Journal of Universal Computer Science, vol. 16, no. 21 (2010), 3210-3244
submitted: 14/4/09, accepted: 24/11/10, appeared: 1/12/10 © J.UCS

Refinement and Extension of SMDM, a Method for Defining Valid Measures

Luis Reynoso
(Facultad de Informática, University of Comahue, Neuquén, Argentina
lreynoso@uncoma.edu.ar)

Marcela Genero and Mario Piattini
(Alarcos Research Group, University of Castilla-La Mancha, Ciudad Real, Spain
{Marcela.Genero, Mario.Piattini}@uclm.es)

Abstract: Although literature contains a huge amount of measures for measuring quality characteristics of software artifacts throughout the development life-cycle, the majority go no further than the step of definition. The key to obtaining valid measures which may be useful in practice is to carry out a definition of these by following a rigorous method. In a previous work we defined a method for obtaining valid measures, called SMDM (Software Measure Definition Method). In this paper we present the extensions and refinements of this method, which has been redefined in the light of seven years of application to various software artifacts, such as OCL expressions, UML diagrams, ER diagrams, Relational database schemas, Datawarehouse conceptual models, etc. In order to illustrate the redefined method, an example of the definition of a measure for the import-coupling of OCL expressions is presented.

Keywords: software measures, measure definition, theoretical validation, empirical validation, psychological explanation
Categories: D.2.8

1 Introduction

One of the current concerns of software factories is that of evaluating and improving the quality of their software products throughout the development life-cycle. To do this, they need valid software measures which will allow them to assess the quality characteristics of software products in an objective and quantitative manner. Over the past fifteen years we have witnessed the appearance of a wealth of literature dealing with measures which capture the quality of many software artifacts [Henderson Sellers, 1996; Lorenz and Kidd, 1994; Chidamber and Kemerer, 1994; among others]. However, most of the measures are poorly defined, and have consequently been of little use or indeed of no use at all, in achieving the purpose for which they were intended [Briand et al. 2000]. In our opinion, many of these problems arise from not having followed a clearly defined method for establishing the measures, one which would take into account factors which have to be fulfilled if the measurements are to be valid. We outline these desirable issues at a later point in the paper. The lack of a consolidated method with which to measure definition could be considered as characteristic of a discipline like software engineering, which is a human-intensive, young discipline in contrast to other disciplines, whose methods and techniques need to be fully assessed [Briand et al. 2000].

Some issues that contribute towards obtaining valid measures are [Briand et al. 2000]:

- Measures must help to attain a measurement goal. Measurement goals should be clearly connected to an industrial goal, responding to the software organization's needs.
- The underlying hypotheses associated with the measures once they have been defined should be explicit.
- The context or the environment in which the measures can be applied should be declared.
- Measures should be repeatable, i.e. their definition should be as clear as possible, so that if the measurement of an attribute is repeated by a different person, the same result will always be produced.
- Measures should be theoretically valid, i.e. the particular attribute that a measure aims to quantify should be explicit.
- Measures must be valid in practice, i.e. they must be empirically valid.

Although there were isolated proposals taking onboard those factors which would be desirable in defining measurements, there was, nonetheless, no comprehensive and detailed method which included them all. To fill that gap, in [Calero et al. 2001a] we described the first attempt to define a method for measure definition which integrated those existing proposals in the literature that covered each of the relevant issues mentioned above. We called this method SMDM (Software Measure Definition Method). In a later explanation of the method we will set out in detail which of the already-existing proposals were borne in mind in each one of the stages.

Goal Question Metric (GQM) is the most widely-used method in the organization of software measurement programs. By using GQM, measurement goals are defined and each goal is refined into specific questions, whereas metrics are defined to provide the information to answer these questions. Thus, by using GQM, metrics are defined from a top-down perspective and analyzed and interpreted bottom-up. The initial ideas of GQM were first published in 1984 and, since then, much industrial experience has been gained [Solingen and Berghout 2001]. Two important considerations about GQM are those provided by Briand et al. [Briand et al. 2002] and Card [Card 1993]:

- Briand et al. argue that to define sound and useful software measures an intellectual process is needed and that this has to be supported by a solid theory which makes it possible to review and refine them [Briand et al. 2002]. That being the opinion they hold, their approach, the GQM/MEDEA, draws many ideas from the theory for experimental design and from empirical studies in general. The main contribution of GQM/MEDEA to GQM is the definition of an organized process for the definition of software product measures based on GQM goals. The authors highlight one important difference with from GQM: the way the gap between GQM goals and measures is filled. The GQM fills it by means of questions and models, whereas "GQM/MEDEA does not rely on questions to fill that gap, but it defines the steps to carry out the relationships among them, the sources of information, and shows the integrated use of hypotheses, abstractions, and theoretical and empirical validation" [Briand et al. 2002]. They also highlight that the object of study (defining the object or process being measured) defined in the GQM goal template may need to be studied at a finer granularity level.

- Card, on the other hand, argues that a complete and effective strategy for defining a software measurement system must include: (1) *Goals*, (2) *Models* of the process and product to be measured (models identify what can be measured and how the measures relate to one another); (3) *Metrics* (they give no indication of "goodness." They are counts or other measures of the things in models); (4) *Indicators*, i.e. a metric combined with some expectation of what its value should be, based on plans or experience (they can help determine if goals are being satisfied and so give some idea of "goodness"); (5) *Checkpoints*. At checkpoints indicators are examined; (6) *Evaluation* of the measurement system (i.e. whether the measurement program has achieved its goals and on what the utility of individual metrics and indicators is). He therefore suggests supplementing GQM with modeling, in order to select specific practical measures. Developing a model of the object of study makes it possible to select measures on the grounds of how effective they are rather than on how desirable they may be. Models that identify measurable things and describe the relationships among them are essential to measurement interpretation.

During the last decade, a great deal of theory has been developed to underpin a method for measure definition. For example, our method takes into account Briand's idea of having a finer granularity level of the object of study, which is similar to Card's proposal of using a model of the object of study. We believe that there is no sense in modeling the object of study from scratch, when the software community is in the process of accepting standard metamodels (like the one defined by OMG) of many software artifacts. So, our method suggests selecting a metamodel of the object of study, as well as defining the measure using that metamodel. In this article, in the section where the steps of the method are dealt with, we will explain the proposals which had been considered.

From 2001, SMDM has been used, both by our research group and others, to define measures for different software artifacts, such as UML diagrams [Genero et al. 2007; Cruz-Lemus et al. 2005], ER diagrams [Genero et al. 2008], Relational database schemas [Calero et al. 2001b], datawarehouse conceptual models [Serrano et al. 2004], etc. During the application of the original SMDM to different contexts, we realized that, taking it as it was originally defined, the method was difficult to follow and sometimes ambiguous. The level of detail of the tasks to be done in each step was not high enough. For example, the step for measure definition requires more details in order to differentiate high level activities from those of a lower level if it is to be easily understood and used.

Therefore, our experience of using the original version of SMDM in different contexts motivated us to improve it through refinement and extension. The main goal of this paper is to give a thorough description of how this method has in fact been extended and refined.

The paper is organized as follows: The original version of SMDM, proposed in [Calero et al. 2001a], is presented in Section 2. Section 3 outlines the main issues that have been refined and extended in the current version of the SMDM. Sections 4 and 5 give an in-depth description of the new version of SMDM, emphasizing the activities that have been refined or extended to illustrate the entire method. An example of the definition of a measure for the import-coupling of OCL expressions is presented in Section 6. Concluding remarks and future work are presented in Section 7.

2 Summary of the Original Method for Measure Definition

An overall picture of the original method [Calero et al. 2001a] is shown in Figure 1. This method identifies four steps:

- Metrics definition. This step stresses the importance of defining measures correctly. It highlights the importance of considering the characteristics of the software products, the use of standards to identify quality attributes, along with the experience of modelers, designers, developers and product users. This step also proposes the use of the Goal Question Metric (GQM) Approach [Basili and Weiss 1984; Basili and Rombach 1998; Solingen and Berghout 1999] to obtain measures in a methodological manner. More details about the use of the GQM approach are provided in section 4.
- Theoretical validation. In this step it is investigated whether a measure really measures the attribute it purports to measure. Two main tendencies are identified: frameworks based on axiomatic approaches and those based on the measurement theory. This step is further explained in section 5.
- Empirical validation. This is carried out to gather empirical evidence of the measures in practice. Two major strategies are described in [Calero et al. 2001a]: experimentation and case studies. These strategies are explained and also compared. More attention is paid to experimentation and the need to run replicas than to case studies. The most relevant phases of running an experiment are also briefly described, along with some of the threats related to its validity.
- Psychological explanation. The purpose of this step in the original method was to explain the influence of the values of the measures from a psychological point of view. It also highlights the use of cognitive psychology as a reference theory in studying information modeling, as well as the use of the knowledge of human information processing to establish a threshold. However, this step was not explained in detail.

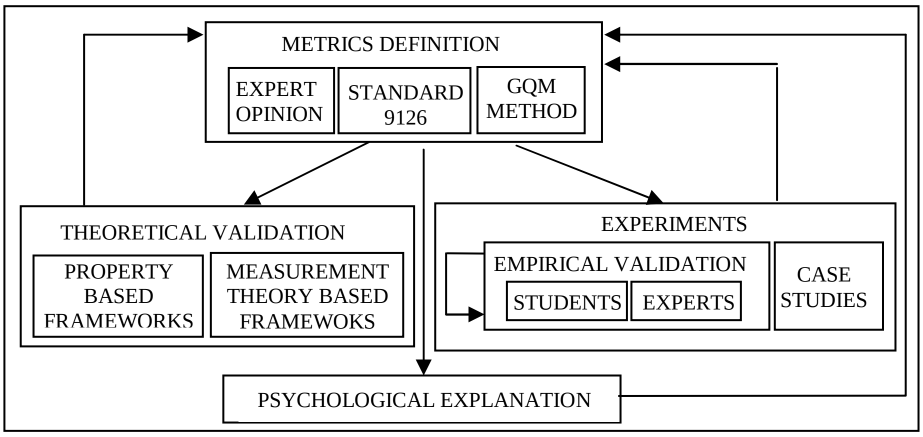


Figure 1: Original method for measure definition [Calero et al. 2001.]

3 Extended and Refined Method for Measures Definition

The original SMDM was refined and extended, due to the fact that we needed a fine-grained method which would provide us with a sufficient level of detail, something that was lacking in the original method, and which would help us to avoid ambiguities in its application and to tackle the measure definition accordingly. Moreover, the application of the original method in different contexts led us to realize not only that many parts should be refined, but also that new steps should be added. The method was therefore divided into several activities which should be performed in order to obtain reliable and consistent measures. The new method was modeled by means of UML activity diagrams.

The following modifications have been carried out on the original SMDM:

- Refinements represent improvements to the method with regard to the activities that should be carried out to attain a particular goal, with a rigorous specification of: (1) the order of execution of the activities, (2) the main data (object flow and major decisions) that should be defined and shared between activities, etc. Some of these refinements were originally conceptualized and designed but were not detailed and explained in relation to other activities.
- Extensions of the method were introduced when we needed to deal with new activities in the measure definition step.

Figure 2 shows the new method's high-level activities. One of the most important decisions taken when we redefined the method was to differentiate between and highlight two initial activities, which were Identification and the Creation of measures. The **Identification** activity (Figure 2, Activity M_1), has the purpose of planning the measurement goals and questions, identifying abstractions and stating general hypotheses by following the most commonly cited methods in literature for measure definition. The latter activity, on the other hand, which is based on the outcomes of the former activity, defines the measures through a rigorous process.

The Identification activity subsumed part of the first step of the original method (that of the definition of measures), but it was only the part which was related to the definition of goal and questions that followed a GQM approach. Moreover, the Identification activity was refined, in order to represent the remaining activities (the identification of abstractions, the stating of a general hypothesis, etc) which were not included in the original method. This activity not only considers a GQM approach, as did the original method, but also makes improvements to it considering the Measurement Model Life Cycle (MMLC) [Cantone and Donzelli 1999] and the GQM/MEDEA [Briand et al. 2002]. The Identification activity was refined because it is a crucial activity and all the ensuing activities will be based upon its results. The Identification activity in Figure 2 shows a rake to the right of the activity, indicating that the activity is described by a more finely detailed activity diagram. This activity is explained in detail in Section 4.

The **Creation** activity (Figure 2, Activity M_2) defines the measures based on clear measurement goals, questions, abstractions and general hypotheses specified in the previous activity. The definition is described in natural language first and also includes a formal definition. Moreover, measures are theoretically and empirically validated, and a plausible psychological explanation of the effort of subjects when dealing with the software artifacts being measured is provided. This activity has, therefore, subsumed part of the definition of the measure (that related to the measure creation within the GQM approach), the psychological explanation and the theoretical and empirical validation of the original method. Since this activity is probably the longest and the most complex, we believe that it should be further broken down into four activities, which are explained in section 5.

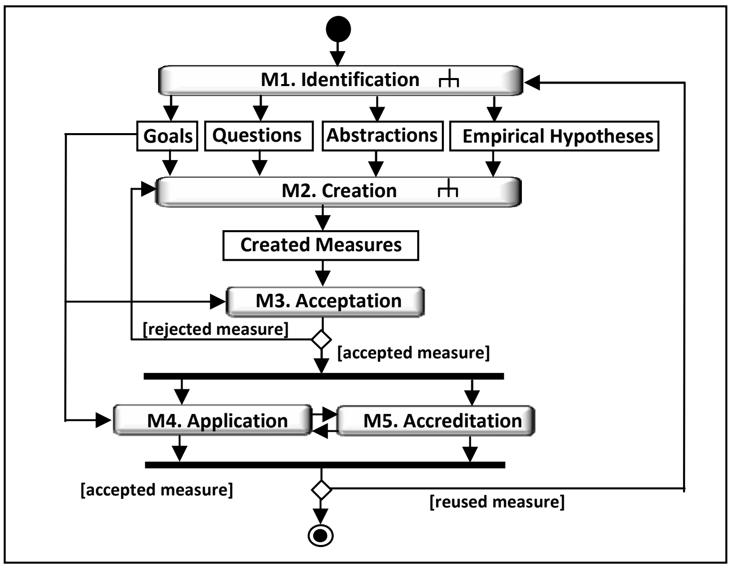


Figure 2: Refined and extended method for measure definition.

The method was also extended with three new activities:

- **Acceptation** (Figure 2, Activity M_3): The aim of this activity is the systematic experimentation of the measure. This is applied to a context which is suitable for the reproduction of the characteristics of the application environment, with real projects and real users, to verify its performance in comparison with the initial goals and stated requirements. After this activity is performed, measures can be accepted or rejected. A decision node therefore follows the Acceptation activity. The branching is based on whether measures are accepted or rejected. Even if the measure is rejected it ought not to be discarded, but should undergo the method from the creation activity.
- **Application** (Figure 2, Activity M_4): The accepted measure is used in real projects in industrial environments.
- **Accreditation** (Figure 2, Activity M_5): The goal of this activity is the maintenance of the measure, to allow it to be adapted to application in changing environments. As the original method explains, the accreditation activity represents a dynamic step that proceeds simultaneously with the application activity. A fork and a join bar in the diagram shown in Figure 2 therefore denote the beginning and the end of a parallel activity, respectively. As a result of this step, the measure can be withdrawn or reused for a new measure definition process.

We shall now provide a detailed description of the core activities of Identification and Creation (sections 4 and 5 respectively), stressing those aspects which were refined and extended. Whenever an activity is explained, its identification number, which coincides with the number shown in the UML activity diagrams, will be shown on its right.

4 Identification

As was described previously, the most important activity is that of Identification (Figure 2, Activity M_1), since it influences all other activities. The UML activity diagram for the Identification activity is shown in Figure 3. It is advisable to be able to achieve the definition of clear measurement goals, to avoid producing a measure definition that does not actually achieve our desired aim, i.e. we should follow a goal-oriented definition of measures. As is described in [Pfleeger et al. 1997], a commonly-used approach which can guide us in deriving and applying a goal-oriented definition of measures is the GQM approach [Basili and Weiss 1984; Basili and Rombach 1998; Solingen and Berghout 1999]. This approach (already explained in [Calero et al. 2001]) has been widely applied as a means of deducing measures by using a top-down perspective and of analyzing and interpreting them by using a bottom-up perspective. Goals are stated in a conceptual level and are, in turn, refined in an operational, tractable manner into a set of quantified questions [Mendonça and Basili 2000], defined in this Identification activity. In spite of the great usefulness of GQM, Card [Card 1993] recommended that GQM must be supplemented with another activity, to select specific practical measures, and he also suggests that one effective supplemental activity is that of modeling. Developing a model, i.e. defining the objects being measured, makes it possible to select measures on the grounds of how effective they are rather than on how desirable they may be [Card 1993], and helps us to describe the relationships between measurable things. Briand et al. similarly provide another mechanism with which to generate models [Briand et al. 2002]. In both approaches, the modeling of the measured artifact is included as a complementary activity for the GQM paradigm. After applying our original method for defining measures to various software artifacts, we found similarities with all the aforementioned remarks from measurement literature. Thus, in order to define measurement goals, in the new version of SMDM we propose the following activities:

- **Select the entity of study** (Figure 3, Activity I_1): According to the ISO 9126 [ISO IEC 2001] an entity is an object (for instance, a product, process, project or resource) that is to be characterized by measuring its attributes. The selected entity is undoubtedly the product of the organization's stakeholders' eliciting requirements.
- **Determine the quality focus** (Figure 3, Activity I_2): Generally, the quality focus corresponds to the quality attributes (abstract properties of an entity) upon which the measurement activities are focused. In order to conceptualize and differentiate when determining such attributes, quality models, such as the ISO 9126 [ISO IEC 2001], Kim [Kim 1999], McCall [McCall et al. 1977], Boehm [Boehm et al. 1978], suggest ways in which to describe different quality characteristics of software products.
- **State the GQM goal(s) at a conceptual level** (Figure 3, Activity I_3): The two previous activities are used to state the GQM Goal(s), which is (are) defined by using the following template [Basili and Weiss 1984]: Analyze the 'object of study' for 'purpose' with regard to 'quality focus' from the point of view of 'point of view'. In other words, a GQM goal specifies what objects are measured for what purposes, from which viewpoints, with regard to which focuses [Saeki 2003].

Once the goal(s) has/have been defined, it/they should be refined into a set of questions. Nevertheless, before addressing the definition of questions, which in fact allows GQM goals to be quantified, it is necessary to consider the structural properties [Darcy and Slaughter 2005] of the software artifact to be studied:

- **Determine the structural properties to be studied** (Figure 3, Activity I_4): The properties (or internal attributes) that we intend to measure should be defined, because we usually interpret software data at that attribute level [Kitchenham et al. 1995]. That is, should we study the coupling, cohesion, size, or length, of the software artifacts?
- **Identify abstractions for measuring the structural properties** (Figure 3, Activity I_5): In helping to identify the structural properties clearly, we should take into account the definition of abstractions for measuring the structural properties as recommended by Briand et al. [Briand et al. 2002] and Card [Card 1993]. For instance, if coupling is the structural property to be studied, the abstraction should identify the different kinds of connections that constitute coupling, the locus of impact of coupling, the granularity of coupling, etc. [Briand et al. 1996].
- **Refine the goal(s) into questions at an operation level** (Figure 3, Activity I_6): Once the structural properties have been selected and abstractions for measuring them are defined, GQM questions can be established. Questions should fit the GQM goals, otherwise they should be redefined or discarded. This situation is modeled through a decision, using the diamond notation shown in Figure 3.

- **State general hypotheses** (Figure 3, Activity I_7): Finally, general hypotheses should be stated, relating structural properties and the quality focus. The definition of a precise, testable research hypothesis is required before any empirical study can be performed. An Empirical hypothesis is a statement which is believed to be true about the relationship between one or more attributes of the object of study and the quality focus. In other words, empirical hypotheses relate the (independent) attributes of some entities to other (dependent) attributes of the same or different software product or activities [Briand et al. 2002].

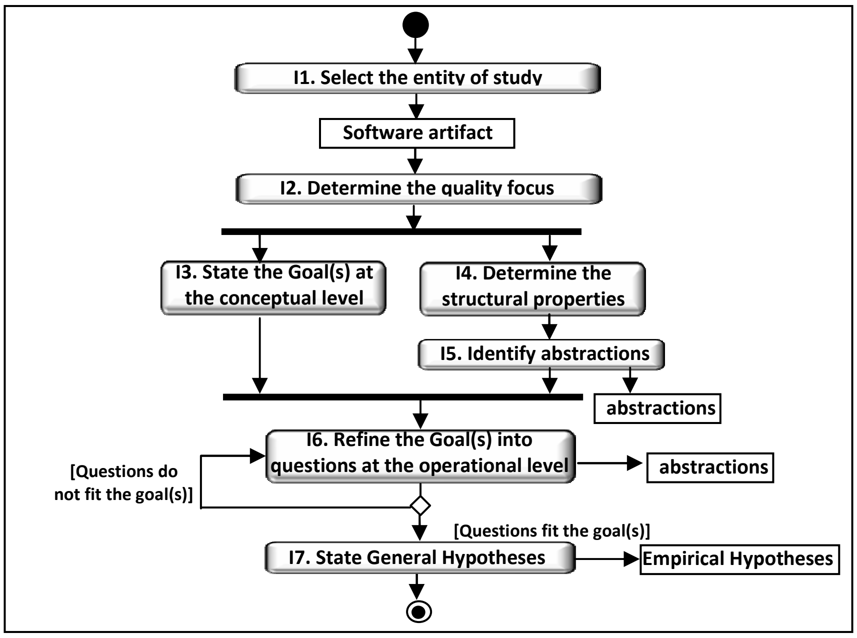

I1. Select the entity of study
Software artifact
I2. Determine the quality focus
I3. State the Goal(s) at the conceptual level
I4. Determine the structural properties
I5. Identify abstractions
abstractions
I6. Refine the Goal(s) into questions at the operational level
abstractions
[Questions do not fit the goal(s)]
[Questions fit the goal(s)]
I7. State General Hypotheses
Empirical Hypotheses

Figure 3: Identification activity.

Most of the main components of the identification activity involve interaction with the stakeholders in an organization, to elicit a shared view about their organization's needs [Berander and Jönsson 2006]. Setting goals and stating general hypotheses are not simple activities, because they are products of expert knowledge. Their identification is a high level activity within the method, in which goal-driven requirement engineering can be applied [Kavakli 2004].

5 Creation

Creation (Figure 2, Activity M_2) relies on the following four activities:

- **Measure Definition** (Figure 4, Activity C_1): In order to clearly define a measure, it is important to tackle two important issues: a clear specification of what is captured by the measure - its purpose, and a formal specification of the measure (i.e. how it is defined). With regard to the former issue, measures are defined by taking into account the goal(s) and questions provided by the identification activity. On the latter, in measurement literature various different approaches have been applied in defining measures: natural language, mathematical approaches, and formal languages. The measures should be defined in a consistent and coherent manner, to avoid misunderstandings and the misinterpretation of meaning.

- **Theoretical Validation** (Figure 4, Activity C_2): Once a measure has been defined, it must be verified whether it fulfills the properties that are associated with the attribute it is intended to measure [Mendonça and Basili 2000]. This task is called theoretical validation, internal validation or formal validation. In the context of an empirical study, the theoretical validation of measures establishes their construct validity [Wholin et al. 2000], i.e. it proves that they are valid measures for the constructs that are used as variables in the study. Theoretical validation is also useful for determining the scale type of the measure, and helps us to discover when and how to apply measures. For instance, the type of scale is useful in identifying the statistical techniques which should be applied in empirical studies.
- **Psychological Explanation** (Figure 4, Activity C_3): Ideally, we should be able to explain how the subjects deal with the software artifacts that are the focus of our measurement activities. As Cant et al. [Cant et al. 1992] remark, measuring structural properties should affect attributes of human comprehension. As a reference discipline in this step, cognitive psychology can be used to obtain a plausible explanation of the effort of the subjects dealing with the software artifact being measured. A clear understanding of the cognitive complexity[1] of the subjects dealing with the software artifact will help us to understand how difficult it is, for instance, to maintain that software artifact, since anything that is difficult to comprehend will affect its maintainability. The psychological explanation is also useful, in that it provides a clear interpretation of the results of empirical studies. This activity can be carried out at the same time as the theoretical validation and it is directly strengthened when qualitative methods are applied in empirical studies [Seaman 1999].
- **Empirical Validation** (Figure 4, Activity C_4): This task is also called empirical validation or external validation. It is an activity that investigates whether the measure is actually effective in practice, i.e. the study assesses whether the measures are related to certain external attributes. Thus, the main purpose of this activity is to run quantitative empirical studies. The activity takes into account the empirical hypotheses provided by the identification activity.

These activities will be described in detail in sections 5.1, 5.2, 5.3, 5.4. The activity of creating measures is evolutionary and iterative and as a result of the feedback, the method could refine, reject or define new measures. We identify two situations in which a review of the creation activity should be performed. The first is after finishing the Theoretical Validation activity since: (1) the measure may not be theoretically valid or (2) the measure may be theoretically valid but it might not capture an expected attribute (the attribute that the measure aims to quantify). The

[1] Cognitive complexity is defined as the mental burden of a person dealing with a software artifact [Briand et al. 1999d].

second situation arises after the empirical validation is performed. Different situations may arise in this case: a measure might not be empirically valid, various measures may capture the same dimension of a concept, derived measures need to be defined as a more precise indicator of independent variables, etc. The two aforementioned situations were modeled through the two bottom diamond decisions in the UML activity diagram shown in Figure 4.

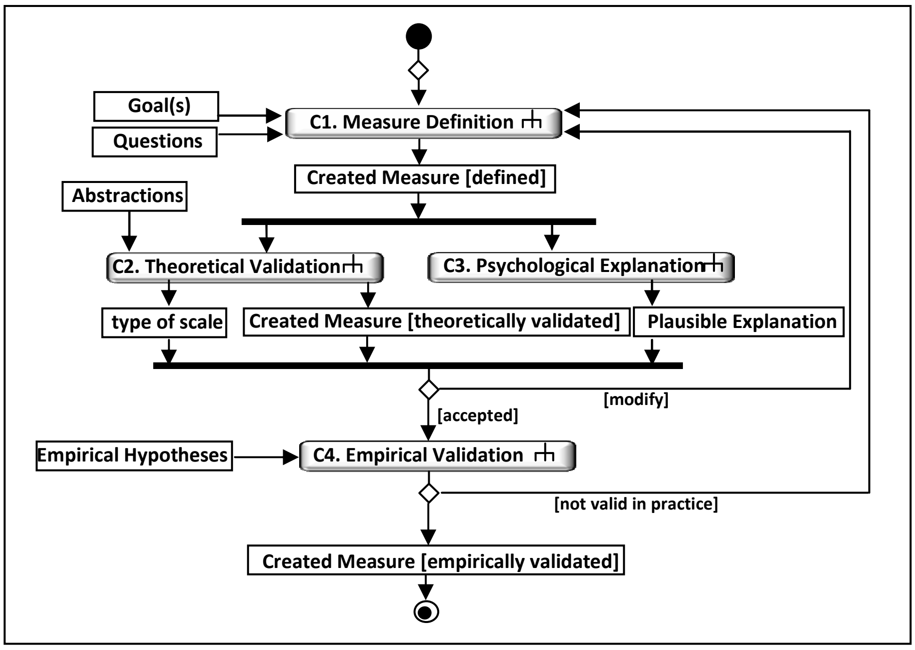


Figure 4: Creation activity.

5.1 Measure Definition

This section explains the Measure Definition (Figure 4, Activity C_1).When measures are defined, the most important goal is that they should (at a quantitative level) provide the information to answer the stated GQM questions. However, the activity of defining measures is not simple. Measures must initially be defined by using natural language and they should then be formally defined. Moreover, both activities have their own preconditions, which constrain the order in which they should be performed:

- **Select a metamodel of the software artifact** (Figure 5, Activity D_1): The definition of a measure must be sufficiently clear and detailed for any concept of the software artifact (the object of study) mentioned in the natural language definition to be quantifiable, i.e. able to be measured [Briand et al. 1996]. In order to fulfill this, a metamodel of the software artifact being measured should be selected as an activity prior to any measure definition. As is defined in [Jacquet and Abran 1997], a metamodel constitutes the set of characteristics selected to represent a software or software piece and the set of their relationships, providing an overall description of the software artifact to which the measurement method will be applied. By using a metamodel, we will be able to ensure that any concept mentioned in the measure definition using natural language should also be an element of the selected metamodel.

- **Define the measure in natural language** (Figure 5, Activity D_2): The activity of defining a measure includes its proper definition, its measure goal, explains how the measure value is obtained and includes a name and its corresponding acronym. Figure 5 assumes that many measures can be defined, so the activity D_2 occurs iteratively for each measure. The activity has a rake in one corner, so its description as an activity is shown in Figure 5 and is explained in subsection 5.1.1.
- **Select a formal language for the formal definition** (Figure 5, Activity D_3): Before any measures are formally defined, we should select a formal language with which to perform the activity. The selection of the formal language may be carried out in parallel with activities D_1 and D_2.

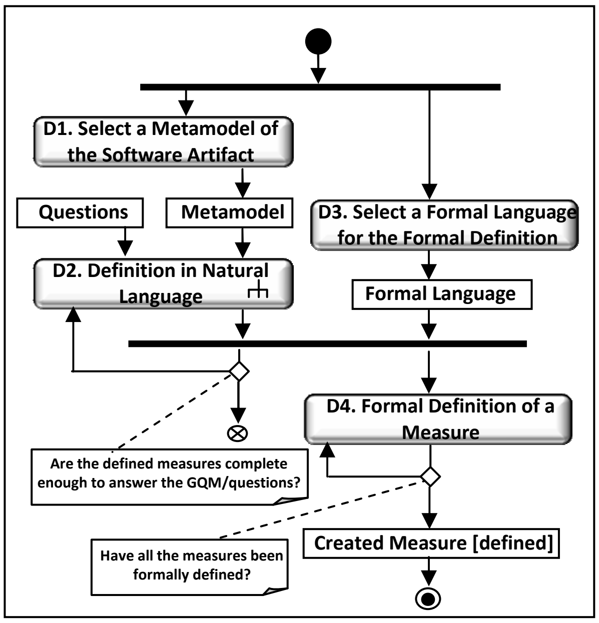

D1. Select a Metamodel of the Software Artifact
Questions
Metamodel
D3. Select a Formal Language for the Formal Definition
D2. Definition in Natural Language
Formal Language
D4. Formal Definition of a Measure
Are the defined measures complete enough to answer the GQM/questions?
Created Measure [defined]
Have all the measures been formally defined?

Figure 5: Measure definition activity.

- **Carry out a formal definition of a measure** (Figure 5, Activity D_4): The purpose of this activity is to provide a precise definition of each measure upon the metamodel. The formal definition of the measures is closely related to its definition in natural language, since the formal specification should be coherent with the natural language description which explains the way in which the measure values should be obtained. Although this activity is not further detailed in an activity, section 5.1.2 explains the underlying reasons for introducing this activity as part of the method.

We will be able to formally define a measure once (1) the first measure being defined using natural language is obtained (Figure 5, Activity D_2), and (2) when both a metamodel and a formal language have been selected. These preconditions are modeled in Figure 5 through the last join. The whole activity terminates when the final measure is formally defined, this being the condition evaluated in the last diamond.

5.1.1 Definition in Natural Language

The Definition in Natural Language (Figure 5, Activity D_2) defines the measures using natural language and it contains the following activities:

- **Define what is captured by the measure** (Figure 6, Activity N_1): The definition of the measure should include a clear description in natural language of what is captured by that measure.
- **Verify the definition explains how the measure value is obtained** (Figure 6, Activity N_2): Each concept and relationship mentioned in the definition must be quantifiable. The measure definition should also give a precise description of how the value of a measure is obtained, e.g. if the measure is defined as a rate, a specification of its formula will be provided.

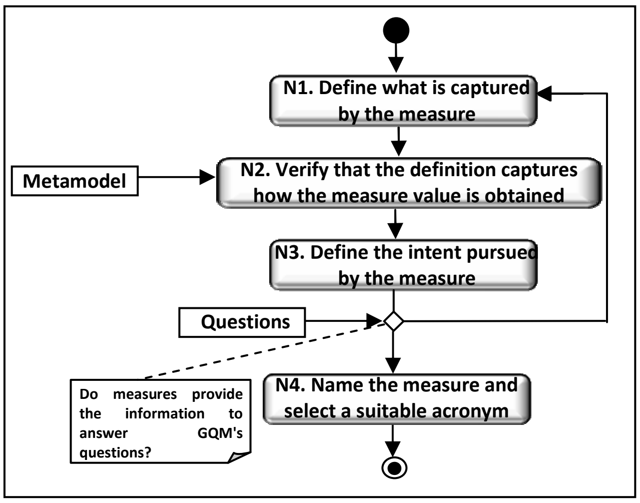

N1. Define what is captured by the measure
Metamodel
N2. Verify that the definition captures how the measure value is obtained
N3. Define the intent pursued by the measure
Questions
Do measures provide the information to answer GQM's questions?
N4. Name the measure and select a suitable acronym

Figure 6: Definition in natural language activity.

- **Define the goal pursued by the measure** (Figure 6, Activity N_3): The measure's intent should be consistent with the GQM question to which the measure provides information. The measure's intent should also be described by considering the cognitive complexity of the modelers dealing with the aspects and concepts captured by the measure. If the measure's intent does not provide information to answer the questions, i.e. if it does not fit our desired aims, we should review its definition or eventually discard it. This decision is represented in the bottom diamond of Figure 6 and verifies that each measure's intent is aligned with the GQM-questions.

- **Name the measure and select a suitable acronym** (Figure 6, Activity N_4): The last activity of a measure definition is to name the measure and select a suitable acronym.

Many measures can be defined to answer different GQM-questions. It is also possible for a set of measures to be used to answer a GQM-question. This set should be complete enough to answer that specific GQM-question. The method therefore allows for the creation of different measures to answer a GQM-question, and verifies that each GQM-question can be answered with a set of measures. This situation is modeled in the diamond shown in Figure 6. By applying the GQM paradigm we ensure that the measures obtained are useful, simple and direct. The paradigm, however, is not intended to define measures at a level of detail which is suitable to ensure that they are trustworthy and, in particular, whether or not they are repeatable (i.e., if the measurement of an attribute were repeated by a different person whether the same result would be produced each time [Kitchenham et al. 1995]). To ensure repeatability, software measures need to be fully defined and specified, not simply named. This is one of the purposes of a formal definition of measures, which is explained below.

5.1.2 Formal Definition of a Measure

The purpose of the Formal Definition (Figure 5, Activity D_4) is to define the measures formally. Many difficulties arise when the measure is defined in an unclear or imprecise way:

- Experimental findings can be misunderstood, due to the fact that it may not be clear what the measures really captured are [Baroni 2002].
- Measure extraction tools can arrive at different results. Kitchenham et al. remark [Kitchenham et al. 2006] that most data collection problems arise from poor definitions of software measures. Data validation, data storage and data analysis problems are consequently involved.
- The replication of experiments is hampered [Baroni 2002].

These are also common problems when we evaluate or consider the methods (or frameworks) used in defining measures. Most of the existing measures differ in the degree of formality used in their definition. Two extreme approaches were used: informal and rigorous definitions. None of these approaches has been widely accepted, however. On the one hand, measures using an informal definition, such as measures defined in natural languages, may be ambiguously defined, and it is universally considered that the use of this practice could cause misinterpretations and misunderstanding. At the other extreme, in a rigorous approach,some authors have used a combination of set theory and simple algebra to express their measures [Chidamber and Kemerer 1994; Henderson-Sellers 1996]. This approach has not been popular, since the majority of members of the OO community may not have the background required to understand the underpinning of the complex mathematical formalism used. An example of how the use of natural language in a measure definition introduces ambiguity is considered in [Baroni 2002], which uses as example the "Number of Times a Class is Reused" metric proposed in [Lorenz and

Kidd 1994]. This measure is defined as the number of references to a class. We agree with Baroni that it is not clear "what references are and how the metric should be computed, and many questions arise, such as: Should internal and external references be counted? Should references be considered in different modules, packages or subsystems? Does the inheritance relationship count as a reference?".

One important contribution, which solves those problems related to the formality degree used to define measures, is the use of a formal language (e.g. OCL) upon a metamodel of the software artifacts to be measured. For instance, any measure defined for a UML artifact (e.g. UML statechart diagram) can use this approach, and we can provide a formal definition of the measures by using OCL upon the corresponding UML metamodel (e.g. UML statechart diagram metamodel [Reynoso et al., 2008]; OCL metamodel [Reynoso, 2007]).

5.2 Theoretical Validation

As was previously described, Theoretical Validation (Figure 4, Activity C_2) is carried out to assess whether a measure actually measures the attribute it claims to [Briand et al. 1995]. There are two main tendencies in measure validation which represent the most widely applied frameworks (modeled in Figure 7):

- **Use of property-based frameworks** (Figure 7, Activity T_1): Some of this kind of frameworks are those proposed in [Weyuker 1988], [Briand et al. 1996] and [Morasca and Briand 1997].
- **Use of frameworks based on measurement theory** (Figure 7, Activity T_2): Poels and Dedene [Poels and Dedene 2000]; Zuse [Zuse 1997]; Whitmire [Whitmire 1997] proposed frameworks based on measurement theory.

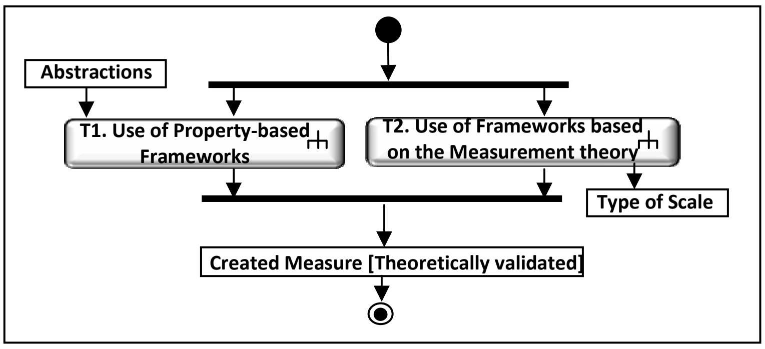

Abstractions
T1. Use of Property-based Frameworks
T2. Use of Frameworks based on the Measurement theory
Type of Scale
Created Measure [Theoretically validated]

Figure 7: Theoretical validation activity.

The use of property-based frameworks does not contradict measurement theory [Briand et al. 2002]. Similarly, measurement theory does not contradict property-based frameworks. The activity of theoretical validation using different frameworks can thus be performed simultaneously. The activity T_2, which represents the application of the measurement theory, also helps us to determine the scale type of a measure. Both activities, T_1 and T_2, show a rake on their right-hand side, meaning that they are further broken down. However, for the sake of brevity, a description is omitted in this article. We believe that both represent well-known processes in measurement literature. Property-based approaches propose a measure property set that is necessary but not sufficient [Briand et al. 1996; Poels and Dedene 2000]. They can be used as a filter to reject proposed measures [Kitchenham 1997], but they are not sufficient to prove the validity of the measure.

5.3 Psychological Explanation

This section discusses Psychological Explanation (Figure 4, Activity C_3). The structural properties of software artifacts influence the cognitive complexity of the software engineers dealing with those artifacts [Briand et al. 1999d; Briand et al. 1999e; Briand et al. 2001], e.g. high structural complexity makes a software artifact more difficult to comprehend. As was previously mentioned, cognitive complexity is defined as the mental burden of a person dealing with a software artifact. We believe that this mental burden will also make an impact on the software quality attribute that is being studied as a GQM-goal (e.g. the high cognitive complexity of a person dealing with an artifact will cause the artifact to exhibit undesirable external qualities, such as the artifact being more difficult to maintain).

Cognitive complexity is therefore at the core of defining measures. The understanding of cognitive complexity has two advantages:

1. It is useful for defining the rationale behind each measure definition (Figure 6, Activity N_3) and in fact, as Klemola [Klemola 2000] remarks, many measures are supported by the fact that they are clearly related to cognitive limitations.
2. Cognitive complexity provides us with the theoretical knowledge to explain the findings of empirical studies, i.e. if we are able to describe and to understand how software engineers comprehend the software artifacts that are being measured, we will be better prepared to interpret and to analyze the empirical studies performed with subjects dealing with those artifacts.

A plausible explanation of the measures from a psychological point of view, such as the understanding of the cognitive demands that software places on software engineers [Glasberg et al. 2000] is necessary. Otherwise, as is argued in [Sebrechts and Black 1982], we only consider surface features of the software measured. By understanding cognitive psychology theories we can justify the influence of structural properties on external quality attributes (such as maintainability) through the study of cognitive complexity. Moreover, Darcy et al. suggest [Darcy and Slaughter 2005] the consideration of multiple theoretical perspectives, including human cognition, to provide a solid foundation upon which to derive an integrative model relating internal and external attributes of software quality.

A detailed cognitive model is a necessary basis for developing software product measures [Darcy and Slaughter 2005]. One way in which to operationalize cognitive complexity is to equate it with the ease of comprehending the software artifact that is measured, as Glasberg et al. note [Glasberg et al 2000]. Cognitive models and mental models are two important theoretical bases for program comprehension. Darcy et al. argue [Darcy and Slaughter 2005] that some of the programming comprehension models are sufficiently generalizable for them to be used also to understand and explain maintenance cognition.

We have identified the following activities for obtaining a plausible explanation of the measure:

- **Select the cognitive theory to use in a plausible explanation** (Figure 8, Activity PE_1): The selection of a cognitive psychology theory should be carefully justified, and the selection will be dependent on the software artifact (Activity I_1, Figure 3) to be measured and on the GQM-goal (Activity I_4, Figure 3) pursued in the measurement process.
- **Relate the cognitive theory to the software artifact and measures** (Figure 8, Activity PE_2): Once the cognitive theory has been selected and each of its components have been described, it is useful to use the cognitive theory to explain how the subjects deal with the artifacts measured and also to establish a relationship between the elements of the theory and the concepts captured in each measure.
- **Use Qualitative Methods to Understand Cognitive Complexity** (Figure 8, Activity PE_3): Seaman argues [Seaman 1999] that in order to delve into the complexity of the human role in software engineering rather than abstract it, qualitative methods should be used. It could be argued that human behavior is one of the few phenomena that is complex enough to require qualitative methods to study it. Bearing these arguments in mind, we have included an activity (PE_3) in which qualitative methods should be applied to help us to understand completely the cognitive complexity of software engineers dealing with a measured software artifact. A thorough study of qualitative methods for data collection and analysis which may be incorporated into empirical studies of software engineering is presented in [Seaman 1999]. The most common qualitative methods employed are observations, in-depth interviews and focus groups [Taylor and Bogdan 1984].

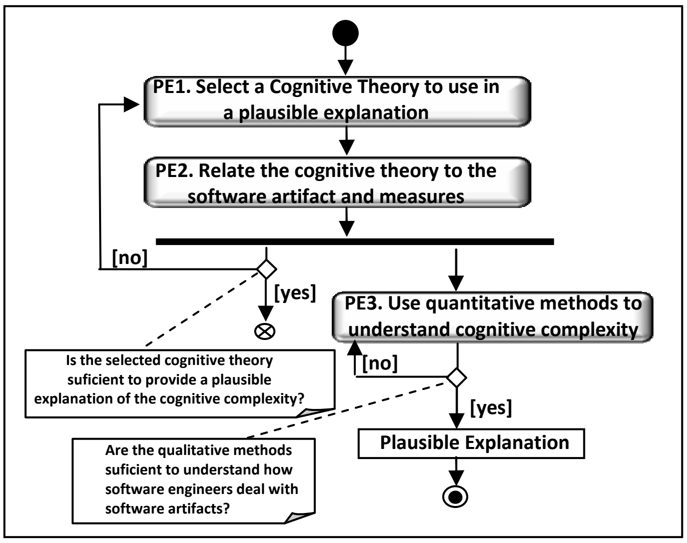

PE1. Select a Cognitive Theory to use in a plausible explanation
PE2. Relate the cognitive theory to the software artifact and measures
[no]
[yes]
PE3. Use quantitative methods to understand cognitive complexity
Is the selected cognitive theory suficient to provide a plausible explanation of the cognitive complexity?
[no]
[yes]
Plausible Explanation
Are the qualitative methods suficient to understand how software engineers deal with software artifacts?

Figure 8: Psychological explanation activity.

Since cognitive complexity constitutes one of the most important aspects that underpin the influence of structural properties on external quality attributes, this activity is considered to be crucial within the method.

5.4 Empirical Validation

In order to prove thoroughly that a measure is useful, an empirical validation (Figure 4, Activity C_4) must be carried out. It is not reliable to use common wisdom, intuition, speculation, or proof of concepts as sources of credible knowledge [Basili et al. 1999]. The measures must be placed under empirical validation. Empirical validation is an on-going activity [Briand et al. 1995] performed to demonstrate the usefulness of a measure. In other words, it addresses the following question: is the measure useful in the sense that it is related to other variables in expected ways? [Briand et al. 1995].

Empirical validation can also be used to demonstrate with real evidence that the measures we have proposed serve the purpose they were defined for. This phase is necessary before any attempt is made to use measures as objective and early indicators of quality. Empirical validation is therefore crucial for the success of any software measurement project [Schneidewind 1992; Kitchenham et al. 1995; Basili et al. 1999]. However, in general, insufficient empirical evidence exists to support the helpfulness of a vast number of proposed measures [Briand et al. 1999d]. Briand et al. therefore argue [Briand et al. 1999a] that empirical studies in software engineering need to be performed, analyzed, and reported better.

Empirical validation is used to obtain objective information concerning the usefulness of the proposed measures, since a measure may be valid from a theoretical point of view, but will not have any practical relevance to a specific problem. That being the case, empirical studies are needed, to confirm and understand the implications of the measurement of our products. This is achieved by means of hypotheses in the real world, above and beyond pure theory, which must be verified through the use of empirical data. Note that in our method general empirical hypotheses were defined as part of the I_7 activity of the identification step (Figure 3). These hypotheses should be empirically validated by means of a set of refined empirical hypothesis through different studies.

We have identified the following high level activities, which can be used to carry out any empirical validation:

- **Select a Strategy to Carry Out the Validation** (Figure 9, Activity E_1): There are three major strategies or types [Robson 1993; Wholin et al. 2000] of empirical investigations:
    a. experiment, i.e. a means of testing, using the principles and procedures of experimental design, if the hypothesis concerning the expected benefit of a tool or method can be confirmed;
    b. case study, i.e. a trial use of a tool or method in a full scale project;
    c. survey, i.e. the collection and analysis of data from a wide variety of projects.
- **Conduct the Strategy through a Family of Studies** (Figure 9, Activity E_2): Having selected the strategy, the validation should be run by using a family of studies, i.e. a family of experiments, a family of case studies, a family of surveys, etc. A family of studies is extremely useful and necessary to draw more credible

conclusions [Perry et al. 2000], and to contribute to obtaining more solid findings and expected results.
To perform any empirical strategy, such as an experiment, survey or case study, several steps have to be taken and must take place in a certain order [Wholin et al. 2000; Juristo and Moreno 2001]. Thus, a process for how to perform the experiments is needed. Processes are important, as they can be used as checklists and guidelines of what to do and how to do it. Only careful planning can guarantee successful empirical studies.

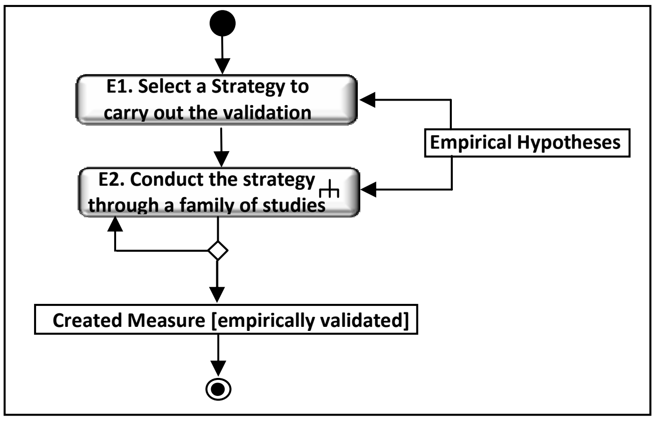

E1. Select a Strategy to carry out the validation
Empirical Hypotheses
E2. Conduct the strategy through a family of studies
Created Measure [empirically validated]

Figure 9: Empirical validation activity.

A wealth of literature on empirical strategies and their processes has been published over the past decade, which is omitted here for the sake of brevity. However, we recommend conducting these strategies appropriately in order to integrate study results into a common body of knowledge [Jedlitschka and Pfahl 2005].

6 Example: Definition of Measures for OCL Expressions

In this section we briefly explain how the new method was used to define measures for assessing the influence of import-coupling on the maintainability of OCL expressions [Reynoso et al. 2005b; Reynoso 2007].

6.1 Identification (M_1)

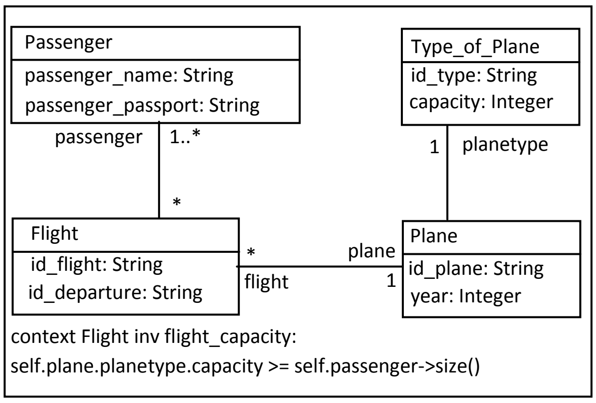
In this section we will present the tasks carried out in the Identification activity (Figure 2, Activity M_1) for the definition of measures for OCL expressions.

- **Select the entity of study** (Figure 3, Activity I_1).The entity of study is an OCL expression. These expressions are the primary elements used by modelers as textual add-on to UML models. Although an expression is attached to a particular contextual type (e.g. a class in a UML diagram), its meaning involves objects (mentioned within its definition) which are usually instances from different classes. The different classes mentioned in an OCL expression constitute the scope of the OCL expression. So, although our focus is an OCL expression, we can not study this artifact in an isolated manner. Its context and its scope are intrinsically involved.

Example: The upper part of Figure 10 shows a UML diagram in which an OCL expression named ‘flight_capacity’, has been defined in the context of the Flight class, meaning that the quantity of passengers on a flight must be lower than or equal to the capacity of the plane’s type on that flight. The contextual type of the expression is Flight, whereas its scope involves the Passenger, Plane and Type_of_plane classes.

- **Determine the quality focus** (Figure 3, Activity I_2). The OCL expression’s maintainability has been chosen as the prime quality attribute of interest. Our study of the OCL expression’s maintainability will help modelers to improve the quality of their models, and this is a major goal in software development using MDA [Kuzniarz 2007] since models are used to drive the entire software development process.

  To our knowledge, not all the maintainability sub-characteristics proposed in the ISO/IEC 9126 [ISO IEC 2001] standard are suitable for OCL expressions. We have considered two sub-characteristics: comprehensibility and modifiability.
- **State the goal** (Figure 3, Activity I_3). The GQM-goal is: Analyze OCL expressions with the purpose of evaluating maintainability from the viewpoint of the OO software modelers in OO software organizations. The object of study and the quality focus were described in the last two paragraphs. The purpose is evaluation, i.e. 'judge the value of'.



```
context Flight inv flight_capacity:
self.plane.planetype.capacity >= self.passenger->size()
```

Figure 10: OCL expression example.

- **Determine the structural property to be studied** (Figure 3, Activity I_4). We focus on the degree to which the elements in a design are connected, i.e. on the structural property of coupling. Coupling is generally recognized as being among the most likely quantifiable indicators for software maintainability. In fact, if one intends to build quality OO models, coupling will very likely be an important structural property to consider [Briand et al. 1999e]. However, coupling is a concept that has many dimensions. We will focus on the degree to which the

OCL expression has knowledge of, uses, or depends on other design elements [Briand et al. 1999b], i.e. on import-coupling, due to:

- The inner nature of OCL expressions: These artifacts are textual add-ons to UML models. Within an expression we can refer to UML artifacts but not the other way around.
- Important empirical findings: We are also interested in the import-coupling, because it has shown to be a strong, stable indicator of fault-proneness of classes [Briand et al. 1999e], and fault-proneness results in low maintainability [Briand et al. 1999b]. Similar results concerning import-coupling were obtained as an indicator of development effort [Briand et al. 2001], where export-coupling measures show a much weaker impact than import-coupling.

Example: The OCL expression in Figure 10 is defined in terms of different UML artifacts: rolenames (plane, planetype, passenger), a UML attribute (capacity). We could say that the expression is tightly coupled to its scope in the diagram. Most of the expressions within a UML/OCL model import-couple different UML artifacts in its definition.

• **Identifying Abstractions for Coupling** (Figure 3, Activity I_5). We have identified several criteria based on [Briand et al. 1999b] to describe abstractions for coupling, such as:
  - Type of Connections: Connections are inherent to any coupling measure. Two entities are usually involved in a connection. A client (or source) entity specifies a connection to a destination entity. The coupling connections we are interested in are connections between an OCL expression and any OO feature of a UML diagram. Consequently, in our case the source entity will always be an OCL expression, while the destination entity varies radically (rolenames, attributes, method names, etc).
  - Locus of impact: The coupling usually defines a client-supplier relationship between the design elements. This criterion defines whether we focus on defining measures for the client or for the server entity (in the connection). If the focus is the client, the locus of impact is import-coupling, otherwise the focus is the server and the locus of impact is export-coupling. As we briefly mentioned before, the intrinsic definition of OCL expressions as a textual add-on to a UML diagram (it allows the modeler to specify explicit references to UML features) constitutes a suitable mechanism through which to focus on the import-coupling. The focus is thus the client entity.
  - Granularity: This criterion involves:
    - o The domain of measure is always an OCL expression. Nevertheless, the expression refers to the semantic properties of its contextual type. Although an OCL expression seems to be a small domain, the scope of objects referred to through an expression (the portion of a UML diagram imported by an OCL expression can vary significantly) may be very large.

      Example: Note that the expression attached to the Flight class refers to three classes in the class diagram (its scope).
    - o The way in which we count connections is as follows: we always count the number of different items at the other end of the connections.

Example: Suppose that within the OCL expression a rolename is used twice within its definition. The two different occurrences of that artifact will be counted only as one.

- **Refine the goal into questions** (Figure 3, Activity I_6). The Briand et al. model [Briand et al. 1999d] was used as a basis for our hypothesis that OCL expression maintainability is influenced by its structural properties which, in turn, depend on the elements of which OCL expressions are composed (navigations, collection operations, variables, etc.). The most important question therefore arises: "Does import-coupling influence OCL expression maintainability?". We have also added two other questions: "Does size influence OCL expression maintainability?" and "Does length (of navigation) influence OCL expression maintainability?".
  The last two questions have come about with two different purposes. The length of navigations is closely related to the depth of coupling, whereas the size property is considered in an attempt to avoid the situation of size aspects biasing the findings related to coupling during experimentation [El Eman et al. 2001].
- **State general hypotheses** (Figure 3, Activity I_7). We hypothesize that high import-coupling of OCL expression affects the maintainability of OCL expressions. We suppose that the greater the import-coupling is, the lower the OCL expression maintainability will be.

6.2 Creation

Creation (Figure 2, Activity M_2) was carried out to measure OCL expressions. This section gives details of the NNR measure.

6.2.1 Measure Definition (C_1)

The Measure Definition (Figure 4, Activity C_1) involves the following activities:

- **Select a metamodel of the software artifacts** (Figure 5, Activity D_1). We have selected the OCL metamodel which defines the core concepts of OCL 2.0 and their relationships in the form of a MOF-compliant metamodel. Thus, all legal OCL expressions can be systematically derived and instantiated from the metamodel.
- **Definition in natural language** (Figure 5, Activity D_2). Each measure was defined using a consistent format composed of:
  - Its acronym and name: this component shows the result of activity N_4 (Fig. 6).
  - Its proper definition: this component involves the result of applying N_1 (define what is captured by the measure) and N_2 (verify that the definition explains how the measure value is obtained) activities of Fig. 6.
  - Its intent: this component describes the goal of the measure, and corresponds to the application of activity N_3 (Fig. 6).
  - By way of example: we have included a sample to illustrate its calculation. The definition of the measures is presented according to the attributes they are related to.

  We exemplify a complete definition through the NNR measure:

  - Acronym and name: NNR stands for Number of Navigated Relationships.
  - Definition: This measure counts the total number of relationships that are navigated at least once in an expression (application of N_1 activity). If a relationship is navigated twice, for example by using different properties of a class or interface, this relationship is counted only once (application of N_2). Whenever an association class is navigated we will consider the association to which the association class is attached.
  - Intent: As Warmer and Kleppe [Warmer and Kleppe 2003] remark: An "argument against complex navigation expressions is that writing, reading and understanding invariants becomes very difficult". The meaning of each relationship involves the understanding of how the objects are coupled to each other. The larger the set of relationships to be navigated, the greater the context to be understood is [Reynoso et al. 2005b].
  - Example: The value of NNR for the expression shown in Figure 10 is 3, because we have used three relationships in two different navigations. A simple navigation, self.passenger, navigates the relationship from Flight to Passenger by using the passenger rolename, whereas a combined navigation, self.plane.planetype, is navigated from Flight to Type_of_Plane through to Plane by using the plane and planetype rolenames.

- **Select a formal language for the formal definition** (Figure 5, Activity D_3): We select OCL as the formal language for the formal definition of measures for OCL expressions.
- **Formal definition of a measure** (Figure 5, Activity D_4): In our approach, when we compute the value of a specific measure we represent an OCL expression as an instantiation of OCL metaclasses. The instantiation has the shape of a tree, an abstract syntax tree (ast). The dynamic hierarchical structure (the ast) is traversed by using a Visitor pattern [Gamma et al. 1995]. We simultaneously visit every element in the tree, and evaluate whether each element of the tree is meaningful for the measure we wish to compute. For more details of the procedure of obtaining the values through the Visitor Pattern, we refer the reader to [Reynoso et al. 2006].

  Within the OCL metamodel, the NavigationCallExp metaclass (Figure 11) is used to represent navigations and constitutes a reference to an AssociationEnd (or an AssociationClass) defined in a UML model. This object reference is used when either a rolename or an association class is used in an OCL expression navigation. An OCL expression ast will have as many NavigationCallExp objects as the navigations contained in its definition. Therefore, following the example of the NNR measure, when we traverse the ast of an OCL expression, instances of NavigationCallExp will be meaningful to obtain the value of NNR.

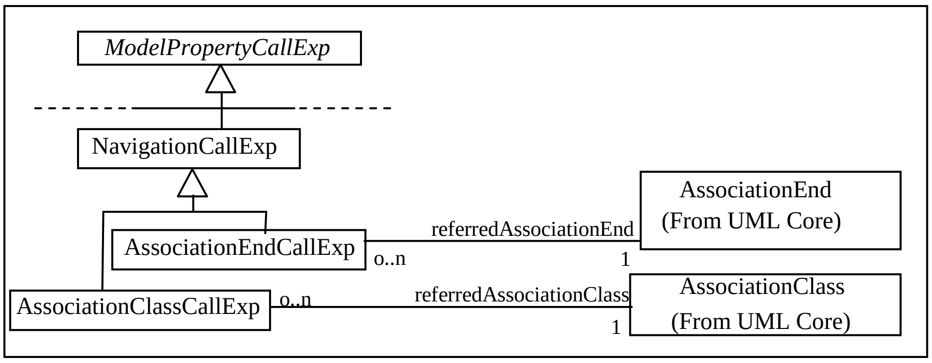


Figure 11: OCL metaclasses related to navigations.

The formal specification of NNR is specified as follows: Whenever a visitor accesses a NavigationCallExp object, it loads in a set (called navigatedClasses) either the name of the classes used in the navigation (if the modeler used a navigation class), or the name of the class of the AssociationEndCall type (i.e. the name of the class to which the rolename refers). However, as the same name of a rolename can be used in different classes, we decided to represent those elements composed of the pair of two strings in the set: the name of the class and the name of the relationship.

```
context Visitor::visitNavigationCallExp(o: NavigationCallExp, metricName:
MetricAcronym)
post:
   metricName = MetricAcronym::NNR
   implies navigatedClasses = navigatedClasses@pre->union(
   (if self.oclIsTypeOf(AssociationEndCallExp)
   then
   source.oclAsType(AssociationEndCallExp).referredAssociationEnd.type.name
   else
   source.oclAsType(AssociationClassCallExp).referredAssociationClass.name
   endif)->append
        (if self.oclIsTypeOf(AssociationEndCallExp)
   then
   source.oclAsType(AssociationEndCallExp).referredAssociationEnd.name
   else source.oclAsType(AssociationClassCallExp).referredAssociationClass.name
   endif))
```

The size of this set is used to obtain the NNR value. More details of the formal definition of OCL measures upon the OCL metamodel can be obtained in [Reynoso et al. 2006].

6.2.2 Theoretical Validation

To develop the Theoretical Validation (Figure 4, Activity C_2) we have applied property-based frameworks (Figure 7, Activity T_1) and a framework based on measurement theory (Figure 7, Activity T_2). In this paragraph we exemplify the application of the former framework. The Briand et al. adaptation framework for interaction-based measures for coupling [Briand et al. 1999c] was used for the theoretical validation of NNR measure.

1. **Nonnegativity:** This is directly proven, as it is impossible to obtain a negative value. An expression e without navigation in its definition has NNR(e) = 0.
2. **Monotonicity:** This is directly verified. Adding import interactions, in this case interactions of navigations, to an OCL expression cannot decrease its import-coupling. If we add a new navigation to an expression, two possible situations may arise: (1) the navigation referred to in the added navigation is a rolename (or association class) already used by an interaction. Thus the NNR applied to the new expression obtained is equal to NNR(e). (2) If the added navigation is new, then the NNR applied to the new expression is greater than NNR (e).
3. **Merging of modules:** Within our context, this property can be expressed in the following way: "the sum of the import-coupling of two modules is no less than the coupling of the module which is composed of the data used in the two modules". The value of the NNR for an expression which consists of the union of two original expressions, is equal to the NNR of each merged expression when the sets of navigations referred to in each original expression are disjointed, otherwise it is less than the NNR of each merged expression.

NNR is therefore validated as an interaction-based measure for coupling.

The theoretical validation following the measurement theory-based framework proposed in [Poels and Dedene 2002] was developed in [Reynoso 2007].

6.2.3 Psychological Explanation

We shall now briefly present the theory used in a plausible psychological explanation (Figure 4, Activity C_3) of the measures for OCL expressions.

- **Selected theory for OCL cognitive complexity** (Figure 8, Activity PE_1): As our hypothesis is that import-coupling, as a structural property, influences the cognitive complexity of modelers during OCL expression comprehension in the maintainability of OCL expressions, we have based our reasoning on the comprehension of OCL expressions using two main theories: cognitive models and mental models. The former concept describes a subject's mental representation of the software artifact to be understood, whereas a cognitive model describes the cognitive processes and temporary information structures in the subject's head that are used to form the mental model [Storey, 2005]. In this paragraph, for the sake of brevity, we shall describe the application of the cognitive model:

  In order to explain how OCL expressions are comprehended and how the navigation is a valuable help in guidance of comprehension we have applied the Cant et al. [Cant et al., 1992] Cognitive Complexity Model (CCM). The basis of the CCM is the definition of two cognitive techniques applied in program comprehension, namely chunking and tracing, which are concurrently and synergistically applied in problem solving.

a. The chunking technique represents the capacity of short term memory, involving the recognition of groups of declarations and extracting information from them which is remembered as a single mental abstraction: a chunk [Cant et al. 1992].
b. The tracing technique involves scanning, either forward or backward, in order to identify relevant chunks [El Eman 2001], resolving some dependencies.

Example: In some aspects, NNR determines the effort of a modeler in carrying out the tracing of the UML diagram. Each time navigation is used in an OCL expression, the modeler should trace a relationship in the associated UML diagram. We believe that OCL navigations are a key facilitator in the tracing of the cognitive technique.

We refer the reader to [Reynoso 2007] for a complete explanation of the selection of this theory.

- **Relate the cognitive theory to the software artifact and measures** (Figure 8, Activity PE_2): During the comprehension of the OCL expression a modeler must find the rolenames, classes and attributes mentioned in the expression (i.e., trace) and then chunk these entities before returning to the original chunk. The relatively large amount of tracing required causes a disruption in the reading of the superchunks, making them more complex [Cant et al. 1992]. While reading an upper-level chunk, a dependency requires the modeler to suspend the reading of the original OCL expression because of the need to undertake tracing, so as to have a complete understanding of the chunk currently being analyzed. The cognitive complexity model can therefore be described qualitatively in terms of a landscape model.

  Example: Figure 12 depicts the landscape associated with the OCL expression shown in Figure 10. Graphically, the top-level chunk (which involves the comprehension of the OCL expression) is interrupted by four lower-level chunks. The first interruption is common to every OCL expression and locates the context of the expression (the UML Classifier –a class, interface, etc.- written after the context keyword) within the UML diagram. The second interruption, depicted as the 'vertical drop' x1P, visually represents the work required in tracing the relevant features in the UML diagram. In this case, it implies following a navigation from the Flight class to another class, in which its opposite-end rolename is defined as ‘plane’. Having found this class, the modeler must chunk not only the class but also the cardinality associated with the rolename. The modeler should then follow a new navigation from Plane to Type_of_plane by using the 'planetype' rolename and, after chunking the meaning of the latter class, s/he must then chunk one of its attributes, that is, 'capacity'. The fourth and last interruption during the comprehension of the flight_capacity’ OCL expression is during the navigation from Flight to the Passenger class (drop x3P), thus obtaining the size of the set of passengers.

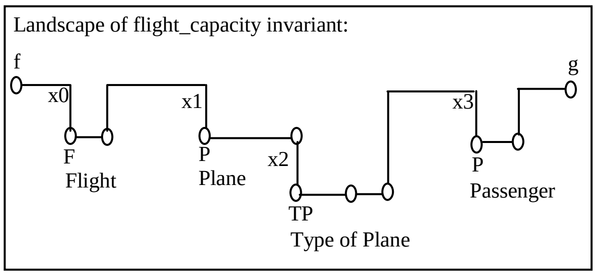


Figure 12: Landscape of an OCL expression.

- **Use qualitative methods to understand cognitive complexity** (Figure 8, Activity PE_3): We decided to apply a qualitative method, a verbal protocol analysis, in which subjects were given three class diagrams and were asked to think aloud to verbalize their thought. The underlying principle of verbal protocol analysis is that any verbalization produced by a subject whilst solving a problem –known as concurrent think aloud- will directly represent the contents of the subject's working memory. So, as OCL expressions consist of suitable short assertions that are not always easy to understand, especially when a lot of objects are coupled within the expression, this qualitative method is used to study the cognitive complexity of modelers dealing with OCL expressions. The experiment is described in [Reynoso et al. 2007]. The aim of the experiment was to validate a categorical model of the main categories of the mental models of the subjects dealing with OCL expressions. We found that the main categories are:
    a. Problem objects: The objects (main concepts) of the problem domain to which the OCL expressions are attached.
    b. Relationships between problem objects: association, composition and inheritance relationships between objects.
    c. Reified objects: These are not problem domain objects per se, but are represented to complete the representation of relationships between problem objects, e.g. OCL collections.

  These categories are based on work by Burkhardt [Burkhardt et al. 2002]. NNR is an example of measuring the relationship between problem objects. We also found that the breadth of familiarity with the UML diagram gained by the subjects before starting to comprehend the OCL expression comprehension activities was different. The range varies in a continuous form, which extends from those subjects who made absolutely no attempt to comprehend the diagram, to those who attempted to comprehend the class diagram systematically before starting to read the OCL expression. The subjects who did not attempt to comprehend the diagram before the comprehension of OCL expressions, followed an as-needed strategy of UML relationships; they focused only on those relationships when they appeared within the OCL expressions. OCL navigation was therefore of valuable assistance in guiding the comprehension of OCL expressions.

6.2.4 Empirical Validation

With regard to the empirical validation (Figure 4, Activity C_4), in [Reynoso et al. 2005a] we described an experiment and two replicas, with the goal of ascertaining whether any relationship exists between the import-coupling (defined in OCL expressions through navigations and collection operations) and the comprehensibility and modifiability of OCL expressions. In this empirical study the subjects were given six class diagrams with one OCL expression each and were asked to comprehend the expression and modify it to satisfy new requirements. The subjects were also asked to evaluate the complexity of comprehensibility and modifiability tasks subjectively. After performing a statistical analysis, we concluded that: (1) the NNR, NNC, WNN, DN, WNCO and NEI measures have a strong correlation with the comprehensibility efficiency (correct answers / comprehensibility time) for almost all of the six models; (2) the NNR, WNN, DN and NCO have a strong correlation with the modifiability efficiency for almost all of the six models. Many factors appear to influence the efficiency of comprehensibility tasks, such as classes, relationships, the navigations, the collection operations and the iterator variables, but only the number of relationships, collection operations and the depth of navigations influence the efficiency of modifiability tasks. The findings also reveal that the NNR, NNC, WNN, DN, WNCO and NEI measures are correlated with the subjective complexity of the subjects. We refer the reader to [Reynoso 2007] for a complete description of the empirical validation of OCL measures.

7 Conclusions

The main contribution of this paper is the refinement and extension of SMDM, a method for measure definition originally proposed in [Calero et al. 2001a], providing more details in the descriptions of the tasks, illustrated by means of UML activity diagrams. Our hope is that the new method will really help to guide us towards a better definition of software measures, ensuring reliability in obtaining well-defined and valid measures. Refinement and extension were a result of the use, over the last ten years, of the method for defining measures for OCL expressions [Reynoso 2007], UML diagrams [Genero et al. 2007; Cruz-Lemus et al. 2005], ER diagrams [Genero et al. 2008], Relational database schemas [Calero et al. 2001b], datawarehouse conceptual models [Serrano et al. 2004], etc.

The refinements of SMDM were introduced in the following steps:

- Creation (C_i activities, i=1..4): Although the main steps of the creation activity were already defined in the original method, we refined important node decisions and object flows between subactivities.
- Empirical Validation (E_i, F_j and EF_k activities, i=1,2, j=1..6, k=1..5): We thoroughly specified and detailed the more relevant activities by carrying out families of experiments and isolated experiments in [Reynoso 2007].

The extensions of the method focused on:

- Identification (I_i activities, i= 1..7): Within the refinement of this activity we specified not only the order in which goals and questions are specified but also a decision action to verify the questioning of the goals. New activities were added, such as the identification of abstractions with which to measure structural properties, the statement of general hypotheses, etc.

- Acceptation, Accreditation and Application (M_i activities, i=1..3).
- Definition in Natural Language (N_i activities, i=1..4): We used a template to define the measures, composed of the acronym, the definition itself, the goal pursued by the measure and one example.
- Formal Definition of Measures (D_i activities, i=1, 3, 4): We identified the most important activities that should be performed in a formal definition of measures.
- Psychological Explanation (PE_i activities, i=1..3): Three relevant activities were detected in a psychological explanation of how subjects deal with the software artifact being measured.
- Theoretical Validation using Property-based Frameworks (P_i activities, i=1..5): Within the method we differentiate between the application of generic properties and context-dependent properties in [Reynoso 2007].

Table 1 summarizes all the activities. The method has been strengthened, not only in the order of its activities but also by identifying object flows between them, as well as important decisions that should be evaluated during the activities.

The method also takes other important aspects into account:

- The phenomena that are studied in software engineering, which is in fact a human-intensive discipline, requires a focus on the human issues that are present in any measurement activity. The method values as important issues:
  - The organizational needs: In order to elicit measurement goal from the organization’s stakeholders,the method follows a GQM-based approach. This helps to institutionalize the measures within the organization.
  - The psychological aspects of those potential subjects who will make use of the defined measures: It is crucial to understand the cognitive complexity of a person when dealing with the software artifact which is the target of our measurement definition activity.

  Human cognition has obviously become more relevant, if we consider that over recent years software engineering empiricists have begun to address the human role in software development in a serious manner [Seaman 1999]. The understanding of cognitive complexity will assist in the definition of the measures’ goal and in the explanation of the empirical findings when applying an empirical strategy (experiment, survey, etc).
- The consideration of a model for the software artifact is appropriate when defining both its structural properties and its abstractions (in order to define measures on the grounds of how effective they are rather than on how desirable they may be [Card 1993]) and the use of a metamodel in order to produce a formal definition of the measure. A formal definition of the measure is of major importance when attempting to obtain replicable measures.
- The importance of testing that a measure captures the attribute that it aims to quantify (theoretical validation) as well as to prove it is valid in practice (empirical validation).

M_1 Identification (section 4)	I_1 Select the entity of study		
	I_2 Determine the quality focus		
	I_3 State the quality focus		
	I_4 State the goal at the conceptual level		
	I_5 Determine the structural properties		
	I_6 Refine the goal(s) at the operational level		
	I_7 State general hypotheses		
M_2 Creation (section 5)	**C_1 Measure Definition (section 5.1)**	D_1 Select a metamodel of the software artifacts	
		D_2 Definition in Natural Language (section 5.1.1)	N_1 Define what is captured by the measure
			N_2 Verify that the definition captures how the measure value is obtained
			N_3 Define the intent pursed by the measure
			N_4 Name the measure and select a suitable acronym
		D_3 Select a formal language for the formal definition	
		D_4 Formal definition of a measure (section 5.1.2)	
	C_2 Theoretical Validation (section 5.2)	T_1 Use of property-based frameworks (see refinements of this activity in [Reynoso 2007])	
		T_2 Use of frameworks based on the measurement theory (see refinements of this activity in [Reynoso 2007])	
	C_3 Psychological Explanation (section 5.3)	PE_1 Select a cognitive theory to use in a plausible explanation	
		PE_2 Relate the cognitive theory to the software artifact and measures	
		PE_3 Use quantitative methods to understand cognitive complexity	
	C_4 Empirical Validation (section 5.4)	E_1 Select a strategy to carry out the validation	
		E_2 Conduct the strategy through a family of experiments (see refinements of this activity in [Reynoso 2007])	

Table 1: Summary of refinements and extensions.

Our future work has a two-fold aim: Firstly, we will apply the method in the definition of new measures (related to other software artifacts), in order to continue validating the method. Apart from this, we encourage other researchers to use the method to obtain greater empirical evidence of its usefulness. Secondly, we will focus on the refinement of the acceptation, application and accreditation activities. These activities are likely to be refined in new UML activity diagrams with regard to the introduction of the proposed measures in real projects developed in software development organizations.

Acknowledgements

This research is part of the following projects: “MEDUSAS (IDI-20090557)” and “Proyecto Integrado ORIGIN (IDI-2010043(1-5))” financed by CDTI and FEDER; “ECCOO (TRA2009_0074)”, “MAGO/PEGASO (TIN2009-13718)” financed by MICINN; MECCA (PII2I09-0075-8394) financed by JCCM and CALIMO funded by the Generalitat Valenciana.the 04/e073 project financed by “Secretaría de Investigación de la Universidad Nacional del Comahue, Neuquén, Argentina”.

References

[Baroni 2002] Baroni, A. L.: “Formal Definition of Object-Oriented Design Metrics”. Master of Science in Computer Science Thesis, Vrije Universiteit Brussel, Belgium, 2002.

[Basili and Rombach 1998] Basili, V. R, Rombach, H. D.: “The TAME Project: Towards Improvement-Oriented Software Environments”. IEEE Transactions on Software Engineering, 14(6), 1998. 758- 773.

[Basili et al. 1999] Basili, V. R., Shull, F., Lanubile, F.: “Building Knowledge through Families of Experiments”. IEEE Transactions on Software Engineering, 25(4), 1999. 456-473.

[Basili and Weiss 1984] Basili, V. R., Weiss, D. M.: “A Methodology for Collecting Valid Software Engineering Data”. IEEE Transactions on Software Engineering, 10(6), 1984. 728-738.

[Berander and Jönsson 2006] Berander, P., Jönsson, P.: “A Goal Question Metric based Approach for Efficient Measurement Framework Definition”. ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM. New York, NY, USA, 2006. 316-325.

[Boehm et al. 1978] Boehm, B. W., Brown, J. R., Kaspar, J. R. “Characteristics of Software Quality”. TRW Series of Software Technology, Amsterdam, North Holland, 1978.

[Briand et al. 1999a] Briand, L. C., Arisholm, E., Counsell, S., Houdek, F., Thevenod-Fosse, P.: “Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions”. Empirical Software Engineering, 4(4), 1999. 387-404.

[Briand et al. 1999b] Briand, L. C., Daly, J. W., Wüst, J.: “A Unified Framework for Coupling Measurement in Object-Oriented Systems”. IEEE Transactions on Software Engineering, 25(1), 1999, 91-121.

[Briand et al. 1995] Briand, L. C., El Emam, K., Morasca, S.: “Theoretical and Empirical Validation of Software Product Measures”. Technical Report ISERN-95-03, ISERN: International Software Engineering Research Network, 1995.

[Briand et al. 1996] Briand, L. C., Morasca, S., Basili, V. R.: “Property-Based Software Engineering Measurement. IEEE Transactions on Software Engineering, 22(1), 1996. 68-86.

[Briand et al. 1999c] Briand, L. C., Morasca, S., Basili, V. R.: “Defining and Validating Measures for Object-Based High-Level Design”. IEEE Transactions on Software Engineering, 25(5), 1999. 722-743.

[Briand et al. 2002] Briand, L. C., Morasca, S., Basili, V. R. : “An Operational Process for Goal-Driven Definition of Measures”. IEEE Transactions on Software Engineering, 28(12), 2002. 1106- 1125.

[Briand et al. 1999d] Briand, L. C., Wüst, J., Ikonomovski, S., Lounis, H.: “A Comprehensive Investigation of Quality Factors in Object-Oriented Designs”. In IEEE ICSE '99: International Conference on Software Engineering, 1999.

[Briand et al. 1999e] Briand, L. C., Wüst, J., Ikonomovski, V., Lounis, H.: “Investigating Quality Factors in Object-Oriented Designs: an Industrial Case Study”. In ICSE '99:

Proceedings of the 21st International Conference on Software Engineering, Los Alamitos, CA, USA. IEEE Computer Society Press, 1999. 345-354.

[Briand et al. 2001] Briand, L. C., Wüst, J., Lounis, H.: “Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs. Empirical Software Engineering, 6(1), 2001, 11-58.

[Burkhardt et al. 2002] Burkhardt, J., Detienne, F., Wiedenbeck, S.: “Object-oriented Program Comprehension: Effect of Expertise, Task and Phase”. In Empirical Software Engineering 7(2), Hingham, MA, USA, Kluwer Academic Publishers, 2002. 115-156.

[Calero et al. 2001a] Calero, C., Piattini, M., Genero, M.: “Method for Obtaining Correct Metrics”. In ICEIS '01: Proceedings of the 3rd International Conference on Enterprise and Information Systems, volume 2, 2001. 779-784.

[Calero et al. 2001b] Calero, C., Piattini, M., Genero, M.: “Empirical validation of referential integrity metrics”. Information and Software Technology. Special Issue on Controlled Experiments in Software Technology. 43 (15), 2001. 949-957.

[Cant et al. 1992] Cant, S. N., Jeffery, D. R., Henderson-Seller, B.: “A Conceptual Model of Cognitive Complexity of Elements of the Programming Process”. Information and Software Technology, 37(7), 1992. 351-362.

[Cantone and Donzelli. 1999] Cantone G., Donzelli, P.: “Goal Oriented Software Measurement Models”. In ESCOM-ENCRESS '98: European Software Control and Metrics Conference, Herstmonceux Castle, East Sussex, UK, 1999.

[Card 1993] Card, D. N.: “What Makes for Effective Measurement?” IEEE Software, 10(6), 1993. 94-95.

[Chidamber and Kemerer 1994] Chidamber, S. R., Kemerer, C. F.: “A Metrics Suite for Object Oriented Design”. IEEE Transactions on Software Engineering, 20(6), 1994. 476-493.

[Cruz-Lemus et al. 2005] Cruz-Lemus, J.A., Genero, M., Manso, M. E., Piattini, M.: “Evaluating the Effect of Composite States on the Understandability of UML Statechart Diagrams”. ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MODELS/UML 2005). LNCS 3713, 2005. 113-125.

[Darcy and Slaughter 2005] Darcy, D. P., Slaughter, S. A: “The Structural Complexity of Software: An Experimental Test”. IEEE Transactions on Software Engineering, 31(11), Member-Chris F. Kemerer and Member-James E. Tomayko, 2005. 982-995.

[El Eman et al. 2001] El Emam, K., Benlarbi, S., Goel, N., Rai, S. N.: “The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics”. IEEE Transactions on Software Engineering, 27(7), 2001, 630-650.

[El Eman 2001] El Emam, K. “Object-Oriented Metrics: A Review of Theory and Practice”. Technical Report NRC 44190, National Research Council Canada. Institute for Information Technology, 2001.

[Fenton and Pfleeger 1998] Fenton, N. E., Pfleeger, S.: “Software Metrics: A Rigorous and Practical Approach”, PWS Publishing Co., Boston, MA, USA, 1998.

[Glasberg et al 2000] Glasberg, D., El Emam, K., Melo, W., Madhavji, N.: “Validating Object-Oriented Design Metrics on a Commercial Java Application”. Technical Report NRC/ERB-1080, National Research Council of Canada, 2000.

[Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison Wesley Longman, Publishing Co., Inc., Boston, MA, 1995.

[Genero et al. 2007] Genero, M., Manso, M. E., Piattini, M., Visaggio, A., Canfora, G: “Building Measure-Based Prediction Models For UML Class Diagram Maintainability”. Empirical Software Engineering, 2007. 12(5), 2007. 517-549.

[Genero et al. 2008] Genero, M., Poels, G., Piattini, M.: “Defining and Validating Metrics for Assessing the Understandability of Entity-Relationship Diagrams”. Data and Knowledge Engineering. 64(3), 2008. 534-557.

[Henderson-Sellers 1996] Henderson-Sellers, B.: “Object-Oriented Metrics: Measures of Complexity”. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[ISO IEC 2001] ISO. IEC 9126-1 “Information Technology - Software Product Quality - Part 1: Quality Model”.

[Jacquet and Abran 1997] Jacquet, J. P., Abran, A.: “From Software Metrics to Software Measurement Methods”. In ISESS '97: Proceedings of the 3rd International Software Engineering Standards Symposium (ISESS '97), Washington, DC, USA, IEEE Computer Society, 1997. 128-135.

[Juristo and Moreno 2001] Juristo, N., Moreno, A.: “Basics of Software Engineering Experimentation”. Kluwer Academic Publishers, 2001.

[Jedlitschka and Pfahl 2005] Jedlitschka, A., Pfahl, D.: “Reporting Guidelines for Controlled Experiments in Software Engineering”. In ISESE '05: International Symposium on Empirical Software Engineering (ISESE 2005), November 2005, Noosa Heads, Australia, 2005. 95-104.

[Kavakli 2004] Kavakli, E.: “Modeling organizational goals: analysis of current methods”, SAC '04: Proceedings of the 2004 ACM symposium on Applied computing. ACM, New York, NY, USA, 2004. 1339-1343.

[Kitchenham et al. 2006] Kitchenham, B., Al-Khilidar, H., Ali Babar, M., Berry, M., Cox, K., Keung, J., Kurniawati, F., Staples, M., Zhang, H., Zhu, L.: “Evaluating Guidelines for Empirical Software Engineering Studies”. In ISESE '06: Proceedings of the 2006 ACM/IEEE International Symposium on International Symposium on Empirical Software Engineering, New York, NY, USA. ACM Press, 2006. 38-47.

[Kim 1999] Kim, H.: “Representing and Reasoning about Quality using Enterprise Models”. PhD Thesis, Dept. Mechanical and Industrial Engineering, University of Toronto, Canada. 1999.

[Klemola 2000] Klemola, T.: “A Cognitive Model for Complexity Metrics”. In QAOOSE '00: Workshop on Quantitative Approaches in Object-Oriented Software Engineering (ECOOP '00). Cannes, France. Springer-Verlag, 2000.

[Kitchenham et al. 1995] Kitchenham, B., Pfleger, S. L., Fenton, N.: “Towards a Framework for Software Measurement Validation”. IEEE Transactions on Software Engineering, 21(12), 1995. 929-944.

[Klemola and Rilling 2002] Klemola, T., Rilling, J.: “Modeling Comprehension Processes in Software Development”. In ICCI '02: Proceedings of the 1st IEEE International Conference on Cognitive Informatics, Washington, DC, USA. IEEE Computer Society. 2002. 329-339.

[Kitchenham 1997] Kitchenham, B., Stell, J.: “The Danger of Using Axioms in Software Metrics”. In IEEE Proceedings on Software Engineering, Volume 144, 1997. 279-285.

[Kuzniarz 2007] Kuzniarz, L., Sourouille, J. L., Staron, M.: “1st Workshop on Quality in Modeling”, Co-located with the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Models 2006, LNCS 4364, 2007. 76-79.

[Lorenz and Kidd 1994] Lorenz, M., Kidd, J.: “Object-Oriented Software Metrics: A Practical Guide”. Prentice-Hall, Inc, Upper Saddle River, NJ, USA. 1994.

[McCall et al. 1977] McCall, J. A., Richards, P. K., Walters, G. F. “Factors in Software Quality, Volume III: Preliminary Handbook on Software Quality for an Acquisition Manager”. Technical Report RADC-TR-77-396, Vol. III., Hanscom AFB, MA 01731. 1977.

[Mendonça and Basili 2000] Mendonça, M. G., Basili, V. R.: “Validation of an Approach for Improving Existing Measurement Frameworks”. IEEE Transactions on Software Engineering, 28(6), 2000. 484-509.

[Morasca and Briand 1997] Morasca, S., Briand, L. C.: “Towards a Theoretical Framework for Measuring Software Attributes”. In METRICS '97: Proceedings of the 4th International Symposium on Software Metrics, Washington, DC, USA, IEEE Computer Society, 1997. 119-126.

[Perry et al. 2000] Perry, D. E., Porter, A. A., Votta, L. G.: “Empirical Studies of Software Engineering: a Roadmap”. In ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, ACM Press, 2000. 345-355,

[Pfleeger et al. 1997] Pfleeger, S. L., Jeffery, R., Curtis, B., Kitchenham, B.: “Status Report on Software Measurement”. IEEE Software, 14(2), 1997. 33-43.

[Poels and Dedene 2000] Poels, G., Dedene, G.: “Distance-based Software Measurement: Necessary and Sufficient Properties for Software Measures. Information and Software Technology, 42(1), 2000. 35-46.

[Reynoso et al. 2008] Reynoso, L., Cruz-Lemus, J. A., Genero, M., Piattini, M.: “Formal definition of measures for UML statechart diagrams using OCL”.SAC '08: Proceedings of the 2008 ACM Symposium on Applied Computing. 2008. 846-847

[Reynoso et al. 2005a] Reynoso, L., Genero, M., Piattini, M.: “Assessing the impact of coupling on the understandability and modifiability of OCL expressions within UML/OCL combined models”. 11th IEEE International Software Metrics Symposium, 2005. 14.

[Reynoso et al. 2005b] Reynoso, L., Genero, M., Piattini, M.: “Measuring OCL Expressions: An Approach Based on Cognitive Techniques”, 2005. M. Genero, M. Piattini and C. Calero (Eds.) Chapter 5 in Metrics for Software Conceptual Models. Imperial College Press, UK. 2005. 161-206.

[Reynoso et al. 2006] Reynoso, L., Genero, M., Piattini, M.: “OCL2: Using OCL in the Formal Definition of OCL Expression Measures”, 1st Workshop on Quality in Modeling QIM co-located with the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MODELs 2006). 1st. October, Genova, Italy. 2006. 95-115.

[Reynoso 2007] Reynoso, L.: “Measurement-based Approach for Assessing the Influence of Import-Coupling on the Maintainability of OCL Expressions”. Ph.D. Thesis, Universidad de Castilla La Mancha, Spain. 2007.

[Reynoso et al. 2007] Reynoso, L., Genero, M., Piattini, M.: “Using Verbal Protocols for Assessing the Influence of Import-Coupling on the OCL Expression Comprehensibility”. Sixth IEEE International Conference on Cognitive Informatics. IEEE ICCI 2007 6-9 August 2007 Lake Tahoe, CA, USA. 2007. 440-449.

[Robson 1993] Robson, C.: “Real World Research: A Resource for Social Scientists and Practitioners-Researchers”. Blackwell, Oxford. 1993.

[Saeki 2003] Saeki, M.: “Embedding Metrics into Information System Development Methods: An Application of Method Engineering Technique”. Lecture Notes in Computer Science 2681, 2003. 374-389.

[Schneidewind 1992] Schneidewind, N. F.: “Methodology for Validating Software Metrics”. IEEE Transactions on Software Engineering, 18(5), 1992. 410-422.

[Seaman 1999] Seaman, C. B.: “Qualitative Methods in Empirical Studies of Software Engineering”. IEEE Transactions on Software Engineering, 25(4), 1999. 557-572.

[Sebrechts and Black 1982] Sebrechts, M., Black, J.: “Software Psychology: A Rich New Domain for Applied Psychology”. Applied Psycholinguistics, 3, 1982. 223-232.

[Serrano et al. 2004] Serrano, M. A., Calero, C., Trujillo, J., Luján-Mora, S., Piattini, M.: “Empirical Validation of Metrics for Conceptual Models of Data Warehouses”. En 16th International Conference on Advanced Information Systems Engineering (CAiSE 2004), Riga (Letonia). LNCS 3084, 2004. 506-520.

[Solingen and Berghout 1999] Solingen, R. V., Berghout, E.: “The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development”. McGraw-Hill. 1999.

[Solingen and Berghout 2001] Solingen, R. V., Berghout, E.: “Integrating Goal-Oriented Measurement in Industrial Software Engineering: Industrial Experiences with and Additions to the Goal/Question/Metric Method (GQM)”. In METRICS '01: Proceedings of the 7th International Symposium on Software Metrics, Washington, DC, USA, IEEE Computer Society, 2001. 246-259.

[Storey 2005] Storey, M. A.: “Theories, Methods and Tools in Program Comprehension: Past, Present and Future”. In IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension, Washington, DC, USA, IEEE Computer Society. 2005. 181-191.

[Taylor and Bogdan 1984] Taylor, S. J., Bogdan, R.: “Introduction to Qualitative Research Methods”. New York: John Wiley and Sons. 1984.

[Warmer and Kleppe, 2003] Warmer, J., Kleppe, A.: “The Object Constraint Language. Second Edition. Getting Your Models Ready for MDA”, Addison-Wesley, Massachusetts. 2003.

[Weyuker 1988] Weyuker, E. J.: “Evaluating Software Complexity Measures”. IEEE Transactions on Software Engineering, 14(9), 1988. 1357-1365.

[Whitmire 1997] Whitmire, S. A.: “Object Oriented Design Measurement”. John Wiley and Sons, Inc., New York, NY, USA. 1997.

[Wholin et al. 2000] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, M.: “Experimentation in Software Engineering: an Introduction”. Kluwer Academic Publishers. 2000.

[Zuse 1997] Zuse, H.: “A Framework of Software Measurement”. Walter de Gruyter & Co. Hawthorne, NJ, USA. 1997.