Measurement Processes are Software, Too*

Maurizio Morisio  
Dipartimento di Automatica e Informatica  
Politecnico di Torino  
Corso Duca degli Abruzzi, 24, 10129 Torino, Italy  
e-mail: morisio@polito.it  
fax: + 39 11 5647099  
voice: + 39 11 5647033

Abstract
Software process improvement and measurement are closely linked: measures are the only way to prove improvements in a process. Despite this link, and the interest in process improvement, measurement is not widely applied in industrial software production. This paper describes a method designed to guide the definition, implementation and operation of measurement processes. The method, which builds upon Fenton’s measurement framework and GQM, starts from the point that measuring a software process is in its turn a process in the software process. The three basic ideas of the method derive from this assumption: -the measurement process should reuse and suitably adapt the same phases of the software process: requirements definition, design, implementation, etc.,  
-a descriptive process model should be the essential starting point of a measurement process,  
-many concepts and tools which derive from the object oriented approach should be effectively used in the measurement process.  
An experimental application in an industrial process has shown that building the process model was the hardest part of the measurement process, and that it has improved the quality of measurement by reducing misunderstandings. Object oriented concepts and tools make it possible to automate certain tasks (for instance the definition of the schema of the measurement database) and to improve robustness against changes in the measurement process.

*This work was partially supported by the European Commission, DGIII/A, under contracts Esprit/ESSI AEFTA n. 10070, MOODS n. 23819 and PSP-NC n. 24060.

This paper is published on the Journal of Systems and Software, vol. 49(1), December 1999.
1 INTRODUCTION

To demonstrate improvements in the software process, it has to be measured. In the Capability Maturity Model of the Software Engineering Institute (Paulk et al. 1993), measurement is a key process area of level 4 organisations. It is the tool which can be used to understand and control processes and products by means of quantitative data. In the practical application of ISO9001 (ISO 1994) to software processes, certification bodies increasingly require that the quality policy of an organisation be stated in terms of measurable quality objectives. In the Personal Software Process (PSP) (Humphrey 1995) measurement is established, at an individual level, at the very beginning and is continuously used to monitor and improve the personal process. Nevertheless, the current situation in the software industry reveals there is limited use of measurement. One reason is that measurement was not part of the software engineering practice at its establishment, another reason is that applied measurement is difficult and demanding both in effort and know-how. An organisation trying to set up a measurement program is faced with the problem of selecting a reasonable and useful number of measures from the huge set of metrics proposed in the literature, finding tools to compute and collect them, motivating people, modifying processes to encompass measurement activities, documents and roles. The future widespread use of measurement in an industrial context is subordinated to the availability of frameworks and methods to guide the introduction and operation of measurement.

Fenton (1995, 1997) defines a simple but extremely powerful framework which gives a structure to the huge, flat and confusing set of measures available. Moreover he applies the representational theory of measurement to software measures, in order to isolate from that set the measures which satisfy the representation condition, and on which meaningful statements are defined. Zuse (1990) does a similar work on complexity measures.

Basili and Rombach (1988) (see also Basili 1995) define the Goal Question Metric (GQM) approach, a method used to select or define measures adapted to a given context starting from the high level goals stated for the measurement program. The amii consortium (Rowe and Whitty 1993) defines a method that builds on the GQM, defining 12 steps towards the introduction of a measurement program, to be iterated to improve and adapt it. Hetzel (1993) defines a partially bottom up method (define, collect, present, use, validate). The Software Engineering Institute proposes a standard set of core measures (Carleton 1992, Florac 1992, Goethert 1992, McAndrews 1993). Other methods are proposed by Westfall (1996), Fisher and McCalla (1996) while Roche (1994) compares some of them.

If standards are to be considered, IEEE 1061 (IEEE 1993) defines a method to select and validate metrics and lists (in the appendix, not part as of the standard) a set of metrics and their correlation with process or product attributes. IEEE 1045 (IEEE 1992) defines common terms for size and productivity measures, ISO 9126 (ISO 1991) defines high level external product attributes and proposes related measures. All these standards recognize the impossibility of defining a definite set of measures and leave to the user the final responsibility of selecting or adapting measures according
to the context. Surprisingly, the standard ISO12207 (ISO 1995) does not include any process or activity dedicated to measurement.

This paper presents a method which can be adopted to guide the introduction and operation of measurement in a software organisation. It builds upon Fenton’s framework and GQM, and the innovative aspect of the method is that measuring a software process is in its turn a process in the software process. The three basic ideas of the method are derived from this assumption.

- The measurement process should suitably reuse and adapt the same primitives of the software process: phases (requirements definition, design, implementation, etc.), products, tools, roles.
- A descriptive process model should be the foundation of a measurement process.
- Many concepts and tools part of the object oriented (OO) approach should be effectively used in the measurement process.

These three points are discussed below.

**Measurement as a process**

Usually measurement is considered as a function in the software process. In fact an effective measurement program requires resources and roles (a project manager, or measurement manager, tools, machines, effort from staff for data collection, effort for data presentation and discussion), products (the measurement plan, the measure database, measurement reports), and phases (definition, collection, analysis of measures). All these elements define a measurement process (MP), more precisely, given a software organisation, an organisation level MP and a project level MP per project.

We propose that, as "software processes are software too" (Osterweil 1987), also measurement processes are software too. We use the latest interpretation of Osterweil’s statement, i.e. software processes (and measurement processes) are not only coding but also requirements, design, etc.. The consequence of this point of view is that most concepts and tools used for the software process can be profitably reused for the MP: lifecycles, formalisms, techniques, tools, methods. Section 2 of this paper is devoted to highlighting some of these possibilities, but we believe many more could be proposed.

In particular, as the software process takes advantage of defining a process model, the measurement process takes advantage of defining a (measurement) process model.

**Process modeling and measurement**

The issue of process modelling is receiving growing attention in the software community. The process model is seen as the starting point from which to analyse, improve and enact the process, but the need of strict coupling between process modelling and process measurement is just emerging.
In the CMM (Paulk et al. 1993) this coupling is implicit, since measurement is performed in level 4 organisations, which are supposed to have defined their process in level 3. Basili and Weiss (1984) consider the measurement process and its validation, but do not couple the measurement process and the surrounding software process. Bache and Bazzana (1994), Hetzel (1993) cite a process model but do not define or use it. Pfleeger and McGowan (1990) associate sets of measures with the levels of the CMM but do not define the measures on a process model. Pfleeger (1995) refines the same approach by defining an SADT process model in order to help in selecting the metrics suitable for a process maturity level. The process model becomes more detailed as process maturity grows, and measurement is applicable only to entities which appear in the process model. The SADT model is also used to depict which measure is collected in which process phase. Wolf and Rosenberg (1993) define (implicitly) a process model in terms of the production of events, then collect and analyse them in order to assess the process. Cook and Wolf (1994) continue along the same lines by defining measures of distance of sequences of real events from process model events. Amadeus (Selby 1991) offers an environment in which to define the conditions on the process, collecting the related events, either to react to them or to measure the process. Broeckers et al. (1995) define MVP-L, as a language for process modelling. One of the basic requirements for the definition of the language was the description of real world processes, instead of the prescription and enactment of small, artificial processes; another requirement was the support given to instrumenting the process for data collection. Matsumoto et al. (1993) were the first authors, to our knowledge, to use a process model to define and collect process measures. The model is represented by Petri nets; duration measures are coupled with transition firing. The Petri net representation allows the process to be enacted and measures to be automatically collected. The drawback of this approach is in the use of plain Petri nets, which are not adequate to express structure and static relationships, nor to model complex systems such as a real world process model. Harrison (1988) applies a similar approach for product metrics, but uses a data oriented formalism. He models a Cobol program as an entity-relationship diagram, then he defines metrics as scripts on the diagram. Rozum (1996) uses Extended Entity Relationship diagrams to define core measures (Carleton 1992).

The process model as a foundation for measurement
When measures are collected in current software engineering practice, and no process model is used, three situations can occur.
1. The process is not described, measures are applied and their definition is taken from the literature. A definition of a measure relies on the identification or definition of the attribute of an entity, or the entity to be measured (see Fenton and Pfleeger’s framework for measurement in (Fenton and Pfleeger 1997)). An entity defined in the literature and the equivalent entity in the context of
a process can differ, resulting in mismatches between the definition of measures and their application in the actual process context. For instance, to compute the design effort, you must know how design is defined, whether it consists of the initial design only, or of rework to fix faults too? And how is a fault defined? Is an incoherence between design documents a fault or not? If the design documents are not yet validated, is it still a fault? And when is a document considered to be validated? Standard entities, such as design and fault, in fact rely heavily on their context. Given a project, since process measures are in many cases collected manually by different people, it is plausible that each person gives a different interpretation to entities, producing unreliable measures. The situation is even worse if many projects are considered.

2. The process is described in natural language, process measures are applied and their definition is taken from the literature. The risk of mismatches still exists, but it is lower, since the definition of the process prescribes coherency checks. The real problem here is with the complexity of the process, which is not adequately modeled in natural language, with the well-known drawbacks in analysis and communication. This leads again to the risk of different interpretations by different people, producing unreliable measures.

3. The process is described in a semi-formal language. Entities are defined in a language suitable to represent them and to tackle complexity. The risk of different interpretations is greatly reduced. Now the problem lies in the separation between process definition and measures definition which can lead to incoherencies. Adding a measures-process validation task is one solution, but it is even better to merge the measures and process definition, that is to say, measures are defined as an extension of the process model.

Our thesis is that the collection of valid measures relies on a clear, unambiguous, easy to understand and communicate model of the current process, and that the definition of measures should be integrated in the process model. This is also a consequence of one of the tenets of measurement theory, entities to be measured should be modeled, according to a specific and stated viewpoint (Fenton 1995).

We call 'measurement process model' (MPM) a snapshot of the current process. It is defined with the purpose of measuring the current process. It corresponds to a descriptive (as opposed to prescriptive) process model in the terminology of the software process community (Feiler and Humphrey 1993), with the difference that it is defined with the purpose of measuring, and is also influenced by the goals of the measurement program. A descriptive process model, defined with the aim of understanding the process, could be more or less detailed in some parts. In measurement terminology it corresponds to a model of entities and attributes which are to be measured (Fenton 1995). **The object oriented approach and measurement**

As already stated, if one considers the measurement process as a software process, one can reuse and apply to it concepts and tools which originated in software development, in particular in the OO approach.
While a lot of work is produced on OO measures, nothing, to our knowledge, exists about the OO approach applied to the measurement process, while some works exist on OO and process modelling.

The majority of works in the process modelling domain and all commercially available process support tools do not take this point of view and use activity or dynamics oriented formalisms such as Petri nets (see for instance Curtis (1992)). The rationale of this choice is that the model is used to enact the process. De Champeaux et al. (1993) use Object Oriented Analysis to model an object oriented analysis process. SLANG (Bandinelli et al. 1994) integrates Petri nets and data structures described by an OO formalism. E³ (Jaccheri et al. 1996) uses a pure OO approach in process modelling, the main goal being the elicitation of the process and not its enaction.

As discussed above, we base the measurement process on a process model, the MPM. The MPM is descriptive, and has no need to represent temporal relations. This is an essential difference from prescriptive models, used to enact processes, that need sophisticated means to model concurrent, sequential, forbidden activities. An MPM (as can be seen in the working example presented in section 3) deals with entities and the static relationships among them. This corresponds very well with the OO approach, so we propose to use it to define the MPM.

Using an OO modeling technique (in section 3 one of these, OMT, is used), entities in the MPM are modeled as classes, and static relationships among them are modeled as associations\(^1\), measures are naturally defined as scripts on classes and relationships.

The next step in an OO process is design. The schema of the database for measures is derived automatically by the MPM model, and procedures to compute measures are derived from the scripts. We will discuss later the advantages of using the OO approach: the clarity in the definitions of measures, the automation of parts of the MP, and the ease in reacting to changes in the MPM.

\(^{1}\) OMT has also several constructs (such as events, state transitions diagrams, conditions on events, actions, event flow diagrams, event trace diagrams, object interaction diagrams) that make it suitable to represent time and dynamic relations. From this point of view it could be used to model prescriptive process models too. An interesting research issue is how an OMT descriptive process model could be derived by an OMT prescriptive one.
2 THE MEASUREMENT PROCESS

In this section a project level Measurement Process is described. An organisation level MP can be described with the same approach. The MP will be described by exploiting the same primitives used for the software process, i.e. phases, roles, products, lifecycle.

The working example is the PSP0 of Humphrey’s PSP (Personal Software Process) (Humphrey 1995, appendix C). We chose the PSP as an example because its definition is detailed, public and easy to obtain. Moreover PSP0 can easily be modeled by a few classes, and contains the most relevant issues of any process and of any measurement process. We also believe that the PSP is exerting and will in the future continue to exert a great influence on software engineering practice and education. A more complex, real life example of application of the MP can be found in (Capra et al. 1997).

**Figure 1: Phases of the measurement process**

![Diagram of Measurement Process Phases](image)

- Requirements definition
  - goals
  - process model
  - questions, measures
  - thresholds, actions

- Design
  - collection
  - storage
  - presentation
  - human factors

- Implementation
  - installation, adaptation of tools
  - training
  - assignment of roles

- Configuration management
- Project Management
- Quality Assurance

- Operation
  - collection,
  - storage,
  - presentation
The phases of the MP (depicted as rectangles in Figure 1) are now detailed. They can be classified in development phases (requirements definition, design, implementation) and supporting phases (configuration management, project management, quality assurance).

Requirements definition
The requirements definition phase, in the case of an MP, is the definition of what should be measured.
In order to define what should be measured, goals have to be stated. To define measures, a descriptive process model has to be defined. Therefore requirement definition is decomposed into the subphases definition of goals, process modeling, definition of questions and measures. The goals stated initially influence both the definition of the process model and of measures. Useful measures
are adopted for decision taking, therefore the last subphase associates actions on the process to thresholds reached by specific measures.

**Definition of goals**

In this subactivity the goals of the measurement program are defined. They usually mirror the overall quality goals of the company and of the specific project.

**Definition of process model**

First, a descriptive process model is defined. If a prescriptive model exists, it could be used as a starting point, but the ideal approach is to build a brand new model as a snapshot of the current process, with no risk of bias from the prescriptive model.

The model typically describes phases, roles, documents, and their relationship. Since the goal of the model is measurement and not enaction, the dynamic part of the model is very limited, if any, while the data part of the model (entities and relationships in the software process) is dominant. Given these conditions, a data oriented formalism is particularly suitable.

In applying the method the OMT (Rumbaugh 1991) formalism was chosen for the following reasons:

- it allows the OO approach and measurement to be merged, reusing OO concepts and tools in measurement practice
- it supports data oriented primitives (class, object, association, aggregation, generalization)
- it also supports powerful dynamic oriented primitives (hierarchical state machines inside classes, and event exchange among classes), and dataflow primitives.
- it is one of the leading OO formalisms, supported by a variety of tools.

Nevertheless, the focal point of our method is the use of an object oriented process model, and not the use of OMT specifically.

The PSP0 and its model can be described briefly as follows (see Figure 2). PSP0 defines process phases (class **Phase**), products (class **Product**) and resources (class **Resource**). This corresponds to Fenton’s framework, except that the term Phase is used instead of Process. A Phase is characterised by the effort the developer dedicated to it (attribute **time**) and has a detailed definition (attribute **definition**). For instance the definition of the planning phase is “produce or obtain a requirements statement, estimate the required development time, enter the plan data on the Project Plan Summary form, complete the Time Recording Log”.

A phase can be planning, developing, post-mortem (classes **Planning**, **Developing**, **Post-mortem**), developing can be design, coding, compile and test. These relationships are modeled by the specialisation association of OMT (depicted by a triangle). The attributes **time** and **definition** are inherited by each specialisation of **Phase**.

Products (class **Product** and its specialisations) of the PSP0 can be Source code (produced in the Code phase), Time recording log, Defect recording log, Project plan summary. Time recording log contains a row for each interval of time passed in phase. Defect recording log contains a row for each defect found. This is modeled by the relationship **Defect recording log is-an-aggregation-of** **Defect** (depicted by a diamond in OMT). A defect is characterised by date, time to fix, type,
phase in which it was injected, phase in which it was removed. The latter two are modeled by the associations **Defect removed-in Phase**, **Defect injected-in Phase**.

The unique resource (class **Resource**) is the programmer.

An actual, running process is modeled by a set of instances of **Phase**, **Product**, and one instance of **Programmer**. A row in the time recording log corresponds to an instance of class **Phase**, the value of effort is the interval of time spent in the phase. An instance of **Defect** is connected to two instances of **Phase**. Since associations are also inherited, this means that any instance of a specialisation of **Phase** inherits the association with zero or more defects.

As the model is oriented to measurement, it contains only classes, attributes and associations relevant for measurement. For instance, process scripts are not modeled.

**Definition of questions and measures**

In this subphase the goals are detailed as questions and measures. This is the well-known Goal Question Metric approach (Basili and Rombach 1988).

<table>
<thead>
<tr>
<th>Goal</th>
<th>Question</th>
<th>Measure name</th>
<th>Measure definition (OMT script)</th>
<th>Measure definition (SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>productivity?</td>
<td>size</td>
<td>( \Sigma (\text{SourceCode.size}) ) [loc]</td>
<td>SELECT SUM(size) FROM SourceCode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>( \Sigma (\text{Phase.time}) ) [minute]</td>
<td>SELECT SUM(time) FROM Phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>productivity</td>
<td>size/time [loc/hour]</td>
<td></td>
</tr>
<tr>
<td>What is</td>
<td>quality?</td>
<td>defect</td>
<td>Cardinality(Defect) [defect]</td>
<td>SELECT COUNT(*) FROM Defect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defect density</td>
<td>defect/size [defect/loc]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>defect injected</td>
<td>Cardinality(((Defect.injected-in).Phase = Design) ) [defect]</td>
<td>SELECT COUNT(*) FROM PHASE, DEFECT WHERE Phase.PhId=Defect.InjPhId AND Phase.type='Design'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defect removed in</td>
<td>Cardinality(((Defect.removed-in).Phase = Compile) ) [defect]</td>
<td>SELECT COUNT(*) FROM PHASE, DEFECT WHERE Phase.PhId=Defect.RemPhId AND Phase.type='Compile'</td>
</tr>
</tbody>
</table>

**Table 1: Goals, questions and measures for the PSP0.**
Table 1 shows goals, questions, and measures for the PSP0. Goals and questions are based on our reconstruction.

Measures are defined as OMT scripts on the process model (third column of Table 1), and involve one or more attributes of a class (in OMT terminology, entity in Fenton’s terminology) or of several classes linked by associations. In order to make it easier to understand, column 4 of Table 1 contains the equivalent SQL (Date 1995) definitions.

Considering OMT definitions, the measure size sums the size attribute of all instances of class SourceCode; the measure time sums the time attribute of all instances of class Phase. (This includes all the instances of any specialisation class); the measure defect counts the instances of class Defect; the measure defect injected in Design starts from all instances of Defect, it selects all the instances of Phase which are related by the association injected-in, restricts to instances of Design, and then counts them.

For SQL definitions, it is assumed that the OMT model is translated into the relations SourceCode, Defect and Phase, adding the identifiers SrcId, DefId and PhId, respectively. The specialisations of Phase are represented by adding the field type to relation Phase, the associations injected in and removed in are represented by adding the fields RemPhId and InjPhId to the relation Defect. This translation is one of many possible ones, and was chosen to simplify the SQL definition of measures.

Given a process model and scripts containing the definition of measures, it is a straightforward matter to map them to Fenton’s framework. An internal attribute is an attribute in a class (ex. SourceCode.size), an external attribute is an OMT’s association attribute (i.e. an attribute attached to an association)(ex. productivity), a direct measure involves one attribute on one class, an indirect measure involves many attributes, defined in the same class/association, or in several classes/associations (ex. productivity, defect injected in phase); if the attributes are in different classes, the classes should be linked by associations (ex. productivity, defect injected in phase). As a class in OMT can have a single copy of an attribute, measures with absolute scales always involve classes and not attributes (ex. defect).

A predictive measure is an indirect measure of an attribute which does not exist yet (ex. productivity, before starting the process). Predictive measures cannot be recognized on the OMT model, unless they are explicitly tagged, thus requiring an extension to OMT.

Definition of thresholds and actions

The measures defined above are used to track and control the software process. In this phase a feedback control loop on the process is defined for each measure. This is accomplished by defining a lower and upper threshold, or a range of acceptable values for each measure. Then an action should be agreed on if the measure is outside the threshold. For instance if defect density on a module is over the upper threshold, the module could be redesigned, if it is below the lower threshold, scheduled testing effort could be reduced.
This phase provides early validation of the measures defined. If it is difficult to identify actions on the software process because of a measure outside the threshold, it is an indication of a useless or ill-defined measure.

**Requirements verification and validation**

The first verification and validation (v&v) phase to be performed on requirements is the validation of goals. It has to be done jointly by senior management, the project manager (those who are responsible for establishing the goals), the quality manager and the measurement manager (those who are able to judge whether the goals are reasonable and measurable). The importance of this validation is clear, for unsuitable goals would lead to the failure of the measurement program. Then the decomposition of goals into measures has to be verified for internal coherency. Last, the validity of measures has to be verified (Fenton 1995, Kitchenham et al. 1995). If measures are taken from the literature, proofs (or counterproofs) of validity should be found in the literature too. If predictive measures, and the related models, are used in the project, a validation phase, as defined in (Schneidewind 1992), should be initiated. Indeed this phase involves several projects and is part of the organisation level MP. The definition of direct measures on a process model is in itself a preventive validation that avoids misunderstandings and omissions. The definition of indirect measures using a formal language (such as OMT scripts, or SQL, or an object oriented SQL) is another preventive validation, and facilitates communication.

**Design**

So far measures have been defined, but there are many possible ways of obtaining and presenting them are possible. The design phase is dedicated to deciding how measures should be obtained, that is to say selecting a way of collecting, storing and presenting them. This impacts tools, organisation and people. Since people are involved, human factors affecting the measurement process should be considered too.

Designing the MP could require changes to the software process too. This kind of change is not considered here, as such changes are the responsibility of the Meta Software Process. As in the software process, also in the MP it is important to keep the what and the how separate. Changes in requirements (i.e. in goals and measures) involve changes in design, risk of past measures becoming obsolete, and they should be limited. Changes in design do not imply changes in requirements and are less disruptive. As usual, design choices should favour robustness against changes.

**Collection**

For each direct measure, the measurement manager decides whether it should be collected manually or automatically. In the former case (typically process measures such as Phase.time and Defect) the procedure for collection (who is responsible for collection, when, how frequently), and the means for collection (screen forms, paper forms) should be designed. In the latter case (typically product measures such as SourceCode.size) a tool has to be selected and possibly customized.
For each direct measure the measurement manager has to decide a suitable technique to validate the data collected, who validates it and when. Basili and Weiss (1984) propose, for measures collected manually, the use of interviews with the measurement manager, as close as possible to the instant of collection. The same paper also sets out guidelines for form design.

For each indirect measure it should be decided who, when and how it will be computed. Typically, direct measures are stored in a database, indirect measures are computed automatically using a query language. In this case the design choices are related to the translation of indirect measures into the query language. In the next section it will be shown that this translation can be automated.

For example in the PSP0, the direct measures \textbf{Phase.time} and \textbf{Phase.type} are collected manually, by the engineer, who register on the Time recording log paper sheet, each time he or she is interrupted. The indirect measure \textit{time} is computed manually, at the end of the project, using the Project Plan Summary paper sheet. In both cases a different design choice could involve using tools to automate the collection and elaboration of measures.

\textit{Storage}

Although direct measures can be stored in paper and file form, a database is the typical choice for industrial level MPs. When a database has been chosen, the schema for measures has to be defined, an issue discussed in (Fenton and Pfleeger 1997, chapter 5.5).

Interestingly, if an MPM has been defined, it becomes, with very few modifications, if any, the conceptual schema of the database, and if the database is object oriented, the schema is derived automatically by the MPM (Aarsten and Morisio 1996). If the database is relational, the schema can be derived semi-automatically.

Queries to compute indirect measures can be produced automatically, too. OMT scripts can be automatically translated into C++ (Aarsten and Morisio 1996), and similar translations could be defined for SQL.

It could be argued that the process could be modeled in relations, and measures defined in SQL directly. The MP-Software process analogy reveals that this would correspond to skipping the requirements phase, but this would result in the lower readability of a model without relationships when compared with an OMT model.

In PSP0 measures are stored on the paper forms used to collect and analyze them. The PSP community has already produced standard files readable from spreadsheets or database tools.

\textit{Presentation}

Presentation deals with deciding the most suitable way measures can be presented. A graphic form (plots, box-plots, scatter-plots, pie charts, bar charts) is usually preferred to numbers and is the key to transform a measure into a usable tool. Presentation decisions determine how the structure, contents and presentation form (paper, web site, etc) of one or more measurement reports are designed; accessibility to data is a function of role (individual measures should not be published, not all measures are of interest to all roles), reporting frequency.
In PSP0 there is no formal prescription of how measures should be presented. Nevertheless, throughout the book (Humphrey 1995), measures are presented as plots of a measure against a program number (the program can be assimilated to a project) for individual engineers, and averages for classes of engineers. These plots are obtained on a spreadsheet. Scatterplots are also used to analyze the relationship between pairs of measures.

**Human factors**

Measurement, as well as software production, is a human intensive activity, so human factors are fundamental and should be considered early on in the MP.

There are three human roles involved in an MP, developers, managers and the measurement manager. While the MP is the work of the measurement manager, it should not hinder the work of developers, and it should help the work of managers.

Defensively, factors to be considered to avoid useless noise in the work of developers are the design of forms and procedures to collect manual, direct measures. Instead, on a positive point of view, the MP should be perceived as a help in their work. This requires training in concepts and goals of the MP, the involvement of developers in MP design. It also requires an assurance that the productivity and quality measures will not be used to monitor individual performance, and that measurement reports are readily available.

As regards managers, the risk is that they do not use measurement reports for decision making. In order to minimize this risk, they should be involved in the MP (basically in the definition of goals), they should also receive timely measurement reports, they should be able to refine the goals until they actually fit their needs.

When these roles have been considered, channels to reach them, (training, diffusion of reports and documents, newsletter, discussion group, etc..), the material to be diffused, and the diffusion means (web server, paper, classes, meetings) have to be decided.

**Design verification**

The main design choices should be verified to see that they are consistent with the requirements (schema of the database and computation of indirect measures), to see that there is internal consistency and completeness (presence of a validation phase during data collection), and that sound MP design (ergonomicity of forms and sheets, suitability of training and motivation, readability of measurement reports design) is ensured.

**Implementation**

Tools are bought, installed and customised, training and staff motivation is provided, procedures are put in place, organisational roles are assigned, the database is implemented, and presentation procedures are prepared. At the end of this phase all products should be verified against design and requirements. The best validation is achieved running the MP on a limited size pilot project.

In PSP0, implementation corresponds to reading the book, understanding the PSP0 and preparing copies of the forms and sheets to be used.
Configuration management, project management

When it comes to configuration management and project management, the MP can be treated in the same way as any other software project.

From the project management point of view, the MP is a full scale project, with its planning, resource allocation, tracking and replanning. In particular a few (meta)measures (effort, duration) are essential to characterize the MP. From the configuration management point of view, a number of documents (the measurement plan, the database, forms, procedures) should undergo configuration management.

Quality assurance

Assuring the quality of a MP means assuring that the measures collected are valid (i.e. they satisfy the representation condition) and useful (they are used for decision making).

Before the MP is operated, most phases in the method (definition of the process model, use of Fenton’s framework, verifications and validations after requirements definition, design and implementation) are performed to assure, a priori, the selection or definition of valid measures and this can be classified under the heading quality assurance. The same holds for the GQM approach, which is used to assure that the measures collected are useful.

During operation of the MP, quality assurance consists of validating the data collected (using the technique chosen in MP design) and repeating at regular intervals the phase Requirements verification and validation. Moreover, measures and defects regarding the MP are collected and analysed.

Since we are dealing with a measurement process, it is obvious that measures should be used to assess its quality, or metameasures (Hetzel 1993). Basili and Weiss (1984) report on the ante-litteram use of metameasures (e.g. the number of misclassified errors or changes in a change measurement process) to tune the process itself.

Table 2 contains the metameasures which have been defined by using the GQM approach. In order to define them, the PM of figure 2 has to be extended as follows. A class Measurement is added as a specialization of Phase to model all phases part of the MP (Requirements definition, Design, etc..). Three values are added as possible values of Defect.type: missing (defined as measure added after start of operation of MP), invalid (measure changed/abandoned after start of operation of MP), useless (measure never used for decision
### Table 2: Goals questions and measures for the MP

<table>
<thead>
<tr>
<th>Goal</th>
<th>Question</th>
<th>Measure name</th>
<th>Measure definition (OMT script) [unit]</th>
<th>Measure definition (SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assure quality of the MP</td>
<td>is the MP efficient ?</td>
<td>MP_time</td>
<td>$\sum(\text{Measurement.time})$ [minute]</td>
<td>SELECT SUM(time) FROM Phase WHERE Phase.type='Measurement'</td>
</tr>
<tr>
<td></td>
<td>are valid measures collected?</td>
<td>relative MP_time</td>
<td>MP_time/time Cardinality([Defect.type = invalid]) [defect]</td>
<td>SELECT COUNT(*) FROM DEFECT WHERE Defect.type='invalid'</td>
</tr>
<tr>
<td></td>
<td>are useful measures collected?</td>
<td>n invalid measures</td>
<td>Cardinality([Defect.type = missing]) [defect]</td>
<td>SELECT COUNT(*) FROM DEFECT WHERE Defect.type='missing'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n missing measures</td>
<td>Cardinality([Defect.type = useless]) [defect]</td>
<td>SELECT COUNT(*) FROM DEFECT WHERE Defect.type='useless'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n useless measures</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Operation

The MP is operated, as defined in requirements and design, in parallel with a project. It collects, stores and presents data which come from the project and produces, on a regular basis, measurement reports for the use of the developers and managers. Meanwhile, metameasures are collected, stored and presented too. In the PSP0, this means using the PSP0 on one of the exercises which come with the book.

### Products

The main products of the MP and their contents are listed below.

- **Requirements document**: goals, process model, definition of measures.
- **Design document**: design of paper/screen forms for data collection; schema of database; list of selected tools and their use; structure and presentation means of measurement reports; contents and schedule of training sessions.
- **Implementation**: installed tools, working database of measures, working procedures and tools for data collection; trained and motivated developers.
- **Operation**: constantly updated database of measures, measurement reports issued at regular intervals.
- **Quality assurance**: reports from verification and validation phases before operation, reports from requirements v&v during operation, MP defect reports, metameasures.
Roles and responsibilities

In general the roles of the MP are the measurement manager (MM) (the one who has the ultimate responsibility for the MP), managers (senior and project management), developers and the quality manager.

Requirements definition is the responsibility of the MM, except for the definition of goals, where senior and project managers are deeply involved, and are essential for the success of the MP. Developers are involved in data collection and are responsible for the precision and accuracy of the data collected, while the MM is responsible for validating it. Design and implementation are the responsibility of, and usually performed by, the MM. QA activities are the responsibility of the quality manager, and project management is the responsibility of the MM. Configuration management in fact is outside the MP.

When assigning roles to persons, it should be remembered that each of the above managing roles has a control function over at least another role. The quality manager controls all the activities performed by the MM, the MM controls the goals declared by senior and project management, the MM controls the data collected by the software engineers.

Lifecycle

The phases listed above and depicted in figure 1 do not imply a temporal sequence. Nevertheless, the ideal MP performs the phases Requirements definition, Design, Implementation, Operation in sequence. Concurrently, the phases Project management, Configuration management, Quality Assurance are performed. Depending on a major defect detected or a major change requested and approved, a new cycle of phases can start. Therefore the lifecycle can be considered to be cyclic-waterfall. At each cycle metameasures and other findings about the MP are used to improve it.

Using the analogy MP - software process, any other lifecycle could be reused and adapted to the MP. Research and experimentation should be performed to find the most suitable one, for each specific context.

3 EXPERIMENTAL VALIDATION

The method proposed in this paper has been experimentally validated by applying it to four industrial projects, each one offering a different industrial context. We present here the results obtained when the method was applied to the first project, the only one which is finished at the time of writing.

The AEFITA project (Morisio et al. 1996) developed a factory automation product which used object technology (OMT and C++), with a staff of seven during an 18 month period. It was part of the ESSI initiative of the European Community, and for this reason it was thoroughly measured using
the method presented in this paper. The context of the project is an ISO9001 certified company, which is usually situated between level 2 and level 3 of the CMM. In the following paragraphs the way in which the specific characteristics of the method influenced the MP during the experiment are discussed.

Definition of measures on a process model

The first reason for using a process model as the foundations with which to define measures is to clarify definitions, in the actual context of the project, to promote communication and avoid misinterpretation. In the AEFTA project the effort dedicated to the MP (this includes effort dedicated to the phases defined in section 2 by the measurement manager and the staff, but it does not include effort dedicated by senior management to define goals) was around 10% of the overall project effort. Of this 10% dedicated to the MP, the requirements definition phase took around 50%, and specifically the definition of the PM 33%, the definition of goals, questions and measures accounted for 17%.

For 3.3% of the whole project effort to be dedicated to the definition of the process model is a lot. Moreover three major iterations in the PM were performed, and dozens of fault reports on the PM were collected. These evolutive or corrective changes required changes, sometimes important changes, in the definition of measures. These measures confirm the intuitive idea that process modelling was by far the hardest part of the measurement process, due to the difficulty in understanding the process precisely.

This proves that there is a close and essential relationship between the process model and the definition of measures. Given an ISO9001 environment, the process was in fact defined, but at a generic, organisational level (not at a project level) and in natural language. What would happen if measures were defined using such an incomplete and ambiguous process model? We believe this would cause misunderstandings and ambiguities. These ambiguities might be discovered very late, or never. The effort dedicated to process modelling would later be used to fix ambiguous measures, or would be traded-off for less precise, or invalid measures.

Instead, most of faults on the PM and on definitions of measures were identified early (in the first four months of the project), by the staff of developers (not by the measurement manager). This fact confirms that the PM is an effective way to communicate and validate the measures. It should be mentioned that the process model used the same formalism, OMT, used by the staff on the project, thus favouring common understanding.

OO process model and evolution of the MP

“A good measurement program will always be changing” (Hetzel, 1993).
Given the measurement process - software process analogy, changes in the MP can be classified as evolutive, corrective, perfective; another classification axis is the phase to which the change applies (requirements, design, ..).

In the AEFTA project, as the MP was treated as a project, changes and faults were recorded and classified. The majority of faults were related to the process model (requirements), the majority of change requests were related to forms (design). Changes in the process model were corrective in the initial period, then evolutive.

As in the software process, changes in requirements (definitions of measures and process model) are, potentially, the most disruptive. In particular changes to the process model can influence all subsequent choices.

The use of oo concepts and technology is the main answer suggested by the method to the problem.

*Reification*

Whenever possible, entities in the process are described as classes or associations: for instance actions are objectified (as class *Phase* in the PSP example). The process model is mainly made up of the object model. The dynamic model is limited to some classes which have a rich or complex behaviour: for instance the why and when an anomaly report has to be issued. Moreover the dynamic model acts as documentation and is not yet used in the definition of measures. Classes and associations are much more stable against changes than dynamic models, thus limiting the necessity for changes on the PM and their impact on other phases of the MP.

*Generalisation/specialisation*

Adding or deleting classes or associations can have two kinds of effect. They can require changes in design and implementation, they can cause measures already taken and stored in the repository to become obsolete. The latter problem is tackled by using the generalisation/specialisation concept and defining a two level process model. One level, (abstract classes *Phase*, abstract associations *removed in, introduced in*) is designed to be very robust against changes. The other level, or concrete level (specialisations of the above classes and the related specialised associations), is designed to change (by adding and deleting classes and associations) to accommodate successive evolution. Measures are defined, whenever possible, at the abstract level (they are called abstract measures). Abstract measures are preserved, provided abstract classes and their relationships do not change, even if concrete classes change. On the contrary concrete measures can become incoherent if changes affect the concrete classes on which they are defined. Abstract measures have the same behavior of abstract classes, they do not change when their specialisations change, and they provide a mechanism to separate stable and unstable parts of a model. The advantage is that the evolution of a process can be tracked by changes to specialisation classes, while abstract measures do not change and preserve data collected in the past.

In the AEFTA project, after abstract measures had been introduced, it was not necessary to make any changes to abstract classes, while many specialisation classes were added. In the PSP example, *Phase* is a typical abstract class. When migrating to PSP1 and PSP2, specialisation classes are
added to **Phase** (for instance **Code Review, Design Review** of PSP2). No change is needed to PSP0 measures (such as time), while PSP2 measures (such as Appraisal to failure ratio) are added.

**Automatic translation**

OO technology today offers many CASE tools which are able to translate from an analysis model to a design model to an implementation language.

In the Aefta project the CASE tool was able to generate C++ code from an OMT design model, and the design model from an OMT analysis model. Using a C++ library to add persistence, the database for measures could be derived automatically from the process model, since the process model corresponds to an analysis model. Direct and indirect measures defined as OMT scripts were translated by hand as function members of the persistent classes (conceptually, this translation can be automated too).

These features eased the design and implementation phases of the MP, and reduced the effort necessary to propagate changes in requirements. The design and implementation phases accounted for only 25% of the MP effort because automatic translations reduce the design and implementation effort, while let the requirement effort become higher (50%). Nevertheless, the requirement effort remains high as compared with the overall project effort.

**Synergy between software process and measurement process**

The measurement process uses the same concepts and technology as the software process. In particular the same development tool and language (C++) are used. In addition to the evident advantage of saving resources (same hardware, same tools, same training on tools and OMT method), a definite advantage is that the same mindset for programming and measuring is shared. Process measures are collected mainly through the manual filling of sheets by the staff, and they rely heavily on the process model for the definition of terms and relationships. Since the process model is described using OMT, a staff member can understand it applying the same skill (s)he uses to understand analysis and design documents of the project. This skill improves with time and produces more reliable measures.

**4 CONCLUSION**

Our method is based on a strong analogy between the software process and the measurement process. The consequence of this analogy is that the starting point of a measurement process is the (measurement) process model. Last, the method uses object oriented concepts and tool.

From the process modelling point of view the process model provides the foundations with which the software process can be understood, improved and enacted. From the measurement point of view a model is needed to clarify the context in which entities, and the related measures, are defined. Our proposal merges the two approaches by using a process model, which is a snapshot of
the software process, as the foundations on which to define measures, and as a starting point to understand and (partially) automate the measurement process.

The main phases of the method are the definition of requirements and then the design, implementation, operation, quality assurance. The requirements definition phase produces a process model which has a set of measures defined on it. The design phase involves choices about how to collect, store and present measures. On the human factors side, design choices are concerned with how the active collaboration of people part of the measurement process can be obtained. The implementation phase puts in place design choices (training, the installation of tools, the implementation of the database and queries on it, modifications to procedures and roles in the software process) before operation. Configuration management, project management and quality assurance are performed in parallel. The quality assurance phase verifies the suitability of goals, of their decomposition into measures, the validity of the measures chosen or defined; it verifies the consistency of design transformations and choices with requirements; it also defines a number of metameasures to monitor and improve the measurement process itself.

We analyse here the advantages stemming from the three main points of the method.

**Measurement process - software process analogy.**

- Phases, roles, documents for the MP are clearly defined, helping managers to allocate the right resources to it. The staff involved in the MP can reuse the same mindset and know how coming from the familiar software development process.

- The different meanings of the *validity* concepts already introduced in measurement literature can be specified, and new ones can be introduced. This is achieved by defining the validation activities to be performed on products of the project level and organisation level measurement process. At the project level: requirements validation (use of valid measures in the sense of Fenton (1995) and Kitchenham et al. (1995), use of measures corresponding to goals in the sense of GQM, use of a valid process model as a foundation on which to build valid measures, use of valid goals), design validation (validation of database schema and queries, validation of effectiveness of forms and reports), implementation validation (consistency of implemented process and required process), quality assurance (verification and validation activities above, validation of data collected (in the sense of Basili and Weiss (1984)), definition and use of metameasures).

  At the organisation level: validation of predictive models in the sense of Schneidewind (1992).

**Use of a process model.**

- Both relevant concepts and terms of the software process and measures are clearly defined, communicated and agreed on by people designing the MP, and by people applying it. This avoids misunderstandings in the interpretation of measures and helps to collect them reliably.

- The characteristics of measures and measured attributes (direct/indirect, internal/external) can be recognized on the process model. This is an advantage for understanding and validating measures.

**Use of OO concepts and tools.**
• The schema of the database for storing measures is derived with few modifications from the process model, queries to compute indirect measures can be derived automatically from their definitions on the process model. This helps in reducing sharply the effort to implement the MP. Also, modifications can be propagated more easily, starting from higher level documents instead than code.

• Reification can be used to model events and activities as classes. Abstract measures, the analogous of abstract classes for measures, can be defined. Both mechanisms provide a powerful means to isolate stable from unstable parts of a model, which makes it more robust against changes. This is a key advantage, as the software process and the measurement process change continuously during the life of projects and companies.

• Object oriented concepts and tools are more and more diffused in software development. If they are applied to measurement too, both the mindset of people, and tools can be reused, providing better quality of measures and savings.

The method has been and is being applied to a number of industrial processes. We interpret here the empirical evidence we are collecting in function of the characteristics of the method.

• the effort for definition of requirements, and process modelling in particular, is high; intuitively, this phase is considered to be the hardest of all, while the definition of measures, given the goals and the process model, is much easier. We interpret this fact as a confirmation that process modelling is essential for measuring. Conversely, skipping process modelling could produce the need for high effort for modifications later, or poor measures.

• change requests and fault reports dealing with the measurement process are concentrated in the early phases (requirements, design, initial operation), and are produced by software engineers. This means that the process reaches stability earlier. We think that this is because the process model used, and the precise vocabulary it defines, makes it easier to understand and communicate the definitions of measures, and it also allows software engineers to participate in the validation process. Software engineers can understand the measurement process better, as it is represented reusing concepts, formalisms and tools similar to the more familiar software development process.

• the effort for implementation is low. This is due partly to the high effort in the upper phases, and partly to the automation of certain tasks (definition of the database schema and of queries on it), that depends on the use of oo tools and techniques.

ACKNOWLEDGEMENTS

The author wishes to thank Walter Ellis, Giuseppe La Commare, Maria Letizia Jaccheri, Rosa Meo and Dieter Welzel for their suggestions and encouragements.

REFERENCES
Aarsten, A., and Morisio, M., Using Object Oriented Technology to Measure the Software Process, 
Proceedings AQuIS96 Achievieng Quality in Software, Florence, Italy, January 1996, 
http://www.cim.polito.it/Articles.html/aquis96.ps.


Bandinelli, S., Fuggetta, A., Ghézzi, C., and Lavazza L., SPADE: An Environment for Software 
Process Analysis, Design and Enactment, in Software Process Modelling and Technology, 

Basili, V.R., and Weiss, M.W., A methodology for collecting valid software engineering data, IEEE 

Basili, V.R., and Rombach, H.D., The TAME Project: Towards Improvement-Oriented Software 

Basili, V.R., Applying the GQM paradigm in the experience factory, in Software Quality 
Assurance and Measurement, (N. Fenton, R. Whitty and Y. Iizuka eds.), Thomson 

2, University of Kaiserlautern, TR 265/95, February 1995.

Capra, M., Escala, D., and Morisio, M., Adapting the PSP to an SME context, in Proceedings of 

Carleton, A.D., Park, R.E., and Goethert, W.B., Software Measurement for DoD Systems: 
Recommendation for Initial Core Measures, Software Engineering Institute, Pittsburgh, PA, 

Cook, J. E., and Wolf, A.L., Towards Metrics for Process Validation, 3rd Int. Conf. on the 


De Champeaux, D., Lea, D., and Faure, P., Object Oriented Systems Development, Addison- 
Wesley, 1993.

Feiler, P.H., and Humphrey, W.S., Software Process Development and Enactment: Concepts and 


Fischer, K., and McCalla, L.F., A Model for Implementing Quantitative Process Management, 

Florac, W., Software Quality Measurement: A Framework for Counting Problems and 
Defects, Software Engineering Institute, Pittsburgh, PA, CMU/SEI-92-TR-22, September 

Staff-Hours and Reporting Schedule Information, Software Engineering Institute, Pittsburgh, 


**BIOGRAPHY**

*Maurizio Morisio* is a research assistant at Politecnico di Torino. His research interests include experimental software engineering and software metrics, software reuse, CASE tools evaluation and selection, software process modelling, requirements specification and design formalisms. He serves as measurement manager for the Esprit/ESSI projects Aefta, Moods and PSP-NC. He received his Ph.D. in software engineering from Politecnico di Torino.