Integrated Measurement for the Evaluation and Improvement of Software Processes

Félix García, Francisco Ruiz, José Antonio Cruz, and Mario Piattini

Alarcos Research Group, University of Castilla-La Mancha, Paseo de la Universidad, 4
13071 Ciudad Real, Spain
{Felix.Garcia, Francisco.RuizG, Mario.Piattini}@uclm.es,
jacruz@proyectos.inf-cr.uclm.es
http://alarcos.inf-cr.uclm.es/english/

Abstract. Software processes have an important influence on the quality of the final software product, and it has motivated companies to be more and more concerned about software process improvement when they are promoting the improvement of the final products. The management of software processes is a complex activity due to the great number of different aspects to be considered and, for this reason it is useful to establish a conceptual architecture which includes all the aspects necessary for the management of this complexity. In this paper we present a conceptual framework in which the software process modeling and measurement are treated in an integrated way for their improvement. As a support to the improvement a collection of software process model metrics is proposed. For the management of the measurement process, GenMETRIC, an extensible tool for the definition, calculation and presentation of software metrics, has been developed.

1 Introduction

The research about the software process has acquired a great importance in the last few years due to the growing interest of software companies in the improvement of their quality. Software processes have an important influence on the quality of the final software product, and for this reason companies are becoming more and more concerned about software process improvement, when they are promoting the improvement of the final products. Software applications are very complex products, and this fact is directly related to their development and maintenance. The software process is therefore a process, yet with special characteristics stemming from the particular complexity of the software products obtained. To support the software process evaluation and improvement, a great variety of initiatives have arisen establishing reference frameworks. Among these initiatives, of special note are CMM [22], CMMI [23], the ISO 15504 [10] standard, and, given its importance, the improvement has been incorporated into the new family of ISO 9000:2000 [12], [13] standards that promote the adoption of a focus based on processes when developing, implementing or improving a quality management system. Among the above-mentioned improvement initiatives, CMMI (Capability Maturity Model Integration) stands out as being especially impor-

F. Oquendo (Ed.): EWSPT 2003, LNCS 2786, pp. 94–111, 2003.

tant. Within the context of CMMI, the company should continuously **understand**, **control** and **improve** its processes, in order to reach the aims of each level of maturity. As a result of effective and efficient processes, a company will, in return, receive high quality products that satisfy both the needs of the client and of the company itself.

The successful management of the software process is necessary in order to satisfy the final quality, cost, and time of the marketing of the software products. In order to carry out said management, four key responsibilities need to be assumed [6]: **Definition**, **Measurement**, **Control** and **Improvement** of the process. Taking these responsibilities into account, it is very important to consider the integrated management of the following aspects to be able to promote process improvement:

- **Process Modeling**. Given the particular complexity of software processes, deriving from the high diversity of elements that have to be considered when managing them, it is necessary to effectively carry out a definition process of the software process. From the process modeling point of view, it is necessary to know which elements are involved before processing them. A software process can be defined as the coherent set of policies, structures, organisation, technology, procedures and artifacts needed to conceive, develop, package and maintain a software product [4]. Software process modeling has become a very acceptable solution for treating the inherent complexity of software processes, and a great variety of modeling languages and formalities can be found in the literature. They are known as "Process Modeling Languages" (PML) and their objective is to precisely represent, without ambiguity, the different elements related to a software process. In general, the following elements (general concepts, although, with different notations and terms) can be identified in a software process in the different PMLs [4]: **Activity**, **Product**, **Resource** and **Organisations and Roles**. Faced with the diversity of existing process modeling proposals, a process metamodel becomes necessary. This metamodel can serve as a common reference, and should include all of the aspects needed to define, as semantically as possible, the way in which the software is developed and maintained. With this goal, the Object Management Group recently proposed the SPEM (Software Process Engineering Metamodel Specification) [17] metamodel, that constitutes a language for the creation of concrete process models in a company.
- **Process Evaluation**. In order to promote software process improvement, it is very important to previously establish a framework for analysis (with the aim of determining its strong and weak points). An effective framework for the measurement of the software processes and products of a company, must be provided, in order to carry this out. The other key aspect to be considered, is the importance of defining and validating software process metrics, in order to evaluate their quality. The previous step of the software processes improvement, is their evaluation, and this goal requires the definition of metrics related to the different elements involved in software processes. Due to the great diversity of elements involved in software processes, the establishment of a common terminology for the definition, calculation and exploitation of

metrics is fundamental for the integrated and effective management of the measurement process.

The integration of the modeling and evaluation of software processes is a fundamental factor for a company to reach a high degree of maturity in its processes, as identified by CMMI. Therefore, it is vital that processes be well understood and improved. This means that it is necessary for them to be well defined, and that an effective measurement process should be carried out previously.

In this article we propose a conceptual framework which integrates the modeling and measurement of the software processes to promote their improvement. This framework incorporates the elements necessary to facilitate the definition and evaluation of software processes. Besides, in order to support the evaluation of the process from a conceptual point of view, a set of metrics have been defined.

Firstly, we present a general view of the conceptual framework. In Section 3, a generic metamodel for the integration of the measurement, is described. It has been defined and incorporated into the conceptual architecture, aiming to establish the needed reference for integrated measurement in an organisation. In the following section, a set of representative metrics for the evaluation of software process models, are presented. In Section 5 the *GenMetric* tool, an extensible tool developed to support integrated measurement in a company, is described. Finally, some conclusions and further works are outlined.

2 Conceptual Framework for the Modeling and Measurement of Software Processes

In order for a company to carry out integrated management of its software processes, it is very important for it to establish a rigorous base for:

- the definition of its process models, using singular terminology and precise and well-defined semantics.

- the integrated management of measurement in the company, using a measurement metamodel that is the framework of reference for the creation of concrete measurement models (database measurement, design, analysis result or work product measurements, process model measurements, etc…).

A conceptual architecture with four levels of abstraction has been defined in order to integrate these two very important aspects in a software process. This architecture is based on the MOF (Meta Object Facility) standard for metamodeling, based on object technology [16] proposed by the Object Management Group (OMG).

The aim of MOF is to specify and manage metadata on different levels of abstraction. MOF describes an abstract modeling language (based on the nucleus of UML). In Table 1 the MOF standard conceptual architecture and its application to the framework of work proposed for the improvement of the software process is shown:

Table 1. MOF conceptual levels and their application for integrated improvement.

Level	MOF	Environment Application
M3	MOF-model (meta-meta-model)	MOF-model
M2	Meta-model	Software Process Engineering Metamodel (SPEM) Generic Measurement Metamodel (ISO 15939)
M1	Model	Concrete Process Models Concrete Measurement Models
M0	Data	Instances of Process Models (concrete projects in the real world) Instances of Measurement Models (results of the application of the measurement model)

The lower level of the conceptual architecture, M0, includes the results for:

- The application of a process model, for example, a model for evaluation and improvement [7], or a maintenance model [20] to a concrete software project. At this level of architecture, the results of the execution of a concrete process model will be registered.

- The application of a measurement process. At this level the values obtained following the application of a concrete measurement model will be registered. For example, the values from the measurement of a relational database or the values from the measurement of UML class diagrams.

The data managed at level M0 are instances of the data represented in the next level up, M1. At this level, according to the conceptual architecture proposed, concrete models for the definition of the software process and concrete models for their measurement will be included. From the definition point of view, at this level the company will include its process models, for example, the model for development, maintenance, evaluation and improvement process, etc. From the measurement point of view, this level will include the concrete measurement models used by the company. For example, this could include concrete measurement models for the measurement of relational [3], object-relational [19], active [5] databases, etc. and concrete models for measuring software artifacts such as UML class diagrams [8], state transition diagrams [9], etc. Moreover, at this level the company could also dispose of measurement models for the defined process models themselves. A collection of metrics of software process models is described in Section 4.

All of the models defined in level M1 are instances of the concepts represented in M2. Therefore, in the M2 level of abstraction of the conceptual architecture, generic metamodels for the creation of concrete models should be included. In our framework the generic metamodels required are:

- **Software Process Metamodel**, with which concrete process models can be defined. SPEM [17] has been chosen as a software process metamodel due to

its wide acceptation in the industry. This metamodel contains the constructors needed to define any concrete software process model. The conceptual model of SPEM is based on the idea that a software development process consists of the collaboration between abstract and active entities, referred to as process roles, that carry out operations, called activities, on tangible entities, called work products. SPEM is basically structured in 5 packages that are: **Basic Elements**, which includes the basic elements needed to describe processes; **Dependences**, that contains the dependences necessary in order to define the relationships between the different process modeling elements, like for example, the ”precedes” dependence, which is a relationship between activities, or between work definitions, and indicates “beginning-beginning”, “end-beginning” or “end-end” dependences; **Process Structure**, which includes the structural elements through which a process description is constructed.; **Process Components**, which contains the elements needed to divide one or more process descriptions into self-contained parts, upon which configuration management processes or version controls can be applied; and **Process Life Cycle**, that includes the process definition elements that help to define how the processes will be executed. In short, SPEM makes the software process integrated management, within the proposed conceptual architecture easier, since the concepts of the different models are grouped under a common terminology.

- **Measurement Metamodel**, with which it is possible to define concrete measurement models. This metamodel is described in detail in Section 3.

In the final conceptual level of the architecture, M3, all of the concepts of the process metamodel and measurement metamodel are represented. This is done using the MOF abstract language, which is basically composed of two structures: MOF class and MOF association (these are the main elements for us, although others do exist such as: package, type of data, etc…). In this way, all of the concepts in level M2 are instances of MOF class or MOF association, for example, the SPEM concepts like, “Activity”, “Work Product” and concepts of the measurement metamodel like “Metric”, “Indicator”, “Measurement Unit” are instances of MOF class, and the relationships “Activity precedes Activity”, “Work Product precedes Work Product” or “Metric has a Measurement Unit”, are instances of MOF association.

With this architecture, it is possible to perform integrated management of the software process improvement, since the process definition and its measurement are systematically integrated. The MANTIS-Metamod [7] tool, which allows for the definition of metamodels (based on the MOF language constructors) and of models (based on the constructors of their metamodels), has been developed as a means of support to this conceptual architecture. A repository manager [21] that uses the XMI standard (XML Meta-data Interchange) [18] to promote portability of the defined models and metamodels is used for management of the storage and exchange of the metadata from the conceptual architecture.

3 Measurement Metamodel for the Integrated Process Evaluation and Improvement

A fundamental element to take into consideration when establishing a framework for process improvement, is the possibility of defining objective indicators of the processes that allow a software company to efficiently evaluate and improve its processes at any given moment. Evaluation standards like CMM, CMMI, ISO 15504, ISO 9000:2000 have assigned an important role to measurements in order to determinate the status of the software processes. A measurement process framework must be established in order to do so.

A good base for developing a measurement process is the one provided by CMMI [23]. In CMMI a new key process area called "Measurement and Analysis" is included. The aim of this area is to develop and establish a measurement capacity that can be used to support the company's information needs, and this implies broadening the concepts included in the CMM model. According to CMMI, the first step in the measurement process is to identify the measurement objectives, so that, in a second step, a measurement and analysis process can be implemented. This requires the measurement to be integrated in the different work processes of a company. It is very important for a company wishing to implant an effective measurement process to be able to precisely define concrete measurement models that, being supported by an integrated measurement tool, allow the appropriate and necessary automation for process evaluation.

Most of the problems in collecting data on a measurement process are mainly due to a poor definition of the software measures being applied. Therefore, it is important not only to gather the values pertaining to the measurement process, but also to appropriately represent the metadata associated to this data. In [14] a method for the specification of measurement models is defined with the aim of capturing the definitions and relationships between software measurements. The proposed framework is made up of three levels of abstraction for measurement, starting from a generic measurement model and moving up to automation of the gathering of metric values on a project level. This idea of abstraction is fundamental in order to be able to effectively integrate the measurement process into the organisation.

Therefore, it is very convenient to introduce a generic metamodel for measurement, making it possible to derive concrete measurement models that make up the base for assessment and improvement processes in an organisation. In Figure 1 our proposal for a measurement metamodel based on the ISO 15939 [11] standard is represented in UML.

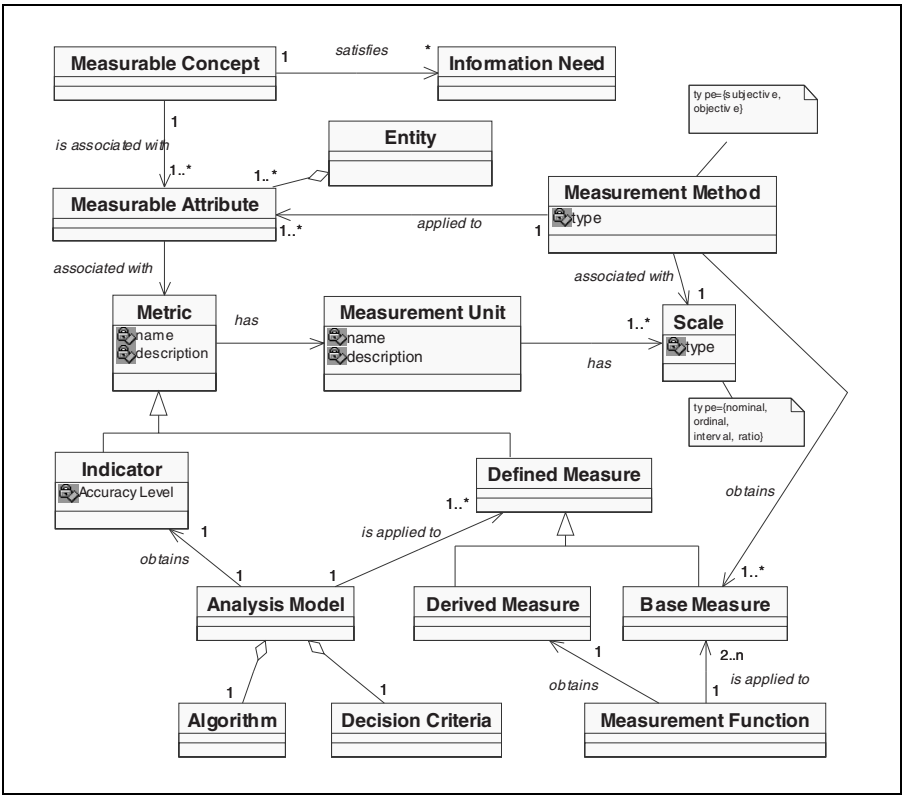

Fig. 1. Generic Metamodel for the Measurement Process

As can be observed in Figure 1, under the measurement point of view, the elements on which properties can be measured are "**Entities"**. An entity is an object (for example, a process, product, project or resource), that can be characterised through the measurement of its **"Measurable Attributes"** which describe properties or characteristics of entities, which can be distinguished quantitatively or qualitatively by human or automatic means. The aim of attributes is to satisfy specific information needs such as, "*the need to compare software development productivity with respect to a determined value*". This abstract relation between attributes and information needs is represented by the element called **"Measurable Concept"**, that, in this case, would be *"productivity ratio of software development"*. As measurable attributes, attributes of the developed product size or of development effort could be used.

All measurable attributes are associated to a metric, which is an abstraction of the different types of measurements used to quantify, and to make decisions concerning the entities. All metrics are associated to a unit of measure (for example, code lines), which at the same time belong to a determined scale. In accordance with the standard, the 4 scales distinguished are: nominal, ordinal, interval and ratio, although other classifications can be established like in [14]. The three types of metrics are:

- **Base Measurement**, defined in the function of an attribute, and the method needed to quantify it (a measurement is a variable to which a value is assigned).

- **Derived Measurement**, a defined measurement in function of two or more values of base measurements.

- **Indicator**, a measurement that provides an estimate or assessment of specific attributes, derived from a model with respect to information needs. The indicators are the base for analysis and decision-making. These measurements are the ones that are presented to the users in charge of the measurement process.

The procedures for calculating each of the metric types are:

- The values of the base measurements are reached with "***Measurement Methods***" that consist of a logical sequence of operations, generically described, used to quantify an attribute with respect to a specific scale. These operations can imply activities such as, counting occurrences or observing the passing of time. The same measurement method can be applied to multiple attributes.

- The derived measurements are obtained by applying a "***Measurement Function***", which is an algorithm or calculation carried out to combine two or more base measurements. The scale and unit of the derived measurement depends on the scales and units of the base measurements.

- The indicators are obtained with an "***Analysis Model***". An analysis model produces estimates and assessments relevant to the defined information needs. It consists of an algorithm or calculation that combines one or more base measurements and/or derivates with determined decision-making criteria. All decision-making criteria is composed of a series of limit values, or used objects for determining the need to research, or to describe the confidence level with regard to a determined result. These criteria help to interpret the measurement results.

Using this reference metamodel it is possible to measure any element of a process or data model. Taking into account that our main objective is the software process improvement, and therefore, the evaluation, itis necessary to establish the relationship between the main elements of the software process metamodel and the main elements of the software measurement metamodel. This relationship is represented in Figure 2.

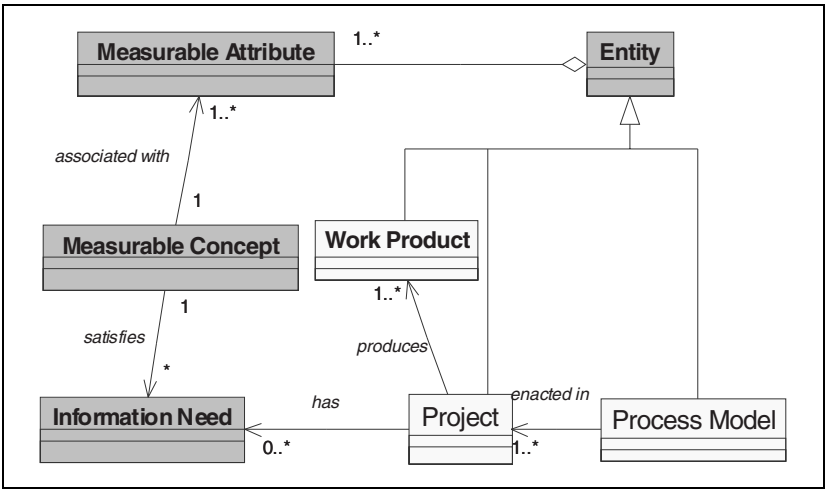

Fig. 2. Relation between the software process metamodel and the measurement metamodel

As we can observe in Figure 2, any software process model is enacted in concrete software projects. As a result of carrying out a software project, certain work products are produced and all software projects are required to satisfy some information needs. The main candidate elements to be measured in order to evaluate the software process are:

- **The Software Process Model**. It could be very convenient to research if the model of software processes has an influence on its final quality. For this reason, with the framework proposed, it is possible to define metrics related with the constructors of the software process metamodel. For example, if we apply the measurement metamodel to the elements of the SPEM model, we could measure important elements like the class *Activity,* and the classes *Work Product and Process Role.* These elements of the model have a set of measurable attributes, such as for an activity: *"the number of activities with which there is a precede type dependence"*. This attribute would be calculated with a metric to satisfy an information necessity like, *"Evaluate the software process coupling"* and, in this case, the unit of measure would be of *"Ratio"* type. This issue will be treated in the following section.

- **The Work Product**. This is a fundamental factor in the quality of the software process. Work Products are the result of the process (final or intermediate), and their measurement is fundamental in order to evaluate the software processes. With the framework proposed, it's possible to measure the quality attributes related with the artifacts or work products, by defining the metamodels related with the different work products. For example, if we have to evaluate the quality of a UML class diagram we have to incorporate into the framework, the UML metamodel and metrics necessary.

In this way, with the framework proposed, the work of an assessment and improvement process is eased, since the fulfillment of the software processes carried out in a determined organisation are quantitatively registered.

4 Proposal of Software Metrics for the Evaluation of Software Processes Models

The study of the possible influence of the software process model complexity in its execution could be very useful. For this reason the first step is the definition of a collection of useful metrics in order to characterise the software process models. In this section a collection of metrics of software process models are going to be defined in order to evaluate their complexity. These metrics have been defined according to the SPEM terminology, but they can be directly applied to other process modeling languages. The metrics proposed could be classified like model level metrics, if they evaluate the characteristics of a software process model, or like fundamental element (activity, process role and work product) metrics, if they describe the characteristics of a model element. For this reason they will be described separately.

4.1 Model Level Metrics

The process model level metrics (PM) proposed are:

- **NA(PM)**. Number of **Activities** of the process model.
- **NSTP(PM)**. Number of steps (tasks) of the process model.
- **NDRA(PM)**. Number of dependence relationships between activities of the process model.
- **RSTPA(PM).** Ratio of steps and activities. Average of the steps and the activities of the process model.

$$RSTPA(PM) = \frac{NSTP(PM)}{NA(PM)}$$

- **AC (PM)**: Activity Coupling in the process model. This metric is defined as:

$$AC(PM) = \frac{NA(PM)}{NDRA(PM)}$$

- **NWP(PM):** Number of **Work Products** of the process model.
- **RWPA(PM):** Ratio of work products and activities. Average of the work products consumed (input), modified (input/output) or produced (output) by the activities.

$$RWPA(PM) = \frac{NWP(PM)}{NA(PM)}$$

- **NPR(PM):** Number of **Process Roles** of the process model.
- **RPRA (PM):** Ratio of process roles and activities. Average of the process roles and the activities of the process model.

$$RPRA(PM) = \frac{NPR(PM)}{NA(PM)}$$

4.2 Fundamental Element Level Metrics

- **Activity Metrics**:
  - **NSTP(A)**. Number of **Steps** (tasks) of an Activity.
  - **NWPIn(A)**. Number of Input Work Products of the Activity.
  - **NWPOut(A).** Number of Output Work Products of the Activity.
  - **NWPInOut(A).** Number of Input-Output Work Products of the Activity.
  - **NWP(A).** Total Number of **Work Products** related to an Activity.

$$NWP(A) = NWPIn(A) + NWPOut(A) - NWPInOut(A)$$

  - **RWPIn(A)**. Average of the Input Work Products in activity A.

$$RWPIn(A) = \frac{NWPIn(A)}{NWP(A)}$$

  - **RWPOut(A)**. Average of the Output Work Products in activity A.

$$RWPIn(A) = \frac{NWPOut(A)}{NWP(A)}$$

  - **RWPInOut(A)**. Average of the Output Work Products respect to the total number of Work Products in activity A.

$$RWPInOut(A) = \frac{NWPInOut(A)}{NWP(A)}$$

  - **NR(A).** Number of responsible **Roles** of an Activity.
  - **NPD(A)**. Number of **Activities** which are **predecessors** (activity dependences of input) of Activity A.

- **NSD(A)**. Number of **Activities** which are **successors** (activity dependences of output) of Activity A.
- **ND(A)**. Total number of dependences of activity A.

$$ND(A) = NPD(A) + NSD(A)$$

- **PR(A)**. Average of the predecessors activities with respect to the total number of dependences in activity A.

$$PR(A) = \frac{NPD(A)}{ND(A)}$$

- **PS(A)**. Average of the successors activities with respect to the total of dependences in activity A.

$$PS(A) = \frac{NSD(A)}{ND(A)}$$

- **Process Role Metrics**:
  - **NARP(R)**. Number of **Activities** who's responsibility is role R.
  - **RRPR(R)**. Ratio of responsibility of the process role. Ratio between the activities in which role R is responsible and the total number of activities in the model.

$$RRPR(A) = \frac{NARP(R)}{NA(PM)}$$

- **Work Product Metrics**:
  - **NAWPIn(WP)**. Number of **Activities** in which the work product is of input.
  - **NAWPOut(WP).** Number of **Activities** in which the work product is of output.
  - **NAWPInOut(WP).** Number of **Activities** in which the work product is of input/output.
  - **NAWP(WP)**. Number of Activities related with the Work Product.

$$NAWP(WP) = NAWPIn(WP) + NAWPOut(WP) - NAWPInOut$$

  - **RDWPA(WP)**. Ratio of dependence of the work product. Ratio between the activities related with the work product and the total number of activities in the model.

$$RDWPA(WP) = \frac{NAWP(WP)}{NA(PM)}$$

4.3 Example

Figure 3 shows an example of a simplified software process model which belongs to the Rational Unified Process [2]. For the graphical representation of the model the SPEM notation [17] has been used. The values of the metrics proposed are shown in the tables 2, 3 and 4.

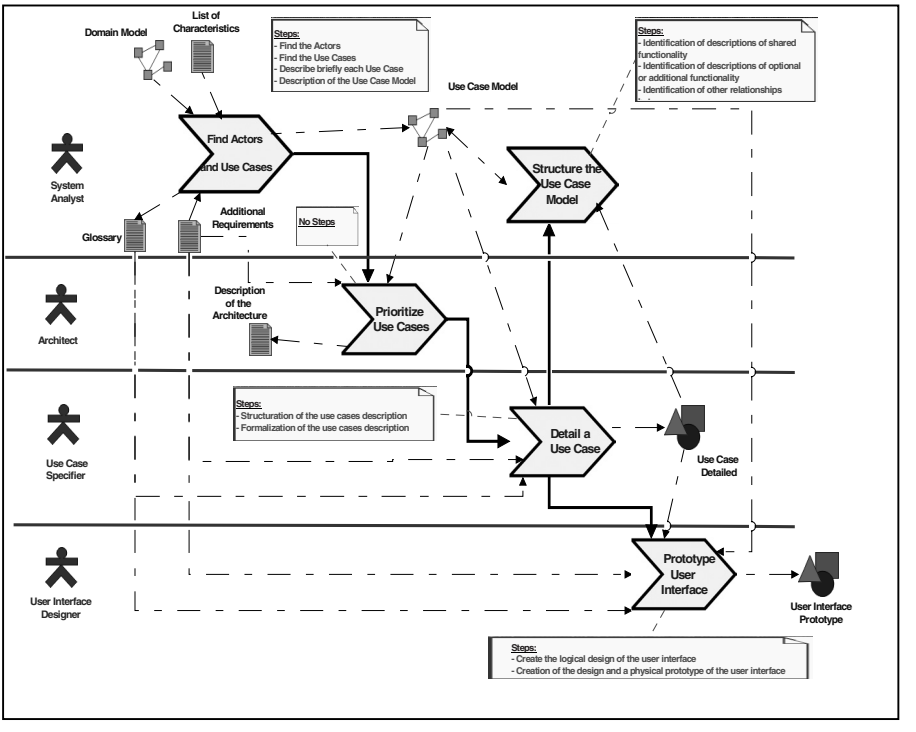

Fig. 3. Example of a Software Process Model represented with SPEM.

Table 2. Model Level Metrics

Metric	Value	Metric	Value
NA(PM)	5	NWP(PM)	8
NSTP(PM)	11	RWPA(PM)	8/5=1,6
NDRA(PM)	4	NPR(PM)	4
RSTPA(PM)	11/5=2,2	RPRA(PM)	4/5= 0,8
AC(PM)	5/4= 1,25		

Table 3. Metrics of the Activity "Detail a Use Case".

Metric	Value	Metric	Value
NSTP(A)	2	RWPInOut(A)	0
NWPIn(A)	3	NR(A)	1
WPOut(A)	1	NPD(A)	1
NWPInOut(A)	0	NSD(A)	2
NWP(A)	4	ND(A)	3
RWPIn(A)	3/4= 0,75	PR(A)	1/3=0,33..
RWPOut(A)	1/4= 0,25	PS(A)	2/3=0,66..

Table 4. Metrics of Work Product and Process Role examples.

Process Role "System Analyst" Metrics	Value	Work Product "Use Case Model" Metrics	Value
NARP(R)	2	NAWPIn(WP)	4
RRPR(R)	2/5=0,4	NAWPOut(WP)	2
		NAWPInOut(WP)	1
		NAWP(WP)	4+2-1=5
		RDWPA(WP)	5/5=1

This is only the first step in the overall metrics definition process [3]. The following step is the formal validation of the metrics, and then, it is fundamental to run empirical studies in order to prove the practical utility of the metrics defined. As a result of this step (and of the complete method) we will be able to accept, discard or redefine the metrics presented in this paper. An important number of metrics have been proposed, and the validation process is fundamental for the selection of the adequate metrics which fulfill our objective.

5 GenMETRIC. Extensible Tool for the Integrated Management of the Measurement Process

Aiming to offer automatic support to the integrated measurement process commented on in the previous sections, we have developed the GenMETRIC tool. GenMETRIC is an extensible tool for the definition, calculation and visualisation of software metrics. This tool for the integrated management of the measurement process supports the definition and management of software metrics. Moreover, the tool supports the measurement metamodel based on ISO 15939 that has been proposed for better support and management of the integrated measurement process.

For the management of the measurement process, the tool can import information on the following elements, represented in XMI document form [18]:

- **Domain Metamodels** on which metrics are defined. The elements of each metamodel are stored to be able to carry out a measurement process on them. For example, if a relational database is to be measured, it will be necessary to previously define the elements of the relational metamodel, like: Table, Attribute, Interrelation, etc...
- **Domain Models**. The models are instances of the metamodels, and it is of interest to carry out a measurement process on them. For example, the schema (model) of the database of a bank would be an instance of the relational metamodel on which we could carry out a measurement process.
- **Metric Models**. Metric models allow the defined metrics to be consistently stored. In order to do so, the metric models used by the tool are instances of the measurement metamodel proposed in the previous section.

The information imported by the tool on the different domain metamodels, and on the metrics is persistently stored in a **XMI** based **Repository**. The calculation of the metrics defined is performed by using the information in the *Repository.* The different models and metamodels needed are defined and represented in XMI with the MANTIS-Metamod [7] tool. The relationship between GenMetric and MANTIS-Metamnod is represented in the following figure:

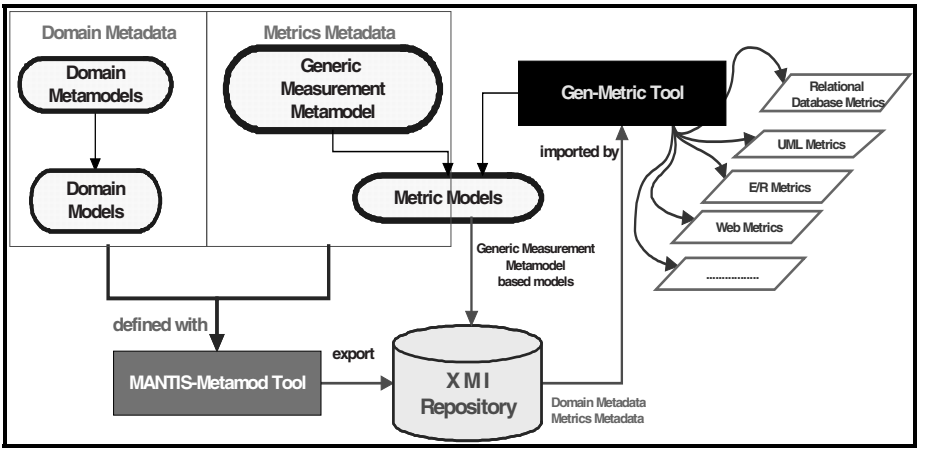

Fig. 4. Relationship between MANTIS-Metamod and Gen-Metric

As we can observe in Figure 4, the repository is the key element for the integrated management of the measurement process. The metadata are defined and exported in XMI with MANTIS-Metamod tool. The information of the repository is imported by GenMetric for the management of the metrics needed, and with GenMetric, the user can build metrics models (based on the generic metamodel). These models are exported to the repository.

GenMetric provides the user with a powerful interface for the definition, calculation and visualisation of metrics. From the perspective of the use of the tool, two roles

have been defined. They are: *Administrator,* that completely controls the functionality of the tool, allowing it to define, calculate and visualise any metric, and *User,* that has access to the calculation and visualisation of the metrics that have already been defined. In Figure 5 the interface for the definition of metrics is represented:

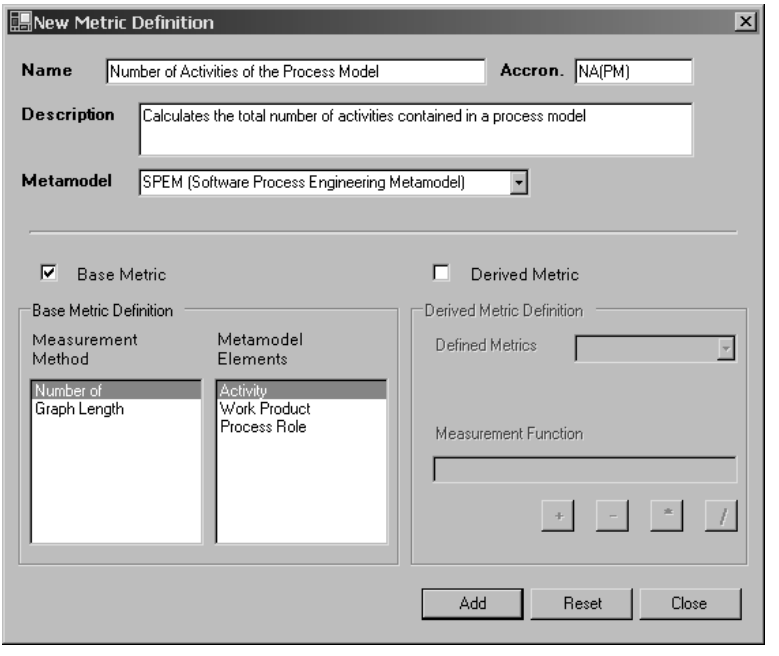

Fig. 5. Definition of a new metric with GenMetric.

An integrated and automatic environment for measurement is provided with the proposed tool. Being a generic tool, the definition of any new metric on the existing domain metamodels is possible, without having to code new modules. Furthermore, the tool is extensible, which eases the incorporation of new domain metamodels, for example a metamodel for defining web elements (formed by web pages, links between pages, etc.) and in this way it is possible for concrete domain models, web sites for example, to be measured. Moreover, as it works with XMI documents, it eases communication and the possibility of openly importing new domain metamodels, or domain models and metrics stored in other repositories based on MOF.

6 Conclusions and Future Work

In this work, a conceptual framework to promote the improvement based on integrated modeling and measurement of software processes has been presented. The SPEM metamodel is used for the modeling of processes under a common terminology, and a metamodel based on the ISO 15939 standard, easing the management of an integrated

measurement process to promote improvement in a company's processes, has been defined as an integrated framework for measurement.

In order to evaluate the influence of the complexity in the software process models in their enactment, some metrics have been proposed. These metrics are focused on the main elements included in a model of software processes, and may provide the quantitative base necessary to evaluate the changes in the software processes in companies with high maturity levels, which are applying continuous improvement actions [1].

As a means of support for the integrated measurement, the *GenMetric* tool has been developed to define, calculate and visualise software metrics. With the tool it is possible to incorporate new types of metrics and new types of elements to measure, since its architecture has a generic and extensible design.

With the proposed framework, any company dedicated to the development and/or maintenance of software can effectively define and evaluate its processes as a step prior to promoting their improvement. Furthermore, as the framework is based on the MOF standard, the simple extension and modification of its elements is possible, with the incorporation and modification of the necessary metamodels, since all of them are represented following the common terminology provided by the MOF model. Along the lines for improvement for future studies, we can point out the following:

- Description of concrete measurement models, using the generic metamodel, to effectively support the evaluation and improvement of software processes.
- Formal and empirical validation of the metrics proposed to study the relationship between the influences of the software process models complexity in their enactment.

Acknowledgements. This work has been partially funded by the TAMANSI project financed by "Consejería de Ciencia y Tecnología, Junta de Comunidades de Castilla-La Mancha" of Spain (project reference PBC-02-001) and by the DOLMEN project (Languages, Methods and Environments), which is partially supported by FEDER with number TIC2000-1676-C06-06.

References

1. Pfleeger, S.L.: Integrating Process and Measurement. In Software Measurement. A. Melton (ed). London. International Thomson Computer Press (1996) 53–74
2. Jacobson, I, G. Booch and J. Rumbaugh,. The Unified Software Development Process. Addison Wesley (1999)
3. Calero, C., Piattini, M. and Genero, M.: Empirical Validation of referential metrics. Information Software and Technology". Special Issue on Controlled Experiments in Software Technology. Vol.43, Nº 15 (2001)
4. Derniame, J.C., Kaba, B.A. and Wastell, D.: Software Process: Principles, methodology and technology. Lecture Notes in Computer Science 1500 . Springer (1999)
5. Díaz, O., Piattini, M. and Calero, C.: Measuring triggering-interaction complexity on active databases. Information Systems Journal. Elsevier Science. Vol. 26, Nº 1 (2001)

6. Florac, W. A. and Carleton, A.D.: Measuring the Software Process. Statistical Process Control for Software Process Improvement. SEI Series in Software Engineering. Addison Wesley (1999).
7. García, F., Ruiz, F., Piattini, M. and Polo, M.: Conceptual Architecture for the Assessment and Improvement of Software Maintenance. 4th International Conference on Enterprise Information Systems (ICEIS'02). Ciudad Real, Spain, April (2002), 610–617
8. Genero, M., Olivas, J., Piattini, M. and Romero, F.: Using metrics to predict OO information systems maintainability. 13th International Conference Advanced Information Systems Engineering (CAiSE'01), (2001), 388–401
9. Genero, M., Miranda, D. and Piattini, M.: Defining and Validating Metrics for UML Statechart Diagrams. 6th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE), (2002), 120–136
10. ISO/IEC: ISO IEC 15504 TR2:1998, part 2: A reference model for processes and process capability, (1998)
11. ISO IEC 15939, Information Technology – *Software Measurement Process*, Committee Draft, December (2000)
12. International Organization for Standardization (ISO). *Quality management systems - Fundamentals and vocabulary*. ISO 9000:2000, (2000). See http://www.iso.ch/iso/en/iso9000-14000/iso9000/selection_use/iso9000family.html
13. International Organization for Standardization (ISO). 2000. *Quality management systems - Requirements* ISO 9001:2000, (2000)
14. Kitchenham, B. A., Hughes, R.T. and Linkman, S.G.: Modeling Software Measurement Data. IEEE Transactions on Software Engineering. 27(9), (2001), 788–804
15. OMG Unified Modeling Language Specification; version 1.4, Object Management Group. September (2001). Available in http://www.omg.org/technology/documents/formal/uml.htm
16. Meta Object Facility (MOF) Specification; version 1.4. Object Management Group. April (2002). In http://www.omg.org/technology/ documents/formal/mof.htm
17. Software Process Engineering Metamodel Specification; adopted specification, version 1.0. Object Management Group. November (2002). Available in http://cgi.omg.org/cgi-bin/doc?ptc/02-05-03.
18. OMG XML Metadata Interchange (XMI) Specification; version 1.2. Object Management Group. January (2002). In http://www.omg.org/technology/documents/formal/xmi.htm
19. Piattini, M., Calero, C., Sahraoui, H. and Lonis, H.: Object-Relational Database Metrics. L' object. ISSN 1262–1137. HERMES Science Publications, Paris. Vol.7, Nº 4, (2001)
20. Ruiz, F., Piattini, M. and Polo, M. An Conceptual Architecture Proposal for Software Maintenance. International Symposium on Systems Integration (ISSI, Intersymp'2001). Baden-Baden, Germany (2001), VIII:1–8
21. Ruiz, F., Piattini, M., García, F. and Polo, M. An XMI-based Repository for Software Process Metamodeling. Proceedings of 4th International Conference on Product Focused Software Process Improvement (PROFES'2002). Lecture Notes in Computer Science (LNCS 2559), Markku Oivo, Seija Komi-Sirviö (Eds.). Springer. Rovaniemi (Finland). December (2002), 546–558
22. Software Engineering Institute (SEI). The Capability Maturity Model: Guidelines for Improving the Software Process, (1995). In http://www.sei.cmu.edu/cmm/cmm.html
23. Software Engineering Institute (SEI). Capability Maturity Model Integration (CMMI^{SM}), version 1.1. March (2002). In http://www.sei.cmu/cmmi/cmmi.html