclipse Modelling Framework (EMF) Main Page." from <http://www.eclipse.org/emf>.
- Auer, M., B. Graser, et al. (2003). A Survey on the Fitness of Commercial Software Metric Tools for Service in Heterogeneous Environments: Common Pitfalls Ninth International Software Metrics Symposium.(Metrics '03).
- Bézivin, J., F. Jouault, et al. (2005). Principles, standards and tools for model engineering. 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'2005).
- Boronat, A. and J. Meseguer (2007). Algebraic Semantics of EMOF/OCL Metamodels, CS Dept., University of Illinois at Urbana-Champaign.
- Brown, M. and G. Dennis (2004). "Measurement and Analysis: What Can and Does Go Wrong?" 10th IEEE International Symposium on Software Metrics (METRICS'04): 131-138.
- Eclipse. (2007). "Eclipse Graphical Modeling Framework (GMF) Main Page." from <http://www.eclipse.org/gmf/>
- Fenton, N. and S. L. Pfleeger (1997). Software Metrics: A Rigorous & Practical Approach, Second Edition, PWS Publishing Company.
- Florac, W. A., A. D. Carleton, et al. (2000). "Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process." IEEE Software 17 (4)(4).
- Garcia, F., M. F. Bertoa, et al. (2006). "Towards a consistent terminology for software measurement." Information and Software Technology 48(8): 631-644
- Garcia, F., M. Piattini, et al. (2006). "FMESP: Framework for the modeling and evaluation of software processes." Journal of Systems Architecture - Agile Methodologies for Software Production 52(11):627-639
- Garcia, F., M. Serrano, et al. (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. and C. Lassenius (1999). Using the internet to communicate software metrics in a large organization. Proceedings of GlobeCom'99.
- Jouault, F. and J. Bézivin (2006). "KM3: a DSL for Metamodel Specification."
- Kempkens, R., P. Rösch, et al. (2000). Instrumenting Measurement Programs with Tools. PROFES 2000, Oulu, Finland.
- Kitchenham, B., R. T. Hughes, et al. (2001). "Modeling Software Measurement Data." IEEE Transactions on Software Engineering 27(9): 788-804.
- Kolovos, D. S., R. F. Paige, et al. (2006). Requirements for Domain-Specific Languages. First ECOOP Workshop on Domain-Specific Program Development (ECOOP'06), Nantes, France.
- Komi-Sirviö, S., P. Parviainen, et al. (2001). Measurement Automation: Methodological Background and Practical Solutions-A Multiple Case Study. Seventh International Software Metrics Symposium (METRICS'01), London.
- Lavazza, L. and A. Agostini (2005). "Automated Measurement of UML Models: an open toolset approach." Object Technology 4(4): 115-134.
- Mora, B., F. Garcia, et al. (2008). Software Measurement by using QVT Transformation in an MDA context. ICEIS 2008 (In Press), Barcelona (Spain).

- Mora, B., F. Ruiz, et al. (2008). SMML: Software Measurement Modeling Language, Department of Computer Science. University of Castilla - La Mancha.
- OMG (2007). Architecture-Driven Modernization (ADM): Software Metrics Meta-Model (SMM). OMG Document: dmtf/2007-03-02, Object Management Group.
- Palza, E., C. Fuhrman, et al. (2003). Establishing a Generic and Multidimensional Measurement Repository in CMMI context 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), Greenbelt (Maryland, USA).
- Scotto, M., A. Sillitti, et al. (2004). A relational approach to software metrics. Proceedings of the 2004 ACM symposium on Applied computing (SAC'2004), Nicosia, Cyprus.
- Vépa, É., J. Bézivin, et al. (2006). Measuring Model Repositories. Model Size Metrics Workshop at the MODELS/UML 2006 conference, Genova, Italy.



SciTeP
Science and Technology Publications