Departamento de Tecnologías y Sistemas de Información

Universidad de Castilla La-Mancha

Tesis Doctoral

A Measurement-based Approach for Assessing the Influence of Import-Coupling on the Maintainability of OCL Expressions

Doctorando:

Luis Reynoso

Directores:

Dr. Mario Piattini Velthuis

Dra. Marcela Genero Bocco

Departamento de Tecnologías y Sistemas de Información
Universidad de Castilla La-Mancha
Paseo de la Universidad, 4. 13701, Ciudad Real

November, 2007
Ciudad Real, Spain.

Dedication

To my mother, Emilia, for her confidence in me, her support, her vision and her strength which helped me grow up.
To all my family, my sisters, Norma and Miriam, my brother in law, Jose, my nephew, Pablo and my nieces, Natalia, Brana, Hanna and Luba. They are always close to me.
I am forever grateful to all my friends for their kindness and care while I was abroad on study.

.

Acknowledgements

I am deeply indebted to my supervisors and mentors, Marcela Genero and Mario Piattini. Thanks to their training I have learnt many concepts and I have improved this dissertation and its results. Marcela, thank you so much for your generosity and all your ideas and comments!. Mario, thank you for your valuable help and time that you always share with all of us!.
I would also like to express my gratitude to all those who gave me the possibility to complete this thesis: Jose Carsi from Valencia University (Spain), Macario Polo Usaola from Castilla La-Mancha University (Spain), Cristina Cachero from Alicante University (Spain), Luis Alvarez from Austral University (Chile) for their assistance in teaching courses and running experiments.
I would like to thank all the members of Alarcos Research Group, they gave me the feeling of being at home at work. I will always be interested in collaborating with all of you. I would also like to thank all the doctorate students who worked and shared with me in the Alarcos' laboratory.
I want to thank my colleagues of the Department of Computer Science at the University of Comahue for giving me permission and support to do this post-graduated study. In particular, I am very grateful to Ing. Jorgelina Georgetti, the headmaster of the Department, who provides me her help to give many courses and running experiments in the university I belong. She has always been concerned in helping all the teachers and making the group more cohesive. I would also like to thank Alejandra Cechich and Laura Sanchez, because beside them I have learned to carry out research. Her academic supervision has been really important for me.
I wish to thank the Goverment of the province of Neuquén (Argentina) who partially gave me the financial support to study for four years.

Contents

List of Figures

List of Tables

Summary

Although the combination of UML and OCL languages provides the expressiveness needed to capture all model constraints, their combined use do not necessarily guarantee that a 'good' model will be produced. In order to assess the quality of OCL expressions within UML models, the availability at early stages of reliable indicators are essential. A huge amount of measures have been defined for measuring quality characteristics of UML models, but all of them have only been focused on the model elements expressed by diagrammatic notation, without considering OCL facilities. This thesis is focused on assessing the influence of import-coupling on OCL expressions maintainability (modifiability and comprehensibility) through a methodological definition of a set of measures for OCL expressions. This influence is indeed produced due to the fact that structural properties of OCL expressions affect the cognitive complexity of modelers, and high cognitive complexity leads to the OCL expression to exhibit undesirable external qualities such as a low maintainability. For that reason, its is important to explain the measures considering not only the structural properties of OCL expressions but the cognitive complexity of modelers through a psychological explanation for obtaining a precise definition.

In this Ph.D. thesis new kinds of coupling measures are defined in terms of core concepts of OCL: navigations, collections operations and contextual objects. Moreover, we define the measures through a thoughtful formal definition and a theoretical and empirical validation. For their theoretical validation the adaptation of existing frameworks of coupling measures was needed and is carefully presented. Regarding their empirical validation, we ascertain through three families of experiments if any relation exists between OCL measures, and two maintainability sub-characteristics. The empirical results reveal that there is empirical evidence that import-coupling is strongly correlated with the comprehensibility and modifiability of OCL expressions, and coupling, the most relevant characteristic in an object oriented system, is still a crucial aspect that should be carefully considered when OCL expressions are defined in UML/OCL combined models.

Chapter 1

Introduction

We are likely witnessing the birth of a paradigm shift, software development will shift its focus from code to models [WK03]. Within this change, the two initiatives being built under supervision of the Object Management Group (OMG), the Model-Driven Development (MDD) [AK03] and the Model-Driven Architecture (MDA) [OMG03a], consider that models are the backbone of the Object-Oriented (OO) system development, emphasizing the importance of building good models. Models will constitute the basis of software development focusing on producing a platform-independent model (PIM) and then transforming it to a platform-specific model (PSM) and code (Fig. 1.1). A PIM captures the main concepts of the domain relevant to the system creating an abstract description of some portion of the real world. This model forms the foundation for all later implementation using platform-specific constructors (PSM), so, PIM model is a major determinant of the quality of the overall OO software system design. We believe that the quality of PIM models is crucial because they could have an influence on the quality of the software product which is finally delivered. In truly model-driven software engineering the quality of the models used is of great importance as it will ultimately determine the quality of the software systems produced. In particular, it is widely believed that the system quality is highly dependent on many decisions made early in its development, specifically when artifacts and their constraints are defined. Errors introduced as early stages are usually more expensive to correct than implementation stage' errors, being desirable to prevent, detect and correct errors as early as possible in the development process [QT06].

The appearance of the Unified Modelling Language (UML) [OMG03c] as a modelling language has been a quantum leap for building higher quality PIM models. As experience with UML grew and the issues and needs of software modeling became better understood, new requirements for UML emerged. This led to the issuing of formal requests for the first major revision of the standard, greater clarity of the

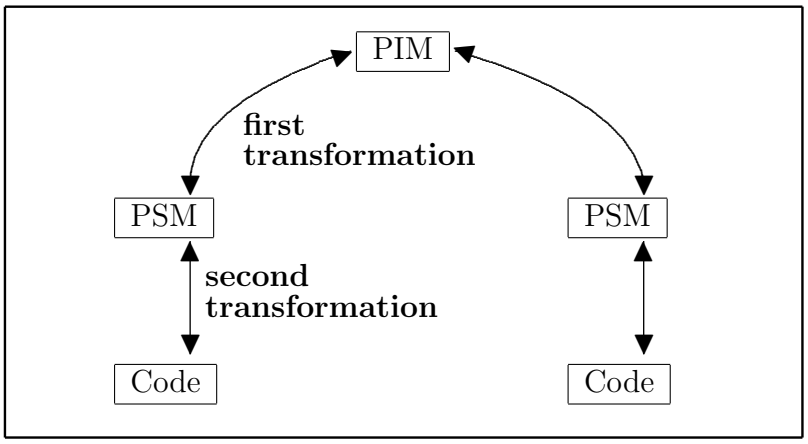


Figure 1.1: Relationship between PIM, PSM and Code

specification, and some new modeling capabilities. However designers realize that although UML has became a standard language for OO software system specification, many design decisions, constraints and essential aspects of software systems cannot be expressed in a UML diagram [WK03], [CKM+02] through diagrammatic notations. So, a new language emerged, to be used as a textual add-on to UML diagrams, the Object Constraint Language (OCL) [OMG03b], and modelers improved the quality of their UML models specifying them in a combination of the UML and OCL languages, i.e., through a UML/OCL combined model. Nowadays, OCL is recognized as an essential language in building consistent and coherent PIM models and helping to raise the level of maturity of the software process [WK03].

A UML/OCL combined model is considered a complete, detailed, consistent and precise description of a system, its OCL expressions are written referencing to the model features, constraining, querying and defining their semantic properties. Warmer et al. argue in [WK03] that without OCL expressions the model would be severely underspecified. Nonetheless, although the use of both languages, UML and OCL, is a necessary condition for obtaining solid and precise models, it is not sufficient for obtaining models of high quality because several factors influence their quality; for example low coupling is a concern that should be present in all modeling tasks [Nun03]. Coupling is one of the earliest indicators of design quality and it is considered one of the more complex software attributes in OO systems [BDW99]. Its role as an essential indicator of the dimension of the structural property has historically been described and discussed [DS05]. Within coupling it was empirically shown that the extent to which the rest of the software system depends on the software part (i.e. the export-coupling) shows a much weaker impact on development effort than the extent to which a software part depends on the rest of the software system (i.e. the import-coupling) [BW01], [BWDP00], [BWL01]. Moreover, import-

coupling has shown to be a strong, stable indicator of fault proneness of classes [BWSL99]. So, we believe that import-coupling should be carefully considered in studying OCL expressions.

Nevertheless despite of the direction of coupling, import-coupling or export coupling [Ari02], scanty information of coupling is available at early stages of software development using only UML graphical notations, and many times, many coupling decisions are made during implementation [WK03]. However, the availability of more coupling information of a model at early stages (e.g. to decide which classes should undergo more intensive verification or validation) would be useful. We believe that a UML/OCL model reveals more coupling information than a model specified using only UML, due to the fact that OCL navigation defines coupling between the objects involved [WK03], and the coupled objects are usually manipulated in an OCL expression through collections and its collection operations (to handle its elements) for defining constraints between the objects.

So, we have studied OCL expressions as a crucial add-on to the UML diagrams, due to the fact OCL gives expressiveness and coupling information to a UML model. Although expressiveness of the modeling technique used (e.g. the notation, etc.) affects one of the most important characteristics of a model, its comprehensibility [Sel03], and it was also empirically proved that OCL has the potential to significantly improve UML-based model comprehension and maintainability [BLYP04], we believe OCL expressions themselves can be difficult to write and maintain [GL05].

Software maintenance, is an important part of the software lifecycle, typically accounting for at least 50 percent of the total lifetime cost of [GB01], and maintenance depends, in part, on comprehension [CW00]. Mohan et al. consider that program comprehension is responsible for up to half the total cost of software maintenance [MG04]. Consequently, it is desirable to reduce the cost of software maintenance whilst preserving the quality of the software system, and maintainer´s comprehension [GB01].

The aim of this thesis is focussed on assessing if import-coupling has an influence on OCL expressions maintainability (mainly in modifiability and comprehensibility). In order to evaluate the influence we define a set of measures for OCL expressions. The rigorous definition of these measures was achieved by means of a method of measure definition proposed in [CPG01], which mainly consists of the following steps: measure definition, theoretical validation and empirical validation. We extended and refined this method in order to identify and create the measures in a more precise way.

The definition of measures for assessing the coupling in OCL expression maintainability constitutes an important contribution to UML/OCL modeling activities. In fact, over the past ten years there has been a wealth of literature on measures capturing the quality of UML models but none of these measures can be applied

to UML/OCL models. The measures that can be found in the literature can be applied to UML class diagrams [BD02], [BMB99], [BDM97], [CK94] use case diagrams [Mar98], [SW98], [Sae03], statecharts diagrams [Der95], [CS02], [CLGO+04], [CLGO+05], [CLGPM06]. They emphasize that the usage of measures for UML models can help designers make better decisions which is gaining relevance in software measurement area. However, there are no measures to capture quality aspects of OCL expressions.

The set of measures for OCL expressions will support the modeling activity of modelers due to the fact the availability of early indicators of the essential coupling information at early phases of the development will help the modeler to assess the quality of the model that is finally delivered.

We started to study if import-coupling information obtained from OCL expressions affects the maintainability of the delivered model, based on the hypothesis that import-coupling (a structural property [DS05]) of an OCL expression within a UML/OCL model (software artifacts) can influence the cognitive complexity of modelers, and a high cognitive complexity leads to OCL expressions exhibit undesirable external qualities [ISO01], such as less comprehensibility or reduced maintainability. This hypothesis is based on the more relevant theoretical basis for developing quantitative models where a strong effect of measures on external quality attributes exists [BWIL99], [BWSL99], [BWL01], [ISO01]. The mechanism causing this effect is assumed to be the cognitive complexity [CES01], [GEMM00] see Fig. 1.2, i.e. the mental burden of the individuals (modelers, developers, testers, etc.) who have to deal with the artefacts.

1.1 Ph.D. Thesis Framework

This Ph.D. Thesis has been developed within the following framework:

- MESSENGER means 'MEjora de los SiStemas ElectróNicos de GEstión de Relaciones'. This project is financed by the Conserjería de Ciencia y Tecnología of Junta de Comunidades de Castilla La Mancha (Ref. PCC-03-003-1). The intention of this project is the definition of set of techniques for improving relationships management of electronic systems, in order to assure its security and quality. The project goal will be pursued after the definition and validation of suitable techniques and measures, as well as the definition of a methodology and its corresponding control indicators to audit these systems.

- CALIPO stands for 'Calidad en Portales' (Quality of Portals). This project is financed by Ministerio de Educación y Ciencia (Ref TIC 2003-07804-C05-03). It started in 2003 and was completed in 2006. Portals had evolved from

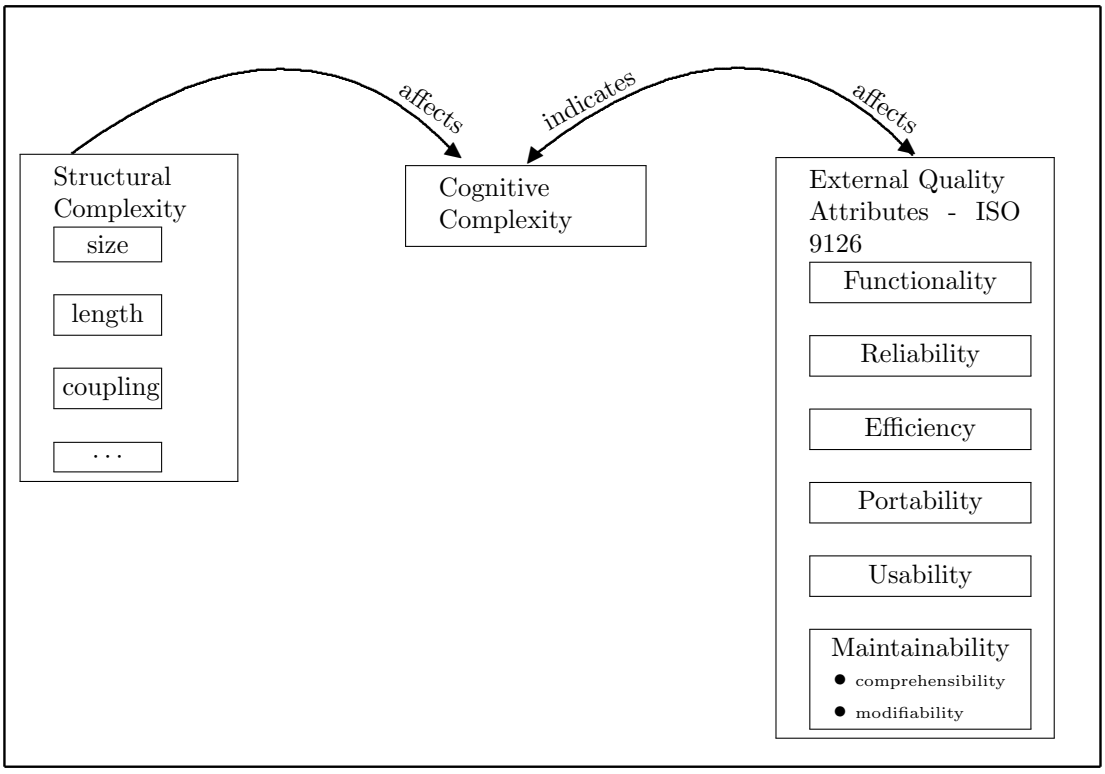


Figure 1.2: Relationship between Structural Properties, Cognitive Complexity, and External Quality Attributes [BWIL99], [BWSL99], [BWL01], [ISO01]

simple web pages providers and corporative databases to support intelligent management, application and collaborative process. The goal of this project is to assure the quality of portals, which also depends on multiple factors, which will be managed by the project. It is widely recognized that static HTML web sites which have been prevalent in Internet for the last decade have been substituted by dynamic web sites which have advanced capabilities in databases and analytic applications. The project defines a model of portal quality. Within the model are included those aspects that are inherent to a portal such as datawarehouses and databases, portal security, XML language, metadata, web services, portal component and portal customization integration.

- This Ph.D. thesis has been funded by the network VII-J-RITOS2 financed by CYTED. This project is financed by the Ministerio de Educación y Ciencia. It started in January 2001 and was completed in 2006.

- COMPETISOFT (Mejora de Procesos para Fomentar la Competitividad de la Pequenã y Mediana Industria -PYMES- del Software de Iberoamérica) project (506PI0287). This project is financed by CYTED (Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo). The aim of the project is to

increase the competitiveness level of the ibero-american PYMES which produce software through the creation and diffusion of a common methodological framework. The framework will have the facility to be applied to the specific needs and will constitute the basis to state a well-known evaluation and certification mechanism of software factories in Iberoamerica.

- ESFINGE stand for 'Evolución de Software Factories mediante INGeniería del software Empírica'. This project is financed by the Ministerio de Educación y Ciencia (Ref TIN 2006-15175-C05-05). It started in October 2006 and will be completed at the end of 2009. The project goals are: to define measures and indicators for different abstraction levels of models and software architectures, to obtain a set of measures threshold values and to develop software tools for their automatic computation. Besides, this project aims to define a framework for systems evolution and reengineering of software factories using a MDSD-based approach as well as the definition of an environment for software testing based on metamodels and the definition of a software environment for the improvement and evolution of business process models. The definition and validation of techniques and measures for the development of model-based security software (PIM, PSM, etc) is also another goal of the project.

In these projects the specification of 'good quality' models is crucial. Building a UML/OCL combined model as a PIM it is possible to produce a PSM in terms of the constructors that are available in one specific technology [WK03], such as databases, web-based systems, electronic systems (ECRM systems), etc. Clear indicators of the quality characteristics of OCL expressions within UML/OCL models will help the aforementioned projects to achieve correct and coherent specifications of the domain specific systems they deal with.

Luis Reynoso enjoyed during the development of the thesis a postgraduate grant from the agreement between the Government of Neuquén (Argentina) and YPF-Repsol.

1.2 Goal, Objectives and Hypothesis

The main goal of the Ph.D. Thesis can be defined as:

ASSESSING THE INFLUENCE OF IMPORT-COUPLING ON THE MAINTAINABILITY OF OCL EXPRESSIONS THROUGH A MEASUREMENT-BASED APPROACH

And, on the basis of the main objective, a series of partial objectives have arisen:

1. Analyse the existing measures for UML models and measures for coupling.
2. Extend and refine the method for the definition of valid measures.
3. Propose a set of measures for measuring the structural properties of OCL expressions within UML/OCL models.
4. Carry out the formal definition of the proposed measures.
5. Perform the theoretical validation of the proposed measures using the most suitable frameworks.
6. Describe the rationale of the measures using a psychological explanation from a cognitive point of view.
7. Perform the empirical validation of the proposed measures to find early indicators of OCL maintainability.

Therefore, in view of these objectives, this is the hypothesis that we propose:

IT IS FEASIBLE TO ASSESS THE INFLUENCE OF IMPORT-COUPLING ON THE MAINTAINABILITY OF OCL EXPRESSIONS THROUGH A MEASUREMENT-BASED APPROACH

1.3 Ph.D. Thesis Outline

The rest of the chapters of this dissertation will present the following content:

- **Chapter 2: Research Method.** The research method that underpin the definition of import-coupling measures is outlined in chapter 2. Each of the steps of the research method that we have applied, extended and refined for the definition of valid measures is presented in the second chapter.
- **Chapter 3: State of the Art.** This chapter attempts to address an introduction to the OCL language and its utility, and more importantly, it describes a thorough study of the existing measures for coupling and UML diagrams.

- **Chapter 4: Measures Proposal.** This chapter presents a thorough definition of measures for OCL expressions we propose, detailing the intent pursued by each measure and illustrating the calculation of their values through examples. In this chapter we provide an informal definition of the measure using natural language.

- **Chapter 5: Formal Definition of Measures.** The fifth chapter refers to the formal definition of the measures using OCL upon the OCL metamodel.

- **Chapter 6: Theoretical Validation.** The theoretical validation of the proposed measures following different approaches are applied: a property-based approach and a measurement-theory based one. In the former approach, we adapt a framework of coupling measurement before using it.

- **Chapter 7: Psychological Explanation.** This chapter gives a plausible explanation of the rationale behind the measures based on cognitive theories. This explanation is underpinned through the application of cognitive and mental models of modelers dealing with OCL expressions.

- **Chapter 8: Empirical Validation.** This chapter covers the empirical validation of the proposed measures through a detailed description of the experimental process followed in the experiments we carried out.

- **Chapter 9: Conclusions.** In this chapter we will detail the main contributions of this PhD thesis, the results obtained in the different studies performed and the research lines that are still open for further research.

- **Bibliography and Appendixes.** The bibliography used in this dissertation is listed before the appendixes. Various appendixes are presented at the end of this dissertation. The first corresponds to the acronyms and important definitions used in this dissertation. The second complements the theoretical validation of chapter 6. The third provides more details about experimentation. The fourth presents the experimental material we handed to the experimental subjects.

Chapter 2

Research Method

As Briand et al. argue in [BMB02] a large number of measures have appeared for capturing software attributes in a quantitative way. However, few measures have successfully survived the initial definition phase and are actually used in industry. The most relevant problems related to the validity of many measures [BMB02], [CD99] are summarized next:

- Measures are not always defined in the context of some explicit and well-defined measurement goal of industrial interest they help reach, e.g., reduction of development effort or faults present in the software products [BMB02].
- Even if the goal is made explicit, the experimental hypotheses are often not made explicit, e.g., what do you expect to learn from the analysis and can you believe it? [BMB02]
- Measurement definitions do not always take into account the environment or context in which they will be applied, e.g., would you use a complexity measure that was defined for non-OO software in an OO context? [BMB02]
- A reasonable theoretical validation of the measure is often not possible because the attribute that a measure aims to quantify is often not well defined, e.g., are you using a measure of complexity that corresponds to your intuition about complexity (attribute)? [BMB02]
- A large number of measures have never been subject to an empirical validation, e.g., how do you know which measures of size predict effort best in your environment? [BMB02]
- Many software attributes are still poorly understood and this often yields to fuzzy defined, not reusable measures [CD99]

The above problems are inherent to any young discipline, especially one that is human intensive. Software measurement is currently in the phase in which terminology, principles, and methods are still being defined and consolidated. The human-intensive nature of software engineering makes its measurement closer to that of the social sciences rather than that of the physical sciences. The phenomena that are studied involve a number of variables that depend on human behaviour and can not be controlled easily [BMB02].

In order to avoid many of the aforementioned problems, measures should be defined following a methodological and disciplined steps. Having in mind this idea many frameworks or methods were defined and validated, however as Kitchenham et al. remark in [KPF95] there is much work to be done to complete a framework for measurement validation, as well as to achieve consensus within the research community on the framework accuracy and usefulness. Besides, new trends in software require a continuous analysis of the frameworks, for instance consider the measurement of new software artifacts such as OCL or the formal definition of the measures upon a metamodel. A continuous analysis of the measurement frameworks and methods will allow to extend and refine a method making it more precise and robust.

In this chapter we will describe a method we used to get valid and reliable measures. In fact, the method represents an extension and refinement of a previous method, defined by Calero et al. [CPG01] and the MMLC (Measure Model Life Cycle) [CD00]. The high level steps of the previous method for defining measures were not modified, however they were refined and new sub-steps were added, such as the 'Formal Definition of Measures', 'Psychological Explanation of Measures', etc. The refinement of the method focused on studying the relationships between several steps as it is also precisely described in the following sections.

Section 2.1 describes an overview of the method whereas sections 2.2 and 2.3 describe the two more intensive steps, identification and creation. The contribution of this chapter to the dissertation is summarized in section 2.4.

2.1 Overview of the Method

We modeled the method for defining and validating measures using UML activity diagrams. So, we will consider the method as divided in several activities which should be performed to obtain reliable and consistent measures. The main activities are briefly described in this section and they represent the main steps of the original method. An overall picture of the method is shown in Figure 2.1, beginning with the definition of the measures goals and finishing with the acceptance of these measures in real projects.

The main activities of the method are:

- **Identification** (Figure 2.1, Activity M_1): This initial activity has the purpose to define the measurement goals. In order to achieve the goals questions are specified, abstractions are identified and general hypotheses are planned. Measurement goals should be clearly connected to an industrial goal, responding to the software organization needs. This premise is emphasized by the most commonly cited methods in literature for measure definition, the Goal-Question-Metric (GQM) paradigm [BW84], [BR98], [SB99] or even improvements on it, the Measurement Model Life Cycle (MMLC) [CD99] or the GQM/MEDEA [BMB02].

  The Identification activity is crucial because all the following activities will be based upon its results. In Figure 2.1 the rake in the bottom corner of M_1 activity indicates that the activity is described by a more finely detailed activity diagram, see section 2.2.

- **Creation** (Figure 2.1, Activity M_2): Using the creation activity measures are defined based on clear measurement goals, questions, abstractions and general hypotheses specified in the previous activity. The definition also includes a formal definition. Besides, measures are theoretically and empirically validated, and a plausible psychological explanation of the effort of the subjects dealing with the software artifacts being measured is provided. This activity, likely the longest and the more complex, is further broken down into four sub-activities which are explained in section 2.3.

- **Acceptation** (Figure 2.1, Activity M_3): The aim of this activity is the systematic experimentation of the measure. This is applied to a context suitable to reproduce the characteristics of the application environment, with real projects and real users, to verify its performance against the initial goals and stated requirements. After this activity is performed measures can be accepted or rejected. For that purpose, a decision node follows the Acceptation activity. The branching is based on whether measures are accepted or rejected. Even if the measure is rejected it should not be discarded but undergo the method from the creation activity.

- **Application** (Figure 2.1, Activity M_4): The accepted measure is used in real projects in industrial environments.

- **Accreditation** (Figure 2.1, Activity M_5): The goal of this activity is the maintenance of the measure, so it can be adapted to application changing environments. As the original method explains [CPG01] the accreditation activity represents a dynamic step that proceeds simultaneously with the application activity. So, a fork and a join bar in the diagram of Figure 2.1 denotes 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.

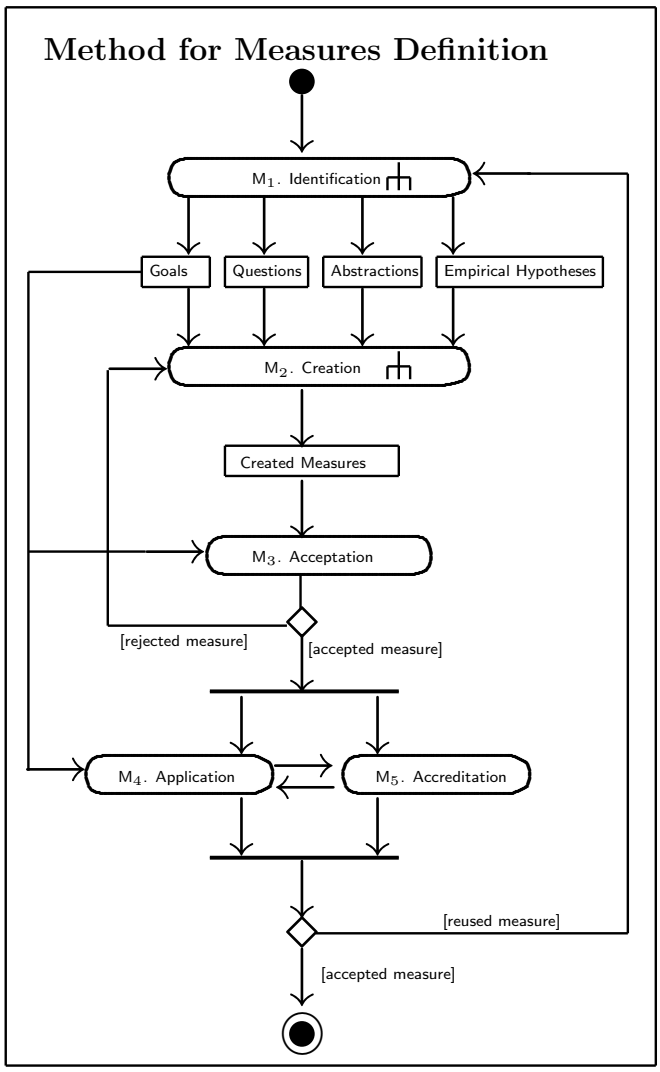


Figure 2.1: Method for Measure Definition

Hereafter, we will describe in detail the core steps Identification and Creation. Whenever an activity is explained its identification number will be show on its right.

2.2 Identification (M_1)

As we previously described the identification activity is the most important one, since it influences all other activities. The UML activity diagram for the Identifica-

tion activity is depicted in Figure 2.2.

It is advisable to be able to achieve the definition of clear measurement goals to avoid coming up with a measure definition that does not actually achieve our desired aim, i.e. we should follow a goal-oriented definition of measures. As it is described by Pfleeger in [PJCK97] a commonly used model which can guide us in deriving and applying a goal-oriented definition is the GQM paradigm. The paradigm was suggested by Basili and Weiss [BW84], [SB99] (later expanded by Basili and Rombach [BW84], [BR98]), and it is used to define appropriate measures for the organization's maturity levels and setting up a measurement program.

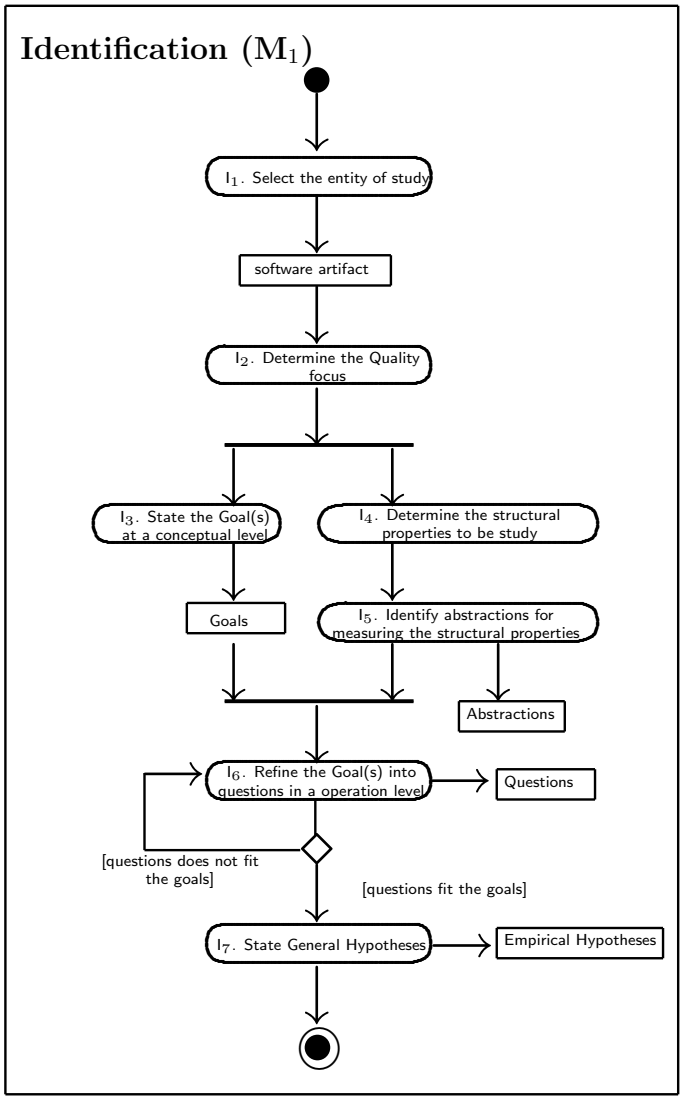


Figure 2.2: Identification Step (M_1)

This paradigm has been widely applied as a way to define what measures to col-

lect, both in empirical studies and for management [Car93]. In our case, the GQM paradigm is used in the measure definition in such a way: each measure is deduced using a top-down perspective and analyzed and interpreted using a bottom-up perspective [SB01]. Goals are stated in a conceptual level, which in turn are refined in an operational, tractable way, into a set of quantified questions [MB00]. Subsequently, measures should provide the information to answer these questions (at a quantitative level). Moreover, the GQM paradigm provides a template and guidelines to define measurement goals and refine them into concrete and realistic questions, which subsequently lead to the definition of measures [BMB02]. Although the definition of measures according to GQM questions is part of the creation activity, the selection of goals (and questions) are two of the main purpose of the identification activity. So, the GQM paradigm phases are split in different activities within the proposed method.

The GQM paradigm is the most widely used paradigm, however several authors have argued that GQM has some important limitations [Car93], [BMB02], and is not enough, by itself, to define effective measures. For instance, Card recommended that the use of GQM must be supplemented with another activity to select specific practical measures, and he also suggests that one effective supplemental activity is modeling. Developing a model, that is defining the objects being measured, makes it possible to select measures for effect rather than desire [Car93], and help us to describe the relationships between measurable things. Likewise, Briand et al. [BMB02] provides a mechanism for generating models as an extension of the GQM paradigm, called GQM/MEDEA, proposing a practical guideline to design and reuse technically sound and useful measures. In both approaches, the modelling of the measured artifact and also abstractions for the captured attribute being measured were included as complementary activities for the GQM paradigm. So, our method takes into account all these remarks from the measurement literature.

In order to define measurement goals the following activities should be performed:

- **Select the entity of study** (Figure 2.2, Activity I_1): According to ISO 9126 [ISO01] an entity is an object (for instance, a product, process, project or resource) that is to be characterized by measuring its attributes.

- **Determine the quality focus** (Figure 2.2, Activity I_2): Generally the quality focus corresponds to the quality attributes (abstract properties of a entity) used in the measurement process. In determining such attributes, quality models[1], such as the ISO 9126 [ISO01], Kim [Kim99], McCall [MRW77], Boehm [BBK78], suggest ways to describe different quality characteristics of software products, such as distinguishing usability from maintainability [FP98].

[1] A quality model according to ISO [ISO01] is the set of characteristics and relationships between them, which provide the basis for specifying quality requirements and evaluating quality.

- **State the GQM goal(s) at a conceptual level** (Figure 2.2, Activity I_3): The two previous activities are used to state the GQM Goal(s), which is (are) defined using the following template: Analyze the 'object of study' in order to 'purpose' with respect 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 respect to which focuses [Sae03].

Once the goal(s) is defined it should be refined into a set of questions. Nevertheless, before addressing the definition of questions, which in fact allows that GQM goals to be quantified, it is necessary to consider the structural properties [DS05] of the software artifact to be studied:

- **Determine the structural properties to be studied** (Figure 2.2, Activity I_4): We need to define the properties (or internal attributes) that we intend to measure because we usually interpret software data at that attribute level [KPF95]. That is, should we study the coupling, cohesion, size, length, etc. of the software products?.

- **Identify abstractions for measuring the structural properties** (Figure 2.2, Activity I_5): In helping to clearly identify the structural properties we should take into account the definition of abstractions for measuring the structural properties as recommended by Briand et al. [BMB02] and Card [Car93]. For instance in the case of coupling being 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. [BMB96].

- **Refine the goal(s) into questions at an operation level** (Figure 2.2, 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 a diamond notation in the UML activity diagram of Figure 2.2.

- **State General Hypotheses** (Figure 2.2, Activity I_7): Finally, general hypotheses should be stated relating structural properties and the quality focus. The definition of precise, testable research hypotheses are required before any empirical study be performed. An Empirical hypothesis is a statement 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 (independent) attributes of some entities (e.g. software properties or software complexity) to other (dependent) attributes of the same or different software product or activities [BMB02].

2.3 Creation (M_2)

The creation activity relies on four sub-activities. An overview of them is depicted in Figure 2.3 and it is briefly explained below. Subsections 2.3.1, 2.3.2, 2.3.3, 2.3.4 describe them in detail.

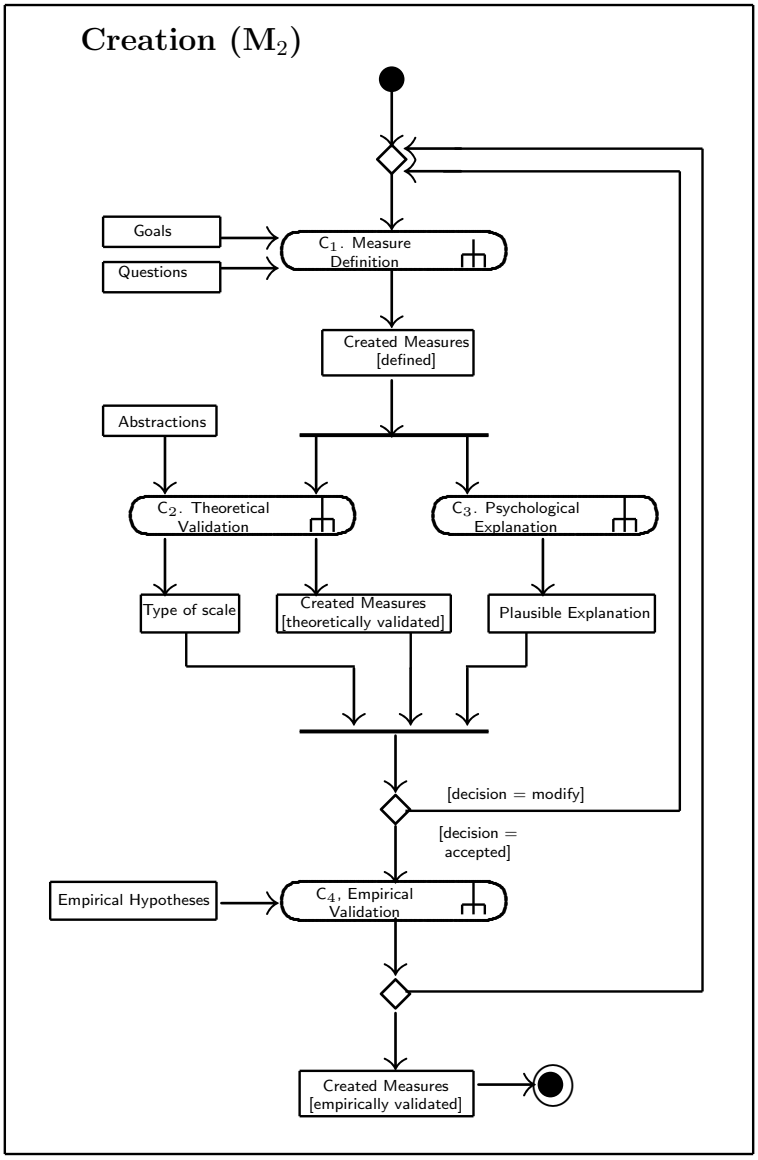


Figure 2.3: Creation Step (M_2)

- **Measure Definition** (Figure 2.3, 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 and its purpose, and a formal specification of the measure (i.e. *how* it is defined). Considering the former issue, measures are defined taking into account the goal(s) and questions provided by the identification activity.

Regarding the latter issue, in the measure´s literature different approaches were applied for defining measures: natural language, mathematical approaches, and formal languages. The measures should be defined in a consistent and coherent way to avoid misunderstanding and misinterpretation of its meaning.

- **Theoretical Validation** (Figure 2.3, Activity C_2): Once a measure has been defined it is necessary to verify whether it fulfills the properties that are associated with the attribute it purposes to measure [MB00]. 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, i.e. it proves that they are valid measures for the constructs that are used as variables in the study. The theoretical validation is also useful to determine the scale type of the measure, and helps us to know when and how to apply measures, for instance the scale of the type is useful to identify the statistical techniques which should be applied.

- **Psychological explanation** (Figure 2.3, Activity C_3): Ideally, we should be able to explain how the subjects deal with the entities that are the focus of our measurement activities. As Cant et al. [CJHS92] 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. This explanation is also useful to provide a clear interpretation of the results of empirical studies. The psychological explanation can be carried out at the same time with the theoretical validation and it is directly strengthened when qualitative methods are applied in empirical studies [Sea99].

- **Empirical validation** (Figure 2.3, Activity C_4): This activity investigates whether the measure is actually effective in practice, i.e. the study assesses whether the measures are related to some external attribute. This task is called empirical validation or external validation. This activity takes into account the empirical hypothesis provided by the identification activity. The purpose of this activity is to prove the practical utility of the proposed measures.

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 where a review of the creation activity should be performed. The first is after finishing the Theoretical Validation activity due to the fact: (1) the measure

may not theoretically valid or (2) the measure can be theoretically valid but does not capture an expected attribute (the attribute that the measure aims to quantify). The second situation is after the empirical validation is performed. Different situations can arise: a measure could not be empirically valid, several measures can capture the same dimension of a concept, derived measures need to be defined as a more precise indicator of independent variables, etc. These two situation were modeled through the bottom two diamond decisions in the UML activity diagram shown in Figure 2.3.

2.3.1 Measure Definition (C_1)

When measures are defined the most important goal is that they should provide, at a quantitative level, the information to answer the stated GQM questions. However, the activity of defining measures is not an easy task. Initially measures must be defined using natural language, then they should be formally defined. Nevertheless both activities have their own preconditions, which constraints the order in which they should be performed:

- **Select a Metamodel of the Software Artifact** (Figure 2.4 (a), Activity D_1): The definition of a measure has to be clear and detailed enough so that any concept of the software artifact (the object of study) mentioned in the natural language definition should be quantifiable, i.e able to be measured [BMB96]. To fulfil this purpose a metamodel of the software artifact being measured should be selected as a previous activity of any measure definition. As it is defined in [JA97] a metamodel constitutes the set of characteristics selected to represent a software or software piece and the set of their relationships, and they are proposed for the description of the software to which the measurement method will be applied.

  Using a metamodel we should scrutinize that any concept mentioned in the measure definition using natural language should also be an element of the selected metamodel. This step, **Selection or definition of the more suitable metamodel** was also considered in [JA97] where Jacquet et al. describe a high-level model for Measurement process.

- **Definition in Natural Language** (Figure 2.4 (a), Activity D_2): In the definition activity, it is assumed that many measures can be defined. Here, the Definition in Natural Language occurs iteratively for each measure. The activity has a rake in its bottom corner. The description as a sub-activity is modelled in Figure 2.4 (b) and explained in subsection 2.3.1.1.

- **Select a Formal Language for the Formal Definition** (Figure 2.4 (a), Activity D_3): Previous to any formal definition of measures we should select a

formal language to perform the activity. The selection of the formal language may be carried out in parallel with activities D_1 and D_2.

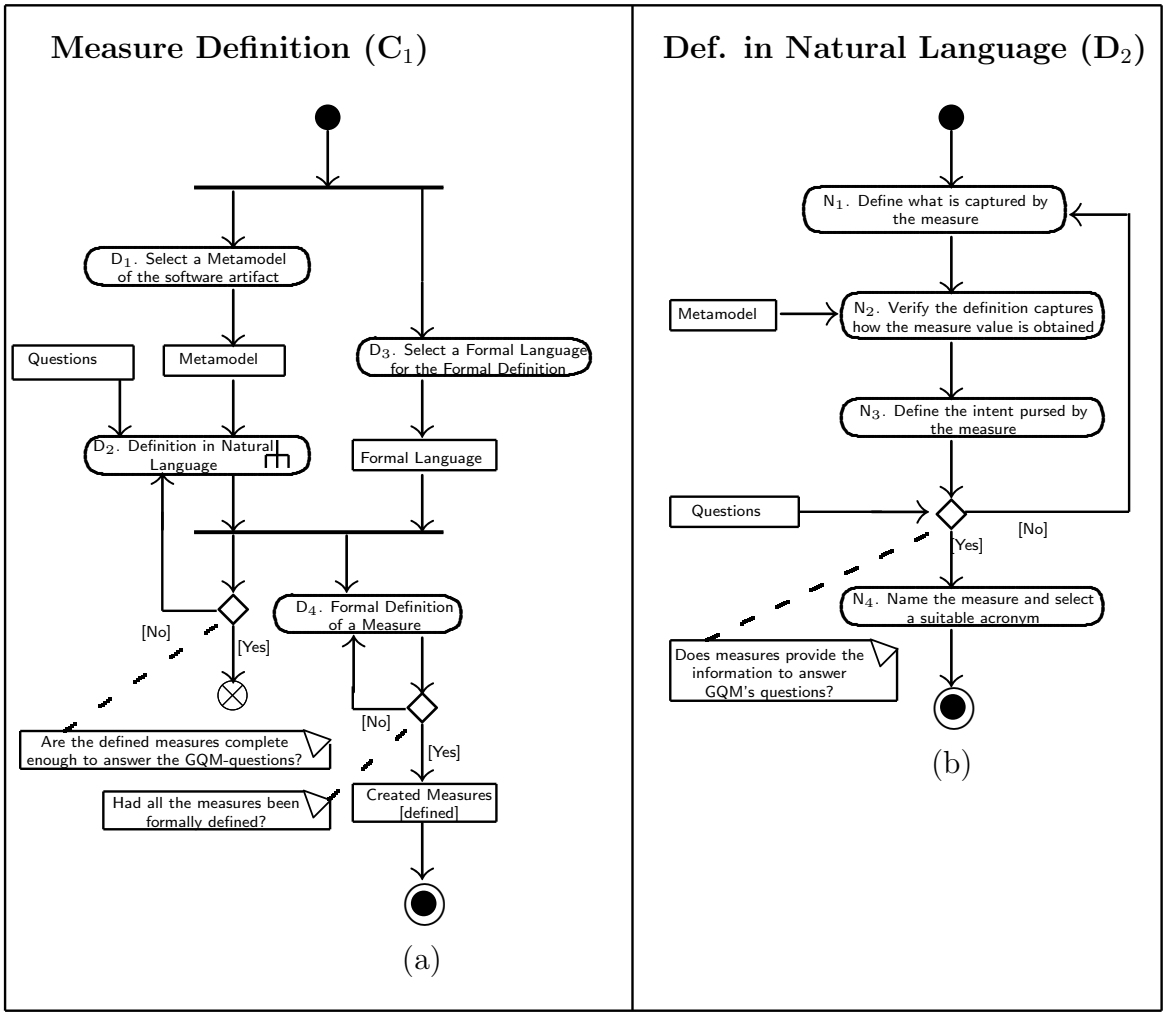


Figure 2.4: Measure Definition Step (C_1)

- **Formal Definition of a Measure** (Figure 2.4 (a), Activity D_4): We will be able to formally define a measure once (1) the first measure being defined using natural language (Activity D_2 of Figure 2.4(a)), and (2) both, metamodel and formal language were selected. These preconditions are modelled in Figure 2.4 (a) through the last join. The whole activity finishes when the last measure was formally defined, being that condition evaluated in the last diamond of Figure 2.4 (a). The formal definition of the measures should be consistent with the definition of the measure in natural language. Furthermore, the formal specification should be coherent with the natural language

description which explains the way in which measure values should be obtained. Although this activity is not further detailed in a subactivity we will explain it in section 2.3.1.2 the underlying reasons of introducing this activity as part of the method.

2.3.1.1 Definition in Natural Language (D_2)

The activity defines the measures using natural language and involves the following activities:

- **Define what is captured by the measure** (Figure 2.4 (b), Activity N_1): The definition of the measure should include a clear description in natural language of what is captured by the measure.
- **Verify the definition captures how the measure value is obtained** (Figure 2.4 (b), Activity N_2): Each concept and relationship mentioned in the definition must be quantifiable. Besides, the measure definition should describe precisely how the value of a measure is obtained.
- **Define the goal pursued by the measure.** (Figure 2.4 (b), Activity N_3): The measure intent should be consistent with the GQM question to which the measure provides information. Besides, the measure intent should be described considering the cognitive complexity of modelers dealing with the aspects and concepts captured by the measure. Whether the measure 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 2.4 (b) and it verifies that each measure intent is aligned with the GQM-questions.
- **Name the measure and select a suitable acronym** (Figure 2.4 (b), 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 in order to answer different GQM-questions. And it is also possible that a set of measures can be used to answer a GQM-question. This set should be complete enough to answer that specific GQM-question. So, the method allows 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 modelled in the left diamond of Figure 2.4 (a).

Applying the GQM paradigm we ensure that the obtained measures are useful, simple and direct. However the paradigm is not intended to define measures at a level of detail suitable to ensure that they are trustworthy, in particular, whether or not

they are repeatable (i.e., whether the measurement of an attribute was repeated by a different person the same result would be produced each time) [KPF95]. In order to ensure repeatability, software measures need to be fully defined and specified, not simply named. This is one of the purpose of a formal definition of measures.

2.3.1.2 Formal Definition of a Measure (D_4)

The purpose of this activity is to formally define the measures. Many difficulties arise when the measure is defined in an unclear or imprecise way, for example Baroni [Bar02] remarks:

- experimental findings can be misunderstood due to the fact that they may be not clear what the measure really capture,
- measures extraction tools can arrive at different results. Kitchenham et al. remark [KAKB+06] that most data collection problems arise from poor definitions of software measures. Data validation, data storage and data analysis problems arise.
- and experiments replication is hampered.

These are also common problems when we evaluate or consider the methods used in defining measures. Most of the existent measures differ in the degree of formality used in their definition. Two extreme approaches were used, informal and rigorous definitions. However none of these approaches have been widely accepted. On the one hand, measures using an informal definition, such as measures defined in natural languages, may be ambiguously defined, and everybody knows that the use of this practice could introduce 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 [CK94], [HS96]. This approach was not popular because the majority of members of OO community may not have the required background to understand the underpinning of the complex mathematical formalism used. An example of how the use of natural language introduces ambiguity in the measure definition is considered in [Bar02] referring to the definition of the measure Number of Times a Class is Reused, proposed by Lorenz and Kidd [LK94]. This measure is defined as the number of references to a class. We agree with Baroni et al. [Bar02] that it is not clear what references are and how the measure should be computed, and many questions arise 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?.

An important contribution to solve these 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 example, any measure defined for UML features can use this approach, we can provide a formal definition of the measures using OCL upon the UML metamodel.

2.3.2 Theoretical Validation (C_2)

As previously described, the theoretical validation is carried out to assess whether a measure actually measures what it claims to measure. In other words, it shows that a measure is really measuring the attribute it is purporting to measure [BEM95].

There are two main tendencies in measures validation which represent the most widely knowledged frameworks to be applied. Both tendencies constitute the more important activities in the theoretical validation and are modelled in Figure 2.5:

- **Use Property-based Frameworks** (Figure 2.5, Activity T_1): Weyuker [Wey88]; Briand et al. [BMB96], Morasca and Briand [MB97].
- **Use Frameworks based on Measurement Theory** (Figure 2.5, Activity T_2): Poels and Dedene [PD00]; Zuse [Zus97]; Whitmire [Whi97].

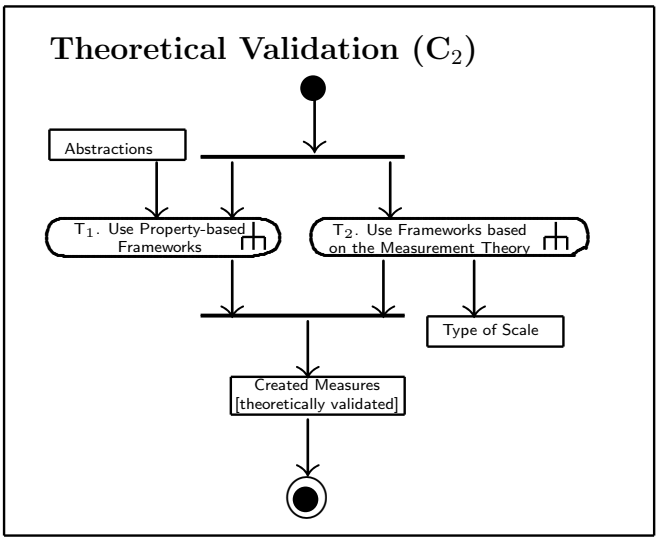


Figure 2.5: Theoretical Validation Step (C_2)

The use of property-based frameworks is not in contradiction with measurement theory [BMB02]. Similarly the measurement theory is not in contradiction with property-based frameworks. So, the activity of the theoretical validation using different frameworks can 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 in its right side a rake meaning that they are further broken down, and are explained in following two subsections.

Property-based approaches propose a measure property set that is necessary but not sufficient ([BMB96]; Poels and Dedene [PD00]). They can be used as a filter to reject proposed measures [KS97], but they are not sufficient to prove the validity of the measure.

2.3.2.1 Property-based Frameworks (T_1)

Two situations arise when choosing the more suitable property-based framework to be use for theoretical validation:

- There are frameworks that can easily be applied to software entities of different level of design [BMB96]. However others can be applied to high level design-software such as [BMB99]. So, we should take into account the software engineering stage where measurement is applied.
- The theoretical validation is close related to the I_3 and I_4 activities of the identification step (Figure 2.2). In fact, using the framework we should be able to prove the measure captures the attribute that it claims, i.e. the attribute of the structural properties (coupling, cohesion, etc.) as it is considered in [DS05] should be evaluated according to a set of mathematical properties to assure that the measure is valid.

Any application of a framework for a theoretical validation of a measure requires:

- The instantiation of the theoretical framework.
- The verification of the mathematical properties of the measure using the previous instantiation.

However, not always a framework is straightforwardly applied. Sometimes a framework is sufficient for our needs, but other times we must extend a framework in order to handle more complex situations or more complex software artifacts. For example, the application of the Briand et al.' property-based framework [BMB96] includes the definition of what is considered a system, a module, a modular system, etc. This is what we call framework instantiation in our method. Sometimes this correspondence or mapping is straightforward, and the verification of the generic properties of the framework is clear. However, we must be sure that such instantiation does not hide any problem [BMB02]. Sometimes, it is necessary that the set of properties associated with an attribute (e.g. coupling) be expanded adding new

properties, which formalize the additional knowledge about the characteristic of the measure for that attribute in a specific context, allowing to adapt the attributes of the generic properties to a specific quality focus. These properties which are believed true in a measurement context are called context-dependent properties [BMB02]. It is only necessary to define context-dependent properties when these properties are not implied by generic properties. For example, Briand et al. [BMB99] define generic properties (appendix B, section B.1.1 defines the generic properties for size and length) whereas Briand et al. [BMB96] define context-dependent properties for coupling in an object-oriented context for a ADA' high level designs (see appendix B, section B.1.2). These two possibilities, the application of generic properties or context-dependent properties are modelled in Figure 2.6:

- **Generic Properties**: P_1 and P_2 activities correspond to the application of generic properties:
  - **Instantiation of the Framework** (Figure 2.6, Activity P_1).
  - **Validate a Measure using Generic Properties** (Figure 2.6, Activity P_1). The activity consists of the verification of the mathematical properties of the measure using the previous instantiation.

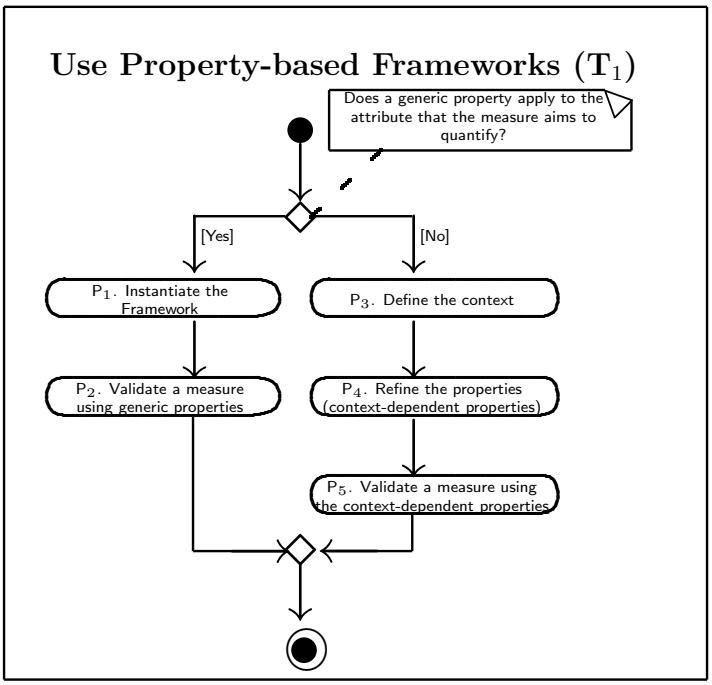


Figure 2.6: Theoretical Validation with Property-based Frameworks (T_1)

- **Context-dependent Properties**: P_3, P_4 and P_5 activities are used in the case where context-dependent properties have been defined:

- **Define the context** (Figure 2.6, Activity P_3): It includes the definition of the context where the properties will be applied.
- **Refine the properties** (Figure 2.6, Activity P_4): It defines the refinement of the properties (i.e. context-dependent properties).
- **Validate the measure using the context-dependent properties** (Figure 2.6, Activity P_5): It implies the validation of a measure using these refined properties.

2.3.2.2 Frameworks based on the Measurement Theory (T_1)

As we previously described there are many frameworks grounded in Measurement theory [KLST06], [Rob79]. In this subsection we briefly describe the DISTANCE framework of Poels and Dedene [PD00]. A detailed explanation of the framework is included in appendix B, section B.2.

2.3.2.2.1 DISTANCE Framework This theory, originating from the discipline called Philosophy of Science, is a normative theory prescribing the conditions that must be satisfied in order to use mathematical functions as 'measures'. Measurement theoretic approaches to software measure validation, such as DISTANCE, propose methods to verify whether these conditions hold for software measures.

The measure construction procedure prescribes five activities. The procedure is modelled by the activity diagram of Figure 2.7 and it is triggered by a request of defining a measure for a property that characterize the elements of some set of objects. The activities are the DISTANCE procedures which are briefly summarized below. For notational convenience, let P be a set of objects that are characterized by some property *pty* for which a measure needs to be constructed.

- **Find a Measurement Abstraction** (Figure 2.7, Activity MT_1): The objects of interest must be modelled in such a way that the property for which a measure is needed is emphasized [Zus97]. A suitable representation, called measurement abstraction hereafter, should allow to what extent an object is characterized by the property to be observed. By comparing measurement abstractions, we should be able to tell whether an object is more, equally, or less characterized by the property than another object. The outcome of this activity is a set of objects M that can be used as measurement abstractions for the objects in P with respect to the property *pty*. Let *abs*: $P \rightarrow M$ be the function that formally describes the rules of the mapping.
- **Define Distances between Measurement Abstractions** (Figure 2.7, Activity MT_2): This activity is based on a generic definition of distance that

holds for elements in a set. To define distances between elements in a set, the concept of an elementary transformation function is used. This is a homogeneous function on a set representing an atomic change of an element in some prescribed way. By applying an elementary transformation function to an element of a set, the element is transformed into another element of the set by changing it. Moreover, this change is atomic, meaning that it cannot be subdivided into a series of 'smaller' changes. Elements of a set can be changed in multiple ways. The second activity of the DISTANCE procedure requires that a set T_e of elementary transformation functions be found that is sufficient to change any element of M into any other element of M. If such a set T_e is found, then the distance between two elements of M is defined as a shortest sequence of elementary transformations (i.e. applications of the elementary transformation functions in T_e) to transform one element into the other. In general, there are multiple sequences of elementary transformations to go from one element to another. For the notion of distance used by the procedure, only the shortest sequences are taken into account, i.e. those that require the minimum number of elementary transformations.

- **Quantify distances between Measurement Abstractions** (Figure 2.7, Activity MT_3): This activity requires the definition of a distance measure for the elements of M. Basically this means that the distances defined in the previous activity are now quantified by representing (i.e. measuring) them as the number of elementary transformations in the shortest sequences of elementary transformations between elements. Formally, the activity results in the definition of a measure (in the mathematical sense) δ:M $\times$ M $\rightarrow \Re$ that can be used to map (the distance between) a pair of elements in M to a real number.

- **Find a Reference Abstraction** (Figure 2.7, Activity MT_4): This activity requires a kind of thought experiment. We need to determine what the measurement abstraction for the objects in P would look like if they were characterized by the theoretical lowest amount of *pty* (again, on condition that the property is a quantity). If such a hypothetical measurement abstraction (i.e. an object in M) can be found, then this object is called the reference abstraction for P with respect to *pty*. The idea of DISTANCE is now to use this reference abstraction as a reference point for measurement. More particularly, DISTANCE uses the distance between the measurement abstraction of an object p in P and the reference abstraction of P as a formal definition of the property *pty* of the object p. This means that the larger the distance between the measurement abstraction and the reference abstraction, the more the property characterize the object (i.e. the greater the amount of *pty* in p). Let ref: P $\rightarrow$ M be the function that describes the rules of the mapping.

- **Define a measure for the property** (Figure 2.7, Activity MT_5): The final

activity consists of defining a measure for *pty*. Since properties are formally defined as distances, and these distances are quantified with a measure function, the formal outcome of this activity is the definition of a function μ: P $\rightarrow$ $\Re$ defined such that $\forall$ p $\in$ P: μ(p) = δ(*abs*(p), *ref*(p)).

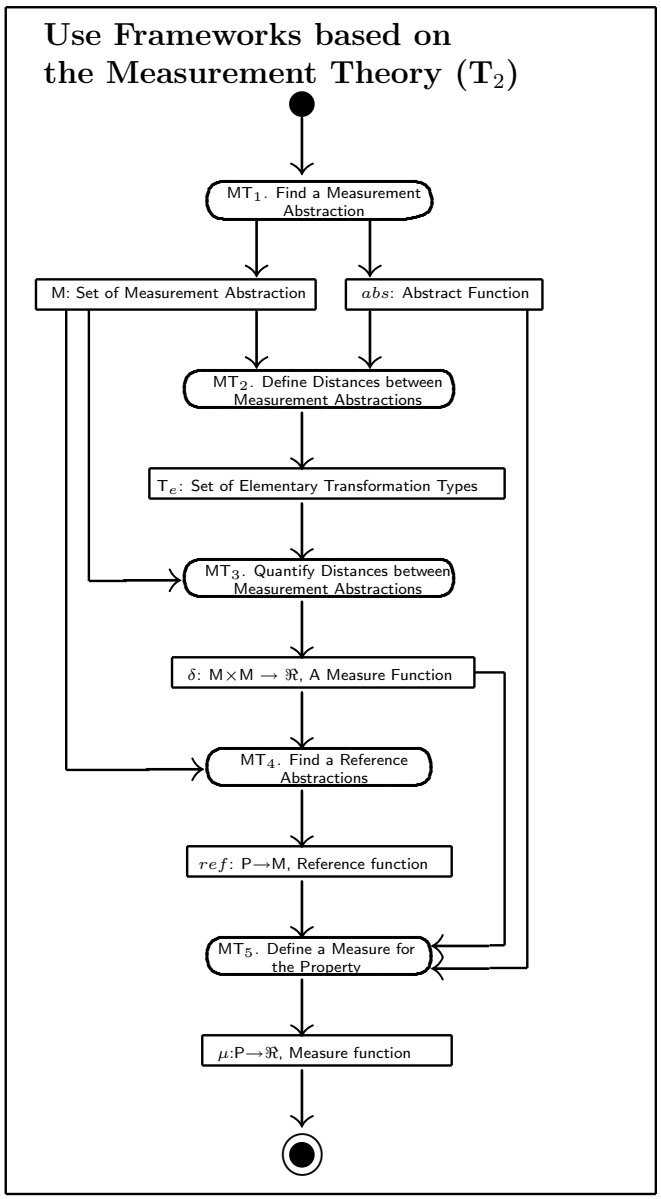


Figure 2.7: Theoretical Validation with DISTANCE Framework (T_2)

2.3.3 Psychological Explanation (C_3)

Structural properties of software artifacts influence the cognitive complexity of software engineers dealing with the artifacts [BWIL99], [BWSL99], [BWL01]. Cognitive complexity is defined as the mental burden of a person dealing with a software artifact. So, we believe that the mental burden will impact on the software quality attribute that is currently studied as a GQM-goal. The understanding of the cognitive complexity of modelers dealing with software artifact has two benefits: (1) it is useful to define the rationale behind each measure definition, indeed some traditional measures are supported by the fact that they are clearly related to cognitive limitations [Kle00], (2) will provide us the theoretical knowledge to explain the obtained results in empirical studies.

So, 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 [GEMM00] is necessary, otherwise as it is argued in [SB82] we only surface features of the software measured. It has been argued that a detailed cognitive model is a necessary basis for developing software product measures [DS05]. Understanding cognitive psychology theories, we could justify the influence of structural properties on external quality attributes (such as maintainability) through the study of cognitive complexity. In fact as Darcy et al. argue in [DS05] the consideration of multiple theoretical perspectives, including human cognition, provides a solid foundation upon which to derive an integrative model relating internal and external attributes of software quality.

Glasberg [GEMM00] argues that one way to operationalize cognitive complexity is to equate it with the ease of comprehending the software artifact that is measured. Moreover, if we are able to describe and to understand how software engineers comprehend the software artifacts that are measured, we will be able to interpret and to analyze the empirical studies performed with subjects dealing with those artifacts.

Cognitive models and mental models are two important theoretical basis for program comprehension. Nevertheless, as Darcy et al. argue [DS05], some of the programming comprehension models are sufficiently generalizable so that they can also be used to understand and explain maintenance cognition.

We had identified the following activities in order to obtain a plausible explanation:

- **Select the Cognitive Theory to Use in a Plausible Explanation** (Figure 2.8, Activity PE_1): The selection of a cognitive psychology theory should be carefully justified, and the selection will be dependent on the software artifact (object of study) to be measured and the GQM-goal pursued in the measurement process.

- **Relate the Cognitive Theory to the Software Artifact and Measures** (Figure 2.8, Activity PE_2): Once the cognitive theory is selected and each of its elements are described, it is useful to use the cognitive theory to explain how the subjects deal with the measured artifacts 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 2.8, Activity PE_3): Human cognition become more relevant considering that in last years software engineering empiricists are beginning to address the human role in software development in a serious way. Seaman argues [Sea99] that in order to delve into the complexity of human role in software engineering rather than abstract it away, qualitative methods should be used. It could be argued that human behaviour is one of the few phenomena that is complex enough to require qualitative methods to study it. Nevertheless we have included an activity PE_3 in which qualitative methods should be applied in order to really understand the cognitive complexity of software engineers dealing with a measured software artifact. A thorough study about qualitative methods for data collection and analysis which may be incorporated into empirical studies of software engineering is presented in [Sea99]. The most common qualitative methods employed are observations, in-depth interviews and focus groups [TB84].

Due to the fact that cognitive complexity constitutes one of the most important aspects that underpin the influence of structural properties on external quality attribute in the following subsection we explain it in detail.

2.3.3.1 The Cognitive Complexity

Cognitive complexity is defined as the mental burden of the individuals (developers, testers, inspectors, maintainers, etc) who have to deal with the component [EMM01]. The concept of cognitive complexity is crucial in measuring software products, because many empirical work hypothesize that high cognitive complexity leads to a component exhibiting undesirable external qualities, such as reduced maintainability [Ema01].

In general, the cognitive complexity occurs in the context of finite short-term memory (STM), long-term memory (LTM) which is practically unlimited in capacity though fades over time, and long-term working memory (LT-WM) which is used to represent acquired expertise [KR02]. When processing information in general, concepts are first stored in STM. Groups of related concepts are grouped into chunks and stored in LTM. Depending on the familiarity of information encountered, the

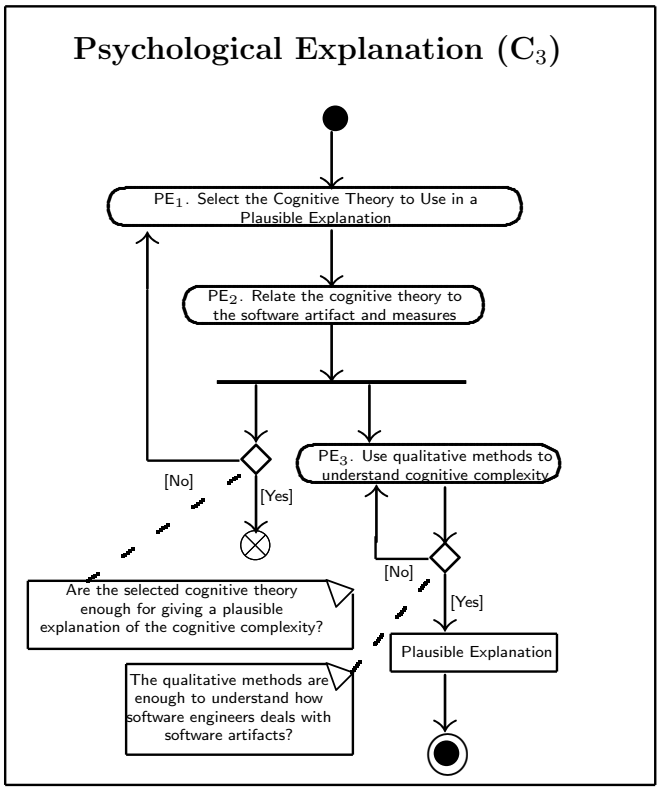


Figure 2.8: Psychological Explanation (C_3)

experience with the domain comes into play (LT-WM), determining ones ease of comprehension. Here we describe, salient aspects of STM, LTM and LT-WM:

- STM capacity was measured by Miller at 7 +/- 2 concepts, however perfect recall was only achieved 50% of the time [Mil56]. Broadbent later found that 4 was the greatest number of unfamiliar concepts that could be reliably recalled [Bro75]. One consequence of STM limitations is that text that contains a high concentration of unfamiliar concepts usage will be more difficult to correctly understand [Gop91]. On the other hand, when a concept is repeatedly recalled, it becomes easier to recall again and can be easily assimilated.

- LTM consists of concepts and associations between concepts represented as schemes or patterns. The performance of LTM depends on the density of associations between concepts therein. LTM fades over time. Moreover, activities that rely on remembering something that is seldom repeated is inherently risky. While a dense association between concepts in LTM improves LTM performance [EK95], a dense association between concepts in text hinders absorption when the concepts or associations are unfamiliar [Gop91].

- LT-WM is used to represent cognitive skill or expertise. When an individual has many years of problem solving experience in the same domain, they can quickly solve problems that normally take much longer [EK95]. One of the consequences of expertise is that individuals highly familiar with obscure knowledge or methods will perform much more efficiently in their domain of expertise than novices. This makes subjective evaluations of methods and procedures more difficult to assess when the holder of the opinion is very familiar with them [KR02].

The concept of knowledge of the limitations of the human information processing capacity has also been used as a helpful underpinning for establishing measures threshold values. Therefore, several authors have argued that when the interacting concepts overflow short-term memory, this will lead to an increase in external quality aspect, for instance comprehensibility. The implication of this is that a certain amount of concepts does not affect cognitive burden, until a non-zero threshold is exceeded. This was the reasoning applied by El Emam in [Ema01] for coupling measures. However, the issue of OO thresholds has been discussed and evaluated in [BEGR00], [EBG+02], and as expected, no evidence supporting thresholds was found [GEMM00].

Nevertheless, cognitive complexity is the main factor determining the comprehension of software artifacts, such source code, UML models, or even OCL expressions.

2.3.3.2 Selected Theory for OCL Cognitive Complexity (PE_1)

In this section we present the theory we used in a plausible psychological explanation of the measures we presented in chapter 4. Due to some traditional measures are supported by the fact that they are clearly related to cognitive limitations [Kle00] we started the explanation considering the concept of cognitive complexity and the capacity of human memory. However as our hypothesis is that import-coupling, as a structural property, influences the cognitive complexity of the modelers during OCL expressions comprehension in maintainability of OCL expressions, we based our reasoning on the comprehension of OCL expressions using two main topics: *mental models* and *cognitive 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 from the mental model [Sto05]. Understanding cognitive psychology theories, for instance to explain which are the mental model of modelers during comprehension or the cognitive process of modelers, we could justify the influence of import-coupling (a structural property) on the maintainability of OCL expressions.

During the comprehension of OCL expressions within UML models many cognitive

dimensions are opportunistically used, and their theory will be also discussed in this section.

Siau [Sia99] proposes the use of cognitive psychology as a reference discipline in method engineering and the study of information modeling. In general the understanding attributes of cognitive process can lead to new software measures that allow the prediction of human performance in software development [RK03] for assessing and improving the maintainability of software artifacts.

2.3.3.2.1 Cognitive Complexity Model

In order to carefully 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. [CJHS92] Cognitive Complexity Model (CCM). In this section we will first explain the CCM for software complexity which give a general framework to explain the process of OCL comprehension and we apply it in chapter 6.

The CCM was defined by Cant et al. [CJHS92] and gives a general cognitive theory of software complexity that relates to the impact of structure on understandability [Ema01]. Although the study of Cant et al. [CJHS92] has been considered a reasonable point of departure for understanding the impact of structural properties on understandability of code and the coding process, we believe that this model can also be applied to UML developers when they try to understand OCL expressions. The basis of the CCM is the definition of two cognitive techniques applied in program comprehension. Besides, the cognitive complexity model can be described qualitatively in terms of a landscape model.

2.3.3.2.1.1 Cognitive Techniques The underlying rationale for the CCM argues that comprehension consists of two techniques or processes: chunking and tracing, that are concurrently and synergistically applied in problem solving. Cant el al. [CJHS92] argue that both techniques have implication for software complexity:

- **Chunking technique**: a capacity of short term memory, involves recognizing groups of statements (not necessarily sequential) and extracting information from them which is remembered as a single mental abstraction: a chunk [CJHS92].
- **Tracing technique**: involves scanning, either forward or backward, in order to identify relevant chunks [Ema01], resolving some dependencies.

Cant et al. [CJHS92] argue that it is difficult to determine what constitutes a chunk since it is a product of semantic knowledge. For our purposes we will consider an OCL expression as a chunk unit, whilst the comprehension of an operation,

an attribute or a relationship along with their associated OCL expressions are also considered chunks. Henderson-Sellers [HS96] notes that tracing disrupts the process of chunking. Tracing has been observed as a fundamental activity in program comprehension [RK03]. The comprehension of a particular chunk is the sum of three components: (1) the difficulty of understanding the chunk itself; (2) the difficulty of understanding all the dependencies on the chunks upon which a particular chunk depends, and (3) the difficulty of tracing these dependencies to those chunks [Ema01]. Tracing is applied when a method calls for another method to be used in a different class, or when an inherited property needs to be understood [Ema01]. UML modelers or developers also commonly perform these cognitive techniques during the understandability of OCL specifications.

2.3.3.2.1.2 Landscape Models In the CCM model Cant et al. explain that when a programmer is primarily chunking, there are control and variable dependencies that, to be resolved, require the programmer to perform a certain amount of tracing forward or backward to find the relevant sections of code. Having found that code, programmers will once again chunk to comprehend it. Conversely, when programmers are primarily tracing, they will need to chunk to understand the effect of the identified code fragments. The effects of chunking and tracing difficulty on complexity can be demonstrated graphically by modeling the various programmer tasks as landscape (see Figure 2.9). In this visualization of the chunking and tracing cognitive processes, each chunk is delineated by a pair of markers at a single level.

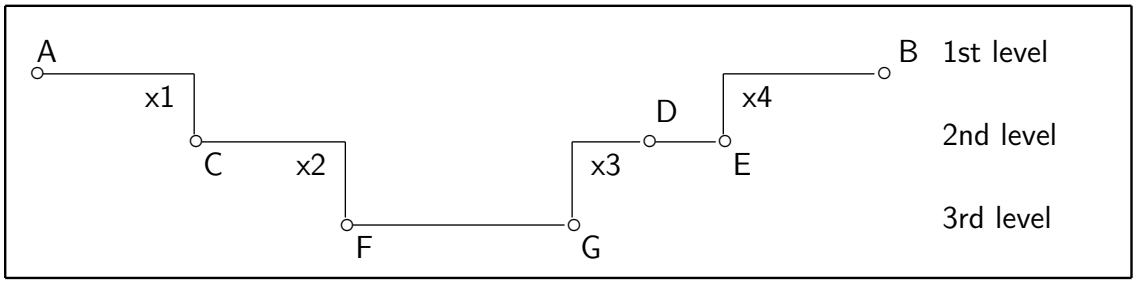


Figure 2.9: Landscape Model of Program Comprehension

For example, in Figure 2.9, at the top-level there is a single chunk visible, delineated by the two markers (A and B); at the second level there are two chunks delineated by the two pairs of markers (C,D and D,E); and at the lowest level there is a single chunk: F,G. Note that although the chunk CD is interrupted by a lower-level chunk, its integrity remains as a result of its semantic integrity. The complexity of the top-level chunk is thus represented by the sum of the two line segments Ax1 and x4B; the overall system complexity being visualized by the total distance between the end points of the chunk: A and B.

The 'vertical drop' (e.g., x1C) represents visually the work required in tracing the relevant code section. The length of time and amount of work required to solve the dependency is a function of the aggregate *depth* and *breadth* of the dependency valley. The total depth of the *nested valley* depends on the length of the chain of dependencies that must be traced to satisfy the programmer´s enquiry, and the difficulty of performing the tracing involved in each link of the chain. In addition, the number of 'steps' involved indicates the number of chunks that need to be considered. The total breadth of the nested valleys is determined by the effort required to understand each chunk in the dependency chain.

2.3.3.2.2 Mental Models

A *mental model* is a *predictive representation of real world systems*. In other words, people create internal representations of objects and information in the world, and they use these mental representations to reason about, explain, and predict the behavior of external systems [RLW04]. So, mental models (also referred to as schemes) play an important role in software comprehension and correspondingly in comprehension-related tasks, such as modification and debugging. Many studies were performed largely with procedural applications [CW99b], [CW00]. However, recently a number of studies have tested elements of the above mental model structure on OO software.

Software comprehension is an essential part of software maintenance because software that is not comprehended cannot be changed [RW02]. It is also considered as an important part of the entire software engineering [KM02].

In this section we describe the more important dimensions of program comprehension which were used as a basis to explain the comprehension of OCL expressions within UML models. Mental models are part of one of these dimensions, the dimension of direction of comprehension.

A summary of the scope and direction dimensions and the most important mental model representations are shown in Figure 2.10.

2.3.3.2.2.1 Dimension of Scope of Comprehension The scope of comprehension refers to the breadth of familiarity with the program gained by the programmer during comprehension activities [CW00]. Littman et al. [LPLS87] found two strategies used by programmers concerning scope of comprehension, **systematic** and **as-needed**. Erdos et al. [ES98] propose **partial comprehension** as the only feasible approach when systems are large or when deadlines have to be met.

Using the systematic strategy, the programmer attempts to gain a broad under-

standing of the program before carrying out modifications. The goal is to understand the design of the original programmer so that modifications fit with the existing code. On the other hand, using the as-needed strategy, the programmer attempts to minimize the amount of code that has to be understood. The programmer does not attempt to understand the overall design of the program but concentrates instead on the functioning of selected local parts of the code that are critically involved in the modification. Littman et al. [LPLS87] found that programmers who used the systematic approach carried out modifications more successfully. The authors argue that programmers using the systematic strategy were more successful because their systematic study increased their ability to detect interactions between the code central to the modification and code elsewhere in the program [CW00]. Nevertheless, maintenance programmers are in practice faced with enormously complex programs that cannot be understood in their entirety. Erdos et al. [ES98] argue that maintenance programmers only comprehend those sectors affected by the maintenance request. This cognitive activity is called partial comprehension.

2.3.3.2.2.2 Dimension of Direction of Comprehension Rilling et al. explain in [RK03] that traditionally the following cognitive models are used to group approaches of how programmers comprehend software. Three well-known strategic approaches are: the bottom-up, top-down[2] and opportunistic cognitive models.

- The **bottom-up** theories of comprehension propose that understanding is built from the bottom up, by reading source code and then mentally chunking or grouping these statements into higher level abstractions. These abstractions are aggregated further until a high level understanding of the program is attained [SFM99]. Three bottom-up cognitive models are, the work of Shneiderman and Mayer´s cognitive framework [SM79], the Pennington´s model [Pen87], and the Burkhardt et al.´s model [BDW02]. These models are described in detail below.

- The **top-down** approach is applying a goal-oriented approach by utilizing domain/application specific knowledge that is used to identify parts of the program that are relevant to achieve the goal in leading to the identification of the relevant source code artifacts.

- While the top-down and bottom-up models have been very influential, today mixed models of program comprehension are increasingly studied, showing that software engineers switch between these different models depending on the problem-solving task [SV98], [MV96]. This **opportunistic** approach can be described as exploiting both top-down and bottom-up cues as they become

[2]Both top-down and bottom-up comprehension models have been used in an attempt to define how a software engineer understands software systems.

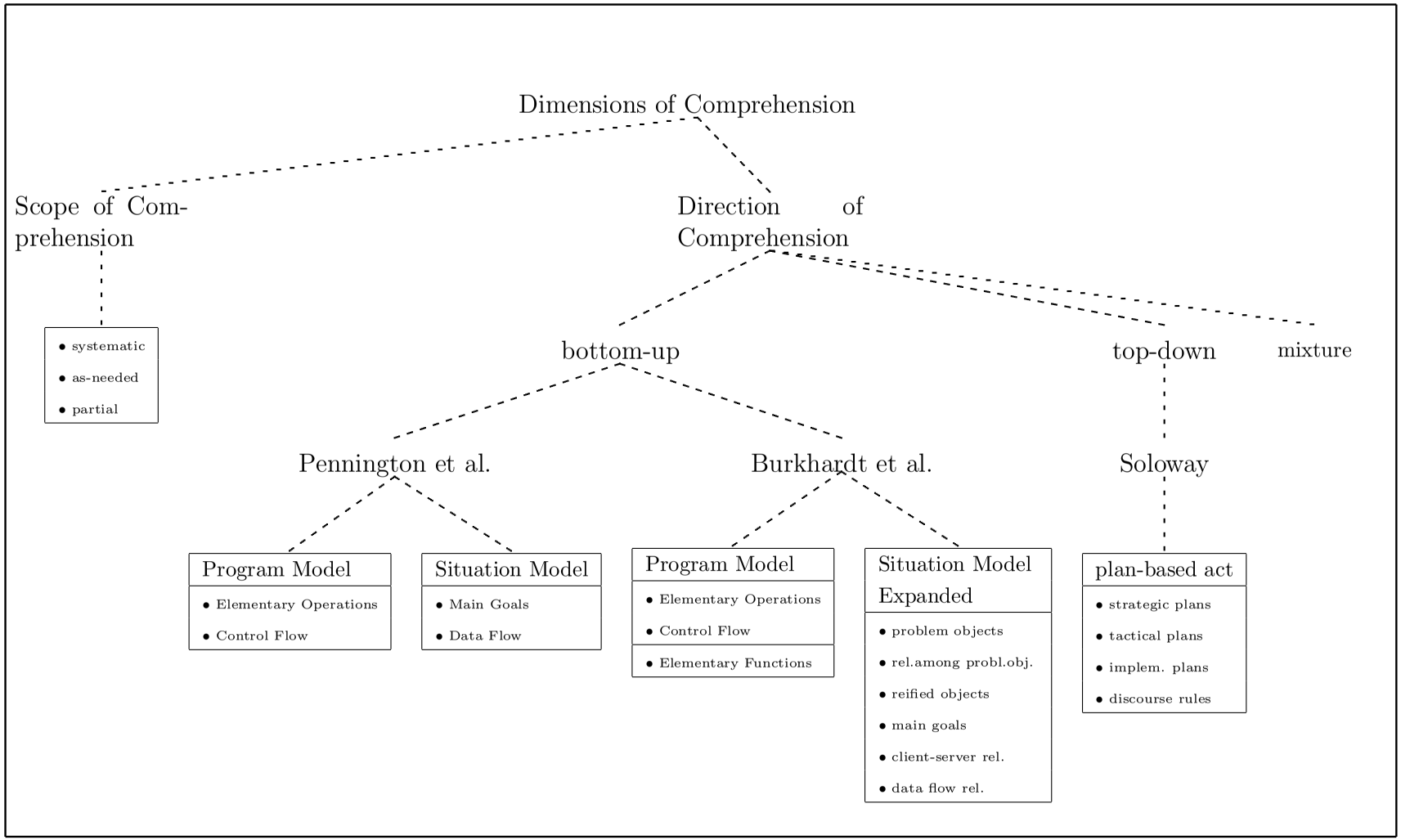


Figure 2.10: Dimensions of Comprehension in Mental Models

available. Mayrhauser and Vans [MV96] observed that the use of bottom-up and top-down comprehension varies with the domain knowledge and language knowledge. If the former is higher programmers take a top-down approach, if the latter is low programmers are more bottom-up in comprehension.

- **Bottom-up Direction of Comprehension**: Within the work of Shneiderman and Mayer´ cognitive framework [SM79] the authors differentiate between *syntactic* and *semantic knowledge of programs*. Syntactic knowledge is language dependent and concerns the statements and basic units in a program. Semantic knowledge is language independent and is built in progressive layers until a mental model is formed which describes the application domain. The final mental model is acquired through the chunking and aggregation of other semantic components and syntactic fragments of text.

  In the Pennington´s model, the author researched the role of programming knowledge and the nature of mental representations in program comprehension. She observed that programmers first develop a control- flow abstraction of the program which captures the sequence of operations in the program. This model is referred to as the *program model* and contains text structure knowledge. Pennington defined text structure knowledge in terms of the microstructure of the program text: elementary operations, generally consisting of single lines of code, and control flow between these operations. Once the program model has been fully assimilated, the *situation model* is developed. The *situation model* encompasses knowledge about data-flow abstractions and functional abstractions (the program goal hierarchy, the function of the program). The development of the situation model requires knowledge of the application domain and is also built from the bottom-up.

  The existence of these two program abstractions during comprehension, (the program and situation model) and also the formation of the program model before the domain model was also studied by Bergantz and Hassell [BH91].

  Burkhardt et al.´s model [BDW02] is based on the model of Pennington´s model. It expands the original model to take into account additional factors. The Program model is expanded to consider the representation of larger units such as routines, known as macrostructure or Elementary Functions. These functions correspond to units in the program structure, i.e., routines attached to objects. Likewise, the Pennington´s situation model is expanded to include aspects of objects and their relationships, as well as client server relationships of objects, as defined below:

  - **Problem Objects**: These objects directly model objects of the problem domain.

  - **Relationships between Problem Objects**: These consist of the inheritance and composition relationships between objects.
  - **Reified Objects**: An example of a computing, or reified, object is a string class, which is not a problem domain object per se. Reified objects are represented at the situation model level in as much as they are necessary to complete the representation of the relationships between problem objects, i.e., they bundle together program-level elements needed by the domain objects.
  - **Main Goals**: The main goals of the problem correspond to functions accomplished by the program viewed at a high level of granularity.
  - **Client server Relationships**: Communication between objects corresponds to client server relationships in which one object processes and supplies data needed by another object. These connections between objects are the links connecting units of complex delocalized plans. Client server relationships represent those delocalized connections.
  - **Data Flow Relationships**: Communication between variables corresponds to data flow relationships connecting units of local plans within a routine.

- **Top-down Direction of Comprehension**: Soloway et al. [SBGE82] present a top-down theory of program comprehension, they treat comprehension as a plan-based activity. Plans are schematic knowledge about how to carry out stereotypical actions in a program. Plans exist at different levels. *Strategic plans* are global plans for the solution of a problem; *tactical plans* are language independent specifications of algorithms for solution of local parts of a problem; *implementation plans* are plans for carrying out a tactical plan in a given programming language. Also, *discourse rules* are programming conventions that govern how plans are expressed and combined. Program understanding begins with the programmer hypothesizing a high-level program goal, then breaking it down into more specific subgoals that should be present in a program having a given high-level goal. Having identified expected goals and subgoals, the programmer must determine whether they exist in the program. The programmer uses knowledge of stored plans and discourse rules to try to satisfy the subgoals and ultimately the top-level goal [CW00]. Brooks [Bro83] also proposed a top down theory of program comprehension, his theory is hypothesis driven and he theorizes that programmers use increasingly specific hypotheses to derive the functionality of the code. His proposal is centered on beacons as knowledge structures.

2.3.3.2.2.3 Dimension of Guidance of Comprehension Burkhardt et al. [BDW98b] distinguish several approach used to guidance reading and comprehension[3]. One of these approaches was called relationship among classes. This approach involves reading the program in a manner which highlights the relationships among classes. Two specific types of guidance are possible reflecting composition and inheritance relationships in a program. We will use this dimension to conceptualize and introduce a guidance approach used in OCL comprehension: Navigation. The navigation of relationships within OCL expressions (see definition in 4.2.1) constitutes one of the most important guidance to comprehend them.

Using another meaning or sense of navigation, Mosemann et al. [MW01] describes different methods of navigation[4] as a process of program comprehension, where navigation is conceptualized as the process of collecting information about pieces of the program.

However, we consider this dimension as a relevant aspect of cognitive models instead of mental model, because it is more related to a cognitive process than a mental representation.

2.3.4 Empirical Validation (C_4)

In order to thoroughly prove that a measure is useful, it is not reliable to use common wisdom, intuition, speculation, or proof of concepts as sources of credible knowledge [BSL99]. It is necessary to place the measures under empirical validation. Empirical validation is an on-going activity [BEM95] 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? [BEM95]. Through empirical validation we can also demonstrate with real evidence that the measures we have proposed serve the purpose they were defined for. This phase is necessary before any attempt to use measures as objective and early indicators of quality.

So, empirical validation is crucial for the success of any software measurement project [Sch92], [KPF95], [BSL99]. However, in general there exists insufficient empirical evidence supporting the usefulness of a vast number of proposed measures [BWIL99]. For that reason, Briand et al. argue [BAC+99] that empirical studies in software engineering need to be better performed, analyzed, and reported.

[3]This third dimension, the guidance of reading & comprehension, was mentioned along with the scope and direction of comprehension dimensions, however it was not used as a new dimension in the recent mental model defined in [BDW02], nevertheless many of the concepts mentions as guidance where included when the situation model of the new model was defined. We only use the concept of *guidance of comprehension* as a term to refer a facilitator activity in OCL expression comprehension.

[4]Here, navigation has another sense or meaning, is not used as OCL navigation.

The empirical validation is used to obtain objective information concerning the usefulness of the proposed measures, because it may occur that a measure was valid from a theoretical point of view, but not to have practical relevance to a specific problem. Therefore, empirical studies are necessary 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 using empirical data. Empirical hypotheses were defined as part of the I_7 activity of the identification step (see Figure 2.2). That hypotheses should be empirically validated through a set of refined hypothesis through different studies. Empirical hypotheses relate (independent) attributes of some entities (e.g. structural properties) to other (dependent) attributes of the same or different activities [BMB02].

We identified the following high level activities in order to carry out any empirical validation:

- **Select a Strategy to Carry Out the Validation** (Figure 2.11, Activity E_1): There are three major strategies or types [Rob93], [WRH+00] of empirical investigations:
  - **experiment**, i.e. a means of testing, using the principles and procedures of experimental design, whether the hypothesis about the expected benefit of a tool or method can be confirmed;
  - **case study**, i.e. a trial use of a tool or method on a full scale project;
  - **survey**, i.e. the collection and analysis of data from a wide variety of projects.
- **Conduct the Strategy through a Family of Studies** (Figure 2.11, Activity E_2): Having selected the strategy, the validation should be run 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 are really useful and necessary to draw more credible conclusions [PPV00], and contribute to obtain more solid findings and expected results. In this thesis we will focus on families of experiments so section 2.3.4.1 describes them in detail.

To perform any empirical strategy such as an experiment, survey or case study, several steps have to be taken and they have to be in a certain order [WRH+00]. 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. When an empirical study is not conducted appropriately or is not precisely reported many problems arise with regard to extracting crucial information from them or even to integrate study results into a common body of knowledge: (1) it is difficult to locate relevant information, (2) important information is often missing, etc. [JP05].

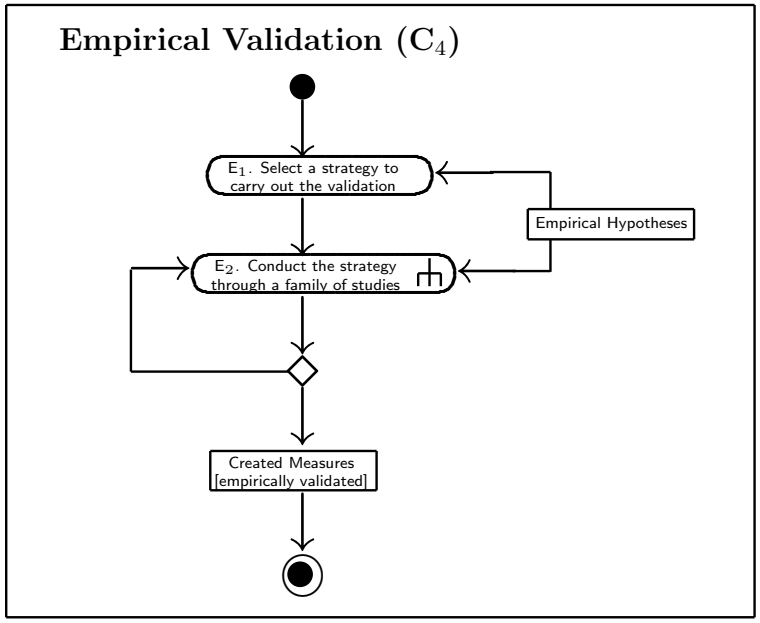


Figure 2.11: Empirical Validation Step (C_4)

Text books on empirical software engineering such as Wohlin et al. [WRH+00] and Juristo et al.[JM01] define a process which is focused on experiments, nevertheless the same basic steps must be performed in any empirical study. Thus, these basic processes may be used for other types of studies being conducted. Various improvements have been achieved to standardize a consistent format for controlled experiments: [Sin99], [JP05], [KPF95], [KAKB+06], [BSH86]. However, the literature about how to carry out a family of experiments is scarce, Ciolkowoski et al. [CSB02] explains the more relevant steps in running families of experiments.

Due to the fact that the empirical validation of this Ph.D. thesis was performed using experiments we will describe them in more detail in the following section.

2.3.4.1 Families of Experiments (E_2)

Experiments can be viewed as part of common families of experiments, rather than being isolated events [BSL99]. Common families of studies can contribute to important and relevant results that may not be suggested by individual experiments.

For individual empirical studies, we know quite well how to proceed, and the experimental process is briefly defined in section 2.3.4.2 and defined in detail in appendix C. However, we do not know what steps are necessary for experiment families. In this section, we present an initial process that allows to systematically define an experiment family according to Ciolkowski et al. [CSB02], integrated with the process of conducting its experiments and modelled through a UML activity diagrams.

A family of experiments is more than a composite of individual studies. Figure 2.12 depicts the process for defining an experiment family [CSB02].

The identified activities include:

- **Prepare the Family of Experiments** (Figure 2.12, Activity F_1): It is necessary to define one or more goals to allow effective analysis afterwards. To guarantee that the data can be compared across all studies of the experiment family, all studies adopt the same experimental framework, including a common GQM [BW84], [BR98], [SB99] plan for measurement.

- **Define the Experimental Context** (Figure 2.12, Activity F_2): It is possible to conduct an analysis that identifies factors across environments and across experiments. That is, the family has to define common measurement framework (or context definition). The relatively low and arbitrary reporting on context variables is a hindrance for metastudies, which are needed to identify which context factors influence which kinds of performance [SHH+05].

  The subjects' variables which are more commonly reported of controlled experiments in software engineering are: gender, age, education, programming experience, task-related experience and task-related training [SHH+05].

- **Setting the Design Framework of the Family** (Figure 2.12, Activity F_3): Additionally, a design framework for the individual studies has to be defined. Furthermore, a set of experimental material has to be created that can be used by the individual experiments.

- **Conduct an Individual Experiment** (Figure 2.12, Activity F_4): Having defined the design framework of the family, the individual experiments can be conducted. Thereby, the same steps are required as in conducting an individual study that is not part of a family. However, the experiment family helps to save effort for the preparation. Individual experiments use the family framework: The preparation of the individual study uses the context definition, design framework, as well as the experimental material from the experiment family.

  After the individual experiment is conducted, a local analysis of experimental data is run. To be able to include specific questions, individual experiments can extend the family framework; for example, they can measure more variables, or use additional material, as long as they include the common framework. Moreover, these extensions may help the experiment family in exploring other variables not considered before.

- **Perform a Family Data Analysis** (Figure 2.12, Activity F_5): Finally, a plan for analysis of the experiment family's results has to be defined. The

final step is to use the data of all studies within the experiment family to conduct the family' data analysis (e.g., analysis on how the context influences the performance of a technique).

- **Define Conclusions for the Family** (Figure 2.12, Activity F_6): After the family data analysis was performed general conclusions can be obtained.

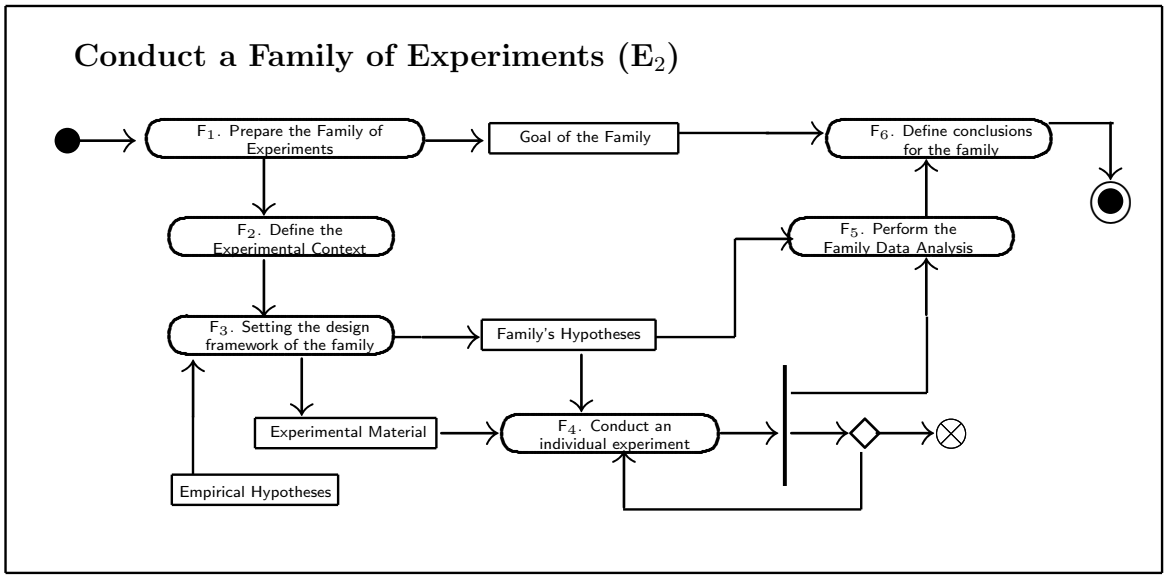


Figure 2.12: Conduct a Family of Experiments (E_2)

The process introduced above shows that experiment families promise to save preparation costs while increasing the benefits of running them. Obviously, defining an experiment family requires more effort than preparing an individual study. We have to describe context variations more systematically than for individual studies. Thereby, context factors of interest have to be defined, maybe using interviews to create questionnaires. Another cost factor is that the design framework has to be defined in such a way that limited variation is possible for individual studies. This also requires more work than for individual studies.

Although the effort for preparing an experiment family is quite high, the benefits are large:

- Researchers who want to participate in the experiment family save work because they can reuse the framework and experimental material. Furthermore, reusing a framework also helps raise the quality of the studies [CSB02].

- Individual studies possess added value when they are part of an experiment family because they are analyzed with respect to the whole experiment family,

not only with respect to their own context. That is, experiment families allow to learn more effectively from individual empirical studies, because studies add to a body of knowledge, instead of providing information limited to one context [CSB02].

- Experiment families allow to answer questions that are beyond the scope of individual experiments, such as which context factors influence the results. Thus, they allow to generalize findings across studies.

2.3.4.2 Conduct Individual Experiments ($\mathbf{F}_4$)

Experiments are launched when we want to have control over the situation and want to manipulate behaviour directly, precisely and systematically. They constitute formal, rigorous and controlled investigations. The objective is to manipulate one or more variables and control other variables at fixed levels. In an experiment the key factors are identified and manipulated. Experiments are appropriate to investigate different aspects, such as to:

- Confirm theories, i.e. to test existing theories.
- Confirm conventional wisdom, i.e. to test peoples conceptions.
- Explore relationships, i.e. to test that a certain relationship holds.
- Evaluate the accuracy of models, i.e. to test that the accuracy of certain models is as expected.
- Validate measures, i.e. to ensure that a measure actually measures what it is supposed to.

The strengths of an experiment is that it can investigate in which situations the claims are true and it can provide a context in which certain standards, methods and tools are recommended for use. We should be able to draw conclusions about the relationship between the cause and the effect for which we stated a hypothesis (which we want to corroborate by means of experiments), only if the experiment is properly set up [WRH+00]. Experiments require a great deal of care and planning if they are to provide meaningful, useful results [FP98].

A characterization of the most important proposals for guidelines on reporting empirical controlled experiments is included in [JP05]. Nevertheless, the following steps are common in most of the aforementioned proposals.

- **Definition** (Figure 2.13, Activity EF_1): In this phase the foundation of the experiment is properly laid, it is defined in terms of problem, objective and goals. Here, the hypothesis has to be stated clearly.

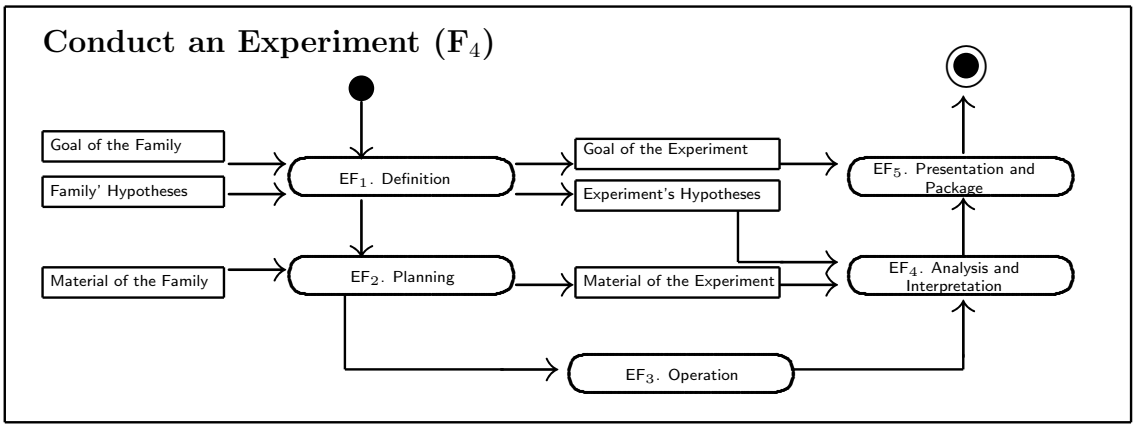


Figure 2.13: Conduct an Individual Experiment (F_4)

- **Planning** (Figure 2.13, Activity EF_2): In this phase the design of the experiment is determined, the instrumentation is chosen, and the threats to the experiment are evaluated.

- **Operation** (Figure 2.13, Activity EF_3): In this phase the experiment is run, and should be conducted according to its design. The measurements are collected.

- **Analysis and Interpretation** (Figure 2.13, Activity EF_4): The data collected during the previous phase is analysed, interpreted and evaluated.

- **Presentation and Package** (Figure 2.13, Activity EF_5): Presenting and Packaging the findings is an important task in order to document the experiment and to facilitate its replications.

Appendix C describes in detail these steps according to [WRH+00].

As Basili et al. [BSL99] remarks experimentation in software engineering is necessary but difficult. One reason for this is the large number of variables of context, which means that to create a cohesive understanding of the experiment results requires a mechanism to explain the studies and incorporate the results.

2.4 Contribution to the Dissertation

The assessment of the influence of import-coupling on the maintainability of OCL expression takes a measurement-based approach. For that reason, in this chapter we defined a method for measure definition. The method is based on a previous

method [CPG01], [CD00] and includes the best practice and experience of different authors and methods of measurement.

The more important refinements and extensions of the method described in this chapter were included in the Identification (M_1) and Creation (M_2) steps. We have identified more than forty activities that further break down the high level activities of the method. We carefully detailed the refined and extended method through UML activity diagrams, allowing a better and complete understanding of the method. The method has been strengthened not only in the order of its steps but also in the specification of their object flows and decision nodes between activities.

The chapter provides the research method that will be applied in the development of this thesis. The activities described in this chapter are applied from chapter four to eight.

Chapter 3

State of the Art

This chapter begins with an introduction of OCL and its utility (section 3.1). Then, a broad and thorough review of most relevant existing works related to coupling measures (section 3.2) and related to measures for UML models (section 3.3) is presented. Last section, 3.4, describes the contribution to the dissertation.

3.1 The Object Constraint Language

OCL is part of the UML, the Object Management Group (OMG) standard for OO analysis and design, and it is publicly available in [OMG03b].

OCL is underpinned by mathematical set theory (OCL provides built-in collections and a set ot theoretical concepts like cardinality, set comprehension, projection, algebra operators, etc.), predicative logic (consider logical operators and quantifiers such as *exists* and *all*) [BeA03a] and operational semantics [Baa00]. It was first developed in 1995 during a business modelling project within IBM. This project was heavily influenced by Syntropy ideas, but unlike Syntropy, OCL does not use unfamiliar mathematical symbols [WK99].

In UML 1.1, OCL appears as the standard for specifying restrictions in one or more values of an OO model. It was designed for usability, the language should be easy to use, easy to learn and easy to understand [WK99] and to be easily grasped by anybody familiar with OO modeling. OCL was available to modelers to increase the meaning that UML diagrams, and it was also used to give added precision to the definition of UML. In parallel to the release of UML 2.0 a new version of the OCL has recently been adopted by the OMG. In this version the understanding is that far more additional information should be included in a model than constraint alone [WK03]. In this version there was a chance to react to many criticisms that the OCL does not have any formal basis for itself [HZ04]. The main differences between OCL

2.0 and OCL 1.4 can be found in [HZ04], [Tch02]. In this thesis we had used the Final Adopted Specification of OCL 2.0 [OMG03b], although we know that the last version of OCL 2.0 is the Proposed Available Specification (FTF Report) of OCL [OMG05b], the former specification was the last version of OCL when we started to define the measures and carried out the experiments.

OCL is a textual specification language defined to solve different problems:

- UML is limited in its expressiveness, and many constraints cannot be defined using only UML graphical features [CKM+02], [WK03].
- Frequently the system properties and constraints that cannot be defined using UML diagrams are defined using natural languages and this leads to misinterpretations, misunderstanding [WK99], and ambiguities [OMG03b].
- The use of formal methods can help to alleviate this problem, in order to specify correctly the system behavior, but the use of formal methods by the object technology community' members requires a strong mathematical background, and formal methods are not a subject with which the average business or system modelers are familiar [OMG03b].
- To provide precise information in the definition of standards, like the UML standard itself, then use of a precise language is required.

OCL was defined as a textual add-on to the UML diagrams [CKM+02]. Its main elements are OCL expressions that represent declarative and side effect-free textual descriptions that are associated to different features of UML diagrams. OCL expressions add precision to UML models beyond the capabilities of the graphical diagrams of UML. Although OCL is considered in [OMG03b] to be a formal language easy to read and write, the misuse of the language can lead to complicated written OCL expressions. Warmer and Kleppe [WK99] give some tips and hints in writing OCL expressions (these recommendations are still valid although OCL has been modified through different versions). Furthermore, they recognize that the way OCL expressions are defined has a large impact on readability, maintainability and the complexity of the associated diagrams.

It is important to introduce several concepts related to an OCL expression:

- Each OCL expression is written in the context of an instance of a specific type. This instance, *self*, provides a point of reference for interpretation of the expression, and is commonly referred to as the contextual instance.
- The context in which an expression is written is introduced through the keyword *context*. Any OCL expression starts with the definition of the context,

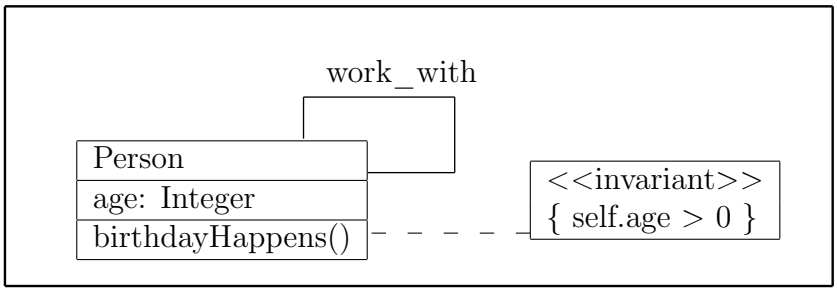


Figure 3.1: Example of an Invariant

that involves the keyword *context* followed by the name of a type (the contextual type), then any specification of *self* within the OCL expression will be associated with the type declared in the context declaration.

- OCL expressions in UML class diagrams are used most importantly: to specify invariants on classes and types in the class diagram, to specify constraints on operations and methods, to describe pre- and post conditions on operations, to specify initial values and derivation rules for attributes, to specify query operations, and to introduce new attributes and operations [OMG03b].
- Invariants, preconditions and postconditions are constraints stereotyped respectively by <<invariant>>, <<precondition>> and <<postcondition>>. Although it is common for stereotypes to be attached to a UML feature using a graphical notation such as a note box, as shown in Figure 3.1 , the quantity of OCL expressions defined for a class diagram is significantly higher and this clutters the understandability of the UML diagram. For that reason all the OCL expressions shown in this chapter will be written in textual form and will not be shown using a note box.

Example 3.1.1 *An invariant definition for the Person class is:*

```
context Person inv:
        self.age > 0
```

The keyword *inv* means that the OCL expression, which comes after the colon, is an invariant expression. In fact, the *inv* keyword denotes the stereotype <<invariant>>. This expression means that all the values of any instance of the class Person must not be zero or lower. *self.age* uses a dot notation to refer to a property *age* of the object represented by *self*, however it is possible that *self* will be implicit in the expression (in the example is possible to simply write *age* instead of *self.age*). In general, whenever a property called *property* is specified without the object to

which it applies -in the form of *property* instead of *object.property*-, the object is left out, or it is implicit in the specification of the property *property*.

Example 3.1.2 *This example introduces a postcondition restriction for the BirthdayHappens operation of Person class. It is a postcondition as the post keyword is used, which in turn refers to a <<postcondition>> stereotype. This postcondition means that the age of a person is incremented by one when the birthday of a person had happened (age@pre is used to refer to the previous value of age just before the execution of BirthdayHappens).*

```
context Person::BirthdayHappens()
post:        age = age@pre + 1
```

Before giving more examples we will introduce two important concepts used in the following sections:

- *Properties*: an attribute, an association-end, and side-effect-free operation or method are considered properties of an object [OMG03b]. The way an object property is specified in an OCL expression is by using a dot notation. *Object.property1* refers to a *property1* of *object* wherever *object* is a valid reference to an object. To illustrate this concept see example 3.1.3.

  Example 3.1.3 *In the following expression the work_with property is used in an expression, meaning that a person cannot work with himself or herself. In this case work_with represents an association-end property.*

  ```
  context Person inv:
          not self.work_with->exists(self)
  ```

- *Classifier*: a classifier is a UML metaclass which represents a type, a class, an interface, an actor (of a use case diagram), an association (acting as types) and datatypes [OMG03b]. Each classifier defined within a UML model represents a distinct OCL type [OMG03b].

3.1.1 Utility of OCL

OCL is used for different domain and in different levels of the metamodelling architecture. As it is widely knowledged a metamodelling approach or Meta-Object Facility (MOF) metamodel consist of four layer (Table 3.1 gives an overview of the four layers). Some examples of how OCL is used in metamodelling are:

Table 3.1: Levels in the UML Metamodel

Level	Layer	Description
M_3	meta-metamodel	Language for metamodels
M_2	metamodel	Language for models
M_1	model	Language for information domains
M_0	user objects	Specific information domain

- **M_1 layer**. In modelling systems (M_1 layer) OCL constraints convey a number of benefits, including precision and design documentation, resulting in better (unambiguous) communication among the involved parts, such as designers, users, programmers, testers and managers and, since OCL is a typed language, it is possible to check constraints for validity during modeling [BeA03a]. In this way OCL is suitable as a lightweight replacement for formal specification languages [HZ04] (like Z, VDM, OBJ, etc.).

- **M_2 layer**. Within UML standard itself, OCL is also used to specify well-formedness rules applied to the UML metamodel [Ric02] and to few other OMG standards being applied on the meta-level [HZ04].

- **M_3 layer**. Within an Meta Object Facility (MOF) OCL is used to specify rules for describing metamodels in various domains [Ric02].

OCL is also used along with different domains and software tools [CBCS04], some examples are:

- **In Model Driven Engineering:** The MDA is a framework that is intended to support the development of software through the transformation of models to executable applications [JCF03]. The essence of MDA approach is that models form the basis of software development. The use of OCL in the MDA approach is crucial. OCL is recommended to be used in MDA process for building PIM [WK03], [MEJ+03]. Using a combination of UML and OCL is possible to build solid and consistent models for MDA [WK03], that is the reason many authors argue that OCL is one of the most interesting candidate languages for appling the contract principle of design for specification of platform-independent description of software components [HZ04], [WK03].

- **In Model Transformation:** A key facilitator of the MDA is a standard to express model transformations. On modelling transformation rules for MDA, OCL can be used in for different purposes: OCL constraints are used to define the relationship between the abstract and concrete metamodels in a declarative manner [GJG04], [GL05], [TFS06]. Here, the OCL constraints play

both the role of transformation invariant and rule postcondition. Cariou et al. [CMSD04] focuses on the specification of model transformation contracts. They investigate the way to define transformations contracts using standard UML and OCL features.

OMG have initiated the standardization of a transformation initiatives, the MOF 2.0 Query/Views/Transformations (QVT) [OMG05a] which will provide a standard language for transformation between MOF-based models. The initiative proposed a pattern-based language with ability to define the transformation partially with a graphical notation and refinement in a textual notation [GO05]. It includes a new pattern matching language and use of OCL as the query language. For QVT the current proposal suggests to use the declarative OCL at least for the querying part.

Patrascoiu [Pat04] proposes the YATL transformation language to do model-to-model transformations. It is a hybrid textual language that uses OCL expressions for querying UML models and new imperative constructions to create target instances. Bezivin et al. [BBDV03] propose the textual ATL transformation language that builds on OCL. Sendall [Sen03] suggests VMT as a graphical UML-to-UML transformation language, which is based on graph matching for source and result selection, metamodel rules, OCL constraints and UML activity models to compose transformations.

Examples of the use of OCL in model transformation:

- OCL is used in the XMI proposal to define the XMI stream production rules. The production rules specify how a model can be transformed into a XML document conforming to the XMI proposal.
- Formal object-oriented specifications are generated in OCL and class diagrams from the use case model of a system through a clearly defined sequence of model transformations. [Rou03]

- **In Metamodelling:** OCL is used in metamodelling facilities for specifying stereotypes and tagged values constraints [SW01], [MRRR02], it is also used for modeling ontologies [WCH02]

- **In Testing:** As it is widely acknowledged early test development and specification enhance the quality and robustness of software [BKW04]. The specification of assertions using OCL [SSR04], [MV04], [VS02], [SC02], [NF06] and its corresponding implementations contribute to the software testing.

- **In Reverse Engineering:** OCL is used for defining a mapping between two models in reverse engineering of UML interaction diagrams [BLM03].

- **In Real Time Applications:** OCL is used and also extended for modelling real-time applications and temporal constraints. For example: To define the well-formedness rules of the abstract model in real-time reactive systems [Mut00], and one extension of OCL that enables modelers to express state-related time-bounded constraints [Fla02] and its semantics [FM02], etc.

- **In Business Domain:** OCL is used in the specification and validation of transactional business software [CW04]. Another approach [HHW04] ensure unambiguity defining well-defined constraints for different business environments through the templates based on an extended version of OCL.

- **In Several Domain Applications:** OCL is used in a wide range of domain-application, for example: In specifying the object interface in multimedia systems [Aag98], in geographical information systems [CWD00], [FCTJ01].

- **In Programming Languages:** Integration of enhancements of OCL in some specification languages can be found in [Ham99] where OCL are integrated in C++, in [Ham04], [BKW04] in Java.

3.2 Coupling Measurement

Software engineering best practices promote that a high quality software design among many other principles, should obey the principle of low coupling [BDW99]. Currently available evidence suggests that coupling measures are good predictors for the maintainability of components in OO systems [DJ03], [Ari02].

In the last fifteen years, a considerable number of studies have been carried out into coupling measures. A general consensus in the software engineering community is that too much coupling is harmful in terms of system structure and increase system complexity [HCN98a]. Stevens et al. [SMC99] gave an initial definition of coupling in the context of structured development techniques, defining it as the measure of the strength of association established by a connection from one module to another. Therefore, the more inter-related as set of modules are, the more difficult these modules are to understand, change and correct and thus the more complex the resulting software system [BDW99].

Briand et al. [BDW99] make a serious attempt to improve the understanding of OO coupling measurement integrating existing three previous frameworks into a unique theoretical framework. The three frameworks were Eder et al. [EKS94], Hitz and Montazeri [HM95], and Briand et al. [BDM97]. The objective of the unified framework are manifold, however two important purposes are: to support the comparison and selection of existing coupling measures and facilitate a more rigorous decision making regarding the definition of new coupling measures.

Acronym	Client item	Server Item	Description
aC	attribute a of a class c	class d, $d \neq c$	class d is the type of a
MpC	method m of a class c	class d, $d \neq c$	class d is the type of a parameter of m, or the return type of m
MvC	method m of a class c	class d, $d \neq c$	class d is the type of a local variable of m
MipC	method m of a class c	class d, $d \neq c$	class d is the type of a parameter of a method invoked by m
Ma	method m of a class c	attribute a of a class d, $d \neq c$	m references a
MiM	method m of a class c	method m' of a class d, $d \neq c$	m invokes m
CC	class c	class d, $d \neq c$	high-level relationships between classes

Table 3.2: Types of Connections

3.2.1 Coupling Framework Criteria

The framework of Briant et al. [BDW99] for coupling consists of six criteria, each criterion determining one basic aspect of the resulting measure.

3.2.1.1 Criterion 1. Type of connection

Connection is a generic term defined as an occurrence of a given type of coupling [BDW99]. For instance, a method invocation represents a connection in a coupling measure. Once we choose the type of connection we are effectively choosing the mechanism that constitutes coupling between two entities.

The Table 3.2 (from [BDW99]) summarizes the possible types of connections, i.e. links between a client and a server item (attribute, method or class). The items are listed in the columns 'client item' and 'server item' respectively. Description column explains the type of connection. The first column lists a short way to refer to each type of connection.

3.2.1.2 Criterion 2. Locus of impact

As Dagpinar et al. [DJ03] argue it is important to make a distinction between the server and client entity from a perspective of a maintenance. For this purpose, Briand et al. [BDM97] paid attention to the locus of impact: whether the class is a user (i.e. a client) or used (i.e. a server) in the coupling relationship. These relationships are called import-coupling (IC) and export-coupling (EC), respectively:

- IC: Import-coupling analyses attributes, methods, or classes in their role as clients (users) of other attributes, methods, or classes.
- EC: Export-coupling analyses the attributes, methods, and classes in their role as servers to other attributes, methods, or classes.

After performing a survey of measures Dagpinar et al. recognize that few empirical studies differentiate between export-coupling and import-coupling [DJ03].

3.2.1.3 Criterion 3. Granularity

The granularity of the measure is the level of detail at which information is gathered. The granularity of the measure is determined by two factors:

- the **domain of the measure**, i.e., what components are to be measured: The possible domains for the coupling measures vary from smaller domains, such as the class level or attribute, to larger domains like the sets of classes and the system level.
- **how exactly the connections are counted**: Available options for this decision can be restricted by the domain of the measure.

  For measures defined at the method or attribute level, two options are listed in Table 3.3, where the difference between options A and B is that multiple connections between two items are counted separately in option A, and counted as one in option B.

  At a class level, there are four options to count connections (see table 3.4) The difference between options D) and E) is that if, for instance, two methods of a class reference the same attribute, the references are counted separately (once for each method) according to D), and counted as one for the class according to E).

  Measures defined for sets of classes or the system can be constructed by adding up the number of connections of the relevant classes, counted according to one of the options C) to F).

3.2.1.4 Criterion 4. Stability of server

Two different categories of class stability are defined: unstable classes and stable classes. Unstable classes are those which are subject to development or modification in the project at hand, whereas stable classes are not subject to change in the project at hand.

option	Description	Import-coupling example	Export-coupling example
A	count individual connections	for each method, the number of references to attributes	for each attribute the number of references to the attribute
B	count the number of distinct items at the other end of the connections	for each method, the number of attributes referenced	for each attribute the number of methods that reference the attribute

Table 3.3: Options for Coupling Connections at the Attribute and Method Levels

option	Description	Import-coupling example	Export-coupling example
C	add up the number of connections counted as in A) for each method or attribute of the class	the total number of attribute references by methods in the class	the total number of references to attributes of the class
D	add up the numbers of connections counted as in B) for each method or attribute of the class	add up the number of attributes referenced by each method of the class	add up or for each attribute of the class: the number of methods that reference the attribute
E	count the number of distinct items at the end of connections starting from or ending in methods or attributes of the class	the number of attributes referenced by the methods of the class	the number of methods referencing attributes of the class
F	for a class c, count the number of other classes to which there is at least one connection	the number of classes which have an attribute that is referenced by a method of class c	the number of classes which have a method that reference an attribute of class c

Table 3.4: Options for Coupling Connections at the Class Level

This criterion has not been commonly addressed in the definition of measures for coupling. Probably due to stability of a server class is a subjective concept which is difficult to measure automatically [BDW99]. We believe that this criterion is more appropriate to use in empirical studies than in definition stages.

3.2.1.5 Criterion 5. Direct or Indirect connections

We have to decide whether to count direct connections only or also indirect connections. For example, if a method m_1 invokes a method m_2, which in turn invokes a method m_3, we can say that m_1 indirectly invokes m_3. Methods m_1 and m_3 are indirectly connected.

Indirect coupling, that has not been studied and suggested that its existence may

be the source of unnecessary maintenance costs. An important work about indirect coupling was developed by Yang et al. [YTB05] they build a prototype for detecting indirect coupling but they recognize that in order to determine whether there is a correlation between indirect coupling and maintainability, they really like to have a measure that is at least an interval scale one.

3.2.1.6 Criterion 6. Inheritance

Two important issue should be considered in this criterion:

3.2.1.6.1 Inheritance based vs. non-inheritance-based coupling First, we have to decide whether to count inheritance-based coupling and/or non-inheritance-based coupling. Inheritance- based coupling analyses connections between classes that are related via inheritance. Likewise, noninheritance- based coupling refers to connections between classes that are not related via inheritance.

3.2.1.6.2 How to assign methods and attributes to classes The final question is to decide to which class an attribute or method belongs. We have to decide, if inherited methods and attributes belong to the inheriting class or not. We distinguish two cases:

- When we compute the coupling of a class, we have to determine what are the methods/attributes of the class, and therefore contribute to the coupling of the class. The available options are:
  - only methods and attributes implemented in the class contribute to the coupling of the class
  - all methods and attributes implemented or declared in the class contribute to the coupling of the class
- When we count the frequency of connections according to option F) (i.e., for a given class, we count the number of other classes it is connected to), we have to assign the items at the other ends of the connections to a class.

3.2.2 Proposal of Coupling Measures

The number of coupling measures that have been proposed for object-oriented products is very large. In this section we give a review of the most important coupling measures defined in the literature.

Name	Definition	Source
CBO	Coupling Between Object classes. This includes inheritance-based coupling (coupling between classes related via inheritance).	Chidamber and Kemerer, (1994 [CK94]
CBO′	Same as CBO, except that inheritance-based coupling is not counted.	Chidamber and Kemerer, (1991) [CK91]
RFC_∞	Response Set for Class. The response set of a class consists of the set M of methods of the class, and the set of methods directly or indirectly invoked by methods in M.	Chidamber and Kemerer, (1991) [CK91]
RFC_1	Same as RFC_∞, except that methods indirectly invoked by methods in M are not included in the response set this time.	Chidamber and Kemerer, (1994) [CK94]
MPC	Message Passing Coupling. The number of method invocations in a class.	Li and Henry [LH93]
DAC	Data Abstraction Coupling The number of attributes in a class that have as their type another class.	
DAC′	The number of different classes that are used as types of attributes in a class.	
ICP	Information flow- based Coupling. The number of method invocations in a class, weighted by the number of parameters of the invoked methods.	Lee et al., (1995) [LLWW95]
IH-ICP	Information flow- based Inheritance Coupling. As ICP, but counts invocations of methods of ancestors of classes (i.e., inheritance-based coupling) only.	
NIH-ICP	Information- flow-based Noninheritance Coupling. As ICP, but counts invocations to classes not related through inheritance.	
NPAVG	measures the average number of parameters per method (not including inherited methods).	Lorenz et al. [LK94]
CDM	Coupling Dependency measure	Binkley et al. [BS98]

Table 3.5: Coupling Measures (Part I)

- Chidamber and Kemerer [CK91] gave an initial definition of coupling as any evidence of a method of one object using methods or instance variables of another object. This measured through a coupling measure named Coupling Between Object classes (CBO'). Within the CBO measures, which was empirically validated in [BBM96], a class A is coupled to class B if A uses B's member functions and/or instance variables. CBO' counts the number of classes to which a given class is coupled.

- In a later work, Chidamber and Kemerer introduced a new version of CBO', CBO, which includes inheritance-based coupling. So, CBO' measures 'non-inheritance based coupling' [CK91] whereas CBO explicitly includes 'coupling due to inheritance' [CK94].

  Several authors acknowledged the need to differentiate between inheritance-based and non-inheritance-based coupling. Lee et al. [LLWW95] proposed three measures that considers inheritance aspects: NIH-ICP counts non-inheritance-based coupling only, IH-ICP counts inheritance based coupling only. ICP is the sum of IH-ICP and NIH-ICP, thus treats both types of coupling equal.

  The suite of measures by Briand et al. [BDM97] also provides measures which

count inheritance-based and non-inheritance based coupling separately.

- Li and Henry [LH93] proposed more fine-grained extensions of the CK coupling measure via measures like Message Passing Coupling (MPC) and Data Abstraction Coupling (DAC) [SK03]. A distinction is made between coupling with other classes, and coupling as a result of messages sent to *self* (i.e., an instance of a class type).

- Briand et al. defines in [BDM97] a suite of measures to quantify the level of class coupling during the design of object-oriented systems. Same of the defined measures are language specific due to different OO design mechanisms provided by the C++ language are considered (e.g., friendship between classes, specialization, and aggregation) and ancestors. A complete suite of measures of coupling measures is included in [BDW99]. Table 3.6 defines each of these measures.

- Dagpinar et al. [DJ03] defines a set of measures (see Table 3.7) that are mainly based on measures defined by Briand el al [BDM97], with the following modifications:

  - A separation has been made in order to distinguish between indirect and direct coupling.
  - Relationship names have been changed from antecedent class, friend class, and other class to inheritance and non-inheritance relationships since friend class is only valid for C++ and we do not want to make our model language specific.
  - In addition to counting the number of interactions between classes, the number of classes that the class interacts with is also counted.

All the above described measures are evaluated using the Briand et al. framework for coupling measurement [BDW99], we summarize this process in tables 3.2.2.1 and 3.9. The three tables structure is the following: column 1 shows the measure acronym of the table, column 2 shows the source work where the measure is defined, column 3 shows whether the measures are theoretically or empirically validated, column 4 shows the type of connection according the acronym used in Table 3.2, column 5 shows the strength of the measures, column 6 describes the locus of impact -see criterion 2 of Briand et al. framework-, finally column 7 and 8 show the direction -see criterion 5 of Briand et al. framework- and inheritance -see criterion 6 of Briand et al. framework- aspects of the measures.

Name	Definition	Source
IFCAIC ACAIC OCAIC FCAEC DCAEC OCAEC IFCMIC ACMIC OCMIC FCMEC DCMEC OCMEC OMMIC IFMMIC AMMIC OMMEC FMMEC DMMEC	These coupling measures are counts of interactions between classes. The measures distinguish the relationship between the classes (friendship, inheritance, none), different types of interactions, and the locus of impact of the interaction. The acronyms for the measures indicates what interactions are counted: • The first or first two letters indicate the relationship (A: coupling to ancestor classes, D: Descendents, F: Friend classes, IF: Inverse Friends (classes that declare a given class c as their friend), O: Others, i.e., none of the other relationships). • The next two letters indicate the type of interaction: CA: There is a Class-Attribute interaction; CM: There is a Class-Method interaction; MM: There is a Method-Method interaction. • The last two letters indicate the locus of impact: IC: Import-coupling, the measure counts for a class c all interactions where c is using another class. EC: Export-coupling: count interactions where class d is the used class.	Briand et al. [BDM97]
PIMAS	the number of direct and indirect static invocations	Briand et al. [BWL99]
SIMAS	the number of direct and indirect static invocations, taking polymorphism into account	
INAG	direct and indirect aggregation relationships	

Table 3.6: Coupling Measures (Part II)

3.2.2.1 Empirical Studies of Coupling Measures

Although there exists insufficient empirical evidence supporting the usefulness of a vast number of proposed OO measures [BAC+99], the most promising results with object-oriented measures were obtained using coupling measures [Ema02]. These relevant empirical studies suggests that there are important relationships between coupling structural attribute and several external quality indicators.

Table 3.10, briefly shows a survey of empirical studies for coupling measures, showing the authors (first column), the system where the the study was run (second column), the dependent variables analysed (third column) as a external quality aspect affected by the coupling measure (the independent variable is shown in the fourth column). Reading Table 3.10 is possible to show that exist evidence that some forms of coupling have an impact on understandability, maintenance effort, fault proneness, impact analysis and quality guidelines. Now, we describe these empirical studies in more detail.

The first empirical study of measures including coupling measures, was run by Li and Henry [LH93] which explore the link between several OO design measures and the extent of code change, which they used as a surrogate measure for maintenance effort.

Many empirical studies were conducted by L. C. Briand, around a number of OO

coupling measures identified in a survey of the literature included in [BDW99]. These studies were empirically validated for predicting different external quality attributes:

- **Impact Analysis:** Briand el al. study in [BWL99] has investigated the use of coupling measurement, for identifying classes likely to contain ripple changes when another class is being changed, i.e. they investigate the relationship between class coupling and ripple change. A commercial C++ system, which has been under maintenance and change data has been collected over several year, was used to investigate this question. The study shows that a number of coupling measures, related to aggregation and invocation coupling, are related to a higher probability of common changes.

Name	Definition	Source
ICAIC	Inheritance class-attribute import-coupling.	Dagpinar et al, 2003 [DJ03]
NICAIC	Non-inheritance class-attribute import-coupling.	
ICAEC	Inheritance class-attribute export-coupling.	
NICAEC	Non-inheritance class-attribute export-coupling.	
ICMIC	Inheritance class-method import-coupling.	
NICMIC	Non-inheritance class-method import-coupling.	
ICMEC	Inheritance class-method export-coupling.	
NICMEC	Non-inheritance class-method export-coupling.	
IMMIC	Inheritance method-method import-coupling.	
NIMMIC	Non-inheritance method-method import-coupling.	
IMMEC	Inheritance method-method export-coupling.	
NIMMEC	Non-inheritance method-method export-coupling.	
IIC	Inheritance import-coupling. IIC = ICAIC + ICMIC + IMMIC	
NIIC	Non-inheritance import-coupling. NIIC = NICAIC + NICMIC + NIMMIC	
IEC	Inheritance export-coupling. IEC = ICAEC + ICMEC + IMMEC	
NIEC	Non-inheritance export-coupling. NIEC = NICAEC + NICMEC + NIMMEC	
TIIC	Total inheritance import-coupling by including indirect coupling relationships.	
TNIIC	Total non-inheritance import-coupling by including indirect coupling relationships.	
TIEC	Total inheritance export-coupling by including indirect coupling relationships.	
TNIEC	Total non-inheritance export-coupling by including indirect coupling relationships.	
DTIIC	Direct total inheritance import-coupling.	
DTNIIC	Direct total non-inheritance import-coupling.	
DTIEC	Direct total inheritance export-coupling.	
DTNIEC	Direct total non-inheritance export-coupling.	
IDTIIC	Indirect total inheritance import-coupling. IDTIIC of class A is calculated by counting	
IDTNIIC	Indirect total non-inheritance import-coupling.	
IDTIEC	Indirect total inheritance export-coupling.	
IDTNIEC	Indirect total non-inheritance export-coupling.	

Table 3.7: Coupling Measures

- **Fault-Proneness:** Briand et al. explores in [BWDP00] the relationships between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing. Results have shown that many coupling and inheritance measures are strongly related to the probability of fault detection in a class. In particular, coupling induced by method invocations (import-coupling), the rate of change in a class due to specialization, and the depth of a class in its inheritance hierarchy appear to be important quality factors. On the other hand, cohesion, as currently captured by existing measures, does not seem to have a significant impact on fault-proneness.

  Also Briand et al. in [BWIL99] study an industrial case and found that three coupling measures were associated with fault-proneness class.

- **Development Effort:** Briand et al. [BW01] investigate the impact that structural properties (coupling was included) has on development effort for a class. A graphical and interactive editor, LIOO implemented in C++ on a Linux platform, developed at the University of Florence was study. They found that simple size measures, predict most of the effort variance, and that coupling and cohesion measures do not help to improve these predictions to a degree that would be practically significant. However, Chidamber et al. [CDK98] observed that higher values of the coupling and the cohesion measures in the CK suite were associated with reduced productivity and increased rework/design effort.

- **Quality Guidelines:** Briand et al. describe in [BBD01] an investigation of the use of quality design principles and their influence on the developer' s ability to understand and modify OO design documents. A set of quality design principles defined by Coad and Yourdon are studied, being the guidelines of low coupling between classes one of the design principles analyzed. Their results shown, with statistical significance, that adherence to good object-oriented design principles provides practically significant benefits to object-oriented design documents in terms of ease of understanding and modifiability. In a replicated study of quality guidelines, Briand et al. [BWL01] provides the following recommendation: A strong emphasis should be put on method invocation, import-coupling since it has shown to be a strong, stable indicator of fault proneness'. We also recommend that the following aspects be measured separately since they capture distinct dimensions in our data sets: import versus export-coupling, coupling to library classes versus application classes, method invocation versus aggregation coupling. As far as cohesion is concerned and measured today, it is very likely not a very good fault-proneness indicator.

Similarly, to the results of Briand et al. [BWDP00] regarding fault-proneness,

Name	Source	Validation		Briand et al. Framework Criteria				
		Theo.	Emp.	connection	strength	Locus	Direction	Inheritance
CBO	[CK94]	yes	yes	Ma, MiM	# coupled classes	both	no	both making no distinction
CBO′	[CK91]	yes	yes	Ma, MiM	# coupled classes	both	no	non-inheritance based
RFC_∞	[CK91]	no	no	MiM	# methods invoked	import	depends	both making no distinction
RFC	[CK94]	yes	yes	MiM	# methods invoked	import	no	both making no distinction
RFC′	[CK94]	yes	no	MiM	# methods invoked	import	no	both making no distinction
MPC	[LC94]	no	yes	MiM	# methods invocations	import	yes	both making no distinction
DAC	[LC94]	no	yes	aC	# attributes	import	no	both making no distinction
DAC′	[LC94]	no	yes	aC	# distinct types	import	no	both making no distinction
COF	[eAGE95]	no	no	aC, Ma, MiM	# coupled classes	both	no	non-inheritance based
ICP	[LLWW95]	yes	no	MiM	# method invocations # parameter passed	import	no	both making no distinction
IH-ICP	[LLWW95]	yes	no	MiM	# method invocations # parameter passed	import	no	inheritance based
NIH-ICP	[LLWW95]	yes	no	MiM	# method invocations # parameter passed	import	no	non-inheritance based
NPAVG	[LK94]	yes	no	–	# parameters	import	no	inheritance based
CDM	[BS98]	yes	yes	CC		import	no	–
IFCAIC	[BDM97]	yes	yes	aC	# attributes	import	no	non-inheritance based
ACAIC	[BDM97]	yes	yes	aC	# attributes	import	no	inheritance based
OCAIC	[BDM97]	yes	yes	aC	# attributes	import	no	non-inheritance based
FCAEC	[BDM97]	yes	yes	aC	# attributes	export	no	non-inheritance based
DCAEC	[BDM97]	yes	yes	aC	# attributes	export	no	inheritance based
OCAEC	[BDM97]	yes	yes	aC	# attributes	export	no	non-inheritance based
IFCMIC	[BDM97]	yes	yes	MpC	# of parameters	import	no	non-inheritance based
ACMIC	[BDM97]	yes	yes	MpC	# of parameters	import	no	inheritance based
OCMIC	[BDM97]	yes	yes	MpC	# of parameters	import	no	non-inheritance based
FCMEC	[BDM97]	yes	yes	MpC	# of parameters	export	no	non-inheritance based
DCMEC	[BDM97]	yes	yes	MpC	# of parameters	export	no	inheritance based
OCMEC	[BDM97]	yes	yes	MpC	# of parameters	export	no	non-inheritance based
OMMIC	[BDM97]	yes	yes	MiM	# method invocations	import	no	non-inheritance based
IFMMIC	[BDM97]	yes	yes	MiM	# pointers passed to a method	import	no	non-inheritance based
AMMIC	[BDM97]	yes	yes	MiM		import	no	inheritance based
OMMEC	[BDM97]	yes	yes	MiM		export	no	non-inheritance based
FMMEC	[BDM97]	yes	yes	MiM		export	no	non-inheritance based
DMMEC	[BDM97]	yes	yes	MiM		export	no	inheritance based
SIMAS	[BWL99]	no	no	MiM	# method invocations	import	yes	inheritance based
PIMAS	[BWL99]	no	no	MiM	# method invocations	import	yes	inheritance based + polymorphism
INAG	[BWL99]	no	no	MiM	# aggregations relationships	import	yes	inheritance based

Table 3.8: A Survey of Coupling Measures.

Name	Source	Validation		Briand et al. Framework Criteria				
		Theo.	Emp.	type of connec.	strength	Locus	Direction	Inheritance
ICAIC	[DJ03]	no	yes	aC	# attributes	import	direct	inheritance based
NICAIC	[DJ03]	no	yes	aC	# attributes	import	direct	non-inheritance based
ICAEC	[DJ03]	no	yes	aC	# attributes	export	direct	inheritance based
NICAEC	[DJ03]	no	yes	aC	# attributes	export	direct	non-inheritance based
ICMIC	[DJ03]	no	yes	MpC	# methods	import	direct	inheritance based
NICMIC	[DJ03]	no	yes	MpC	# methods	import	direct	non-inheritance based
ICMEC	[DJ03]	no	yes	MpC	# methods	export	direct	inheritance based
NICMEC	[DJ03]	no	yes	MpC	# methods	export	direct	non-inheritance based
IMMIC	[DJ03]	no	yes	MiM	# methods invoked	import	direct	inheritance based
NIMMIC	[DJ03]	no	yes	MiM	# methods invoked	import	direct	non-inheritance based
IMMEC	[DJ03]	no	yes	MiM	# methods invoked	export	direct	inheritance based
NIMMEC	[DJ03]	no	yes	MiM	# methods invoked	export	direct	non-inheritance based
IIC	[DJ03]	no	yes	aC + MpC + MiM	R	import	direct	inheritance based
NIIC	[DJ03]	no	yes	aC + MpC + MiM	R	import	direct	non-inheritance based
IEC	[DJ03]	no	yes	aC + MpC + MiM	R	export	direct	inheritance based
NIEC	[DJ03]	no	yes	aC + MpC + MiM	R	export	direct	non-inheritance based
TIIC	[DJ03]	no	yes	aC + MpC + MiM	R	import	indirect	inheritance based
TNIIC	[DJ03]	no	yes	aC + MpC + MiM	R	import	indirect	non-inheritance based
TIEC	[DJ03]	no	yes	aC + MpC + MiM	R	export	indirect	inheritance based
TNIEC	[DJ03]	no	yes	aC + MpC + MiM	R	export	indirect	non-inheritance based
DTIIC	[DJ03]	no	yes	CC	# coupled classes	import	direct	inheritance based
DTNIIC	[DJ03]	no	yes	CC	# coupled classes	import	direct	non-inheritance based
DTIEC	[DJ03]	no	yes	CC	# coupled classes	export	direct	inheritance based
DTNIEC	[DJ03]	no	yes	CC	# coupled classes	export	direct	non-inheritance based
IDTIIC	[DJ03]	no	yes	CC	# coupled classes	import	indirect	inheritance based
IDTNIIC	[DJ03]	no	yes	CC	# coupled classes	import	indirect	non-inheritance based
IDTIEC	[DJ03]	no	yes	CC	# coupled classes	export	indirect	inheritance based
IDTNIEC	[DJ03]	no	yes	CC	# coupled classes	export	indirect	non-inheritance based

Table 3.9: A Survey of Coupling Measures.

Binkley and Schach [BS98] found that the coupling measure was associated with maintenance changes made in classes due to field failures. Their investigation was based in several coupling measures (including CBO) and the NOC measure of the CK suite in two university software applications. Likewise, Basili et al. [4] and Tang et al. [TKC99] found that several of the CK coupling measures were positively associated with fault-proneness of classes.

Harrison et al. [HCN98a] study the CBO measure and compared with an alternative OO design measure called NAS, which measures the Number of Associations between a class and its peers. Results from all systems studied indicate a strong relationship between CBO and NAS, suggesting that they are not orthogonal. Their hypotheses that coupling would be related to understandability, the number of errors and error density, was rejected because they did not found any relationships of the systems between class understandability and coupling.

Dagpinar et al. [DJ03] in order to validate their measures (see table 3.9) conducted an empirical study based on historical data collected from the maintenance history of a medium-sized object-oriented system over a period of three years.

Their results indicate that size and import direct coupling measures are significant predictors for measuring maintainability of classes while inheritance, cohesion, and indirect/export coupling measures are not. These results are quite similar to the one obtained by Briand et al. when they study the influences of measures in quality guidelines.

Subramanyam et al. [SK03] provides empirical evidence supporting the role of OO design complexity measures, specifically a subset of the Chidamber et al. suite, in determining software defects. The analysed measures were WMC, CBO and DIT. They argue that further analyses to understand potential language specific differences are needed. They validate the aforementioned measures across two programing languages, C++ and Java, and found that the language might play a role in the relationship between OO design measures and defects, due to defects were found to differ in the two languages employed.

3.3 Measures for UML Models

The main goal of this section is to introduce the main proposals of measures that can be applied for measuring quality characteristics of UML structural diagrams (class diagrams) and and UML behavioral diagrams (use case diagrams, statechart diagrams).

This section is organized thus: subsections 3.3.1 to 3.3.3 outline the different proposals of measures for UML diagrams: use cases diagrams, class diagrams and statechart diagrams, respectively. Finally, last subsection presents some concluding

Publication	System analysed	Dependent Variable	Independent Variable
Li and Henry [LH93]	two medium-sized, commercial software systems	Maintenance effort (number of lines changed over a period of three years)	Chidamber and Kemerers measures suite [CK91] + MPC, DAC and two size measures (SIZE1, SIZE2)
Li et al. [LHKS95]			Chidamber and Kemerers measures suite [CK91] + MPC
Briand et al. [BW01]	a graphical and interactive editor for music scores	Development Effort	measures suite defined in [BDM97]
Binkley and Schach [BS98]	Four systems were developed (two of them using OO methods)	Maintenance code change	Class coupling
Basili et al. [BBM96]	eight small software systems, developed in a students project	Fault-proneness (probability of detecting a fault in a class)	Chidamber and Kemerers measures suite [CK94]
Briand et al. [BDM97], [BWDP00]			measures suite defined in [BDM97]
Tang et al. [TKC99]	Three real time systems were analyzed for testing and maintenance defects.	Fault-proneness	WNC and RFC measures
Briand et al. [BWL99]	An open multi-agent system development environment	Impact Analysis	measures suite defined in [BDM97]
Briand et al. [BBD01], [BWL01]		Quality Guidelines on the Maintainability	measures suite defined in [BDM97] and others measures
Harrison et al. [HCN98a]	five systems written in C++	understandability, the number of errors and error density	CBO and NAS measures
Dagpinar et al. [DJ03]	two Java systems developed at the University of Paderborn, Germany	maintainability of software	measures of table 3.7
Subramanyam et al. [SK03]	software developed in two languages, C++ and Java	Software defects	CBO and two other measures of Chidamber et al. [CK94]

Table 3.10: Overview of Empirical Validation Studies for Coupling Measures

remarks highlighting emerging trends in the area of measures for UML models.

3.3.1 Measures for UML Use Case Diagrams

Jacobson introduced in 1992 [JCJO92] the concept of "use cases" as primary elements in software development , and a a diagram for visualizing them [FS00]. The use case diagram has been adopted by the UML. A use case diagram shows the relationship among use cases within a system or other semantic external entities [OMG03c]. The main constituents of a use case diagrams are modelled in UML through the following model elements: actors, use cases, use case relationships (association, includes, extend, and generalization).

Table 3.11: Use Case Proposal for Project Management

	structural property							quality focus					
Authors	actors	use cases	use case points	use case complexity and size	type of use case	include and extend rel.	association rel.	System Complexity	Development effort	System size	Maintainability	Prediction equation	Tool Support
Karner [Kar93]			X							X	X		
Marchesi [Mar98]	X	X				X		X					
Schneider et al. [SW98]			X					X	X			X	Sparx System
Smith [Smi99]	X								X			X	
Feldt [Fel00]				X			X			X			
Software Solutions on Time [oT01]		X			X							X	measure Data
Henderson Sellers et al. [HSZKP02]				X					X		X		
In et al. [IKB03]	X	X							X			X	OSMAT
Carbone et al. [CS02]								X					Fast and Serious

But, UML only focus on the diagrammatic notation of use cases. As commented by some several authors use case diagrams must be understood only as a table of contents of use cases, not as an alternative of their textual specification. In use case diagrams, only the name of the use cases, the participating actors and some use case relationships are shown. The essence of use cases, i.e. their sequence of actor-system interactions, can not be in anyway derived from use case diagrams. Therefore, it is necessary to complement use case diagrams with use cases textual specification.

Use Case diagram have been found well suited as a basis for the estimation and

planning of projects, specially many works are centered around software development effort. Use cases are also popular and widely used technique for capturing and describing the functional requirements of a software system [ADSJ01]. These estimations are based on attributes (the model elements) of a use case diagram [ADSJ01].

There are few proposals of specific measures for Use Case Diagrams, such as [Mar98] and [Sae03]. There are also some others which are specifically used for use cases, among others [Fel00], [HSZKP02], [BDG04]. Although a thorough study of measures for use cases can be found in [GPE05], we present in this section two tables which summarize a comparison of use case measures applied with two different purpose. Table 3.11 shows the proposal applied to project management, meanwhile 3.12 shows those related to improve Requirements Engineering Process.

Table 3.12: Use Case Proposal for Improving Requirements Engineering Process

Author	structural properties	quality focus	Prediction Equation	Tool support
Alexander [Ale01]	Number of use cases, number of actors, number of use cases without exceptions, etc	Status of requirements and potential problems	No	DOORS
Kim et al. [KB02]	Number of actors, number of messages in the interaction diagram associated with the use case, number of system classes associated with the use case	Importance of the requirement, impact caused by change a requirement	No	No
Saeki [Sae03]	Number of relationships and dependencies between use cases	Modifiability	No	No
Bernardez et al. [BDG04]	Number of steps of use case steps, rate of each type of step and cyclomatic complexity	Fault-proneness	No	Yes (REM)

As we intend to show the existing measures for use case diagrams, next we will outline the measures brought forward by Marchesi [Mar98] and Saeki [Sae03].

Marchesi [Mar98] proposed a set of measures for Use case diagrams complexity. He commented that the number of use cases (N_{CU}), the number of actors (N_a) and the number of include and extend relationships are good indicators of system complexity.

In [Sae03] a set of measures for use cases diagrams are defined to obtain the rate of modifiability. The basic idea of the defined measures is that if a use case needs a change, probably other use cases will also need a change: those that have a relationship with the originally changed use case. In short, include and extend relationships and control and data dependency relationships are considered. The intuition suggests that, the more existing relationships in the model, the more difficult it will be to make any change.

The type of use case is another factor that has influence in the modifiability of use

cases. Simplifying the idea, if a use case has several goals (types to Saeki), it is more susceptible of changing than if it only has one goal. In order to approximate the modifiability, the defined measures are (Number of Dependencies) and (Number of Use Case Types). The goal achieved by the author was to find an indicator rate ($0 \leq$ Modifiability ≤ 1) that would reveal the modifiability degree of a use cases model.

In our knowledge there is no evidence on the theoretical and empirical validation of these two proposals. Moreover, there is no automatic support for the measures calculation.

Only the measures defined by Karner [Kar93] were validated by Anda et al. [ADSJ01].

3.3.2 Measures for UML Class Diagrams

The main idea of this section is to show a summary of the most relevant existing proposals of measures that can be applied to UML class diagrams at conceptual level, looking at their strengths and weaknesses. Most of the measure proposals we will consider and list below were not originally defined to measure UML class diagrams, nevertheless they can be tailored for this purpose.

A class diagram has become truly central within OO methods [FS00]. It is widely used and their basic model elements are needed and familiar by everyone [FS00]. A class diagram mainly show the *attributes* and *operations* of classes and the constraints that apply to the way objects are connected. These constraints are mainly represented by graphical relationships between classes, that is, *aggregation*, *instantiation*, *association* and *inheritance* relationships.

Table 3.13: CK Measures

measure name	measure Definition
WMC	The Weighted Methods per Class is defined as follows: $$\text{WMC} = \sum_{i=1}^{n} c_i \quad (3.1)$$ Where c_1, ..., c_n be the complexity of the methods of a class with methods M_1, ...,M_n. If all method complexities are considered to be unity, the WMC = n, the number of methods.
DIT	The Depth of Inheritance of a class is the DIT measure for a class. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.
NOC	The Number of Children is the number of immediate subclasses subordinated to a class in the class hierarchy.

- Chidamber and Kemerer´s [CK94] measures. These measures, also called CK measures, were defined at class level and their purpose is to measure design complexity in relation to their impact on external quality attributes such as maintainability, reusability, etc. This proposal is among the ones widely spread and used. Only three of the six CK measures are available for a UML class diagram at conceptual level (see table 3.13).

- Li and Henry´s [LH93] measures. These measures measure different internal attributes such as coupling, complexity and size, and are successfully used to predict maintenance effort. They were defined at class level.

- Brito e Abreu and Carapua´s [eAC94] measures. They were defined to measure the use of OO design mechanisms such as inheritance, information hiding, coupling and polymorphism and the consequent relationship with software quality and development productivity. They can be applied at class diagram level.

- Lorenz and Kidd´s [LK94] measures. They were defined at class level to measure the static characteristics of software design, such as the usage of inheritance, the amount of responsibilities in a class, etc.

- Briand et al.´s [BDM97] measures. These measures are defined at class level, and are counts of interactions between classes. Their aim is the measurement of the coupling between classes.

- Marchesi´s [Mar98] measures. The aim of these measures is the measurement of system complexity, of balancing responsibilities among packages and classes, and of cohesion and coupling between system entities.

- Harrison et al.´s [HCN98a] measures. They have proposed the measure Number of Associations per class as an inter-class coupling measure.

- Genero et al. [GPE05], [GPC00] have defined and validated a set of measures for structural complexity of UML class diagrams due to the use of UML relationships, such as: associations, generalizations, dependencies and aggregations (see Table 3.14).

- Bansiya et al.´s [BD02], [BEDL99] measures. These measures were defined at class level for assessing design properties such as encapsulation, coupling, cohesion, composition and inheritance.

- In et al. [IKB03] uses measure tree to help a project manager early in the development lifecycle. He inputs UML diagrams to output some key indicators [YWG04]. The output indicators are respective total number of class,

inheritance relationships, use relationships, association relationships, roles, operation, parameters, and attributes.

- Zhou [Zho03] propose a measure which only uses one indicator, namely entropy distance based structure complexity measure, to evaluate the complexity of class diagrams. The measure defines weights for various relationships respectively. Then it gives some rules to transform a class diagram into a weighted class dependence graph. The structure complexity of a class diagram is defined as the entropy distance of the corresponding weighted class dependence graph.

- Kang [Kan04] defines a structure complexity measure for the UML class diagrams based on entropy distance. It considers complexity of both classes and relationships between the classes, and presents rules for transforming complexity value of classes and different kinds of relations into a weighted class dependence graphs. This method can measure the structure complexity of class diagrams objectively.

Table 3.14: Measures for the Structural Complexity of UML Class Diagrams

Measure Name	Definition
NAssoc	The total Number of Associations.
NAgg	The total Number of Aggregation relationships within a class diagram (each whole-part pair in an aggregation relationship).
NDep	The total Number of Dependency relationships.
NGen	The total Number of generalization relationships within a class diagram (each parent-child pair in a generalization relationship).
NAggH	The total Number of Aggregation Hierarchies (whole-part structures) within a class diagram.
NGenH	The total Number of Generalization Hierarchies within a class diagram.
MaxDIT	It is the Maximum of the DIT (Depth of Inheritance Tree) values obtained for each class of the class diagram. The DIT value for a class within a generalization hierarchy is the longest path from the class to the root of the hierarchy.
MaxHAgg	It is the maximum of the HAgg values obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the longest path from the class to the leaves.

Table 3.15 reflects if there exist published studies related to the theoretical and the empirical validation of the proposals of measures previously mentioned. Moreover the last column indicates if there exist tool for the automatic calculation of the measures.

Table 3.15: Summary of Proposals of Measures for UML Class Diagrams

Source	Validation				Tool
	Empirical		Theoretical		
	Experiments	Case Studies	Property-Based Approaches	Measurement Theory Based Approaches	
Chidamber and Kemerer [CK94]	Chidamber and Kemerer [CK94], Basili et al. [BBM96], Daly et al. [DBM+96], Cartwright [Car98], Unger and Prechelt [UPP98], Harrison et al. [HCN00], Poels and Dedene [PD01], Briand et al. [BBD01], Bandi et al. [BVT03], Subramanyam et al. [SK03]	[LH93], Chidamber et al. [CDK98], Tang et al. [TKC99], Briand et al. [BWL01], [BWDP00].	Briand et al. [BMB96], Chidamber et al. [CK94]	Zuse [Zus97], Poels et al. [Poe99]	YES
Li and Henry [LH93]		Li and Henry [LH93]			[HCN98a]
Brito e Abreu and Carapua [eAC94]		Brito e Abreu et al. [eAGE95], [eAM96], [eAEG96], Harrison et al. [HCN98b]		Harrison et al. [HCN98a]	YES
Lorenz and Kidd [LK94]		[LK94]			
Briand et al. [BDM97]		Briand et al. [BWL01], [BWDP00] , El-Emman [EBGR99], Galsberg et al. [GEMM00]		[BDW98a]	
Marchesi [Mar98]		[Mar98]			
Harrison et al. [HCN98a]		[HCN98a]			
Bansiya et al. [BD02], [BEDL99]		[BD02], [BEDL99]			YES
Genero et al. [GPE05], [GPC00]	[GPE05], [GPC00]		[GPE05], [GPC00]	[GPE05], [GPC00]	[GPE05], [GPC00]
In et al. [IKB03]		[YWG04]			
Zho [Zho03]	[Zho03]				
Kang [Kan04]		[Mar98]			

3.3.3 Measures for UML Statecharts Diagrams

Statechart diagrams (*std*) describe all of the possible states that a particular object can get into and how the object´s state changes as a result of events that reach the object [FS00]. This kind of diagrams along with others shows the behaviour of a system. The statechart diagram indicates the various states of an order, being the state connected by transition labeled with three parts, all of which are optional: *Event [Guard] / Action*.

measures for UML statechart diagrams are scanty. Derr [Der95] defined the number of states (NS) and the number of transitions (NT) as measures that measure the OMT statechart diagrams complexity (although they can also be applied to UML).

Carbone and Santucci [CS02] have proposed two measures: numSta(*std*), which stands for the total number of states for a class, and the total number of actions for a class (ie. entry and exit actions associated with a state) as numAction(*std*) . These measures are used along with others for classes diagrams, use case diagrams, etc. in order to determine the total complexity of an OO system.

Nevertheless, both Carbone and Santucci´s and Derr´s proposals have not gone beyond the definition.

We think that Miranda et al.´s [MGP03] and Cruz-Lemus´s [CLGO$^+$04], [CLGO$^+$05], [CLGPM06] work is the most complete one. With the hypothesis that the size and the structural complexity of UML statechart diagrams may influence their understandability (and therefore their maintainability), they defined a set of measures for the structural complexity and size of UML statechart diagrams.

As size measures they defined:

- NEntryA. The total Number of Entry Actions, i.e. the actions performed each time a state is entered)
- NExitA. The total number of Exit Actions, i.e. the actions performed each time a state is left.
- NA. The total Number of Activities (do/activity).
- NSS. The total Number of States considering also the Simple states within the composites states
- NCS. The total Number of Composite States
- NE. The total Number of Events.
- NG. The total Numbers of Guard conditions.

As structural complexity measures they defined:

- NT (Number of transitions). Counts the total number of transitions, considering common transitions (source and target states are different), and final transitions, self-transitions (source and target states are the same) and internal transitions (transitions inside a state that respond to an event but without leaving the state).
- CC (Cyclomatic Number of McCabe) [24] . It is defined as |NSS-NT|+2

The theoretical validity of the measures proposed by Miranda et al. [MGP03], [CLGP05] was demonstrated through the validation following Briand et al. s framework [BMB96], [BMB97], concluding that NA, NSS, NCS, NE, NEntryA, NExitA and NG are size measures; and NT and CC are complexity. Moreover, the use of DISTANCE framework [PD99] guarantees that the measures can be used as ratio scale measurement instruments.

In [CLGO+05], [CLGO+04] Cruz-Lemus et al. concludes that the measures that measure size (number of activities, number of simple states and number of guards) and structural complexity (number of transitions) in an UML statechart diagram are highly correlated with understandability efficiency. This means that the UML constructs that seem to have more impact on the subjects' understanding of UML statechart diagrams are simple states, guards, activities and transitions.

In previous works [CLGO+05] the authors studied the relationship between many of the constructs of the UML statechart diagrams and the effect that they have on the understandability of the diagrams. The authors had found that the effect of the composite states on the understandability of the UML statechart diagrams was not clear. So they designed and performed a controlled experiment and a replication in order to evaluate this effect. The results obtained show that the use of composite states improves the understandability efficiency of UML statechart diagrams if the subjects have a certain level of experience in working with this kind of UML diagrams [CLGPT05]. They conclude that using composite states when modeling the behavior of systems through UML statechart diagrams makes them more understandable.

Table 3.16 summarize the three measures proposal for UML statechart diagrams referring the elements they focus on.

3.3.4 Conclusions about Measures for UML models

The main purpose of this paper is to provide a survey of the most important work available on measures for quality attributes of UML diagrams. It aims to provide practitioners with an overall view of what has been done in the field and what measures are available to help them to take decisions in the early phases of OO development. This work should also help researchers obtain a more comprehensive view of the direction that work in UML model measurement is taking. Further

Table 3.16: Summary of Measures for Statechart Diagrams

std concepts		Derr [Der95]	Carbone et al. [CS02]	Miranda et al. [MGP03]
state	simples	NS	numSta	NSS, CC *
	composite			NCS
action	entry		numAction	NEntryA
	exit			NExitA
activity				NA
event				NE
guard				NG
transition		NT		NT, CC *
** The measure CC is derived from two different measures*				

details of some of the proposals presented in this paper can be found either in the bibliographical references where the measures were originally proposed.

As this study shows, UML class diagrams have been the subject of the most extensive research from the measurement point of view in the models field. Measures for UML use case diagrams and UML statechart diagrams have been proposed and, to a lesser extent, validated. Other UML models covering dynamic aspects of OO systems, such as sequence diagrams, activity diagrams, etc., have been largely ignored. Although the number of measures that have been proposed for UML diagrams at conceptual level is low compared to the large number defined for code or advanced design, we believe a shift in effort is required, from defining new measures to investigating their properties and applications in replicated studies. We need to have a better understanding of what measures are really capturing, whether they are really different, and whether they are useful indicators of external quality attributes such as maintainability, productivity, etc. The need for new measurements will then arise from, and be driven by, the results of such studies.

In this area designers also ask for desirable values for each measure. However, we must be aware that the hard part is to associate the qualifications good and bad to numeric ranges. This makes measures all the more useful for OO system designers, to help them make better decisions in their design tasks, which is ultimately the most important goal of any worthwhile measurement proposal. As a final remark we would conclude by saying that clearly the field of quality measures for UML models needs to mature. We believe that further empirical validation is necessary, in particular by applying measures to models obtained from real projects, in order to build up a solid body of knowledge about the usefulness of measures in practical situations.

3.4 Contribution to the Dissertation

In this chapter we provide an introduction to OCL language and we describe the different utilities of OCL, in Model-Driven development, model transformations and in several stages at the Meta-Object Facility (MOF) metamodel. In this chapter we also provide a clear picture of the state-of-the-art about measures for UML models (see section 3.3). Likewise, we provide a survey of coupling measures, in order to take into account the most important framework used for coupling measurement (see section 3.2), the different coupling measures defined in the literature (see section 3.2.2), and their empirical validation (see section 3.2.2.1). From the last criterion we are agree with El Eman et al. [Ema02] when they argue that the most promising results with OO measures were obtained using coupling measures.

These relevant empirical studies suggests that there are important relationships between coupling and several external quality indicators. We show that exists proved evidence that some forms of coupling have an impact on understandability, maintenance effort, fault proneness, impact analysis and quality guidelines. However, there is no evidence of the influence of coupling on the maintainability of OCL expressions, this motivated to propose a set of measures for assessing coupling of OCL expressions (see chapter 4) and later on validate them as maintainability indicators (see chapter 7 and 8).

Chapter 4

A Proposal of Measures

In this chapter we will present the measures we propose for OCL expressions within UML models. We structured the presentation of this chapter into two sections, the first section deals with the Identification activity and the second one describes the Definition subactivity of the Creation activity. We only present the Definition in Natural Language, the formal definition is described in chapter 5. Each section corresponds to the sequence of activities depicted in Fig. 2.2 and Fig. 2.4(b), respectively. Whenever an activity of the method is applied, the corresponding section is titled accordingly and a reference to the number of the method's activity is included between parenthesis.

4.1 Identification (M_1)

For identifying the measures we will follow the steps detailed in the UML activity diagram shown in Figure 2.2.

4.1.1 Select the Entity of Study (I_1)

The measuring activities will be focused on a new kind of software artifact: the OCL expression. Nevertheless, we must recall that these artifacts are the primary elements that modelers use as textual add-on to UML models. Although an expression is attached to a particular contextual type, its meaning involves objects (mentioned inside 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 on OCL expression, we can not study this artifact in an isolated way. Its context and its scope are intrinsically involved.

4.1.2 Determine the Quality Focus (I_2)

The ISO/IEC 9126 [ISO01] defines software quality as composed of six external characteristics of interest, namely: functionality, reliability, efficiency, usability, maintainability and portability. In turn, each of these quality characteristics is refined into sub-characteristics.

Maintainability is a particularly interesting quality attribute due to the fact it has been recognized that software maintenance activities account for the largest cost in today's software development [DJ03]. The IEEE Standard Glossary of Software Engineering defines maintainability as 'the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment'.

In this Ph. D. thesis the quality attribute *OCL expression' maintainability* has been chosen as the prime attribute of interest. Maintainability has been and continues to be an expensive and challenging task, and it is often poorly managed. One reason for poor management is the lack of proven measures for software maintainability. Our study about the OCL expression maintainability will help the modelers to improve the quality of models, and this is a major goal in software development using MDA [GJG04], due to the fact models are used to drive the entire software development process.

To our knowledge, not all the maintainability sub-characteristics proposed in that standard are suitable for OCL expressions. We used two sub-characteristics:

- Understandability (comprehensibility): The capability of the OCL expression to be understood by modelers[1]. Understandability is a sub-characteristic of Usability.

- Modifiability that includes two sub-characteristics of maintainability of the ISO 9126 standard for software quality [ISO01]:

  - Analyzability: The capability of the OCL expressions to be diagnosed for deficiencies or for the identification of the parts to be modified.

  - Changeability: The capability of the OCL expressions to be changed when modifications are required.

[1]Even though understandability has not been considered as a maintainability subcharacteristic by ISO 9126 [ISO01], we included it because there exists a lot of work related to software measurement that considers understandability as a factor that influences maintainability [FP98], [HCN00]

Table 4.1: Goal of OCL Expression Measures

Object of study:	OCL expression
Purpose:	Evaluation
Quality focus:	Maintainability
Viewpoint:	OO software modelers
Environment:	OO software organizations

4.1.3 State the Goal (I_3)

We used the template provided by the QGM paradigm in order to define our measurement goal, which is shown in Table 4.1. The GQM-goal is: Analyse OCL expressions with the purpose of evaluating maintainability from the viewpoint of the OO software modelers in OO software organizations. The object or study and the quality focus were described in the last two subsection. The purpose is evaluation, i.e. 'judge the value of'.

4.1.4 Determine the Structural Properties to be Studied (I_4)

External quality attributes as maintainability subcharacteristics can only be measured late in the IS life cycle. We therefore need to identify early quality indicators based, for example, on the structural properties of OCL expressions. Thus, our purpose is to define measures to quantify OCL expressions structural properties and afterwards to ascertain how each of these measures are related to each of the maintainability sub-characteristics.

We focus on the degree to which the elements in a design are connected, i.e. on coupling structural property. Coupling is generally recognized as being among the most likely quantifiable indicator for software maintainability [BDV04]. In fact, if one intends to build quality OO models, coupling will very likely be an important structural dimension to consider [BWSL99].

However, coupling as we presented in chapter 3 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 [BDW99], i.e. on import-coupling. We focus on import-coupling due to:

- **The inner nature of OCL expressions**: These artifacts are textual add-on to UML models, within an expression we can refer to UML artifacts but not the other way around. OCL expressions are toughly coupled to UML diagrams,

importing artifacts to express a constraint, query, pre- or post-condition, etc.

- **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 [BWSL99], and fault-proneness result in low maintainability [BDW99]. Similar results about import-coupling were obtained as an indicator of development effort [BW01], [BWDP00], [BWL01], where export-coupling measures show a much weaker impact than import-coupling. High import-coupling can have the following effects:
  - **Decreased maintainability**: changes to the supplier may need follow-up changes (ripple effects) to the client. The stability of the supplier is a factor to consider here. High coupling to elements that are not likely to change is less harmful than coupling to variation points.
  - **Decreased understandability, increased fault-proneness**: elements with high import-coupling operate in large context, developers need to know all the services the element relies on, and how to use them.
  - **Decreased reusability**: To reuse a class or package with high import-coupling in a new context, all the required services must also be made available in the new context.

A clear identification of the abstraction we used to measure OCL expression coupling is deferred to the next section. In the following we continue explaining the several aspects related to the structural properties of the software artifact we studied.

We are aware that size was studied as a confounding factor for many coupling measures [EBGR01]. El-Eman et al. [EBGR01] had demonstrated a strong size confounding effect of class size in validation studies of object-oriented measures and cast doubt on the results of previous empirical validation. In [Ema01] El Eman recommends to use coupling measures along with size measures in fault-proneness prediction. So, size structural properties should be carefully considered during the study of coupling measures. We decided to define some size measures to control that size would not bias our findings during experimentation.

As far as cohesion is concerned and measured today, it is very likely not a very good maintainability indicator, such as in fault-proneness [BWSL99]. Nevertheless, we think that a study of cohesion measures in OCL expression maintainability should be done, and this is a future research direction we are interested in. However, in a recent research work Darcy et al. [DS05] argue and prove that when designing and maintaining software to control complexity, both coupling and cohesion should be considered jointly, instead of independently. We believe that this result does not affect our GQM goal due to the fact we are studying OCL expression's import-coupling, that is the dependence of an OCL expression on other software parts.

Nevertheless, when the quality of a set of OCL expressions is being considered, for instance to measure the quality of a class (or a method) within a UML/OCL model, coupling and cohesion should be jointly considered.

4.1.5 Identifying Abstractions for Coupling (I_5)

In order to identify abstraction for measuring coupling within OCL expression we used as a basis the Briand et al. framework [BDW99] for coupling measurement. However the application of this theory is not straightforward, due to the fact that OCL concepts are not addressed in the framework (only the more typical OO artifacts are considered: attributes, methods, classes, etc.).

As Briand et al. argue there are many forms of coupling that can arise in systems [BAC+99]. We believe that one important contribution of this thesis is to establish a new form of coupling in OO systems.

When UML models are complemented by OCL expressions, within the expressions it is possible to use different artifacts of UML models, and the OCL language defines different concepts from which it is possible to define new OO connections.

Briand et al. framework is based on six criteria (defined in section 3.2) which are applied to OCL expressions in the following:

1. **Type of Connections**: Connections are inherent to any coupling measure. Usually in a connection two entities are involved. A client (or source) entity specifies a connection to a destination entity (see Fig. 4.1 (a)). The coupling connections we are interested in are connections between OCL expression and any OO feature of an UML diagram. So, in our case the source entity will be always an OCL expression, meanwhile the destination entity varies radically. Table 4.2 describes the type of connections and the measures which capture them. Although the source entity is an expression, its meaning depend on the UML artifact to which the expression is attached [2].

2. **Locus of impact**: The coupling usually defines a client-supplier relationship between the design elements. This criterion defines if we focus on defining measures for the client or the server entity (in the connection). If the focus is the client (see Fig. 4.1 (b)), the locus of impact is import-coupling, otherwise (see Fig. 4.1 (c)) 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 UML diagram (allows the modeler to specify

[2] expression can be attached to attributes, method, classes or rolenames, depending on the label used (*def*, *pre*, *post*, *body*, *inv*, etc.) before the expression is defined.

explicit references to UML features) constitutes a suitable mechanism to focus on the import-coupling. So, the focus is the client entity.

3. **Granularity**: This criterion involves:

   - The **domain of measure** is always an OCL expression. Nevertheless, the expression refers to semantic properties of its contextual type. Although an OCL expression seems to be a small domain, the scope of objects referred through a expression (the portion of a UML diagram imported by an OCL expression can vary significatively) can be very large.
   - The **way we count connections** -henceforth, WWCC - is the following: we always count the number of different items at the other end of the connections.

4. **Stability of server**: We did not take into account this criterion, see consideration in section 3.2.1.

5. **Direction of the connections**: Our coupling measures are defined considering direct connections with exception of navigation measures.

6. **Inheritance**: Inheritance aspects are only considered in NIO measure, that involves the comparison of contextual type with inherited types. Our purpose is to study coupling mechanism that are declared through OCL expression, and inheritance aspects are defined in the UML models, specially through the class diagram. In the empirical studies we did not focus on the structural properties of expression and UML models at the same time.

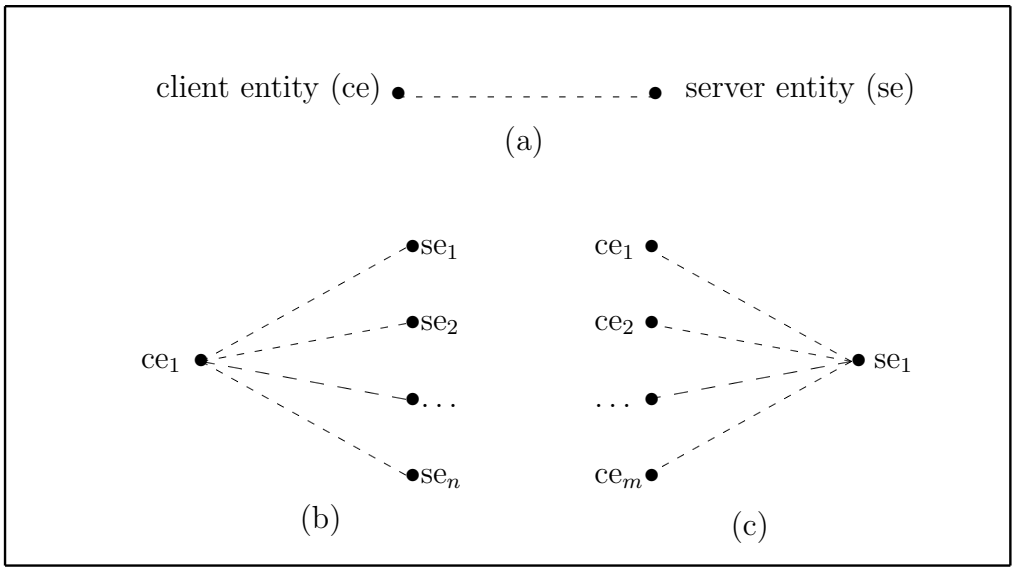


Figure 4.1: Coupling Connections

4.1.5.1 Modeling the Selected Abstractions

As Card recommended in [Car93] one effective supplemental activity of GQM is to develop a model of the entities and relationships measures. And, any modelling activity requires abstraction, being abstraction the mathematical representation of an entity under study [BMB02], often modeled via a graph. Within an abstraction an entity has to be mapped into one or more abstractions so it becomes analyzable and its relevant attributes become quantifiable [MGBB90]. The construction of mapping from the entity to the abstraction is crucial, and needs to be checked for completeness and suitability, i.e. we must verify that the abstraction contains all the relationships that one wants to capture, also the level of granularity of the abstraction should be considered whether it is accurate enough [BMB02], etc. Briand et al. also recognize that although several abstractions capturing control flow, data flow and data dependency information are available in the literature, an even larger variety of abstractions can be derived from software products [BMB02].

Regarding these considerations, we had modelled the entities we are measuring considering as a reference the framework of Briand et al.[BMB99] for coupling and cohesion interaction-based measures. The entities to be studied are software parts, OCL expressions, and the attribute to be studied is import-coupling. The object-based context was defined through dependencies among OCL expressions and UML artifacts (i.e., attributes, operations, rolenames, objects, etc). These dependencies are called interactions and were used to define measures capturing coupling between software parts.

An interaction graph was used as an abstraction to model the elements and relationships that are relevant for capturing import-coupling. The interaction graph contains the elements that are data declarations of a high-level design, in our case UML models, and whose relationships are the interactions among data declarations and data used. The description of interaction graph is deferred to chapter 6 due to the fact it is mainly used to theoretically validate the OCL expression measures (for instance, Figure 6.1 contains the interaction graph corresponding to a UML diagrams with different OCL expression attached).

4.1.6 Refine the Goal into Questions (I_6)

Based on Briand et al. model [BWIL99] we hypothesized that OCL expressions maintainability are influenced by its structural properties, which, in turn depends on elements that compose OCL expressions (navigations, collection operations, variables, etc.). So, the most important question arises:

- Does coupling influence OCL expressions maintainability?

So, we added two other questions:

- Does size influence OCL expressions maintainability?
- Does length (of navigation) influence OCL expressions maintainability?

Nevertheless, the last two questions arise with two different purposes. Length of navigations is closely related with the depth of coupling. Size property was considered in order to control during experimentation that size aspects does not bias the findings related to coupling.

Table 4.2: Type of Connections for OCL Expression Measures

source	**destination**		**measures**
OCL expression	contextual instance		NES, NIS
contextual instance of the OCL expression	contextual objects	references to the objects from the collection	NEI, NII
		parameters	NPT
		data types	NUDTO, NUDTA
	property of an object	attribute	NAS
		method	NOS, N@pre, NON, WNN
		rolename	NNR, WNN, NNC
	manipulation of objects		WNCO
	a classifier in a hierarchy		NIO
OCL expression	OCL expression		NVD

4.1.7 State General Hypotheses (I_7)

We hypothesize that high import-coupling of OCL expression affects the maintainability of OCL expressions.

4.2 Concepts Related to the Measured Attributes

Although our objective is to evaluate OCL expression maintainability we are conscious that maintainability is an external quality attribute and therefore it is influenced by structural properties of the OCL expressions.

As Fenton and Pfleeger [FP98] suggest that it is not advisable to define a single measure for capturing different structural properties, we will define several measures, each of which captures different structural properties for coupling, length and size respectively according to our GQM questions. This section is part of the definition in natural language of measures for OCL expressions. Nevertheless, before starting the definition of the measures we should describe which are the OCL concepts involved with the aforementioned structural properties. For that purpose a metamodel was selected: the OCL metamodel. Although the metamodel is explained in detail in the next chapter, in this section we present it in relation to coupling, size and length attributes.

4.2.1 OCL Concepts Related to Coupling

The OCL concepts related to coupling allow the modeler to specify an OCL expression for a particular contextual type in terms of its context, i.e. to write an expression using properties of the contextual type or properties of other Classifiers which are coupled to the contextual type. By using properties of other classifiers, you import their meaning in the OCL expression scenario. When references to other classifiers, implicit assumptions can turn invalid over time [BDV04].

The concepts related to properties of the Contextual Type are:

- **Accessing attributes and operations belonging to the contextual type.** Using the contextual instance and the dot notation it is possible to refer to attributes and operations of the contextual type. Considering the OCL expression shown in the example 4.2.1, two properties are referred :
  - The level attribute belonging to Person, and
  - The age() operation belonging to the same type.

  Example 4.2.1

```
context Person inv:
        self.level = "Senior" implies self.age() = 21
```

- **<<definition>> constraints.** To allow the reuse of a variable and/or operation over multiple OCL expressions it is possible to define a <<definition>> constraint, using the keyword *def*. In fact, this OCL expression means a stereotype <<definition>>, and the constraint is attached to a Classifier. The keyword *def* can be used after the attribute or operation definition. The constraint is exemplified as follows:

Example 4.2.2 *In the next OCL expression a variable called income is defined.*

```
context Person
        def : income : Integer = self.job.salary-> sum()
```

The income variable is known in the same context as any property of Person. For example in:

```
context Person inv:
        if self.isUnemployee
        then income < 100
        else income >= 100 endif
```

- **Predefined properties that can be applied to any object.** The following predefined operations which are commonly used with inheritance concepts can be applied to any objects:
  - oclIsTypeOf (t :OclType): Boolean: The operation returns *true* if its argument (t) is equal to the type of *self*.
  - oclIsKindOf (t :OclType): Boolean: The operation determines whether t is either the direct type or one of the supertypes of an object.

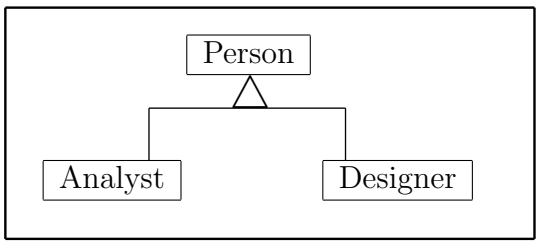


Figure 4.2: An Example of a Class Diagram

Example 4.2.3 *According to the class diagram shown in Fig. 4.2, the following examples are defined in the context of the Designer class :*

```
self.oclIsTypeOf(Person) = false
self.oclIsKindOf(Person) = true
self.oclIsTypeOf(Designer) = true
self.oclIsKindOf(Designer) = true
```

  - oclAsType (t :OclType): instance of OclType: Property of supertypes when they are overridden within a type, they can be accessed through oclAsType().

Example 4.2.4 *If B is supertype of A then it is possible to write:*

```
context B inv:
        self.oclAsType(A).p1
```

in order to refer to the p1 property of A.

- **Accessing previous values in postcondition.** Whenever a property is postfixed with the @pre keyword in a postcondition, the value accessed is the property value before the execution of the operation.

  Example 4.2.5 *In the following example taken from Cook et al. [CKM+02] the usage property refers to the property of Bathroom whereas usage@pre refers to the value of usage before the execution of the uses operation.*

```
context Bathroom::uses (g: Guest)
pre:    .....
post:   usage = usage@pre + 1
```

In the following we describe the concepts involving other objects (different to the contextual type):

- **Navigations.** Starting from a specific object it is possible to navigate an association in the class diagram, to refer to other objects and their properties [OMG03b]. A relation is navigated when we use the rolename of the opposite association-end of a relation, that links the class where the expression is defined with another class in the diagram class (when the association-end is missing we can use the name of the type at the association-end as the rolename). The result of a navigation is a single object or a collection of objects depending on the multiplicity of the association-end [RG98]. The syntax uses the dot notation followed by an association-end property. It is possible to navigate many relationships in order to access as many properties as needed in an expression.

  Navigations can be simple or combined. Whenever we navigate through more than one relationships the navigations is combined, otherwise a simple navigation is used (only one relationship is navigated).

  Example 4.2.6 *The following expression is specified for the class diagram of Figure 4.3.*

```
context LoyaltyProgram inv:
        membership.card-> forAll (goodThru = Date::fromYMD(2007,1, 1))
        and self.customer->forAll (age()>30)
```

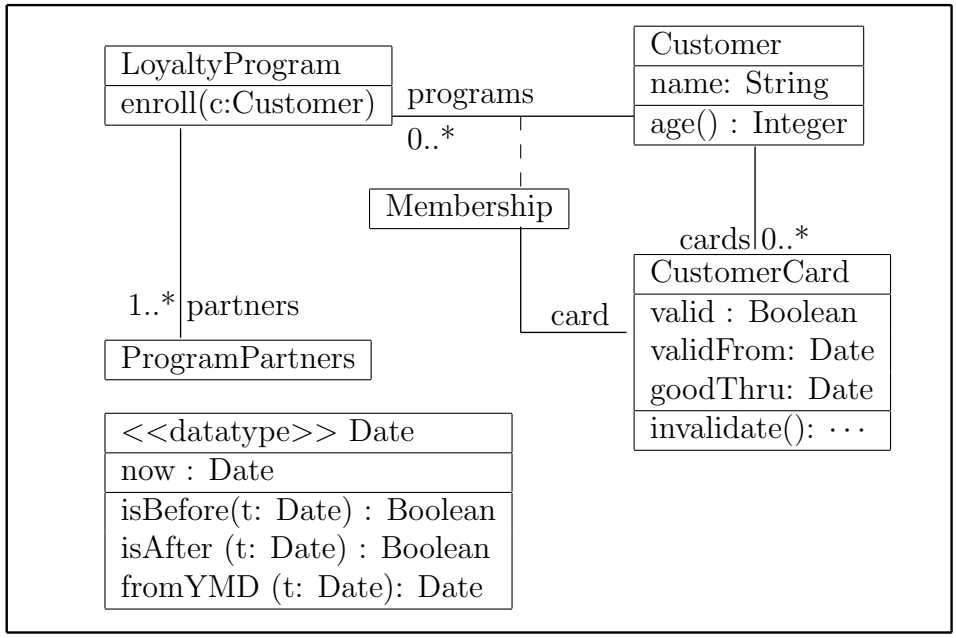


Figure 4.3: Part of the Loyal and Royal Class Diagram

membership.card represents a navigation from LoyaltyProgram to CustomerCard. This is an example of a combined navigation. membership.card navigates two relationships: one from LoyaltyProgram to Membership (an association class), and another from Membership to CustomerCard. In the former relationship there is no rolename attached to the association-end where Membership is the sink class, and for that reason the name of the class is used. Meanwhile, in the latter relationship, the navigation is represented by its rolename "card". The expression also contains another navigation self.customer (a simple navigation).

- **In, Out and In/Out Parameters, and Return Values**. Operations may have *in*, *out*, *in/out* parameters. If the operation has *out* or *in/out* parameters, the result of this operation is a tuple containing all *out*, *in/out* parameters and the return value [OMG03b].

- **Collection Operations.** OCL defines many operations for handling the elements in a collection. The operations allow the modeler to project new collections from the existing one. Operations like *select*, *reject*, *iterate*, *forAll* and *exists*, take each element in a collection and evaluate an expression for them. The expression evaluated for each collection can be defined in terms of new navigations. We will take into account those expressions of collection operations which are defined in terms of other navigations.

All collection operations are specified using the arrow-syntax notation: collection->OperationCollection(Boolean-expression).

Example 4.2.7 *self.customer of example 4.2.6 specifies a forAll collection operation to express that all LoyaltyProgram customer must be greater than 30 years old.*

- **Messages**. OCL message expressions are used to specify the fact that an object has, or will send some message to another object at some moment in time [OMG03b], [WK03].

Example 4.2.8 *The following expression broadcasts a message called update to the observers of a subject.*

```
context Subject:: haschanged()
post:        observer ^update(?:Integer, ?:Integer)
```

- **User-Defined DataType**. A data type is a special kind of classifier, similar to a class, whose instances are pure values (not objects). Usually, a data type is used for specification of the type of an attribute. A data type is denoted using the rectangle symbol with keyword <<dataType>> or, when it is referenced by e.g. an attribute, denoted by a string containing the name of the data type [OMG03c].

Example 4.2.9 *A user-defined data type called Date is included in the class diagram of Figure 4.3. The expression of the example 4.2.6 uses it and access to its property (the fromYMD property).*

4.2.2 OCL Concepts Related to Length

The navigation is also a concept related to length. The length of coupling is defined in terms of the distance from the contextual type to those objects to which the expression navigates through a simple or combined navigation. The greater the combined navigation the greater the distance from the contextual instance to the coupled object.

Table 4.3: Predefined Iterator Expressions

Collection		**Set**	**Bag**	**Sequence**
exists	any	select	select	select
forAll	one	reject	reject	reject
isUnique	collect	collectNested	collectNested	collectNested
		sortedBy	sortedBy	sortedBy

4.2.3 OCL Concepts Related to Size

In this group we have included those OCL concepts that are intrinsic to the language itself. The concepts are:

- **Logical operators.** The Boolean type is a predefined type composed of two literal values: *true* and *false*. OCL defines the following logical operators for Boolean: *or*, *xor*, *and*, *not* and *implies*. It is common to use logical operators in an OCL expression because they represent general connectors of subexpressions.

- **Predefined iterator expressions.** The semantic of predefined iterator expressions is defined in terms of an iterate expression. The set of standard iterator expressions defined in OCL [OMG03b] is included in Table 4.3.

  For example the *reject* operation allows us to obtain a subset from a collection. The subset obtained from the collection using *reject*, is composed of all the elements of the collection from which the expression evaluates to false. However, the operation can adopt three different forms (see definition 4.2.10). The last two forms include an iterator variable, being the iterator, in the last form, specified by its type. The iterator variable is used to refer explicitly to the collection elements. The use of these predefined expressions involves dealing with collections and iterators.

Definition 4.2.10 *Definition of reject operation:*

```
collection->reject( Boolean-expression )
collection->reject( v | Boolean-expression-with-v )
collection->reject( v : Type | Boolean-expression-with-v )
```

4.3 Definition in Natural Language (D_2)

In this section we include the result of the application of the Definition of Measures in Natural Language of the Creation activity (Fig. 2.4(b)). The set of measures defined are included in Table 4.4.

Each measure is defined using a consistent format composed of:

- ACRONYM and NAME: this component shows the result of activity N4 of Fig. 2.4(b).
- Proper DEFINITION: this component involves the result of applying N1 and N2 activities of Fig. 2.4(b).
- INTENT: this component describes the goal of the measure, and corresponds to the application of activity N3 of Fig. 2.4(b).
- EXAMPLE: we had included a sample to illustrate its calculus.

The definition of the measures is presented according to the attributes they are related to.

4.3.1 Measures for Length

Within a UML model, a class is shown in class diagram along with its relationships with its surrounding classes, the class can be coupled to other classes through indirect connections. However, graphical information does not express whether their methods trust in classes that are indirectly connected to the class. For example, in Figure 4.3, the LoyaltyProgram class presents three direct relationships (to Customer, Membership and ProgramPartners) but indeed is also coupled to CustomerCard due to the class verify (through an OCL expression) that all its memberships had a valid card. More precise information is available once its methods are implemented, and we can compute the exact quantity of coupled classes (for instance, if we use CBO measure). But this is obtained in later phases of software development. However, in UML/OCL models the OCL expression reveals more precise coupling information when invariant or pre- post-condition expressions are declared. Using the navigation specified in the expression we can estimate, in a more accurate way, how coupled is the class with its context. This is the rationale of DN, one of the most important measure we define. DN controls the depth of the coupling. It is stated that as one goes further to more distant coupled classes, the more complex a class become and, hence, more difficult its comprehension and modification is.

- DN: Depth of Navigations.

Table 4.4: Measures for OCL Expressions of UML/OCL Models.

MEASURE ACRONYM	MEASURE GROUP	MEASURE DESCRIPTION
DN	Length	Depth of Navigations
NAS	Coupling	Number of Attributes belonging to the classifier that Self represents
NOS	Coupling	Number of Operations belonging to the classifier that Self represents
NIO	Coupling	Number of oclIsTypeOf, oclIsKindOf or oclAsType Operations
N@P	Coupling	Number of properties postfixed by @ Pre
NNR	Coupling	Number of Navigated Relationships
NAN	Coupling	Number of Attributes referred through Navigations
NON	Coupling	Number of referred Operations through Navigations
NNC	Coupling	Number of Navigated Classes
NPT	Coupling	Number of Parameters whose Types are classes defined in a class diagram
NUDTA	Coupling	Number of User-Defined Data Type Attributes
NUDTO	Coupling	Number of User-Defined Data Type Operations
WNN	Coupling	Weighted Number of Navigations
WNCO	Coupling	Weighted Number of Collection Operations
NEI, NII	Coupling	Number of Explicit or Implicit Iterator variables
NKW	Size	Number of OCL KeyWords
NES	Size	Number of Explicit Self
NIS	Size	Number of Implicit Self
NBO	Size	Number of Boolean Operators
NCO	Size	Number of Comparison Operators

1. DEFINITION: Given that in an OCL expression there can be many navigations regarding its definition, we build a tree of navigation using the class name to which we navigate. We will only consider navigations starting from the contextual instance (from *self*). The root of the tree is the contextual type. Then we build a branch for each navigation, where each class we navigate to is a node in the branch. Nodes are connected by "navigation relations". DN is defined as the maximum depth of the tree. When a navigation includes a collection operation expression defined in terms of a new navigation(s), we will build a new tree for the navigation used in the collection operation expression, using the same method, then we will connect both trees using a "definition connection". A dashed line will represent a definition connection. When we obtain the depth of the tree, we will apply the following rule: "Navigation connection is counted once, and definition connection twice".

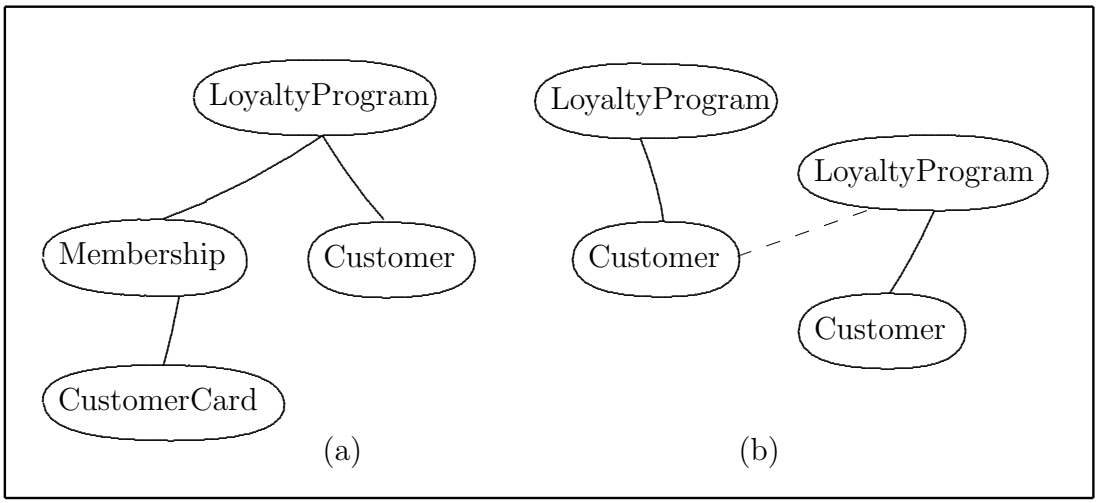


Figure 4.4: Example for Illustrating DN Measure

2. INTENT: A high depth of navigations may involve a complicated navigation. Warmer et al. [WK03] suggest avoiding complex navigation expressions, they also argue that: "using long navigation makes details of distant objects known to the object where we started the navigation".

   This measure was proposed as a measure of OCL expression complexity and class complexity. It is based on the idea that a high value of the measure will be an indicator of how distant the objects known by the contextual type are.

   Example 4.3.1 *A tree built for the following expression -using the method described above- is shown in Figure 4.4 (a). In this example the value of DN is 2.*

```
context LoyaltyProgram inv:
        membership.card -> forAll (
        goodThru = Date::fromYMD (2007,1, 1))
        and self.customer->forAll (age()>30)
```

   Example 4.3.2 *According to the following expression, the tree built is shown in Figure 4.4 (b), where a dashed line represents a definition connection. The DN value for the expression of Figure 4.4 (b) is equal to 4.*

```
context LoyaltyProgram inv:
        self.customer ->forAll( age() <= 30) and
        self.customer ->forAll (c1 | self.customer ->
        forAll (c2| c1 <> c2 implies c1.name <> c2.name )
```

4.3.2 Measures for Coupling

In this section we define a set of measures for OCL expressions considering those elements which involve coupling (see Table 4.4).

- NAS: Number of Attributes belonging to the classifier that *Self* represents.

  1. DEFINITION: This measure counts the total number of attributes belonging to the contextual type. The attributes are directly referred to using the notation self.attributename.
  2. INTENT: A higher number of this kind of attributes will increase the complexity of the expression. The comprehension of attributes used in an expression not only involves the meaning of them as a constituent of a class diagram but also the different OCL expressions that declare restrictions on them.

     Example 4.3.3 *In the following expression, two attributes of Person are used, title and isMale, the former has an implicit self instance, while in the latter it is explicit, thus the value of NAS is 2.*

     ```
     context customer inv:
     title =     (if self.isMale = true
                 then 'Mr.' else 'Ms.'
                 endif)
     ```

- NOS: Number of Operations belonging to the classifier that *self* represents.

  1. DEFINITION: This measure counts the total number of Operations belonging to the contextual type. These operations are directly referred to using the notation self.operationname.
  2. INTENT: The same goal as NAS but considering operations instead of attributes. The comprehension of an operation involves the comprehension of its meaning as a class diagram constituent and also the constraints associated to it (pre and postconditions, body expression, etc.).

     Example 4.3.4 *The value of NOS in the OCL expression of example 4.2.1 is 1, only one operation is used: age().*

- NIO: Number of oclIsTypeOf, oclIsKindOf or oclAsType Operations.

  1. DEFINITION: This measure counts the number of times an *oclIsTypeOf*, *oclIsKindOf* or *oclAsType* operation is used in an expression. These are some predefined properties. NIO uses the contextual type and other types connected to the contextual type through inheritance.

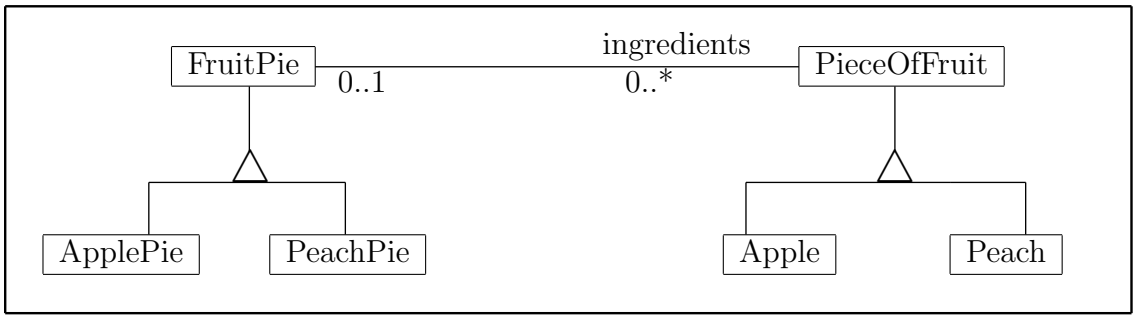


Figure 4.5: A Class Diagram used for Exemplifying NIO Measure

2. INTENT: A high number of this kind of predefined operations can increase the complexity of the expression, as the modelers have to deal with inheritance concepts. The complexity will also depend on the complexity of the inheritance tree in which the contextual type is included.

   Example 4.3.5 *Given the class diagram of Figure 4.5 and the following expression of the ApplePie class:*

   ```
   context ApplePie inv:
           self.ingredients->forAll(oclIsKindOf(Apple))
   ```

   The value of NIO is 1, because the expression uses oclIsKindOf().

- N@P: Number of properties postfixed by @Pre.

  1. DEFINITION: This measure represents the number of different properties postfixed by @pre. This measure can be used exclusively for postconditions.
  2. INTENT: A high number of variables postfixed by @pre could increase the complexity of an OCL expression.

     Example 4.3.6 *In the expression of example 4.2.5 the value of N@P is 1, as the postfix @pre is used with the usage property.*

- NNR: Number of Navigated Relationships.

  1. DEFINITION: This measure counts the total number of relationships that are navigated in an expression. If a relationship is navigated twice, for example using different properties of a class or interface, this relationship is counted only once. Whenever an association class is navigated we will consider the association to which the association class is attached.

2. INTENT: As Warmer and Kleppe [WK03] 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 is the context to be understood.

   Example 4.3.7 *In the following expression valid for Fig. 4.3 two different relationships are navigated: (1) the relationship between LoyaltyProgram and Customer (it is navigated from LoyaltyProgram to Membership, and from LoyaltyProgram to Customer), (2) the relationship between Membership and CustomerCard.*

```
context LoyaltyProgram inv:
        membership.card -> forAll (
        goodThru = Date::fromYMD (2007,1, 1)
        and self.customer->forAll (age()>30)
```

   thus NNR = 3 because three relationships were navigated.

- NAN: Number of Attributes referred through Navigations.

  1. DEFINITION: This measure counts the total number of attributes referred through navigations in an expression.
  2. INTENT: NAN measures the extent of usage of attributes of other classes by the contextual type. The larger the set of attributes referred through navigations, the greater is the context to be understood.

     Example 4.3.8 *In the expression of example 4.3.7 valid for Figure 4.3 only the goodThru attribute is used, thus NAN = 1:*

- NON: Number of referred Operations through Navigations (NON).

  1. DEFINITION: The measure is defined as the count of operations which are referred through navigations.

  2. INTENT: NON measures the extent of usage of operations of other classes by the contextual type. The larger the set of operations (referred through navigations) the greater is the context to be understood.

     Example 4.3.9 *The following operation "income" has a result of type Integer, an in parameter (d) and an out parameter (bonus):*

```
context Person::income(d: Date, bonus: Integer): Integer
post:       result = type { bonus = ···,
            result = ··· }
```

Now, consider an expression in which we navigate to the Person class, and we operate with the two returned values of income:

```
context Salary::calculate()
post:       person.income(aDate).bonus + person.income(aDate).result
```

The value of NON measure for the postcondition expression is 1 although the income operation was used twice.

- NNC: Number of Navigated Classes.

  1. DEFINITION: This measure counts the total number of classes, association classes or interfaces to which an expression navigates. If a class contains a reflexive relation and an expression navigates it, the class will be considered only once in the measure. Also, as a class might be reachable from a starting class/interface from different forms of navigations (i.e. following different relationships) we must consider this situation as a special case: If a class is used in two (or more) different navigations the class is counted only once.
  2. INTENT: Warmer and Kleppe [WK99] argue that "any navigation that traverses the whole class model creates a coupling between the object involved". A high number of navigated classes will increase the coupling between the objects.

  Example 4.3.10 *In the expression of example 4.3.8, the value of NNC = 3, because the classes Membership, Customer and CustomerCard are used.*

- NPT: Number of Parameters whose Types are classes defined in a class diagram.

  1. DEFINITION: This measure is specially used in pre and postcondition expressions and it counts the method parameters, and the return type (also called result) used in an expression, each parameter/result having a type representing a class or interface defined in the class diagram.

2. INTENT: In an OO system a typical method of communication is by using an object as a parameter [GHJV95]. Parameters can be used in the specification of an OCL constraint. However if the quantity of parameters whose types are classes in the class diagram is high, the context of the object involved will affect the understanding of the OCL expression.

   Example 4.3.11 *In the following expressions (both, pre- and post- conditions, are valid expressions for the LoyaltyProgram class of Fig. 4.3.), the value of NPT = 1 because only one parameter (c), whose type is a class in the class diagram (Customer), is used in the expression.*

   ```
   LoyaltyProgram::enroll(c: Customer)
   pre:    not customer->includes( c )
   post:   customer = customer@pre-> including ( c)
   ```

- NUDTA: Number of User-Defined Data Type Attributes.

  1. DEFINITION: This measure counts the total number of attributes belonging to a user-defined data type used in an expression. Attributes are counted once if they belong to the data type class, even if they are used more than once.
  2. INTENT: NUDTA is a measure of the potential reuse of user-defined data type attributes.

     Example 4.3.12 *In the following expression the value of NUDTA = 1 because only one class attribute (now) of a data type (Date) is used.*

     ```
     context CustomerCard inv:
                 validFrom.isBefore(goodThru) or
                 goodThru.isAfter(Date::now) implies self ^invalidate()
     ```

- NUDTO: Number of User-Defined Data Type Operations.

  1. DEFINITION: The definition of this measure is analogous to the NUDTA measure, but considering operations instead of attributes.
  2. INTENT: NUDTO is a measure of the potential reuse of user-defined data type operations.

     Example 4.3.13 *In the expression of example 4.3.12, NUDTO = 2 because the isBefore and isAfter operations (belonging to the data type Date) are used.*

- WNN: Weighted Number of Navigations.

1. DEFINITION: As we explain in the section 3.2.3. an operation collection is composed of an expression which is evaluated for each collection element, and if the evaluated expression involves a new navigation (or many) we will give a higher weight to the new navigation used inside the definition of the outermost expression. As the collection operation can be defined in terms of a new navigation and its collection operations, i.e. in a recursive way, we will refer to the different compositions of navigation as "level". In the case that navigation B is used in the immediate definition of an operation collection for a navigation A, we would say that B is in level 2 and A in level 1. The weight associated with each level is equal to the level number. Therefore the definition of the WNN measure is:
   WNN = Σ weight of the level $\times$ number of navigations of the level.
2. INTENT: This measure is an estimate of overall coupling among objects (those involved through relationships) in the specification of an OCL expression. The value of WNN will provide an indicator of how the relationships are used together for specifying semantics of an expression in terms of a coupling set of objects. A high number of WNN will indicate an intertwining specification of relationships and this could reduce the understandability of an OCL expression.

   Example 4.3.14 *In the precondition expression of example 4.3.11, the value of WNN is 1, and there is only one navigation (self.customer). Now, we will show how the WNN is obtained in the expression of example 4.3.2.*

   Two subexpressions are connected by an AND operator. Each subexpression involves navigations. Whilst the navigation of the first subexpression does not include a new navigation in its evaluation, the second one uses a collection operation defined in terms of another, and the value of WNN is obtained in the following way: **1** ** 2 +* **2** ** 1 = 4. The number shown in bold print font represents the applied weight, and the number shown in normal font indicates the number of navigations.*

- WNCO: Weighted Number of Collection Operations.

1. DEFINITION: The collection operations used in the expression definition are weighted according to the level in which they are defined, so the measure is defined thus:
   WNCO = Σ weight of the level * number of collection operations of the level.

2. INTENT: The value of WCO will provide an indicator of how the operation collections are specified using a sort of composition. A high number

of WNCO will indicate an intertwining specification of operation collections and this could reduce the understandability of an OCL expression.

Example 4.3.15 *In the expression of example 4.3.14 WNCO = 4, and the value is obtained in the following way:* **1** * 2 + **2** * 1 = 4. *The number shown in bold font represents the weight whereas the number shown in normal font indicates the number of operation collections. Three operation collections are used in the specification of the expression, two of them are used in the same subexpression at different levels.*

- NEI, NII: Number of Explicit/Implicit Iterator variables.

  1. DEFINITION: These measures count the total number of iterator variables used in explicit or implicit form respectively. Each iterator variable have a specific type (a classifier in the UML models to which the expression is attached) that is coupled to the contextual type of the expression. The way to determine the values of the NEI and NII measures is similar to NSE and NSI measures. NEI is the number of times an iterator variable appears in an expression (except in the way it is declared), whilst the way to compute the value of the NII measure involves the evaluation of each property with an implicit object in order to determine the object to which the property applies. If the property belongs to an object represented by an iterator variable, the measure is incremented by one. In [OMG03b] there is a clear example of the resolution of ambiguities for an implicit object.
  2. INTENT: As already mentioned, the use of implicit objects and the determination of which object a property is applied to, leads to the interference of the comprehensibility of the expression and could also reduce its comprehension, as it is not always easy to know immediately which is the target object.

  Example 4.3.16 *Given the class diagram of Figure 4.6 and the following expression:*

```
context Person inv:
        self.employer->forAll (iter1 |
        iter1.employee->exists (lastname = name ))
```

  The value of NEI is 1 due to the fact that iter1 is an explicit iterator variable for the forAll operation (iter1 is an explicit variable whose type is Company), and it is used in an explicit form when iter1.employee is used. The value of NII is 2 due to:

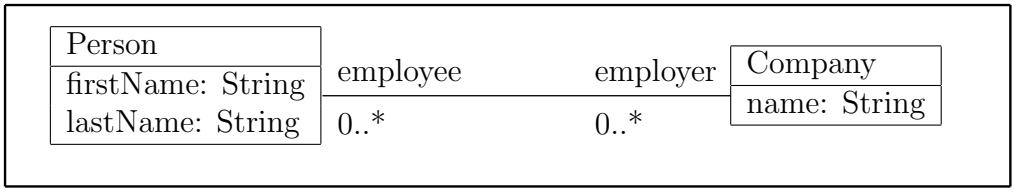


Figure 4.6: A Class Diagram used for Exemplifying NII Measure

- *exists operation does not have an explicit iterator variable (an iterator variable whose type is Person), and lastname refers to this implicit variable.*
- *The name attribute is a property of self and iter1, and this constitutes an ambiguity. To determine to which object name is applied, the most inner scope is used [OMG03b]. The result is: name attribute refers to iter1.*

4.3.3 Measures for Size

In this section we present the measures for OCL expressions related to size structural properties (see Table 4.4).

- NKW: Number of OCL KeyWords.

  1. DEFINITION. This measure counts the total number of OCL keywords used in an expression. The OCL keywords are: *and*, *attr*, *body*, *context*, *def*, *else*, *endif*, *endpackage*, *if*, *implies*, *in*, *inv*, *let*, *not*, *oper*, *or*, *package*, *post*, *pre*, *then*, and *xor*.
  2. INTENT: The number of keywords is an indicator of the complexity of an OCL expression, in terms of its size. A higher number of keywords used in an OCL expression the greater its complexity.

     Example 4.3.17 *In the expression of example 4.3.3 the value of NKW is 6, the keywords used are: context, inv, if, then, else, and endif.*

- NES: Number of Explicit Self.

  1. DEFINITION. This measure counts the number of times *self* is used in an explicit form in an OCL expression.
  2. INTENT: Self, as explained in section 3.1, provides a point of reference for the interpretation of an OCL expression, By using it in an explicit

or implicit form it is possible to access different properties (attributes, operations, and associations-end). The greater the number of times *self* is used may indicate the greater the difficulty of the context to be understood.

Example 4.3.18 *The expression of example 3.1.3 contains a navigation of a reflexive association written as self.work_with.exists(self); in this example the first self could be written in implicit form but the last self must be explicitly declared because self is sent as a parameter. The value of NSE is 2, as self was used in explicit form twice.*

- NIS: Number of Implicit Self.

  1. DEFINITION: This measure counts the number of times *Self* is used in an implicit form in an expression.
  2. INTENT: The goal of NES is also valid for the NIS measure; however, as *self* (and iterator variables in operation collections) can be left implicit, the number of times *self* is left implicit introduces a difficulty for modelers, because they have to evaluate to which object a property is applied.

  Example 4.3.19 *In the following expression the value of NIS is 1, self is implicit when the property work_with is referred.*

```
context Person inv:
        not work_with.exists(self)
```

- NBO: Number of Boolean Operators.

  1. DEFINITION: This measure counts the total number of boolean operators used in an expression. Two occurrence of the same boolean operators are counted separately.
  2. INTENT: We believe that the number of boolean operators is an indicator of the complexity of an OCL expression, in the same way as the number of keywords used in it. In Warmer and Kleppe [WK99], [WK03] it is also recommended to split a constraint with many boolean AND operators, as a correct style for writing less complex expressions. Those expressions with a high number of boolean operators can be candidates to be evaluated in order to be rewritten.

  Example 4.3.20 *NBO = 2 in the following expression:*

```
context ProgramPartner inv:
        partners.deliveredServices->forAll(pointsEarned = 0) and
        membership.card->
```

```
forAll(goodThru = Date::fromYMD(2000,1,1)) and
customer->forAll(age() >55)
```

NBO = 2 because two AND operators are used in the expression. This expression taken from Warmer and Kleppe [WK03], is a an example of an invariant that can be rewritten splitting it into three different invariants. Each of the new invariants will be composed of an operand of the AND logical operator.

- NCO: Number of Comparison Operators.

  1. DEFINITION: This measure counts the number of times an operator like: <, <=, >,>=, = y <> is used in an expression. If an operator is used many times the measures take into account each occurrence of it.
  2. INTENT: It is common to use a comparison operator as a way of expressing a constraint. The goal is similar to that of the previous defined measure.

     Example 4.3.21 *The value of NCO measure is 4 in the following expression:*

```
context Person inv:
        age() > 30 and
        Person.allinstances()->
        forAll(p1, p2 | p1 <>p2 implies p1.dni <> p2.dni) and
        work_with->size() <= 5
```

4.4 Contribution to the Dissertation

In this chapter, we define our measurement goal using GQM, which was stated as the evaluation of OCL expression maintainability. After the definition of questions for reaching the pursued goal, we focus on the definition of coupling measures.

Even though we are conscious that coupling is almost the more important dimension of predicting maintainability, other product properties such as size and length also affects maintainability. We decided to define some size measures to control that size not bias our findings during experimentation due to the fact that size was study as a confounding factor for many coupling measures [EBGR01]. In relation to length, the length of navigations is closely related with the depth of coupling.

We carefully consider the following aspects during the measure definition for assessing import-coupling:

- We analysed the different types of connections that can be specified in OCL expressions in terms of UML model artifacts, defining coupling connections between the contextual instance and its coupled objects.
- Regarding the locus of impact, we put strong emphasis on import-coupling.
- Granularity aspects were evaluated. We take into account what we measure, and in which form we do it, i.e. we distinguish the connection domain and how we count connections.

Having this in mind, we obtained fifteen coupling measures, five size measures and one length measure (see Table 4.4), defining in natural language for each of them a shorthand, a definition, its pursued intent and an example.

Chapter 5

Formal Definition of the Measures

This chapter describes the activities related to the formal definition of measures according to Figure 2.4 (a). Section 5.1 begins explaining the selected formal language for the formal definition (activity D_3 of the method), then section 5.2 describes in detail the selected metamodel (activity D_1) and section 5.3 presents the formal definition of the measures (activity D_4). Finally, the contribution to the dissertation is included in section 5.4.

Although the selected metamodel (activity D_1) is a shared activity with the measure definition in Natural language it is explained here due to the fact that the specification of the measures uses its OCL metaclasses.

Whenever an activity of the method is applied, the corresponding section is titled accordingly and a reference to its number is included between parenthesis.

5.1 Select a Formal Language for the Formal Definition (D_3)

An important contribution to solve the problems introduced of the formality degree in the measure definition (mentioned in chapter 2) is to use the Object Constraint Language upon a design metamodel. We have proposed a set of measures for OCL expressions, trying to find indicators for the comprehensibility and modifiability of OCL expressions. When we decided to formally define them we considered that the use of OCL could have two advantages (activity D_3 of Figure 2.4):

- The first is that OCL itself is precisely defined through metamodelling facilities, as an instance of the meta-metamodel of the OMG Meta Object Facility (MOF), and the measure definition can be suitably placed at the same level (the M2 level) as the OCL definition.

- The second is that a same language, OCL, is used as a formal language to define the UML and OCL semantics (at M2 Level) and is used by modelers for defining constraints on their models (at M1 Level). In fact the OCL was claimed as a language easy to use and easy to learn, and to be easily grasped by anybody familiar with OO modeling [WK03]. So, the familiarity of this language can make the definition of our measures more modeler-friendly.

Thus, the approach of defining measures for OCL expressions using OCL metamodel and OCL language as the formal language allows an unambiguous definition. OCL was previously used by the QUASAR (QUantitative Approaches on Software Engineering And Reengineering) Research Group [BBeA02], [BeA03a], [BeA02], [BeA03b] to define measures. However the metamodel upon which OCL was used was the UML metamodel. In our case, OCL is used as a language for defining measures for OCL expressions upon the OCL metamodel. This chapter describe the formal definition of the measures.

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*). We traverse the dynamic hierarchical structure (the *ast*) and meanwhile we visit every element in the tree, we evaluate if each element of the tree is meaningful for the measure we want to compute. If it is, the measure is incremented otherwise it remains as it is.

Before giving an example of how the dynamic hierarchical structure is built according to an OCL expression, we must describe each of the constituent parts of the OCL metamodel for understanding how the *ast* objects are instantiated from its corresponding metaclasses. So, section 5.2 describes the OCL metamodel and section 5.2.2 gives an example of an OCL expression and its corresponding *ast* tree. Finally, section 5.3 formally defines the proposed measures.

5.2 OCL Metamodel (D_1)

The UML metamodel is defined as one of the layers of a four-layer metamodelling architecture. In this architecture the UML metamodel is an instance of the meta-metamodel of the OMG Meta Object Facility (MOF). The OCL metamodel is placed at the same level (the M2 level) as the UML metamodel and use the MOF as the definition language (provided by the M3 level) [Ric02]. So, the concepts of OCL 2.0 and their relationships have been defined in the form of a MOF-compliant metamodel [HZ04] [1]. The benefit of a metamodel for OCL is that it precisely defines the structures and syntax of all OCL concepts like types, expressions, and values in

[1] Previous versions of OCL had no metamodel representation

an abstract way and by means of UML features. Thus, all legal OCL expressions can be systematically derived and instantiated from the metamodel [Ric02].

This section has two subsections: the OCL metamodel is described in section 5.2.1 whereas section 5.2.2 gives samples of OCL expressions and their corresponding *ast* trees.

5.2.1 OCL Metamodel Metaclasses

In this section we will present the main metaclasses of the Expression Package of the OCL metamodel which are essential for the formal definition of the proposed measures. We present several class diagrams defining the abstract syntax of OCL concepts, giving an explanation of the class in the diagram according to [OMG03b]. Some concrete examples are used to illustrate the application of the metamodel, in the same way Richters does in [Ric02] when he describes the first definition of the OCL metamodel.

5.2.1.1 Expressions Core

Figure 5.1 shows the core part of the Expressions package. The basic structure in the package consists of the classes OclExpression, PropertyCallExp and VariableExp. An OclExpression always has a type, which is usually not explicitly modeled, but derived. Each PropertyCallExp has exactly one source, identified by an OclExpression. A ModelPropertyCallExp generalizes all property calls that refer to Features or AssociationEnds in the UML metamodel. In Figure 5.2 the various subtypes of ModelPropertyCallExp are defined.

Most of the remainder of the expressions package consists of a specification of the different subclasses of PropertyCallExp and their specific structure. From the metamodel it can be deduced that an OCL expression always starts with a variable or literal, on which a property is recursively applied.

- **OclExpression** The abstract superclass OCLExpression defines the set of all legal expression in OCL [Ric02]. It is the top-level element of the OCL Expressions package. Every OclExpression has a type that can be statically determined by analysing the expression and its context. Evaluation of an expression results in a value [2].
  The environment of an OclExpression defines what model elements are visible and can be referred to in an expression. At the topmost level the environment will be defined by the ModelElement to which the OCL expression is attached,

[2]Expressions with boolean result can be used as constraints, e.g. to specify invariants. Expressions of any type can be used to specify queries, initial attribute values, target sets, etc.

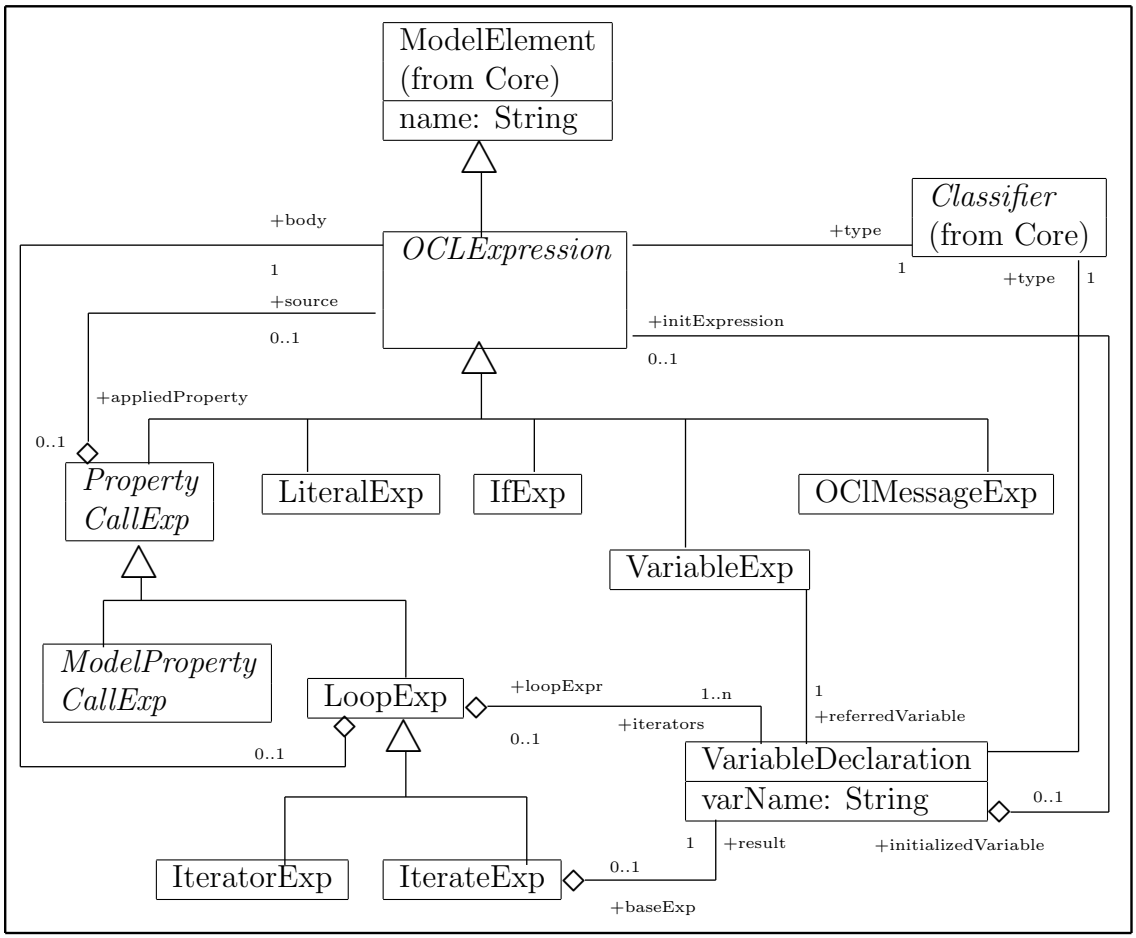


Figure 5.1: The Basic Structure of the Abstract Syntax Kernel Metamodel for Expressions

for example by a Classifier if the OCL expression is used as an invariant. On a lower level, each iterator expression can also introduce one or more iterator variables into the environment. The environment is not modeled as a separate metaclass, because it can be completely derived using derivation rules. The complete derivation rules can be found in chapter 4 of [OMG03b].

Associations	
appliedProperty	The property that is applied to the instance that results from evaluating this OclExpression.
type	The type of the value that is the result of evaluating the OclExpression.
parentOperation	The OperationCallExp where this OclExpression is an argument of.
initializedVariable	The variable of which the result of this expression is the initial value.

- **PropertyCallExp** A PropertyCallExp is an expression that refers to a property (operation, attribute, association end, predefined iterator for collections). Its result value is the evaluation of the corresponding property. This is an abstract metaclass.

- **ModelPropertyCallExp** A ModelPropertyCall expression is an expression that refers to a property that is defined for a Classifier in the UML model to which this expression is attached. Its result value is the evaluation of the corresponding property. Subclasses of ModelPropertyCallExp are defined in 5.2.1.2.

- **LoopExp** A LoopExp is an expression that represent a loop construct over a collection. It has an *iterator variable* that represents the elements of the collection during iteration. The *body* expression is evaluated for each element in the collection. The result of a loop expression depends on the specific kind and its name.

Associations	
iterators	The VariableDeclarations that represents the iterator variables. These variables are, each in its turn, bound to every element value of the *source* collection while evaluating the *body* expression.
body	The OclExpression that is evaluated for each element in the *source* collection.

- **IterateExp** An IterateExp is an expression which evaluates its *body* expression for each element of a collection. It acts as a loop construct that iterates over the elements of its *source* collection and results in a value. An iterate expression evaluates its *body* expression for each element of its source collection. The evaluated value of the body expression in each iteration-step becomes the new value for the *result* variable for the succeeding iteration-step. The result can be of any type and is defined by the *result* association. The IterateExp is the most fundamental collection expression defined in the OCL Expressions package.

Associations

result	The VariableDeclaration that represents the result variable.

Example 5.2.1 *The following example illustrates the general structure of* **IterateExp** *expressions. The* source *attribute is inherited from* **PropertyCallExp**, *whereas the* iterators *and body attribute are inherited from* **LoopExp**.

$$\underbrace{\underbrace{Sequence\{1..5\}}_{source} -> iterate(\underbrace{i : Integer}_{iterators}; \underbrace{acc : Integer = 1}_{result} | \underbrace{acc * i}_{body})}_{iterateExp}$$

- **IteratorExp** The type of the iterator expression depends on the name of the expression, and sometimes on the type of the associated *source* expression. The IteratorExp represents all other predefined collection operations that use an iterator. This includes *select*, *collect*, *reject*, *forAll*, *exists*, etc. The OCL Standard Library defines a number of predefined iterator expressions.

  Example 5.2.2 *The following example illustrates the general structure of IteratorExp expressions. The* source *attribute is inherited from* **PropertyCallExp**, *whereas the* iterators *and body attribute is inherited from* **LoopExp**.

$$\underbrace{\underbrace{self.employees}_{source} -> forAll(\underbrace{p : Person}_{iterators} | \underbrace{p.age > 45}_{body})}_{iteratorExp}$$

- **VariableDeclaration** A VariableDeclaration declares a variable name and binds it to a type. The variable can be used in expressions where the variable is in scope.

  Associations

initExpression	The OclExpression that represents the initial value of the variable. Depending on the role that a variable declaration plays, the init expression might be mandatory.
type	The Classifier which represents the type of the variable.

  Attributes

varName	The String that is the name of the variable.

  Example 5.2.3 *The following example illustrates the general structure of a* **VariableDeclaration** *used in the previous* **IterateExp** *expressions example.*

$$\underbrace{\underbrace{acc}_{varName} : \underbrace{Integer}_{type} = \underbrace{1}_{initExpression}}_{variableDeclaration}$$

- **LiteralExp** A LiteralExp is an expression with no arguments producing a value. In general the result value is identical with the expression symbol. This includes things like the integer 1 or literal strings like *this is a LiteralExp*.

- **IfExp** An IfExp represent an If Expression, it is defined in section 5.2.1.3, but included in this diagram for completeness.

- **VariableExp** A VariableExp is an expression which consists of a reference to a variable. References to the variables *self* and *result* or to variables defined by *Let expressions* are examples of such variable expressions.

  Associations

referredVariable	The VariableDeclaration to which this variable expression refers. In the case of a *self* expression the variable declaration is the definition of the *self* variable.

- **OclMessageExp** OclMessageExp is defined in section 5.2.1.4, but included in this diagram for completeness.

  Associations

source	The result value of the *source* expression is the instance that performs the property call.

5.2.1.2 ModelPropertyCall Expressions

A ModelPropertyCallExp is a specialization of PropertyCallExp. All ModelPropertyCallExp have at least one argument: an OCLExpression, determining the source object. A ModelPropertyCallExp can refer to any of the subtypes of Feature as defined in the UML kernel. This is shown in Figure 5.2 by the three different subtypes, each of which is associated with its own type of ModelElement.

- **AttributeCallExp** An AttributeCallExpression is a reference to an Attribute of a Classifier defined in a UML model. It evaluates to the value of the attribute.

  Associations

referredAttribute	The Attribute to which this AttributeCallExp is a reference.

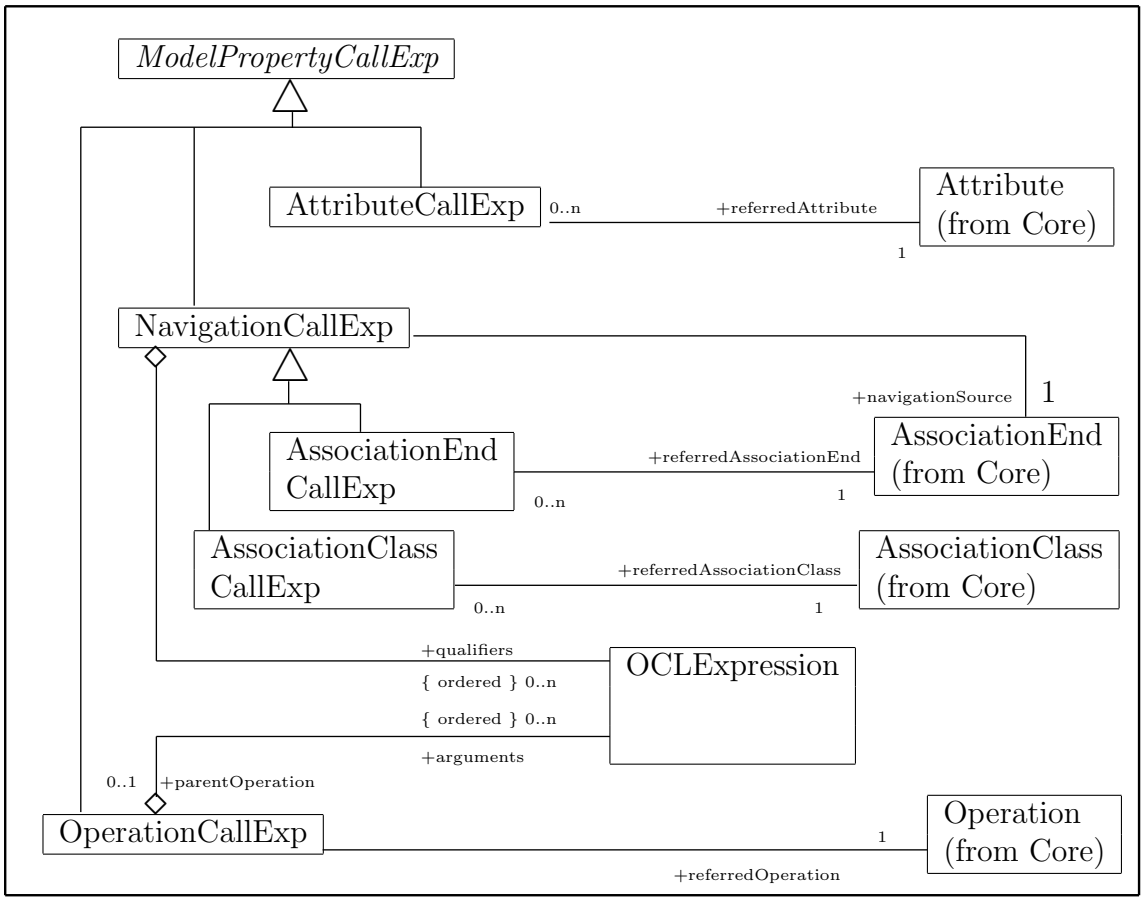


Figure 5.2: Abstract Syntax Metamodel for ModelPropertyCallExp

Example 5.2.4 *The following example illustrates the general structure of an* **AttributeCallExp** *used in an example 3.1.1 valid to the UML class diagram of Figure 3.1.*

$$\underbrace{\underbrace{self}_{source}.\underbrace{age}_{referredAttribute}}_{AttributeCallExp} > 50$$

- **NavigationCallExp** A NavigationCallExp is a reference to an AssociationEnd or an AssociationClass defined in a UML model. It is used to determine objects linked to a target object by an association. If there is a qualifier attached to the source end of the association then additional qualifiers expressions may be used to specify the values of the qualifying attributes.

Associations	
qualifiers	The values for the qualifier attributes if applicable.
navigationSource	The source denotes the AssociationEnd at the end of the object itself. This is used to resolve ambiguities when the same Classifier participates in more than one AssociationEnd in the same association. In other cases it can be derived.

- **AssociationEndCallExp** An AssociationEndCallExp is a reference to an AssociationEnd defined in a UML model. It is used to determine objects linked to a target object by an association. The expression refers to these target objects by the role name of the association end connected to the target class.

Associations	
referredAssociationEnd	The AssociationEnd to which this AssociationEndCallExp is a reference. This refers to an AssociationEnd of an Association that is defined in the UML model.

Example 5.2.5 *The following example illustrates the general structure of an* **AttributeCallExp** *used in an example 4.3.16 valid to the UML class diagram of Figure 4.6.*

$$\underbrace{\underbrace{self}_{source}\,.\;\underbrace{employer}_{referredAssociationEnd}}_{AssociationEndCallExp} \;->\; forAll\cdots$$

- **AssociationClassCallExp** An AssociationClassCallExp is a reference to an AssociationClass defined in a UML model. It is used to determine objects linked to a target object by an association class. The expression refers to these target objects by the name of the target associationclass.

Associations	
referredAssociationClass	The AssociationClass to which this AssociationClassCallExp is a reference. This refers to an AssociationClass that is defined in the UML model.

- **OperationCallExp** A OperationCallExp refers to an operation defined in a Classifier. The expression may contain a list of argument expressions if the operation is defined to have parameters. In this case, the number and types of the

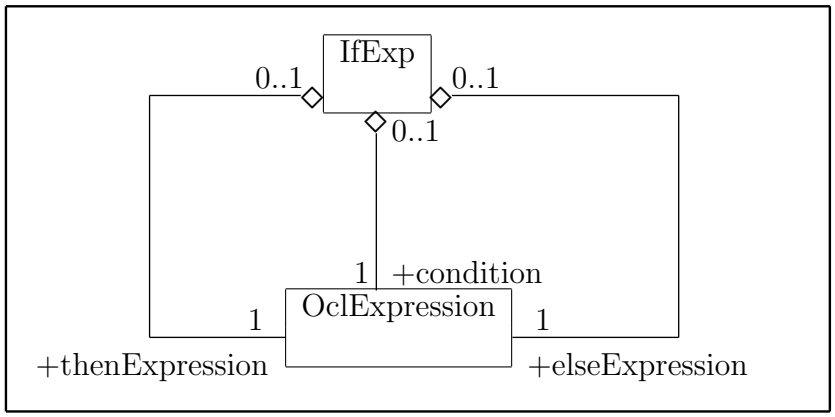


Figure 5.3: The Abstract Syntax of If Expressions

arguments must match the parameters.

Associations	
arguments	The arguments denote the arguments to the operation call. This is only useful when the operation call is related to an Operation that takes parameters.
referredOperation	The Operation to which this OperationCallExp is a reference. This is an Operation of a Classifier that is defined in the UML model.

5.2.1.3 If Expressions

Figure 5.3 shows the structure of the *if* expression.

- **IfExp** An IfExp results in one of two alternative expressions depending on the evaluated value of a condition. Note that both the *thenExpression* and the *elseExpression* are mandatory. The reason behind this is that an if expression should always result in a value, which cannot be guaranteed if the *else* part is left out.

Associations	
condition	The OclExpression that represents the boolean condition. If this condition evaluates to true, the result of the *if* expression is identical to the result of the *thenExpression*. If this condition evaluates to false, the result of the if expression is identical to the result of the *elseExpression*
thenExpression	The OclExpression that represents the *then* part of the *if* expression.
elseExpression	The OclExpression that represents the *else* part of the *if* expression.

Example 5.2.6 *The following example illustrates the general structure of a If expression used in the constraint of 4.3.3 example.*

$$\underbrace{title = if \underbrace{isMale = true}_{condition} then \quad \underbrace{'Mr.'}_{thenExpression} \quad else \quad \underbrace{'Ms.'}_{elseExpression}}_{IfExp}$$

5.2.1.4 Message Expressions

In the specification of communication between instances we unify the notions of asynchronous and synchronous communication. The structure of the message expressions is shown in Figure 5.4.

- **OclMessageExp** An OclMessageExp is an expression that results in an collection of OclMessage value. An OclMessage is the unification of a signal sent, and an operation call. The target of the operation call or signal sent is specified by the target OclExpression. Arguments can be OclExpressions, but may also be unspecified value expressions for arguments whose value is not specified.

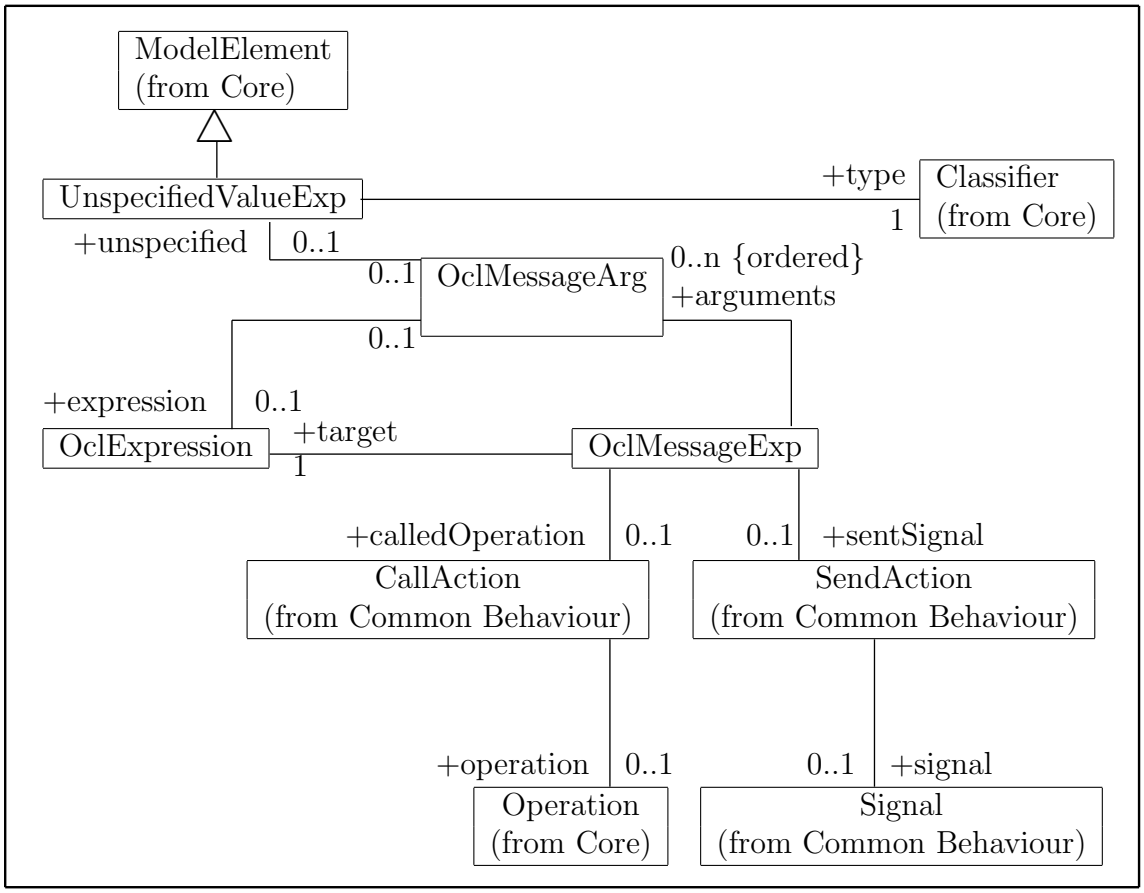


Figure 5.4: The Abstract Syntax of OclMessages

Associations	
target	The OclExpression that represents the target instance to which the signal is sent.
arguments	The SignalArgs that represents the parameters to the Operation or Signal. The number and type of arguments should conform to those defined in the Operation or Signal. The order of the arguments is the same as the order of the parameters of the Operation or the attributes of a Signal.
calledOperation	If this is a message to request an operation call, this is the requested CallAction.
sentSignal	If this is a UML signal sent, this is the SendAction.

- **OclMessageArg** An OclMessageArg is an argument of an OclMessageExp. It is either an OclExpression, or an UnspecifiedValueExp. An OclExpression is used to specify the exact value of the parameter. An UnspecifiedValueExp is

used when one does not want, or is not able to specify the exact value of the parameter at the time of sending the message. An OclMessageArg has either a specified or an unspecified value.

Associations	
expression	The OclExpression that represents an actual parameter to the Operation or Signal. unspecified The UnspecifiedValueExp that represents a random value that conforms to the type of this expression.

- **UnspecifiedValueExp** An UnpecifiedValueExp is an expression whose value is unspecified in an OCL expression. It is used within OCL messages to leave parameters of messages unspecified.

5.2.1.5 Literal Expressions

This section defines the different types of literal expressions of OCL. It also refers to enumeration types and enumeration literals. Figure 5.5 shows all types of literal expressions.

- **BooleanLiteralExp** A BooleanLiteralExp represents the value *true* or *false* of the predefined type Boolean.

Associations	
booleanSymbol	The Boolean that represents the value of the literal.

- **CollectionItem** A CollectionItem represents an individual element of a collection.

- **CollectionKind** A CollectionKind is an enumeration of kinds of collections.

- **CollectionLiteralExp** A CollectionLiteralExp represents a reference to a collection literal.

Attributes	
kind	The kind of collection literal that is specified by this CollectionLiteralExp.

- **CollectionLiteralPart** A CollectionLiteralPart is a member of the collection literal.

Associations	
type	The type of the collection literal.

- **CollectionRange** A CollectionRange represents a range of integers.

- **EnumLiteralExp** An EnumLiteralExp represents a reference to an enumeration literal.

Associations	
referredEnumLiteral	The EnumLiteral to which the enum expression refers.

- **IntegerLiteralExp** A IntegerLiteralExp denotes a value of the predefined type Integer.

Attributes	
integerSymbol	The Integer that represents the value of the literal.

- **NumericLiteralExp** A NumericLiteralExp denotes a value of either the type Integer or the type Real.

- **PrimitiveLiteralExp** A PrimitiveLiteralExp literal denotes a value of a primitive type.

Attributes	
symbol	The String that represents the value of the literal.

- **RealLiteralExp** A RealLiteralExp denotes a value of the predefined type Real.

Attributes	
realSymbol	The Real that represents the value of the literal.

- **StringLiteralExp** A StringLiteralExp denotes a value of the predefined type String.

Attributes	
stringSymbol	The String that represents the value of the literal.

- **TupleLiteralExp** A TupleLiteralExp denotes a tuple value. It contains a name and a value for each part of the tuple type.

5.2.1.6 Let Expressions

The abstract syntax metamodel for Let expressions introduce a new metaclass, the metaclass LetExp, as shown in Figure 5.6. The other metaclasses are re-used from the previous diagrams.

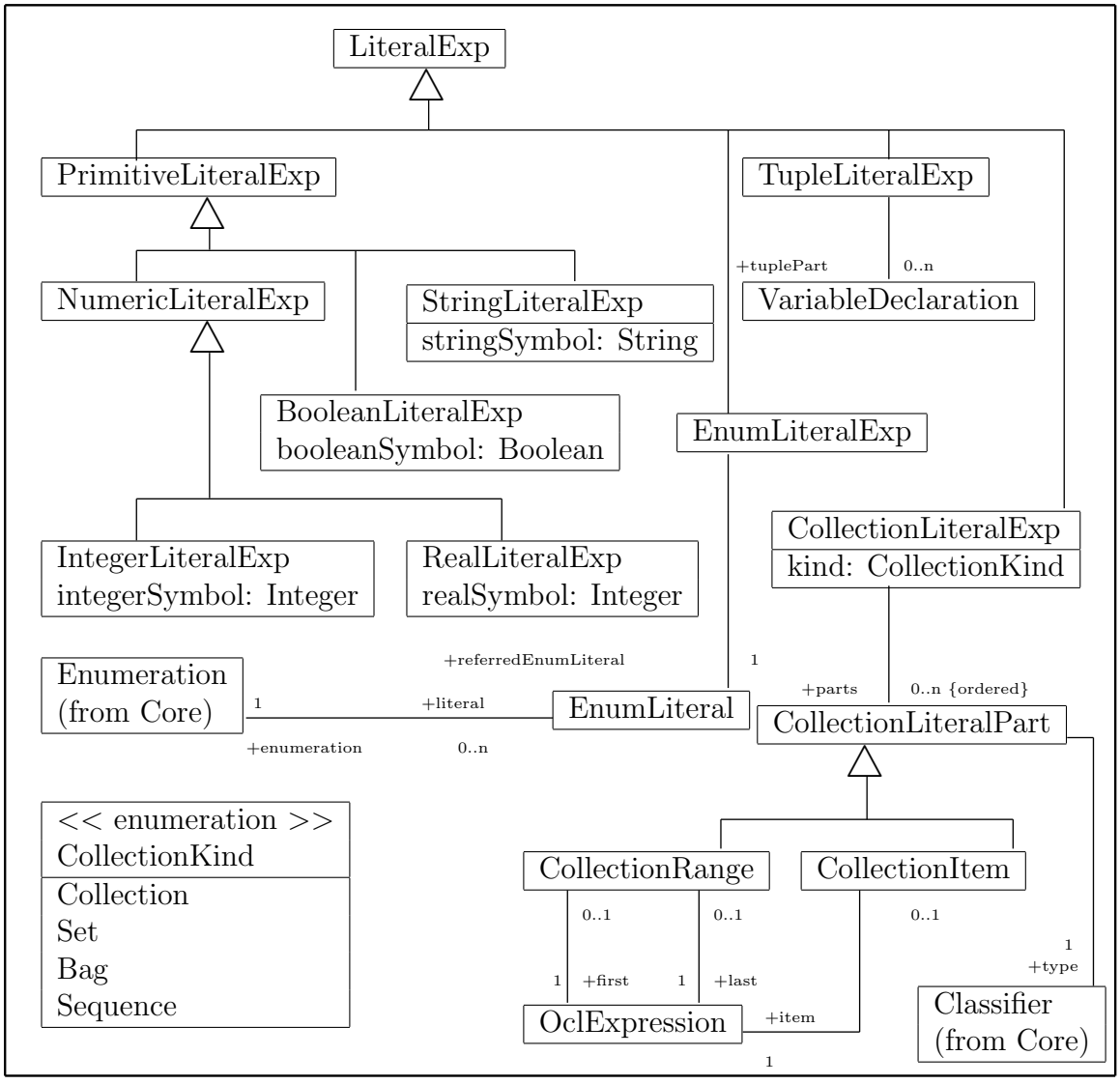


Figure 5.5: The Abstract Syntax Metamodel for Literal Expression

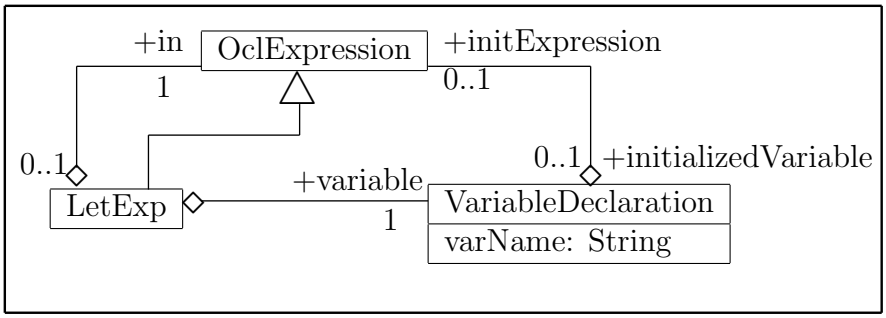


Figure 5.6: Abstract Syntax Metamodel for Let Expression

5.2.1.6.1 LetExp Through a LetExp expression it is possible to define a new variable with an initial value. A variable defined by a LetExp cannot change its value. The value is always the evaluated value of the initial expression. The variable is visible only in the *in expression*.

Associations	
variable	The VariableDeclaration that defined the variable.
in	The OclExpression in whose environment the defined variable is visible.

5.2.2 Samples of Abstract Syntax Tree

Given an OCL expression it is possible to build an abstract syntax tree (*ast*) of it. The abstract syntax of OCL is defined via UML class diagram [Lia04] and was described according to [OMG03b] in the previous section. The purpose of this section is to show two samples of ast built from two different OCL expression.

Example 5.2.7 *Figure 5.7 shows the following invariant as a stereotype associated to the Company class whereas Figure 5.8 shows the standard place where an invariant OCL expression occur in the UML and OCL metamodel.*

```
context Company inv:
    self.numberOfEmployees > 50
```

The object diagram for the abstract syntax of the invariant expression as an instantiation of the metamodel for OCL expressions and types is depicted in Figure 5.9.

The object diagram basically shows an abstract syntax tree (ast). In order to build the tree shown in Figure 5.9, instances of the following OCL metaclasses have been

used: OperationCallExp, AttributeCallExp, Operation, IntegerLiteralExp, VariableExp and VariableDeclaration. Also, in the tree shown in Figure 5.9 there are instances of the Class, Constraint and Attribute UML´s metaclasses.

The root of the tree of Figure 5.9 is the OperationCallExp expression, which has three child branches:

- *First, the source of the operation call expression is an AttributeCallExpression.*
- *The second branch models the referred operation, and*
- *The third branch represents the argument, an Integer Literal expression.*

In order to compute the value of a specific measure, for example the measure Number of Explicit Self (NES), we must visit each of the tree nodes (instances of OCL metaclasses), and verify if each of them belongs to the particular metaclass we are interested in measuring. In the case of computing the value of NES, we need to traverse the tree looking for instances of the VariableDeclaration metaclass having an attribute called name whose value is 'self '. In this example there is only one instance of Variable Declaration with this characteristic. See the first leaf of the ast tree.

Example 5.2.8 *This example corresponds to the invariant OCL expression attached to the Flight class of Figure 5.10. The meaning of the invariant expression is that a flight does not contain more passengers than the number of seats of the airplane' type associated with the airplane of the flight. The basic instantiation of this fragment of the model for our example is consistent with the standard place where an invariant OCL expression occurs in the UML and OCL metamodel. An OCL expression always constitutes the body of a Constraint object associated with one or more ModelElement objects. So, the instantiation includes two important objects:*

- *a class object where its name is Flight, and*
- *a constraint object to represent an invariant constraint.*

The body of the constraint will be represented by the object diagram for the ast of the invariant expression. The object diagram of Figure 5.11 basically shows an ast in the right part.

The tree is built with instances of the following OCL metaclassses: OperationCallExp, AttributeCallExp, AssociationEndCallExp, Operation, IntegerLiteralExp, VariableExp and VariableDeclaration OCL metaclasses. Within the tree there is also an instance of the Attribute UML metaclass which constitutes the attribute referred by the AttributeCallExp; and three instances of the AssociationEnd UML metaclass.

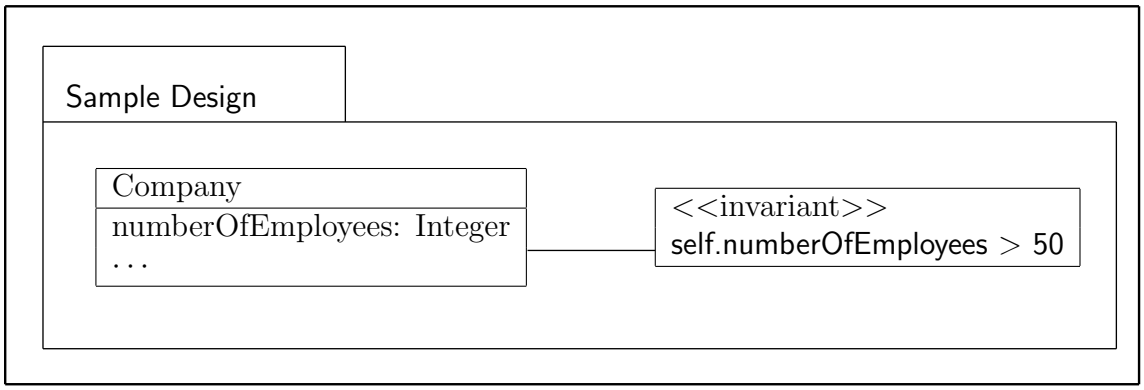


Figure 5.7: Metamodel Objects for a Sample Design

The root of the tree of Figure 5.11 is the OperationCallExp expression, which has three branches:

- *First, the source of the Operation Call expression is the subtree modeling the subexpression self.passenger->size().*
- *the second branch models the referred operation, and*
- *the third branch represents the argument, the subtree modeling the subexpression self.plane.planetype.numberofseats.*

In order to compute the value of a specific measure we must visit each of the tree nodes (instances of OCL metaclasses) and verify if each of them belongs to the par-ticular metaclass we are interested in measuring. The implemented strategy for visiting the elements is shown in the following section.

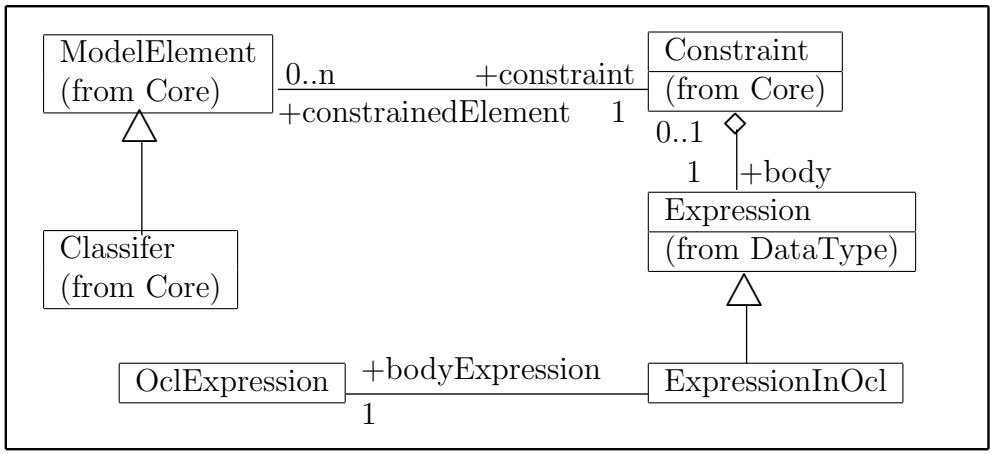


Figure 5.8: Situation of OCL Expression used as Definition or Invariant

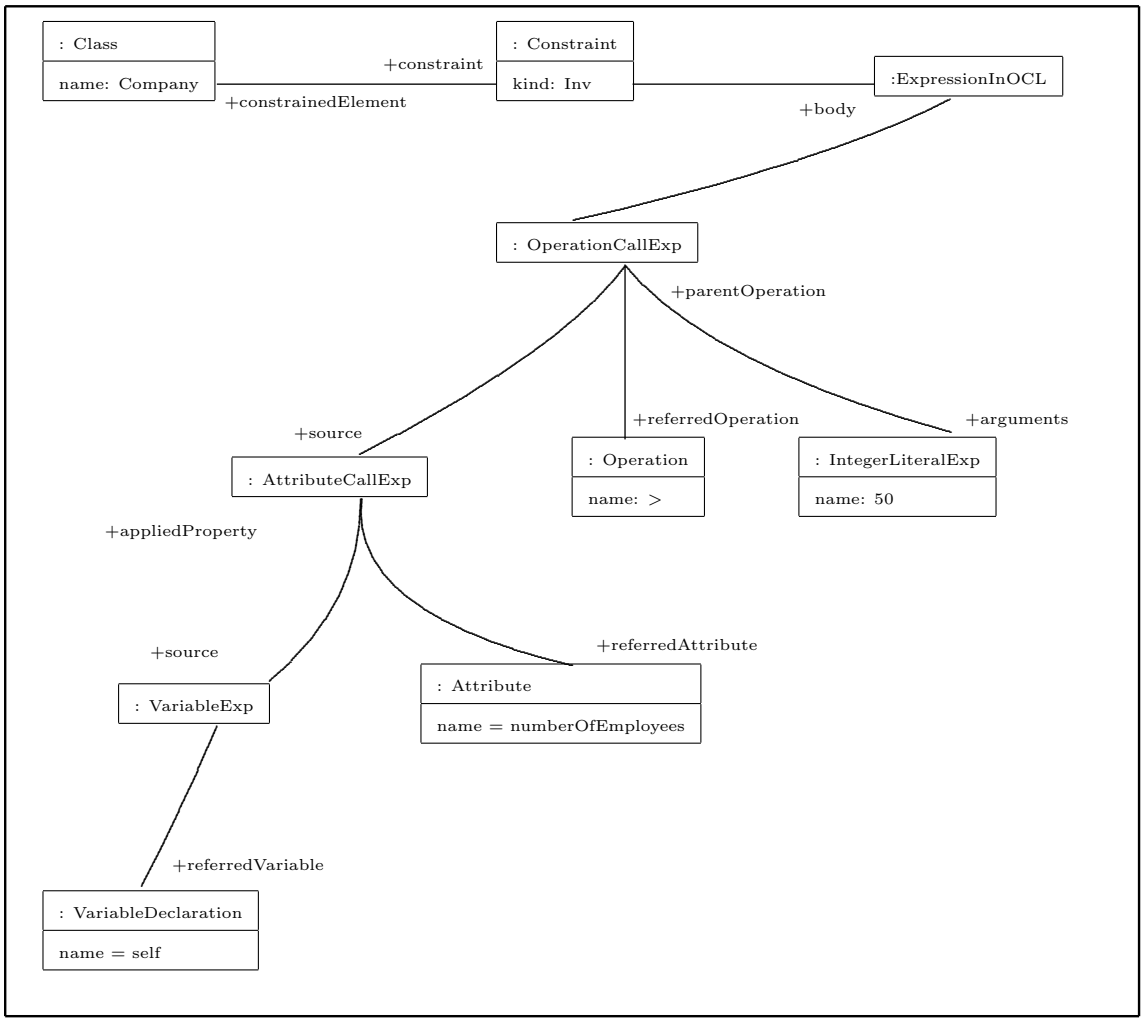


Figure 5.9: *AST* Built for an OCL Invariant

5.3 Formal Definition of the Measures (D_4)

This section formally defines the proposed measures. However, as the definition is defined upon the OCL metamodel the implemented strategy to visit each of the element in the *ast* tree -without clutter the OCL metaclasses- was to use a Visitor Pattern. The strategy employed is described in subsection 5.3.1. The Visitor class which allows us to obtain the value of each measure is specified in section 5.3.3.

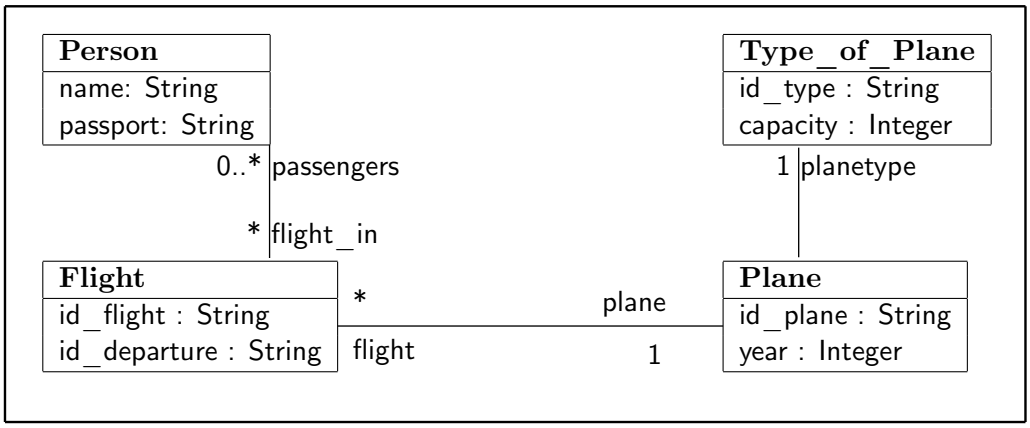


Figure 5.10: Portion of Class Diagram about Flights

5.3.1 Implemented Strategy

There are many different operations we must carry out in order to compute the measures values, and these operations should be defined in many OCL metaclasses, but *we do not want to clutter* the OCL metaclasses with these operations. Moreover, it is possible that during the process of measure definition, many of these operations changes as the process time passes.

A useful implementation which help us to solve this problem is to use a Visitor Pattern [GHJV95]. The operations we must define are located into a separate object (a visitor). The visitor is sent to the tree root, eventually each element forwards the requests to its children and also its calls activates the visitor. The visitor performs operations on the element. An explanation of this pattern can be found in [GHJV95], [RM00].

```
context Constraint::Value_of_NES() :Integer
post:       self.body^accept(self.visitor, MetricAcronym::NES)
            and result = self.visitor.valueMetric
```

Figure 5.12 shows the basic UML design for implementing the strategy. Subsection 5.3.2 shows how the Accept operations are implemented in the OCL´s metaclasses of the Expression Package. Subsection 5.3.3 describes the way measures values are obtained and shows the Visitor Class and its operations.

All the expression used in this section were syntactically verified using ECLIPSE [Ecl00] and the OCTUPUS component [Obj00], a plug-in of ECLIPSE.

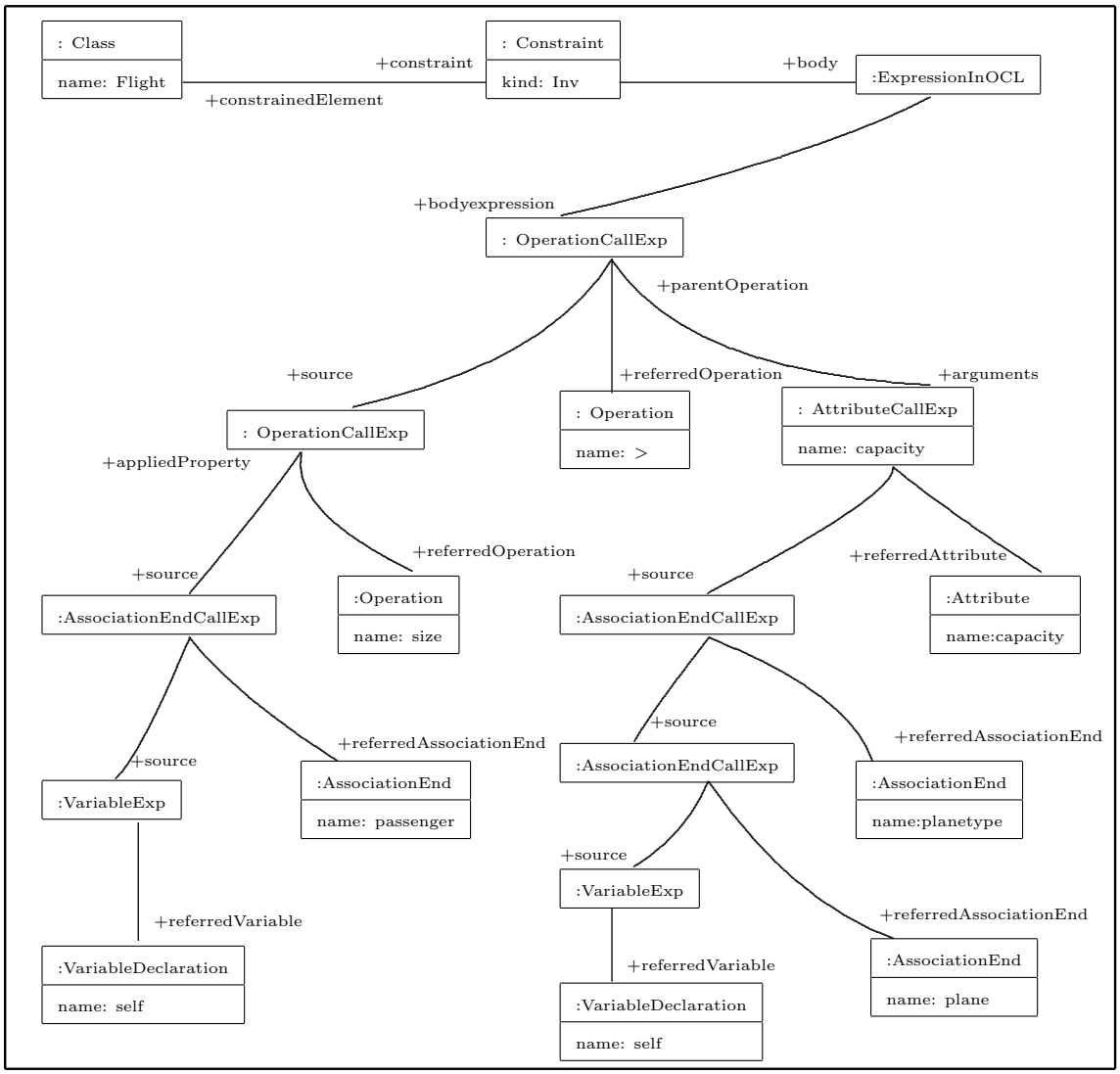


Figure 5.11: 2nd *AST* Built for an OCL Invariant

5.3.2 Implementing the Accept Operations

In this subsection we will show how the Accept´s operations are implemented in the OCL´s metaclasses (described in section 5.2) of the Expression Package. Each OCLExpression subclass define Accept operations in basically the same way: It calls the Visitor operation that corresponds to the class that received the Accept request. Eventually a specific Accept operation also traverses all the parts of its coupled elements. So, in order to understand the Accept operations a clear comprehension of the OCL metaclasses and their relationships is required.

OperationCallExp subclass defines Accept calling the Visitor operation that corresponds to the class, and it implements Accept by iterating over its arguments

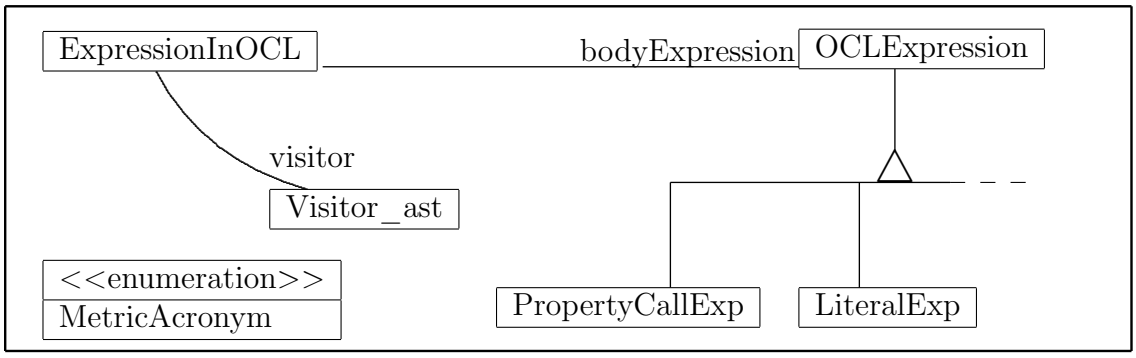


Figure 5.12: Implemented Strategy for Obtaining Measures Values

(whenever the arguments contain an element) and calling Accept on each of them. It also calls Accept operation on its source.

```
context OperationCallExp::accept_new(v: Visitor, metricName: MetricAcronym)
post:       v^visitOperationCallExp(self, metricName) and
            ( arguments->size() >= 1 implies
            arguments->forAll(a | a^accept_new(v, metricName) ))
            and
            (self.source->notEmpty()
            implies self.source^accept_new(v, metricName ))
```

NavigationCallExp subclass defines Accept calling the Visitor operation that corresponds to the class, and it implements Accept by iterating over its qualifiers (whenever the qualifiers contain an element) and calling Accept on each of them. It also calls the Accept operation on its source.

```
context NavigationCallExp::accept_new(v: Visitor_ast, metricName: MetricAcronym)
post:       v^visitNavigationCallExp(self, metricName) and
            (self.qualifiers->size() >= 1 implies
            self.qualifiers->forAll(q | q^accept_new(v, metricName))
            )
            and
            (self.source->notEmpty() implies self.source^accept_new(v, metricName))
```

AttributeCallExp subclass defines Accept calling the Visitor operation that corresponds to the class, and it calls the Visitor operation on its source whether the source is not empty.

```
context AttributeCallExp::accept_new(v: Visitor, metricName: MetricAcronym)
post:       v^visitAttributeCallExp(self, metricName) and
            (self.source->notEmpty() implies
            self.source^accept_new(v, metricName)
            )
```

A LetExp object calls the Visitor operation that corresponds to the class, and it calls Accept operations on its In and variable.initExpression variables.

```
context LetExp::accept_new(v: Visitor, metricName: MetricAcronym)
post:       v^visitLetExp(self, metricName) and
            self.In^accept_new(v, metricName) and
            self.variable^accept_new(v, metricName) and
            (self.variable.initExpression->notEmpty() implies
            self.variable.initExpression^accept_new(v, metricName)
            )
```

A IfExp object calls the Visitor operation that corresponds to the class, and calls Accept operations on its condition, thenExpression and elseExpression variables.

```
context IfExp::accept_new(v: Visitor_ast, metricName: MetricAcronym)
post:       v^visitIfExp(self, metricName) and
            self.condition^accept_new(v, metricName) and
            self.thenExpression^accept_new(v, metricName) and
            self.elseExpression^accept_new(v, metricName)
```

LoopExp subclass defines Accept calling the Visitor operation that corresponds to the class, and it implements Accept by iterating over its iterators (whenever the iterators contain an element) and calling Accept on each of them. It also calls the Accept operation on its Body.

```
context LoopExp::accept_new(v: Visitor_ast, metricName: MetricAcronym)
post:       v^visitLoopExp(self, metricName) and
            self.Body^accept_new(v, metricName) and
            (self.iterators->size() >= 1 implies
            iterators->forAll(a | a.initExpression->notEmpty()
                        implies
                        a.initExpression^accept_new(v, metricName)
                        ))
```

OClMessageExp subclass defines Accept calling the Visitor operation that corresponds to the class, and it implements Accept by iterating over its arguments (whenever the arguments contain an element) and calling Accept on each of them. It also calls the Accept operation on its target.

```
context OclMessageExp::accept_new(v: Visitor, metricName: MetricAcronym)
post:       v^visitOclMessageExp(self, metricName) and
            target^accept_new(v, metricName) and
```

```
(arguments->notEmpty() implies
arguments->forAll(a | a.expression->notEmpty()
        implies a.expression^accept_new(v, metricName)
        ))
```

A CollectionRange subclass calls the Visitor operation that corresponds to the class and calls Accept operations on its first and last variables.

```
context CollectionRange::accept_new(v: Visitor, metricName: MetricAcronym)
post:   v^visitCollectionRange(self, metricName) and
        first^accept_new(v, metricName) and
        last^accept_new(v, metricName)
```

A CollectionItem subclass calls the Visitor operation that corresponds to the class and calls Accept operations on its item.

```
context CollectionItem::accept_new(v: Visitor, metricName: MetricAcronym)
post:   v^visitCollectionItem(self, metricName) and
        item^accept_new(v, metricName)
```

TupleLiteralExp subclass defines Accept calling the Visitor operation that corresponds to the class, and it implements Accept by iterating over its tuplePart (whenever the tuplePart contain an element) and calling Accept on each of them.

```
context TupleLiteralExp::accept_new(v: Visitor, metricName: MetricAcronym)
post:   v^visitTupleLiteralExp(self, metricName) and
        (tuplePart ->notEmpty() implies
        tuplePart->forAll(p | p.initExpression->notEmpty() implies
                p.initExpression^accept_new(v, metricName)
                ))
```

A LiteralExp object calls the Visitor operation that corresponds to its class.

```
context LiteralExp::accept_new(v: Visitor, metricName: MetricAcronym)
post:   v^visitLiteralExp(self, metricName)
```

Table 5.1: A Visitor Class

ClassName: Visitor
<<attributes>>: - valueMetric: Integer; - importedproperties: Set(String); - navigatedClasses: Set(String); – navigatedRelationships: Set(String); - aSetofValues : Set(Integer); - maxDepthatLevel: Sequence(Integer);
<<operations>>: + visitOperationCallExp(o: OperationCallExp, level: Integer, metricName: MetricAcronym) : Integer + visitNavigationCallExp(o: NavigationCallExp, level: Integer, metricName: MetricAcronym) : Integer + visitAttributeCallExp(o: AttributeCallExp,level: Integer, metricName: MetricAcronym) : Integer + visitLetExp(o: LetExp, level: Integer, metricName: MetricAcronym) : Integer + visitIfExp(o: IfExp, level: Integer, metricName: MetricAcronym) : Integer + visitLoopExp(o: LoopExp, level: Integer, metricName: MetricAcronym) : Integer + visitOclMessageExp (o: OclMessageExp, level: Integer, metricName: MetricAcronym) : Integer + visitCollectionRange(o: CollectionRange,level: Integer, metricName: MetricAcronym) : Integer + visit CollectionItem(o: CollectionItem, level: Integer, metricName: MetricAcronym) : Integer + visitTupleLiteralPart (o: TupleLiteralPart,level: Integer, metricName: MetricAcronym) : Integer + visitLiteralExp(o: LiteralExp, level: Integer, metricName: MetricAcronym) : Integer

5.3.3 A Visitor Class for Obtaining the Value of OCL measures

In this subsection a Visitor class is defined. Its methods are shown in Table 5.1, they allow us to compute the value of the OCL expression measures.

In the following we show how the measure values are obtained along with how the visitor class defines a corresponding operation for each OCL metaclass in order to obtain the values.

- **NES:** References to the *self* variables are examples of a **VariableExp**. The *referredVariable* is an instance of the **VariableDeclaration**, being its name *self*.

```
context Visitor::visitLetExp(o: LetExp, metricName: MetricAcronym)
pre:        o.oclIsKindOf(OclAbstractSyntax::Expressions::VariableDeclaration)
```

```
post:          (metricName = MetricAcronym:: NES
               and o.variable.name = 'self')
               implies
               valueMetric = valueMetric@pre + 1
context Visitor::visitTupleLiteralExp(o: TupleLiteralExp,
                    metricName: MetricAcronym)
pre: o.oclIsKindOf(OclAbstractSyntax::Expressions::TupleLiteralExp)
post:          (metricName = MetricAcronym::NES and
               o.tuplePart->notEmpty())
               implies valueMetric = valueMetric@pre +
               o.tuplePart->collect(a | if a.varName = 'self' then 1
                    else 0 endif )->sum()

context Visitor::visitLoopExp(o: LoopExp, metricName: MetricAcronym)
pre: o.oclIsKindOf(OclAbstractSyntax::Expressions::LoopExp)
post:
               (metricName = MetricAcronym::NES and
               o.iterators->size() >= 1)
               implies

               valueMetric = valueMetric@pre +
               o.iterators->collect( a | if a.varName = 'self' then 1
               else 0 endif )->sum()

context Visitor::visitIterateExp(o: IterateExp, metricName: MetricAcronym)
pre:           o.oclIsKindOf(OclAbstractSyntax::Expressions::IterateExp)
post:          (metricName = MetricAcronym::NES
               and o.result.varName = 'self')
               implies valueMetric = valueMetric@pre + 1

context Visitor::visitVariableExp(o: VariableExp, metricName: MetricAcronym)
pre:           o.oclIsKindOf(OclAbstractSyntax::Expressions::VariableExp)
post:          (metricName = MetricAcronym::NES
               and o.referredVariable.varName = 'self')
               implies valueMetric = valueMetric@pre + 1
```

- **NKW:** This measures value is equal to the number of keywords used in an expression. The keywords were defined in section 4.3.3. Many OCL meta-classes represent different keywords, e.g. **LetExp** a Let expression, **IfExp** an If expression, etc.

```
context Visitor::visitLetExp(o: LetExp, metricName: MetricAcronym)
pre:           o.oclIsKindOf(OclAbstractSyntax::Expressions::VariableDeclaration)
post:          metricName = MetricAcronym:: NKW)

               implies
               valueMetric = valueMetric@pre + 1
```

```
context Visitor::visitIfExp(o: IfExp, metricName: MetricAcronym)
pre:        o.oclIsKindOf(OclAbstractSyntax::Expressions::IfExp)
post:       (metricName = MetricAcronym::NKW)
            implies valueMetric = valueMetric@pre + 1

context Visitor::visitOperationCallExp(o: OperationCallExp,
                    metricName: MetricAcronym)
pre:        o.oclIsKindOf(OclAbstractSyntax::Expressions::OperationCallExp)
post:       (metricName = MetricAcronym::NKW) and
            Set{'and', 'or', 'xor', 'implies'}->includes(o.referredOperation.name))
            implies valueMetric = valueMetric@pre + 1
```

- **NBO:** Boolean Operators are instances of **OperationCallExp** metaclass where the name of the *referredOperation* is *and*, *or*, *xor* or *implies*.

```
context Visitor::visitOperationCallExp(o: OperationCallExp,
                    metricName: MetricAcronym)
pre:        o.oclIsKindOf(OclAbstractSyntax::Expressions::OperationCallExp)
post:       (metricName = MetricAcronym::NBO and
            Set{'and', 'or', 'xor', 'implies'}->includes(o.referredOperation.name))

            implies valueMetric = valueMetric@pre + 1
```

- **NCO:** Comparison Operators are instances of OperationCallExp metaclass where the name of the *referredOperation* is '=', '<=', '>=', '<', '>'.

```
context Visitor::visitOperationCallExp(o: OperationCallExp,
                    metricName: MetricAcronym)
pre:        o.oclIsKindOf(OclAbstractSyntax::Expressions::OperationCallExp)
post:       (metricName = MetricAcronym::NCO and
            Set'=', '<=', '>=', '<', '>'->includes(o.referredOperation.name))
            implies valueMetric = valueMetric@pre + 1
```

- **NAS:** The 'number of the Attributes belonging to the Classifier that *self* represents' is obtained by counting the instances of **AssociationCallExp** where *self* is the *name* of the *referredvariable* through its *source* variable.

```
context Visitor::visitAttributeCallExp(o: AttributeCallExp,
                    metricName: MetricAcronym)
post:       (metricName = MetricAcronym::NAS
            and o.source.oclIsTypeOf(VariableExp)
            and o.source.oclAsType(VariableExp).referredVariable.varName = 'self'
            )
            implies importedproperties =
            importedproperties@pre ->including(o.referredAttribute.name)
```

- **NOS:** The 'number of the Operations belonging to the Classifier that *self* represents' is equal to the quantity of instances of **OperationCallExp** where *self* is the *name* of the *referredvariable* through its *source* variable.

```
context Visitor::visitOperationCallExp(o: OperationCallExp,
                    metricName: MetricAcronym)
post:       (metricName = MetricAcronym::NOS
            and o.source.oclIsTypeOf(VariableExp)
            and o.source.oclAsType(VariableExp).referredVariable.varName = 'self'
            )
            implies importedproperties =
            importedproperties@pre->including(o.referredOperation.name)
```

- **NAN:** The 'number of the Attributes belonging to a Classifier which is different to the Classifier that *self* represents' is equal to the quantity of instances of **AssociationEndCallExp** where the type of its source is an **AssociationEndCallExp** or **AssociationClassCallExp**.

```
context Visitor::visitAttributeCallExp(o: AttributeCallExp,
                    metricName: MetricAcronym)
post:       (metricName = MetricAcronym::NAN
            and o.source.oclIsTypeOf(AssociationEndCallExp)
            and o.source.oclIsTypeOf(AssociationClassCallExp)
            )
            implies importedproperties =
            importedproperties@pre ->including(o.referredAttribute.name)
```

- **NON:** The 'number of the Operations belonging to a Classifier which is different to the Classifier that *self* represents' is equal to the quantity of instances of **AssociationEndCallExp** where the type of its source is an **AssociationEndCallExp** or **AssociationClassCallExp**.

```
context Visitor::visitOperationCallExp(o: OperationCallExp,
                    metricName: MetricAcronym)
post:       (metricName = MetricAcronym::NON
            and o.source.oclIsTypeOf(AssociationEndCallExp)
            and o.source.oclIsTypeOf(AssociationClassCallExp)
            )
            implies importedproperties =
            importedproperties@pre->including(o.referredOperation.name)
```

- **NUDTA:** Attributes belonging to a DataType is obtained by counting the instances of **AttributeCallExp** where its type, the *referredvariable*, is a Datatype.

```
context Visitor::visitAttributeCallExp(o: AttributeCallExp,
                    metricName: MetricAcronym)
post:       (metricName = MetricAcronym::NUDTA
            and o.referredAttribute.type.oclIsTypeOf( DataType )
            )
            implies importedproperties =
            importedproperties@pre ->including(o.referredAttribute.name)
```

- **NUDTO:** The 'number of Operations belonging to a DataType' is obtained counting the instances of **OperationCallExp** where the owner of its the *referredOperation* is a Datatype.

```
context Visitor::visitOperationCallExp(o: OperationCallExp,
                    metricName: MetricAcronym)
post:       (metricName = MetricAcronym::NUDTO
            and o.referredOperation.owner.oclIsTypeOf( DataType ) )
            )
            implies importedproperties =
            importedproperties@pre->including(o.referredOperation.name)
```

- **N@P:** Whenever the postfix atPre is specified, an instance of **OperationCallExp** metaclass should be used, having the name of the *referredOperation* the name of 'atPre'.

```
context Visitor::visitOperationCallExp(o: OperationCallExp,
                    metricName: MetricAcronym)
pre:        o.oclIsKindOf(OclAbstractSyntax::Expressions::OperationCallExp)
post:       (metricName = MetricAcronym::NatP
            and o.referredOperation.name = 'atPre')
            implies valueMetric = valueMetric@pre + 1
```

- **NIO:** The number of 'oclIsTypeOf', 'oclIsKindOf' and 'oclAsTupe' operators is obtained counting the instance of OperationCallExp metaclass where the name of the *referredOperation* is the matching name, i.e. the name is 'oclIsTypeOf', 'oclIsKindOf' or 'oclAsTupe'.

```
context Visitor::visitOperationCallExp(o: OperationCallExp,
                    metricName: MetricAcronym)
pre:        o.oclIsKindOf(OclAbstractSyntax::Expressions::OperationCallExp)
post:       (metricName = MetricAcronym::NIO and
            Set'oclIsTypeOf', 'oclIsKindOf', 'oclAsTupe'->
                    includes(o.referredOperation.name))

            implies valueMetric = valueMetric@pre + 1
```

The value of NES, NKW, NBO, NCO, NAS, NOS, NAN, NON, NIO, NUDTO, NUDTA and N@P is obtained in similar way. However the definition for the rest of the measures is more complicated, and we will their definition in details.

Finally, we can show how the value of NCO is obtained:

```
context ExpressionInOcl::value_of_NCO() : Integer
post:       self.bodyExpression^accept_new(self.visitor, 0, MetricAcronym::NCO)
            and result = self.visitor.valueMetric
```

In the same way, using the valueMetric variable, the values of NES, NKW, NBO, NIO and N@P are obtained.

In order to compute the value of NAN the following operation is defined:

```
context ExpressionInOcl::value_of_NAN() : Integer
post:       self.visitor.oclIsNew()
            and self.bodyExpression^accept_new(self.visitor, 0, MetricAcronym::NCO)
            and result = self.visitor.importedproperties->size()
```

In the same way, using the *importedproperties* variable, the values of NAS, NUDTA, NOS, NON and NUDTO are obtained.

5.3.3.1 NNR and NNC measures

Whenever a visitor accesses a NavigationCallExp object, it loads in a set (called navigatedClasses) the name of the classes used in the navigation (whether the modeler use 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). The size of this set is used to obtain the NNC value.

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

A similar operation is used to compute the quantity of relationships (NNR measure). However, as a same name of a rolename can be used in different classes we decided to represent in the set those elements composed by the pair of two strings, 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))
```

5.3.3.2 WNN and WNCO measures

There are some OCL expression measures that its computation use a weighted factor, such as WNN and WNCO using a level factor. In both cases the level represents the degree of compositions of collection operations, i.e. if an OCL expression includes a navigation along with a collection operation whose body includes a new navigation, the collection operation determines a new level. Level 1 refers to the place where the first navigation is defined whereas the body of the collection operation is at the second level. WNN weights the number of navigations of a level by the level itself, meanwhile WNCO weights the number of collection operations of a level by the level itself.

The level is specified through a parameter of each accept operations and consists of a value parameter that each ast node forwards to its descendants. The value is incremented by the IteratorExp accept operation because this class represents all the collection operations.

```
context IteratorExp::accept(v: Visitor_ast, level: Integer,
                    metricName: MetricAcronym)
post:         v^visitIterateExp(self, level , metricName) and
              self.Body^accept_new(v, level + 1, metricName) and
              self.source^accept_new(v, level, metricName) and
```

```
(self.iterators->size() >= 1 implies
iterators->forAll(a | a.initExpression->notEmpty()
        implies
        a.initExpression^accept_new(v, level + 1, metricName)
        ))
```

The same accept operation is defined for IterateExp.

5.3.3.2.1 WNN measure: When each ast node of type NavigationCallExp calls the visitor that corresponds to the class, the visitor uses the level parameter. The first NavigationCallExp object (in an ast) of any simple or combined navigation produces that a level value being included in a set of Values (aSetofValues) to compute WNN measure.

```
context Visitor::visitIterateExp(o: IterateExp, level: Integer, metricName: MetricAcronym)
pre: o.oclIsKindOf(OclAbstractSyntax::Expressions::IterateExp)
post: (metricName = MetricAcronym::WNN
        and (o.appliedProperty.oclIsTypeOf(AttributeCallExp)
        or o.appliedProperty.oclIsTypeOf(OperationCallExp)
        and o.source.oclIsTypeOf(VariableExp)
        )
        implies aSetofValues = aSetofValues@pre->including(level)
```

5.3.3.2.2 WNCO measure: When each ast object of type IteratorExp calls the Visitor operation that corresponds to the class, the visitor uses the level parameter. The visitor includes a level value in the set of values to compute this measure. The same operation is performed in the visitIterateExp.

```
context Visitor::visitIteratorExp(o: IterateExp, level: Integer, metricName: MetricAcronym)
pre: o.oclIsKindOf(OclAbstractSyntax::Expressions::IteratorExp)
post:       metricName = MetricAcronym::WNCO
            implies aSetofValues = aSetofValues@pre->including(level)

context Visitor::visitIterateExp(o: IterateExp, level: Integer, metricName: MetricAcronym)
pre: o.oclIsKindOf(OclAbstractSyntax::Expressions::IterateExp)
post:       metricName = MetricAcronym::WNCO
            implies aSetofValues = aSetofValues@pre->including(level)
```

5.3.3.3 DN measure

The value of DN measure is obtained using an OCL Sequence of Integer, called maxDepthAtLevel. The Sequence contains the max depth of navigation for each

level. A new operation was defined in the NavigationCallExp metaclass. This operation, getdepthofNavigation, is recursive because the navigation could be combined. In each iteration of this operation a new relationship is navigated. When the last rolename of a combined navigation is traversed the recursion finish.

```
context Visitor::visitNavigationCallExp(o: NavigationCallExp, level: Integer,
                    metricName: MetricAcronym)
post:          (metricName = MetricAcronym::DN and
               (o.appliedProperty.oclIsTypeOf(AttributeCallExp)
               or o.appliedProperty.oclIsTypeOf(OperationCallExp))
               implies maxDepthatLevel = maxDepthatLevel@pre->insertat(level,
                    maxDepthatLevel->at(level).max(o.getdepthNavigation(o))

context NavigationCallExp::getdepthNavigation(a: NavigationCallExp) : Integer
body:          (if source.oclIsTypeOf(AssociationEndCallExp)
               then 1 + getdepthNavigation(source.oclAsType(AssociationEndCallExp))
               else
                    (if source.oclIsTypeOf(AssociationClassCallExp)
                    then
                    1 + getdepthNavigation(source.oclAsType(AssociationClassCallExp))
                    else
                    1
                    endif)
               endif)
```

The sequence maxDepthAtLevel is used to obtain the value of DN in the following way: The values of each level are accumulated and for each new level we increment by 2 the value of the measure because a definition connection is used to represent this situation.

5.4 Contribution to the Dissertation

Within the OO software measurement community the formal definition of measures is an important aspect that was almost neglected. Although there is a huge amount of OO measures, the lack of formalization constitutes a serious matter. Only natural language or rigorous mathematical definition were used, being none of these extremes suitable and widely adopted. Sometimes an increase in the focus on one aspect such as a more rigorous definition (e.g. when mathematical background is used) it decreases the focus on another aspect (e.g. not everybody is familiar with the mathematical formalism applied in a rigourous definition). On the other hand, if a less rigorous definition is applied for gaining a wide range of audience, ambiguity is introduced. Our belief is that the combination of the metamodelling facilities and the OCL as a language for defining OCL semantics, such as in defining the UML

and OCL languages, allows also unambiguous measure definition achieving both understandability and formality in their specification. We claim that the formal definition of our measures using OCL language is easy to grasp by anybody familiar with metamodelling. The relevance of the proposed approach for the specification of OCL expressions measures using OCL upon the OCL metamodel is reinforced by the growing field of Model Driven Architecture paradigm and the proliferation of MOF-compliant architectures. In addition, the formal specification of measures allows the development of precise measures extraction tools and we believe that in the future, code implementation for UML and OCL measures extraction can be translated from their formal definition through model transformations (a relevant aspect of model-driven engineering).

In this chapter our goal was to precisely define the OCL expression measures. In order to fulfill this purpose we used the OCL metamodel (at M2 of MOF). So, we had firstly described each of the OCL metaclasses used in any measure definition. Afterwards, we described the strategy used to traverse the ast object (an instance of the OCL metamodel) to compute a measure value. Finally, we had included the measure formal definition.

Chapter 6

Theoretical Validation

The theoretical validation is carried out to assess *whether a measure actually measures what it claims to measure*. The theoretical proof is crucial to avoid construct validity threats. To develop the theoretical validation we have applied the activities described in section 2.3.2. We have used property-based frameworks (Activity T_1 of Figure 2.5) and a framework based on the measurement theory (Activity T_2 of Figure 2.5). These two applications are described in sections 6.1 and 6.2 respectively.

Whenever an activity of the method is applied, the corresponding section is titled accordingly and a reference to its number is included between parenthesis.

6.1 Use Property-based Frameworks (T_1)

In this section we use two frameworks to theoretical validate the measures: the Briand et al. Framework [BMB96], [BMB97] and its adaptation for interaction-based measures for coupling and cohesion [BMB99]. Subsection 6.1.1 uses the property-based framework of Briand et al. [BMB96], [BMB97] for validate measures of size and length, i.e. we used generic properties defined by Briand et al., applying the activities P_1 and P_2 of Figure 2.5 (b).

However, in order to study the relations defined between a software individual part -in our context, an OCL expression- and its associated software system -mainly, attributes, rolenames, and operations and other expressions- we need to apply context-dependent properties as it is described by activities P_3 and P_4 of of Figure 2.5 (b). So, to perform this activity we follow a similar approach to that described in section B.1.2 when context-dependent properties are defined fro ADA' modules. In section 6.1.2 we explain how the Briand et al. [BMB99] adaptation framework for interaction-based measures for coupling and cohesion [BMB99] is used and applied.

We used this framework because we focus on the import-coupling of an OCL expression, and we want to study the interaction of an OCL expression with its associated system, i.e. its surrounding context (remember that OCL is not an stand-alone language). The framework used is rarely applied in other theoretical validation of measures although its authors states that the notion of interaction can be applied to other object-based design methods and formalisms. Briand et al. [BMB99] also remark that 'When working with other design techniques, one can use all the available information on the interactions between elements of the design. If mechanisms for describing such interactions exists, then one can apply our approach, based on more information than is available in our case', obtaining in that way more accurate models.

6.1.1 Applying Generic Properties ($\mathbf{P_1}$,$\mathbf{P_2}$)

In this section the generic properties of size and length [BMB96], [BMB97] (described in section B.1.1.1) are used for the validation of some OCL expression measures. Due to the fact we will apply a generic property we follow the activities P_1 and P_2 of Figure 2.5 (b).

6.1.1.1 NCO as a Size Measure

- Instantiation ($\mathbf{P}_1$): For our purpose and in accordance with the framework of Briand et al. [BMB96] and [BMB97], we consider that an OCL expression is a system composed of comparison operators (elements) and the relationship between elements is just the sequential relationship. A sub-expression will be considered a module.

- Validation using the properties ($\mathbf{P}_2$): We will demonstrate that NCO fulfills all of the axioms that characterize size measures, as follows:

  1. **Nonnegativity:** Is directly proven and it is impossible to obtain a negative value.
  2. **Null value:** An expression e without a message, has a WNM(e) = 0
  3. **Module Additivity:** If we consider that an OCL expression is composed of modules, the number of comparison operator of an OCL expression will always be the sum of the number of comparison operators of its modules.

In a similar way, it is possible to show that WNN, WNCO, NKW, NES, NIS, NEI, NII and NBO are size measures.

6.1.1.2 DN as a Length Measure

- Instantiation ($\mathbf{P}_1$): For our purpose and in accordance with the framework of Briand et al. [BMB96] and [BMB97], we consider that an OCL expression is a system. The elements are the classes (to which the expression navigates) and the relationships are the navigations of a UML relationship.
- Validation using the properties ($\mathbf{P}_2$): We will demonstrate that DN fulfills all of the axioms that characterize length measures, as follows:
  1. **Nonnegativity and Null Value** are straightforwardly satisfied, the depth of a tree can never be negative, and an expression without navigation has an empty tree, and DN is 0.
  2. **Nonincreasing monotonocity for connected components**: If we add relationships between elements of a tree (classes or interfaces) the depth does not vary.
  3. **Nondecreasing monotonocity for non- connected components**: Adding a relationship to two unconnected components (two trees) makes them connected, and its length is not less than the length of the two unconnected components.
  4. **Disjoint modules**: The depth of a tree is given by the component that has more levels from the root to the leaves.

6.1.2 Applying Context-dependent Properties ($\mathbf{P}_3$,$\mathbf{P}_4$,$\mathbf{P}_5$)

In this section we will apply a similar approach as the one taken by Briand et al. in [BMB99]. This approach was described in section B.1.2. Instead of using Ada´s modules and subroutines as a high level design we will use UML/OCL models, being the measures applied to OCL expressions defined in that models. In the approach applied several concepts should be defined, mainly the notion of what is considered a *high-level design*, which are the main components of the object-based design along with the interaction links, etc.
Before applying the framework of Briand et al. [BMB99], we will define the basic terminology used in our context.

6.1.2.1 Definition of the Context ($\mathbf{P}_3$)

In this subsection we define the context to which the properties will be applied, this activity constitutes the application of activity P_3 of Figure 2.5 (b). Our object of study is to define measures for a specific elementary component of a high-level

design. So, first at all, we will define which is considered for us a high-level design (implicitly we are currently defining the *object-based design method and formalism* used in our study):

A High level Design and its elementary components and relationships UML/OCL models will be considered a *high level design*: A UML class diagram with the specification of OCL expressions defining invariants, definitions, queries, pre- and post-condition of operations (which only declares the effect of the operation but not how the operation is performed [WK03]), etc. will be considered a high level design.

Now we will describe the elementary component of our high level design. One important elementary component is a module (we use the same concepts as [BMB99]) represented in our design by a class. The module, a class, can be composed by three different components:

- attributes
- operations
- OCL expressions

Modules and module´s components may be related to each other by IS_COM- PONENT_OF and USES relationships. The definition of these relation can be found in section B.1.2.

We will define measures that can be applied only to OCL expressions. We will study the relations defined between a software individual part -in our context, an OCL expression- and its associated software system -mainly, attributes, rolenames, and operations and other expressions-. For short, we will use the term software part (sp) to denote an OCL expression. An OCL expression, for example a pre- or post-condition, gives to the designer a high level point of view of the content of an operation. In that way, in our context the OCL expression is component of what is called 'subroutine' in the Briand et al.´s framework. Indeed, the information available in our context about a 'subroutine' is higher than one available in Ada´s context of [BMB99].

Interactions
There are three possible kinds of interactions forms in our context:

1. data declarations to data declarations,
2. data declarations to OCL expression,
3. OCL expression to OCL expression,

However we do not study all of them, due to the fact our measures should be applied to OCL expressions the first form of interaction is not relevant for our study. We decided to focus only in the second kind of interaction, and we specifically consider coupling-based interaction.

An OCL expression can access to the components of its associated system through OCL messages, OCL navigations, etc. The *interaction links* are defined as follow:

Definition 6.1.1 DU-interaction. *The interaction from data Declaration to data Used in an OCL expression.*

In our context we consider as a Data declaration any information of the model (attributes, role names, etc.) referred through an OCL property or other OCL mechanisms. Whenever an OCL expression refers to a property, it 'uses' the property. The way a Data declaration of the associated system is used in an expression, can involve different OCL concepts for that reason we have differentiated several kinds of DU-interaction.

Example 6.1.2 *The top part of Figure 6.1 shows piece of a UML/OCL models (a high level design) where six OCL expressions were defined. The bottom part shows a directed graph, similar to the ones depicted in [BMB99], showing the modules and link relationships of that model. Data declaration and Data Used are represented by rounded nodes, operations and OCL expressions by thick lined boxes, modules (classes) by thin lined boxes, and interactions by dotted arcs. For instance, it is apparent that both the precondition and the postcondition use the p parameter declared by the new_customer operation. Two OCL expressions, one definition and one invariant, use the salary attribute of the Job class. These are common examples of interaction between a data declaration and OCL expressions. One example of interaction between two OCL expressions (in the Person class) is shown in the arc between the income declaration of the definition expression and its use in an invariant.*

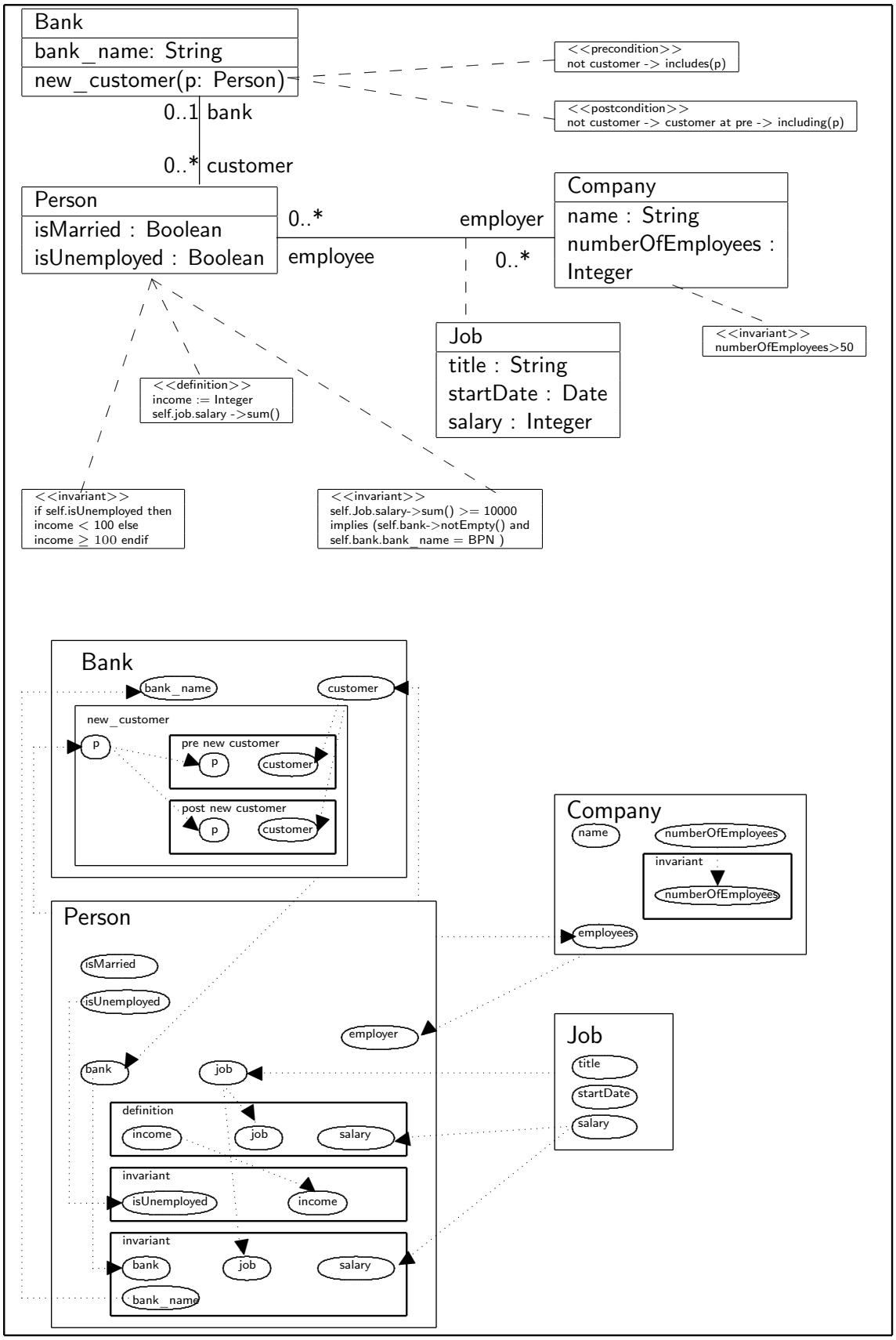


Figure 6.1: DU-interaction

After defining the interaction links of our high level design and giving an exemplification we are able to define a core concept: import-coupling.

Import-coupling: Given a software part *sp*, import-coupling of *sp* is the number of DU-interaction between data declaration external to *sp* and data used within *sp*.

Our hypothesis is similar to the IC and H-ISP hypothesis of [BMB99]:

- The more dependent a software part (an OCL expression) on external data declarations, the more external information needs to be known in order to make the software part consistent with the rest of the system. Any modification on the data declaration could affect the modification of our software part (the OCL expression). Also the comprehension of the software part as a whole involve the comprehension of the external data declaration, from which it depends.
- the larger the number of imported software parts, the larger the number to be understood, the more likely to occurrence of a fault.

6.1.2.2 Context-dependent Properties (P_4)

This activity represents the application of activity P_4 of Figure 2.5 (b). The properties we believe should be satisfied by interaction-based import-coupling measures are defined below. These properties are instantiated for our context, of the properties defined in [BMB96] for coupling:

- **Property OCLCoupling1. Nonnegativity.** Given a software part *sp* (an OCL expression), the measure import_coupling_measure(*sp*) ≥ 0. Import_coupling_ measure(*sp*) $= 0$ if *sp* does not have import interactions with other software parts.
- **Property OCLCoupling2. Monotonicity.** Let m_1 be a module and II(m_1), its set of import interactions. If m_2 is a modified version of m_1 with the same sets of data and subroutine declarations and one more import interaction so that II(m_2) $\supseteq$ II(m_1), then import_coupling_measure(m_2) $\geq$ import_coupling_measure(m_1)[1].
- **Property OCLCoupling3. Merging of Modules.** The sum of the import-couplings of two modules is no less than the coupling of the module which is composed of the data declarations of the two modules.

[1] Adding import interactions to a module cannot decrease its import-coupling.

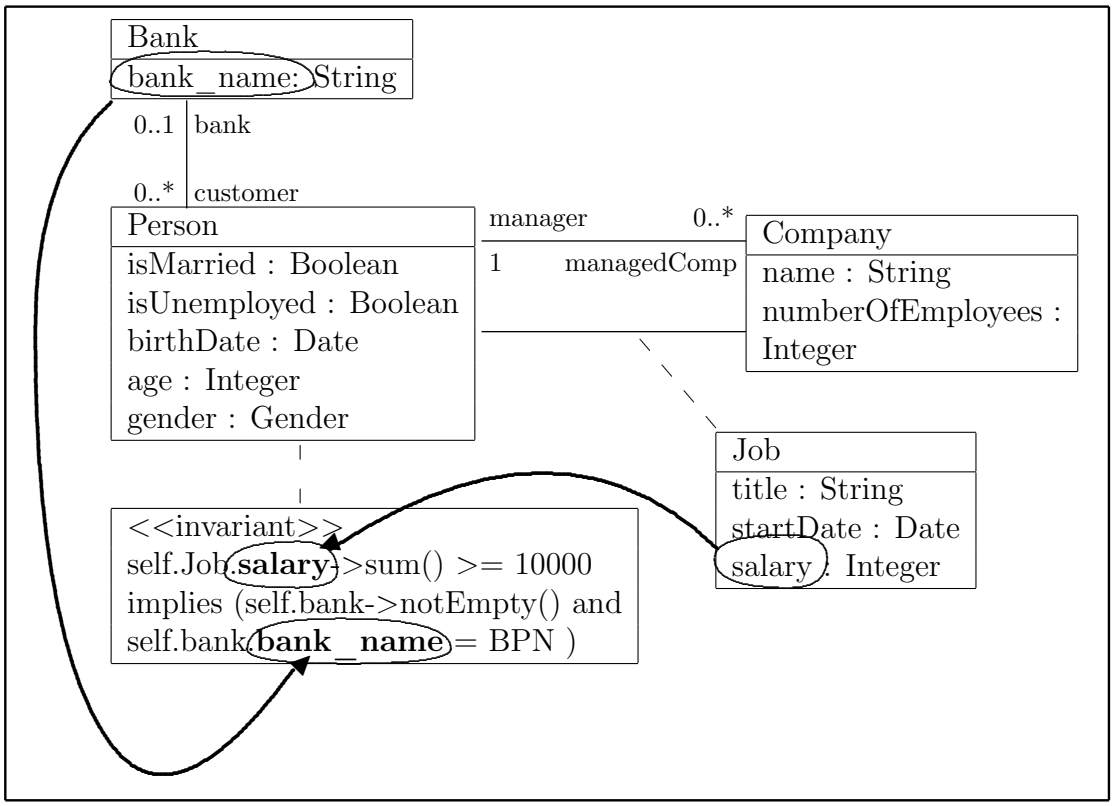


Figure 6.2: Design Used for Explaining the Briand et al.´s Theoretical Validation

Now we will exemplify the application of this framework in the theoretical validation of NAN measure.

6.1.2.3 NAN properties as a coupling interaction-based measure ($\mathbf{P}_5$)

This activity represents the application of **activity $\mathbf{P}_5$** of Figure 2.5 (b). We will make some definitions prior to the application of properties of interaction-based measures for coupling to the NAN (The Number of Attributes referred through Navigations in an expression) measure:

- **Relation:** The relations are defined between a software individual part (in our context, an OCL expression) and its associated software system (attributes which it is possible to access through navigations in the NAN measure).

- **DU-interaction:** The interaction from Data declaration to Data Used (attributes used through navigations) in an OCL expression.

- **Import-coupling:** Given a software part sp (an OCL expression), import-coupling of sp is the number of DU-interactions between data declaration external to sp and data used within sp.

Our hypothesis is similar to the ISP-hypothesis of Briand et al. [BMB99]: The larger the number of "used" software parts, the larger the context to be understood, the more likely the occurrence of a fault.

Example 6.1.3 *Figure 6.2 shows a UML/OCL model and an OCL expression. The value of NAN applied to the OCL expression is 2, because two external attribute data were used in the OCL expression´s invariant, and its use was through navigations. Two curves show the place where the two external data were declared, and the place they were used within the OCL expression.*

Following a similar approach applied in Briand et al. [BMB99] the properties for interaction-based measures for coupling are instantiations, for our specific OCL context, of the properties defined in Briand et al. (1996) and (1997) [BMB96], [BMB97] for coupling.

1. **Nonnegativity:** Is directly proven, as it is impossible to obtain a negative value. An expression sp without navigation (referring to attributes) in its definition has NAN(sp) = 0.

2. **Monotonicity:** Is directly verified, adding import interactions - in this case, DU-interactions of navigations referring to attributes- to an OCL expression cannot decrease its import-coupling. If we add a new navigation referring to an attribute in an expression sp, two possible situations can happen: (1) the attribute referred to in the added navigation is an attribute already used by a DU-interaction. Thus the measure NAN applied to the new expression obtained, is equal to NAN(sp). (2) If the added navigation refers to a new attribute, then NAN applied to the new expression is greater than NAN (sp).

3. **Merging of Modules:** This property can be expressed for our context 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 of the two modules". The value of NAN for an expression which consists of the union of two original expressions, is equal to the NAN of each expression merged when the sets of attributes referred to in each original expression are disjointed, otherwise it is less than NAN of each expression merged.

In a similar way, it is possible to show that NNR, NON, NNC, NPT, NUDTA, NUDTO, NAS, NOS, N@P, WNN, WNCO, NEI, NII and NIO are interaction-based measures for coupling.

6.2 Use Framework based on the Measurement Theory (T_2)

In this section we will follow each of the steps for measurement construction proposed in Poels and Dedene´s DISTANCE framework [PD00], [PD99]. We used the process described in section B.2. In order to exemplify how it is applied two UML/OCL models of Figure 6.3 were used.

- **Find a measurement abstraction** (MT_1): In our case the set of software entities e is the Universe of OCL Expressions (UOE) within UML/OCL models that are relevant for some application domains whereas e is an OCL expression (OCL-E) (i.e. $e \in$ UOE).

  The attribute of interest *attr* is the number of attributes referred through navigations [2], i.e. a particular aspect of OCL expressions structural property. Let U*Attr*r*tN* be the Universe of attributes *rtN* relevant to the Universe of Discourse (UoD) domain. The set of attributes *rtN* in an OCL-E, noted as A*rtN*(OCL-E) is a subset of of U*Attr*r*tN*.

  All the sets of attributes *rtN* within the OCL expressions of UOE are elements of the power set of U*Attr*r*tN*, denoted by $\wp$(U*Attr*r*tN*). We can therefore equate the set of measurement abstractions M to $\wp$(U*Attr*r*tN*) and define the abstraction function as:

  abs_{NAN}: UOE $\rightarrow$ $\wp$(U*Attr*r*tN*): OCL-E $\rightarrow$ A*rtN*(OCL-E)

  This function maps an OCL-E onto its set of Attributes *rtN*. In our example, see Figure 6.3 we have the set of attributes *rtN* of the OCL-E A and of OCL-E B:
  abs_{NAN}(E A) = A*rtN*(OCL-E A) = salary
  abs_{NAN}(E B) = A*rtN*(OCL-E B) = salary, bank_name

- **Model distances between measurement abstractions** (MT_2): The next step is to model distances between the elements of M. We need to find a set of elementary transformation types (T_e) for the set of measurement abstractions $\wp$ (U*Attr*r*tN*) such that any set of attribute *rtN* can be transformed into any other set of attributes *rtN* by means of a finite sequence of elementary transformations. Finding such a set it is quite easy in the case of a domain. Since the elements of $\wp$ (U*Attr*r*tN*) are sets of attributes *rtN*, T_e must only contain two types of elementary transformations: one for adding an attribute

[2] We will use the acronym *rtN* in order to said 'referred through navigations'

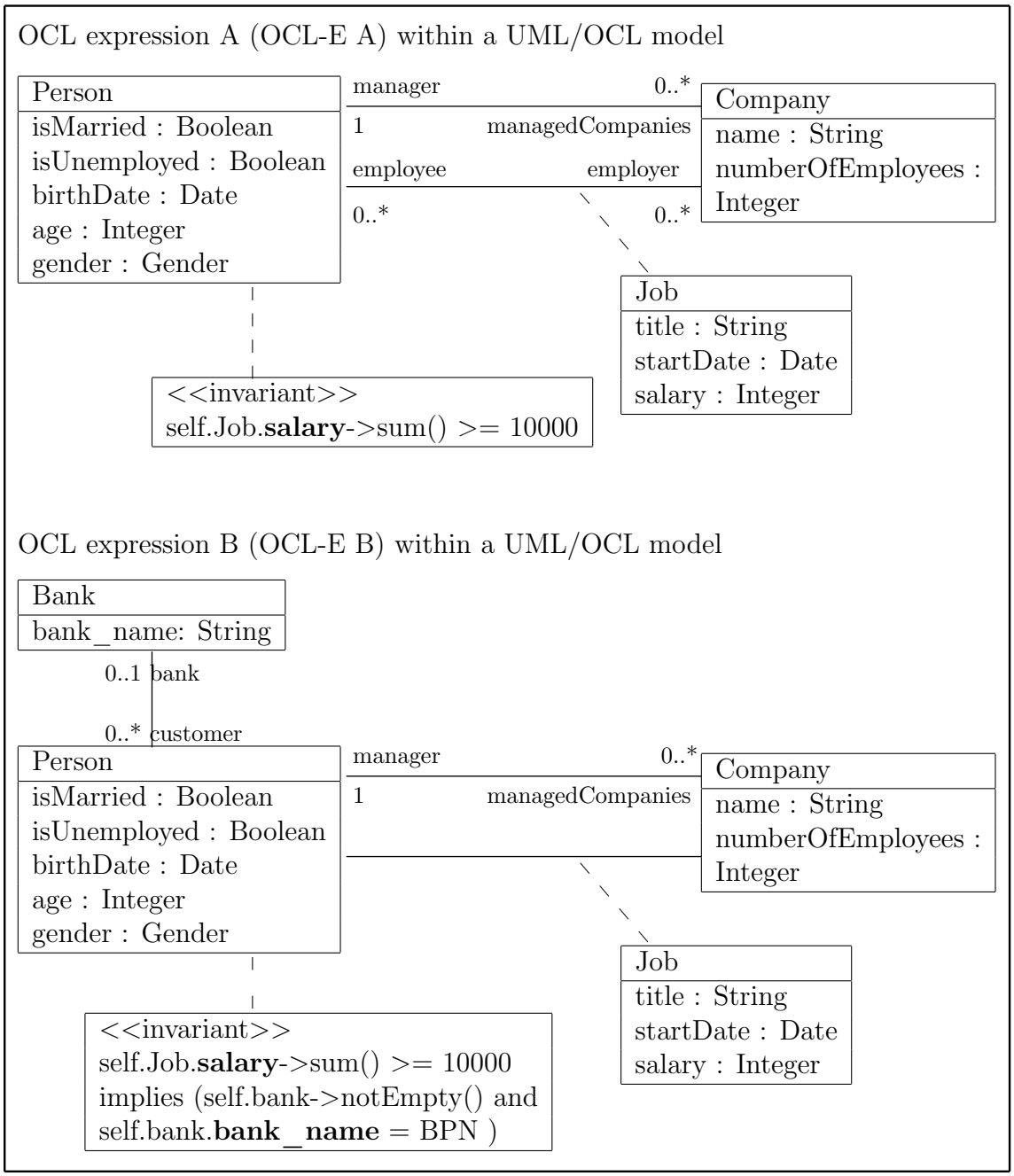


Figure 6.3: Two UML/OCL Models used with the DISTANCE Framework

rtN to a set and one for removing an attribute *rtN* from a set. Given two sets of attribute *rtN* $s_1 \in \wp$ (UAttr*rtN*) and s_2 $\wp$ (UAttr*rtN*), s_1 can always be transformed into s_2 by removing first all attributes *rtN* from s_1 that are not in s_2, and then adding all attribute *rtN* to s_1 that are in s_2, but not in the original s_1. In the "worst case scenario", s_1 must be transformed into s_2 via an empty set of attributes *rtN*.

Formally, $T_e = t_{0-NAN}, t_{1-NAssoc}$, where t_{0-NAN} and t_{1-NAN} are defined as:
$t_{0-NAssoc}$: $\wp$ (UAttr*rtN*) $\rightarrow$ $\wp$ (UAttr*rtN*): s $\rightarrow$ s $\cup$ {a}, with a $\in$ UAttr*rtN*
$t_{1-NAssoc}$: $\wp$ (UAttr*rtN*) $\rightarrow$ $\wp$ (UAttr*rtN*): s $\rightarrow$ s - {a}, with a $\in$ UAttr*rtN*

In our example, the distance between absNAN(OCL-E A) and absNAN(OCL-E B) can be modelled by a sequence of elementary transformations, as is shown below.
m_0 = {salary}
m_1 = {salary, bank_name} = $t_{0-NAN}(m_0)$

This sequences of one elementary transformation is sufficient to transform A*rtN*(OCL-E A) into A*rtN*(OCL-E B). All "shortest" sequences of elementary transformations qualify as models of distance.

- **Quantify distances between measurement abstractions** (MT_3): In this step the distances in $\wp$(UAttr*rtN*) that can be modelled by applying sequences of elementary transformations of the types contained in T_e are quantified. A function δ_{NAN} of these distances is a measure (in a mathematical sense) that is defined by the symmetric difference model, i.e. a particular instance of the model of Tversky (Suppes et al. [KDST06]). It has been provided in [PD99] that the symmetric difference model can always be used to define a measure when the set of measurement abstractions is a power set:

δ_{NAN}: $\wp$(UAttr*rtN*) $\times$ $\wp$(UAttr*rtN*) $\rightarrow \Re$: (s, s') $\rightarrow$ (| s - s'| + | s' - s|)

This definition is equivalent to stating that the distance between two sets of attributes *rtN*, as modelled by a shortest sequence of elementary transformations between these sets, is measured by the count of elementary transformations in the sequence. Note that for any element in s but not in s and for any element in s but not in s, an elementary transformation is needed. The symmetric difference model results in a value of 1 for the distance between the set of associations of OCL-E A and OCL-E B.
δ_{NAN}(abs(OCL-E A), abs(OCL-E B)) =

|{salary} - {salary, bank_name}| + |{salary, bank_name} - {salary}| = | ∅ | + |{bank_name}|= 1

- **Find a reference abstraction** (MT_4): In our example the obvious reference point for measurement is the empty set of attribute *rtN*. It is desirable that an OCL expression without attribute *rtN* will have the lowest possible value for the NAN measure. So that we define the following function:
  ref_{NAN}: UOE $\rightarrow \wp$ (UAttr*rtN*): OCL-E $\rightarrow \emptyset$

- **Define the software measure** (MT_5): In our example, the number of attribute *rtN* in an OCL-E $\in$ UOE can be defined as the distance between its set of associations A*rtN*(OCL-E) and the empty set of associations $\emptyset$. Hence, the NAN measure can be defined as a function that returns the value of the measure δ_{NAN} for the pair of sets A*rtN*(OCL-E) and $\emptyset$ for any OCL-E $\in$ UOE:
  $\forall$ OCL-E $\in$ UOE: NAN(OCL-E) = δ_{NAN}(A*rtN*(OCL-E), $\emptyset$) = |A*rtN*(OCL-E) - $\emptyset$| + | $\emptyset$ - A*rtN*(OCL-E)| = |A*rtN*(OCL-E)|

Consequently, a measure that returns the count of attribute referred through navigations of OCL expression within UML/OCL models qualifies as a number of attributes *rtN* measure. Table 6.1 summarized the definition of the NAN measure following the distance-based process of measurement construction. In fact, table 6.1 is an instantiation of the template shown in table B.2.

The rest of the proposed OCL measures measures can be modelled by means of a set abstraction (see table 6.2, 6.3, 6.4) and as a consequence all the measures take the form of a simple count, so their process construction is analogous followed by the measure NAN.

6.3 Contribution to the Dissertation

We have used two different frameworks in order to theoretically validate the measures defined in chapter 4. These frameworks are the property-based framework of Briand et al. ([BMB96], [BMB97]) and its adaptation for interaction-based measures for coupling and cohesion [BMB99] and the DISTANCE framework based on measurement theory [PD99].

From the results of the validation in the Poels and Dedene´s DISTANCE framework [PD99] we can conclude that all the measures are ratio measures. A summary of the results of the property-based frameworks of Briand et al. is shown in Table 6.5.

We are aware that it is necessary to provide information about the utility of the measures in practice, through their empirical validation. The empirical validation employing experiments or case studies is fundamental to assure that the measures

measure for OCL expressions within UML/OCL models	NAN is the total number of attributes referred trough navigations (rtN) in an OCL expression.		
Software attribute ($attr$)	Number of attributes rtN (a sub-characteristic of an OCL expression (OCL-E))	**Software entity** (p $\in$ P)	OCL-E $\in$ UOE
Output of distance-based process		**Underlying measurement theoretic constructs and formal definitions**	
M	abs_{NAN}: UOE $\rightarrow$ $\wp$(UAttrrtN): OCL-E $\rightarrow$ ArtN(OCL-E) **(remark 1)**		
abs	T$_e$ = t$_{0-NAN}$,t$_{1-NAssoc}$	• $\geq$	• $\geq_{NAN}$
T$_e$	where t$_{0-NAN}$ and t$_{1-NAN}$ are defined as: t$_{0-NAssoc}$: $\wp$(UAttrrtN) $\rightarrow$ $\wp$(UAttrrtN): s $\rightarrow$ s $\cup$ {a} t$_{1-NAssoc}$: $\wp$(UAttrrtN) $\rightarrow$ $\wp$(UAttrrtN): s $\rightarrow$ s - {a} with a $\in$ UAttrrtN **(remark 2)**	SAPS	($\wp$(UOE),• $\geq_{NAN}$)
δ	δ_{NAN}: $\wp$(UAttrrtN) $\times$ $\wp$(UAttrrtN) $\rightarrow$ $\Re$: (s, s') $\rightarrow$ (| s - s'| + | s' - s |)	MSAS	($\wp$(UOE),δ_{NAN})
ref	ref$_{NAN}$: UOE $\rightarrow$ $\wp$ (UAttrrtN): OCL-E $\rightarrow$ $\emptyset$	$ST_{abs(p),ref(p)}$	$ST_{SNAN(OCL-E),\emptyset}$
		Attribute definition	The number of attribute rtN of an OCL-E is the distance between the set of attributes rtN ArtN(OCL-E) and the empty set
δ	δ_{NAN}: UOE $\rightarrow$ $\Re$: OCL-E $\rightarrow$ |ArtN(OCL-E)| **(remark 3)**	Measure definition	The number of attributes rtN in an OCL-E is measured by the count of attributes of ArtN(OCL-E)
Remarks:			
(1) UAttrrtN represents the set of all the attributes rtN of an OCL-E (2) s is a set of attributes rtN, i.e. an element of $\wp$(UAttrrtN) (3) $\forall$ OCL-E $\in$ UOE: δ_{NAN}(ArtN(OCL-E), $\emptyset$) = |ArtN(OCL-E) - $\emptyset$| + | $\emptyset$ - ArtN(OCL-E)| = |ArtN(OCL-E)|			

Table 6.1: Distance-based Definition of NAN Measure

Measure	Abstract Function
NKW	abs_{NKW}: UOE → ℘(UKeywords): OCL-E → SKW(OCL-E) where UKeywords is the Universe of keywords [3] relevant to an UoD. SKW(OCL-E) ⊆ UKeywords is the set of keywords used in an OCL expression.
NES	abs_{NES}: UOE → ℘(UES): OCL-E → SES(OCL-E) where UES is the Universe of explicit occurrences of the contextual instance relevant to an UoD. SES(OCL-E) ⊆ UOE is the set of occurrences of self is used in an OCL expression.
NIS	abs_{NIS}: UOE → ℘(UIS): OCL-E → S(OCL-E) where UIS is the Universe of implicit occurrences of the contextual instance relevant to an UoD. SIS(OCL-E) ⊆ UOE is the set of occurrences of self is used in an OCL expression.
NBO	abs_{NBO}: UOE → ℘(UBO): OCL-E → SBO(OCL-E) where UBO is the Universe of boolean operators occurrences relevant to an UoD. SBO(OCL-E) ⊆ UOE is the set of boolean operators occurrences in an OCL expression.
NCO	abs_{NCO}: UOE → ℘(UCO): OCL-E → SCO(OCL-E) where UCO is the Universe of comparison operators relevant to an UoD. SCO(OCL-E) ⊆ UOE is the set of comparison operators occurrences in an OCL expression.
NEI	abs_{NEI}: UOE → ℘(UEI): OCL-E → SEI(OCL-E) where UEI is the Universe of explicit iterators relevant to an UoD. SEI(OCL-E) ⊆ UOE is the set of explicit iterators defined in an OCL expression.
NII	abs_{NII}: UOE → ℘(UII): OCL-E → SII(OCL-E) where UII is the Universe of implicit iterators occurrences relevant to an UoD. SII(OCL-E) ⊆ UOE is the set of implicit iterators occurrences in an OCL expression.
NAS	abs_{NAS}: UOE → ℘(UAS): OCL-E → SAS(OCL-E) where UAS is the Universe of attributes belonging to the classifier that *Self* represents, relevant to an UoD. SAS(OCL-E) ⊆ UOE is the set of attributes belonging to the classifier that *Self* represents in an OCL expression.

Table 6.2: Abstract Function for the Measures (Part 1)

Measure	Abstract Function
NOS	abs_{NOS}: UOE $\rightarrow \wp$(UOS): OCL-E $\rightarrow$ SOS(OCL-E) where UOS is the Universe of operations belonging to the classifier that *Self* represents, relevant to an UoD. SOS(OCL-E) $\subseteq$ UOE is the set of operations belonging to the classifier that *Self* represents, in an OCL expression.
NIO	abs_{NIO}: UOE $\rightarrow \wp$(UO): OCL-E $\rightarrow$ SIO(OCL-E) where UO is the Universe of number of occurrences of *oclIsTypeOf*, *oclIsKindOf* or *oclAsType* Operations relevant to an UoD. SIO(OCL-E) $\subseteq$ UOE is the set of number of times an *oclIsTypeOf*, *oclIsKindOf* or *oclAsType* operation is used in an OCL expression.
N@P	$\text{abs}_{N@P}$: UOE $\rightarrow \wp$(U@P): OCL-E $\rightarrow$ S@P(OCL-E) where U@P is the Universe of properties postfixed by @Pre, relevant to an UoD. S@P Number of properties postfixed by @Pre(OCL-E) $\subseteq$ UOE is the set of properties postfixed by @Pre in an OCL expression.
NNR	abs_{NNR}: UOE $\rightarrow \wp$(UNR): OCL-E $\rightarrow$ SNR(OCL-E) where UNR is the Universe of Navigated Relationships relevant to an UoD. SNR(OCL-E) $\subseteq$ UOE is the Set of Navigated Relationships in an OCL expression.
NNC	abs_{NNC}: UOE $\rightarrow \wp$(UNC): OCL-E $\rightarrow$ SNC(OCL-E) where UNC is the Universe of Navigated Classes relevant to an UoD. SNC(OCL-E) $\subseteq$ UOE is the Set of Navigated Classes within a OCL expression.
NPT	abs_{NPT}: UOE $\rightarrow \wp$(UPT): OCL-E $\rightarrow$ SPT(OCL-E) where UPT is the Universe of Parameters whose Types are classes relevant to an UoD. SPT(OCL-E) $\subseteq$ UOE is the set of Parameters whose Types are classes in the UML/OCL model where the OCL expression is defined.

Table 6.3: Abstract Function for the Measures (Part II)

Measure	Abstract Function
NUDTA	abs_{NUDTA}: UOE → ℘(UAODT): OCL-E → SAUDT(OCL-E) where UAOUDT is the Universe of attributes or operations belonging to a user-defined data type, used in any expression relevant to an UoD. SAUDT(OCL-E) ⊆ UOE is the Set of Attributes belonging to a User-defined Data Type used in an OCL expression.
NUDTO	$abs_{N...}$: UOE → ℘(UAODT): OCL-E → SOUDT(OCL-E) where UAOUDT is the Universe of attributes or operations belonging to a user-defined data type, used in any expression relevant to an UoD. SOUDT(OCL-E) ⊆ UOE is the Set of Attributes belonging to a User-defined Data Type used in an OCL expression.
DN	abs_{NDN}: UOE → ℘(UCN): OCL-E → SCN(OCL-E) where UCN is the Universe of set of Classes belonging to any navigation tree built for any OCL expression relevant to the UoD. SCN(OCL-E) ⊆ UOE is the set of Classes to which the OCL expression Navigates to.
NON	abs_{NON}: UOE → ℘(UON): OCL-E → SON(OCL-E) where UON is the Universe of referred Operations through Navigations, which are relevant to the UoD. SON(OCL-E) ⊆ UOE is the Set of referred Operations through Navigations in an OCL expression.
WNN	abs_{WNN}: UOE → ℘(UN): OCL-E → SN(OCL-E) where UN is the Universe of Navigations relevant to an UoD. SN(OCL-E) ⊆ UOE is the set of Navigations of an OCL expression.
WNCO	$abs_{N...}$: UOE → ℘(UCO): OCL-E → SCO(OCL-E) where UCO is the Universe of Collection Operations relevant to an UoD. SCO(OCL-E) ⊆ UOE is the Set of Collection Operations in an OCL expression.

Table 6.4: Abstract Function for the Measures (Part III)

Measure Classification	OCL Expression Measures		
	NKW, NES, NIS, NBO, NCO, NEI, NII, WNCO, WNN	NNR, NAN, NON, NNC, NUDTA, NUDTO, NAS, NOS, NIO, N@P, WNN, WNCO, NPT	DN
Interaction-based measures for coupling	×	√	×
Length	×	×	×
Size	√	×	√

Table 6.5: Theoretical Validation of Measures According to Briand et al.´s Frameworks [BMB99],[BMB96],[BMB97]

are really significant and useful in practice, and this is the subject of the following chapter.

Chapter 7

Psychological Explanation

Cognitive complexity is assumed to be the mechanism causing the effect of structural properties on software quality attribute [CES01], [GEMM00]. So, cognitive complexity of modelers dealing with software artifacts should be carefully considered to explain the rationale behind any defined measure.

This chapter describes the psychological explanation activity of the method for measure definition (explained in section 2.3.3):

- The selected Theory to use in a Plausible Explanation (Activity PE_1) was already explained in section 2.3.3.2.

- The relation of the Cognitive Theory to the Software Artifact and Measure (Activity PE_2) is included in section 7.1 of this chapter. Section 7.1 explains the different cognitive dimensions of OCL expressions comprehension according to cognitive and mental models theories, and also defines a mapping between the components of these selected models and the proposed measure.

- The activity of using qualitative methods to understand cognitive complexity (Activity PE_3) is applied in section 7.2. This section gives a plausible explanation of the main categories of the mental model of modelers when they comprehend OCL expressions through the use of verbal protocols.

Finally, section 7.3 describes the contribution to the dissertation.

7.1 Relate the Cognitive Theory to the Software Artifact and Measures (PE_2)

The relationships between the Cognitive Theory and the software artifact and measure is divided in the following parts:

- Section 7.1.1 describes the more important dimensions which affects the comprehension of OCL expressions.
- Section 7.1.2 details how the cognitive techniques of the cognitive model of Cant et al. [CJHS92] are used to explain how modelers deal with OCL expressions. It also shows the OCL concepts related to each technique.

7.1.1 Dimensions of OCL Expression Comprehension

We are conscious that the application of a mental model described in chapter 2 is not straightforward, due to the fact they were defined for program comprehension, and the OCL expressions are declarative without getting embroiled in implementations details [Ham99] as a program does. However, we consider that the mental models are generally enough to be applied to the comprehension of a declarative language like OCL. In fact, the essence of program comprehension is identifying artifacts, discovering relationships, and generating abstractions [TH03]. We believe that, whatever the language is, imperative or declarative, the comprehension of its specifications (or products) involve different cognitive dimensions, such as scope and direction of comprehension.

Before giving an analysis of the dimensions of the OCL expression comprehension we must recall that the main purpose of using OCL is the precise specifications of constraints on object-oriented models through its modeling artifacts [Ham99]. Now we describe the different dimensions which influence on OCL expression comprehension:

- **Type of Information:** During OCL comprehension modelers had to deal with two kinds of information: **textual information**, i.e. the OCL expressions itself written using ASCII code, and **graphical information**, the artifacts of the UML diagram which the expression constraints. Both, diagrams and OCL expressions, are necessary. Without OCL expressions, the model would be severely underspecified; without the UML diagrams, the OCL expressions would refer to non-existing model elements [1] [WK03]. In fact, as

[1] As there is no way in OCL to specify classes and associations

Hamie argues in [Ham99], OCL selects ideas from formal methods to combine with diagrammatic, object-oriented modelling.

Textual and diagrammatic or graphical information present different characteristics. Meanwhile diagrams have topological and geometric relations and the information is indexed according to location [LS87], the text used by a language as OCL consists mainly of signs and meaning. So, during the comprehension of OCL expressions, modelers should constantly switch between textual specification (OCL expressions) and graphical specification (UML diagram).

Graphical representations of software artifacts are often advocated as an effective means of aiding program understanding [MTH04], and we believe that they facilitate the comprehension of OCL expressions. Moreover, early research indicated that illustrations and text should be presented in proximity rather than separately [QaMKI04], otherwise they can contribute detrimentally to cognitive load.

Although the graphical representation as diagrammatic notations has became an important artifact as a medium of knowledge representation [HK99], the visual appearance of a diagram influences its level of computational efficiency and may thus produce different behavioral and performance outcomes [LS87]. Hahn and Kim present in [HK99] a theoretical framework and an empirical exploration of diagrammatic manipulation in the domain of systems analysis and design. We plan to study as a future work different cognitive aspects related to graphical issues.

- **Levels of abstraction:** During the OCL expression comprehension, the modelers deal with two different levels of abstractions: classes and objects. At a class diagram level we can think of a system as made up of classes and their associations, however at object-level perspective see the system as made up of objects linked to each other and interacting [Tor04]. The class-level of abstraction is used to describe features that apply to all instances, and are depicted in the UML class diagram facilitating to build a *situation model* and the comprehension of OCL expressions. However, the OCL expression refers to an object-level perspective, because a expression is declared for a particular object of the contextual type (the contextual instance) and its relationships with other objects. The object-level perspective is not depicted but should be part of the mental representation of the modeler. Nevertheless the UML diagrams contribute significantly to support the conformation of the mental representation of the object-level.

- **Direction of Comprehension:** It is likely that during middle phases of OCL expressions comprehension, modelers tend to utilize a *bottom up* direction, by reading the textual representation of the OCL expressions and then mentally

grouping these statements into higher level abstractions. While the modelers chunk the OCL expression they constantly switched to the graphical UML diagram to which the expression is associated with, looking for artifacts referred in the OCL expressions, i.e. rolename of relationships, methods, attributes, etc. However, in the last phases of OCL expression comprehension process the modelers increasingly use an *opportunistic* direction of comprehension. Nevertheless, the direction of comprehension can be influenced by the familiarity of the modelers with the class diagram. Modelers having a broad knowledge of the class diagrams are likely to use a bottom up direction.

In Table 7.1 we provide a mapping between the Burkhardt et al.´s model [BDW02] and the proposed measures. The analysis of this mapping, i.e. why each component of the Burkhardt et al.´s model is related to each measure is described here:

- **Problem Objects:** The main problem objects within an OCL expression are:
  - the contextual instance, measured by NES and NIS.
  - coupled objects of the problem domain which are obtained through navigations (measured by NNC).
  - coupled objects that are received as a parameter (measured by NPT).
  - any enumeration type is a special user-defined type which is often used as a type for attributes [WK03].
  - the iterators can be used as explicit reference to problem objects. Iterators are commonly specified in collection operations that loops over the collection [WK03].
    Attributes references in OCL can be interpreted as client-server component of Burkhardt, due to the fact Warmer and Kleppe [WK03] explain that any attribute reference in OCL needs to be mapped to the corresponding *get* operation when implementing OCL expressions. Nevertheless, we consider that object properties, such as methods and attributes, are considered as part of references of problem objects' properties. So, NAN, NON, NAS and NOS from the point of view they are attributes and operation references belong to this category.
- **Relationship between Problem Objects:** Relationships between problem objects are:
  - Any navigation within an OCL expression uses association-ends of UML relationships (measured by NNR, WNN and DN).
  - Inheritance relationships should be considered through OCL expression when certain operations are employed. This is measured by NIO measure.

  - **Reified Objects:** OCL Collections constitute reified objects, measured by WNCO measure.
  - **Elementary Operations:** Boolean and comparison operators are elementary operations (measured by NBO and NCO). OCL keywords are also part of this component (measured by NKW).

- **Scope of Comprehension:** We believe that modelers can take different approaches to comprehend an OCL expression. We think that a *systematic* strategy can be applied as well as an *as-needed* strategy. In a systematic strategy the modelers do attempt to understand the overall design that is associated to the OCL expression, whereas in a as-needed strategy the modelers focused on those UML artifacts mentioned in the OCL expressions. However we think that OCL expression *modifications* demand a wide scope of comprehension, and modelers should apply a more *systematic* strategy of the surrounding classes of the contextual type.

- **Experience and Knowledge:** The comprehension of OCL expressions likewise in program comprehension, depends on several factors, including one´s cognitive abilities and preferences, one´s familiarity with the application domain (LT-WM knowledge), and the set of support facilities provided by the software engineering environment. These factors often determine the approach taken to understanding a complex artifact [TH03].

7.1.2 The Application of the Cant et al.'s Model

This section explains the cognitive techniques applied by modelers during OCL comprehension, based on and aligned with the Cognitive Complexity Model of Cant et al. [CJHS92].

We believe that OCL expressions are key facilitators to the construction of chunks and also compound chunks. Moreover, the constraint declared by an OCL expression is often captured by the mnemonics of the OCL expression´s name[2], and this can help to associate high level concepts with program concepts[3].

In our cognitive model, we argue that when a modeler is primarily chunking an OCL expression within a UML/OCL model, there are dependencies that, to be resolved, require the modeler to perform a certain amount of tracing (in different directions) to find relevant features as rolenames, attributes, etc. Having found their features, modelers will once again chunk to comprehend it. Conversely, when modelers are primarily tracing, they will need to chunk to understand the effect of the identified

[2]All OCL expressions can be named as was described in chapter 2.

[3]So, we recommend to properly name OCL expressions as part of a practical guidelines.

Table 7.1: Measures for OCL Expressions within UML Models

MEASURE ACRONYM	**MEASURE DESCRIPTION**	**TEXT RELATION**	**MODEL**
NNC	# of Navigated Classes	Problem Objects	Situation Model
NES	# of Explicit Self	Problem Object	Situation Model
NIS	# of Implicit Self	Problem Object	Situation Model
N@P	# of properties postfixed by @ Pre	Problem Objects	Situation Model
NUDTA	# of User-Defined Data Type Attributes	Problem Objects	Situation Model
NUDTO	# of User-Defined Data Type Operations	Problem Objects	Situation Model
NNR	# of Navigated Relationships	Rel. between Problem Object	Situation Model
NAN	# of Attributes referred through Navigations	Problem Object	Situation Model
NON	# of Operations referred through Navigations	Problem Object	Situation Model
NPT	# of Parameters whose Types are classes defined in a class diagram	Rel. between Problem Object	Situation Model
WNCO	Weighted # of Collection Operations	Reified Objects	Situation Model
NIO	# of oclIsTypeOf, oclIsKindOf or oclAsType Operations	Rel. between Problem Object	Situation Model
NEI, NII	# of Explicit or Implicit Iterator variables	Rel. between Problem Object	Situation Model
WNN	Weighted # of Navigations	Rel. between Problem Object	Situation Model
DN	Depth of Navigations	Rel. between Problem Object	Situation Model
NAS	# of Attributes belonging to the classifier that Self represents	Problem Objects	Situation Model
NOS	# of Operations belonging to the classifier that Self represents	Problem Objects	Situation Model
NBO	# of Boolean Operators	Elementary Operation	Program Model
NCO	# of Comparison Operators	Elementary Operation	Program Model
NKW	# of OCL KeyWords	Elementary operations	Program Model

chunks. Moreover, as El-Eman recognize in [Ema02] a certain amount of coupled chunks do not affect cognitive burden, until a limit is exceeded and overflow short-term memory.

Many times the modelers should constantly switch between textual specification (OCL expressions) and graphical specification (UML diagram). The effects of chunking and tracing difficulty on complexity can be graphically modeled using a landscape model, as we shown in section 2.3.3 (see Figure 2.9).

While reading an upper-level chunk, a dependency requires that the modeler suspend reading of the original OCL expression because of the need to undertake tracing to fully understand the chunk currently being analysed. This is exemplified in the following:

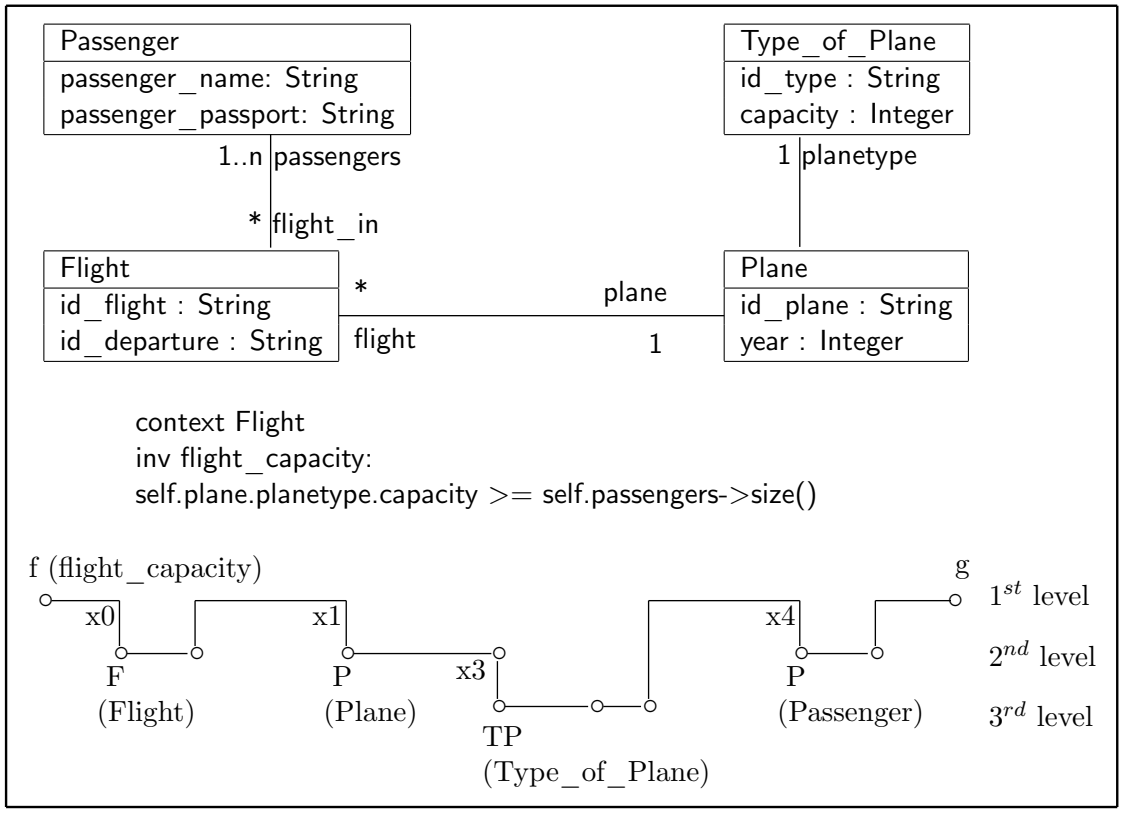


Figure 7.1: A Sample of a Landscape Model Modeled for an OCL Expression within a UML/OCL Model

Example 7.1.1 *The upper part of Figure 7.1 shows an UML/OCL combined model where an OCL expression had been defined for the Flight class, meaning that the quantity of passengers of a flight must be lower or equal to the capacity of the*

plane´s type of that flight. The landscape model for the OCL expression, named 'flight_capacity', is shown in the bottom part of Figure 7.1. Graphically at the top-level there is a single chunk visible, the OCL expression, delineated by the two markers (f, g). This chunk is interrupted by three lower-level chunks. The first interruption is common to every OCL expression where the modeler trace to locale the context of the expression (the Classifier written after the context keyword) within the UML diagram. The second interruption, depicted as the 'vertical drop' x1P *represents visually the work required in tracing the relevant features in the UML diagram, in this case, implies following a navigation from the Flight class to another class where its opposite-end rolename is defined as "plane", having found this class the modeler must chunk it, also chunk the cardinality associated to the mentioned rolename. Then, the modeler should follow a new navigation from Flight to Type_of_plane using the 'planetype' rolename, and after chunking the meaning of the latter class, the modeler should chunk one of its attributes, 'capacity'. The third and last interruption during the comprehension of the flight_capacity OCL expression is during the navigation (drop* x4P*) to the Passenger class, for obtaining the size of the set of passengers.*

As previously described the modeler should switch between textual and graphical information (between ASCII declarations -the OCL expressions- and UML diagrammatic notations) in order to fully capture the meaning of an OCL expression. For instance while reading a navigation in an OCL expression the modeler should follow the rolenames used in the UML diagram. Cant et al. [CHSJ94] consider in their cognitive model the **spatial distance** of tracing as a factor difficulty this cognitive technique, being this factor the distance, measured in program comprehension possibly through lines of code, between two chunks for which there is a dependency. The spatial distance of tracing in OCL expression is captured from the contextual type to the most distant coupled object through the DN measure.

We should also take into account that, in general, OCL expressions are not graphically attached to UML models, rather they are included in the underlying model repository [WK03], so the automated tool used to storage the model repository should facilitate the visualization of both the OCL expression and UML diagram at the same time otherwise the spatial distance could negatively influence the cognitive load.

7.1.2.1 OCL Concepts Related to Cognitive Techniques

The understanding of an OCL expression as a chunk involves a strong intertwining of tracing and chunking techniques. We believe that is important to understand which OCL concepts, specified in its metamodel [OMG03b], are relevant to each of these techniques.

- **Chunking**: As mentioned in the previous section, in order to describe the OCL concepts which involve chunking we have basically considered those concepts which belong to one expression (the chunk) and which do not require solving dependencies to other chunks. In this group we have included those OCL concepts that are intrinsic to the language itself. This group is related to the microstructures in the program model of Pennington´s mental model, or to the Syntactic knowledge of Shneiderman and Mayer´s cognitive model. The OCL concepts which this group involve are: OCL keywords, variable definitions, boolean operators and comparison operators.

- **Tracing**: The OCL concepts related to tracing techniques allow the modeler to write an expression using properties belonging to other classes or interfaces, different to the contextual type. With the purpose of analysing tracing, we have considered those OCL concepts that imply solving dependencies to other chunks. For example, when a modeler must interrupt the reading of an OCL expression in order to follow different parts of a UML diagram, such as the evaluation of a navigation using rolenames and its multiplicity, this is an activity related to tracing. Other instances (objects -as parameters- or object collections), whose types are different to the contextual type, are commonly accessed by tracing techniques. The concepts related to tracing are: Navigations, Parameters and Return Values, Messages and User Defined DataTypes.

- **Chunking & Tracing**: Any reference to the contextual type of the OCL expression, will be considered as part of the chunking & tracing cognitive process. The reason is the following: (1) as it was aforementioned, expression and diagrams are close related, and a constraint and its main constrained object should be considered as part of the same process, that is chunking; however, (2) the reference to an attribute or operation property of the contextual type, also constitute an interruption during comprehension of the OCL expression[4] and the modeler should trace to the classifier representing the contextual type, that is tracing.

  In other words, due to the fact that the OCL expressions are textual add-ons to UML class diagrams there should be OCL mechanisms for referring, for

[4]The spatial distance between the OCL expression and the contextual type is shorter than the distance between the OCL expression and any other coupled object' type due to at least no navigations is used in the connections of these entities

Table 7.2: OCL Concepts which Involve Tracing or Chunking

		OCL concept related to the cognitive technique	Common characteristics of the group
CHUNKING		Variable definitions, boolean operators, comparison operators	OCL facilities related to the language itself.
CHUNKING	TRACING	Reference to attributes or operations of the contextual instance, values postfixed by pre variables defined through << definition >> constraints.	OCL concepts related to the contextual instance and some of its properties, values before the execution of an operation (that is, properties postfixed by @ pre) of the contextual instance, variables defined through <<definition>> constraints in the type represented by the contextual instance, etc.
	TRACING	Navigation and collection operation, parameters whose type are classifiers predefined iterator variables, Messaging, etc	OCL concepts which allow an expression to use properties belonging to other classes or interfaces, different to the type of the contextual instance.

example, to class diagram elements. Implicitly an OCL expression is associated to a specific artifact (of a UML model) through the contextual instance *self*, because *self* is the main point of reference for the comprehension of the OCL expression, in this way *self* is used as the main carrier for chunking the OCL expression itself. However, each of the reference of the contextual instance´s properties involve tracing from the OCL expression to the contextual type in the UML model.

In this group, we have included those OCL concepts that allow one to refer to some properties of the contextual type: Attribute of the Contextual Type, Operations of the Contextual Type, <<definition>> Constraints, Predefined Properties, Values in Postconditions and User Defined DataType.

Table 7.2 shows a synopsis of the main OCL concepts involved in these cognitive techniques. whereas Table 7.3 show the cognitive technique(s) to which each measure is concerned.

7.2 Applying Qualitative Methods (PE_3)

We had used verbal protocol analysis, a qualitative method, whose underlying principle is that any verbalization produced by a subject whilst solving a problem -known

Table 7.3: Measures for OCL Expressions of UML/OCL Models

MEASURE ACRONYM	CHUNKING	TRACING	MEASURE DESCRIPTION
NNR		X	# of Navigated Relationships
NAN		X	# of Attributes referred through Navigations
NON		X	# of Operations referred through Navigations
NNC		X	# of Navigated Classes
NPT		X	# of Parameters whose Types are classes defined in a class diagram
NUDTA		X	# of User-Defined Data Type Attributes
NUDTO		X	# of User-Defined Data Type Operations
WNN		X	Weighted # of Navigations
DN		X	Depth of Navigations
WNCO		X	Weighted Number of Collection Operations
NEI, NII	X		# of Explicit or Implicit Iterator variables
NKW	X		# of OCL KeyWords
NES	X		# of Explicit Self
NIS	X		# of Implicit Self
NBO	X		# of Boolean Operators
NCO	X		# of Comparison Operators
NAS	X	X	# of Attributes belonging to the contextual type that Self represents
NOS	X	X	# of Operations belonging to the classifier
			that Self represents
NIO	X	X	# of oclIsTypeOf, oclIsKindOf or oclAsType Operations
N@P	X	X	# of properties postfixed by @ Pre

as concurrent 'think aloud'- will directly represents the contents of the subject's working memory [ES93]. Using the technique, a researcher can obtain an insight into the subject's cognitive process and use this to address a research question. For example, to investigate a subject's understanding of the problem space [HP03] or how the current state of the problem solution is evaluated, etc.

We have used verbal protocol analysis in order to give a plausible explanation of the main categories conforming the mental model of subjects when they deal with OCL expressions. This chapter is decomposed in:

- Section 7.2.1 explains the technique of verbal Protocols. The technique basically requires few special arrangements: recorded verbalizations are transcribed into protocols, the protocols are coded using preestablished coding scheme, and then analysed in accordance with the relevant research question [HP03].

- Section 7.2.2 describes a think aloud experiment where the verbal protocol analysis is applied. Subjects were asked to verbalize his/her thoughts whilst he/she is concurrently at work on a task which consists of comprehending three OCL expressions. His/her verbal behaviour forms the basic process data to be analyzed. The experiment was run at the Castilla La-Mancha University.

7.2.1 Overview of the Verbal Protocol Analysis Technique

Verbal protocol analysis has been applied to a range of problem types, and the data obtained has been used to create, confirm, or refute hypotheses across a wide range of research domains. A thorough review of the literature used to appraise the application of the verbal protocols analysis technique to software engineering can be obtained in [HP03].

Now we briefly summarize the stages of a verbal protocol, Figure 7.3 depicts the principal data used for the entire technique:

1. **Preparing and Running the session:** We shall now describe the practical procedures which must be applied in experiments where subjects are asked to think aloud:

   (a) **Setting:** The first thing to do when one wants to get a subject to think aloud is to make sure that the setting is such that the subject feels comfortable. The room should be quiet, a glass of water should be at hand, the chair should be comfortable. This recommendation are particularly important to remember when thinking aloud experiments are going to take quite some time and will be tiresome for the voice and throat of the subject [SBS94]. An explanation can be given about the purpose of the research, about what is going to happen and about the protection of the data. It is important to explain that the obtained data will be handled in strict confidentiality.

   (b) **Instructions:** The instruction related to thinking aloud should be quite simple. The essence of the instruction is: Perform the task and say out loud what comes into your mind. The instruction should be write down beforehand and should be read to the subject. An example of a instruction is: 'I will give you a problem. Please keep talking out loud while solving the problem'. Instruction should not be too long to avoid the situation of the subjects making up their own interpretations about what is requested from them.

   (c) **Warming up:** Most subjects need a little training before the real experiment starts. It is important to give the subject an opportunity to practice thinking aloud. In general it is wise to look for a task which

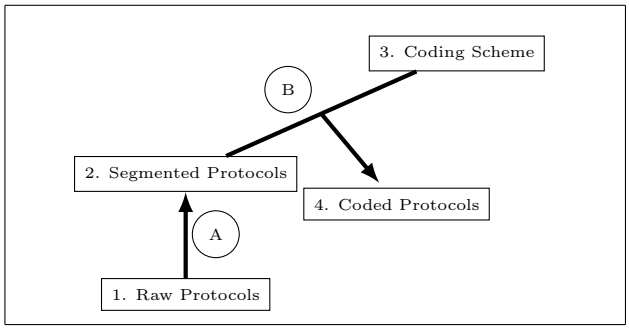


Figure 7.2: Data Transformation in Verbal Protocols

is not too different from the target task. Although most people do not have much difficulty rendering their thoughts, there are some subjects for whom this method does not work. This is the case when after a quarter of an hour a subject finds it hard to verbalize his thoughts. If this is the case, Someren et al. recommend to stop because is unlikely to provide useful protocols [SBS94]. The instruction for practice task is the same as for the main task.

(d) **Behaviour of the experimenter and prompting:** When the subject is working on the task, the role of the experimenter is a restrained one. Interference should only occur when it may be necessary to give the subject a short reminder to continue verbalizing if there is a period of silence. Then the experimenter should prompt the subject by simply saying: 'Please, think aloud' or 'Please, keep on talking'.

(e) **Recording:** The sessions are usually recorded on audio- and video-tape. It may be wise to include the instruction and practising phase, in order to be able to check afterwards whether the procedure was performed correctly.

2. **Transcribing the Protocol:** The actions of transcribing the verbalizations recorded during the session (depicted as Raw Protocols in Figure 7.3) are performed without any difficulty. However this task is tedious and time-consuming, because transcribing a protocol usually means typing it out in a manner which is as verbatim as possible. As Someren et al. [SBS94] remark the task of transcribing may take 10 times as much time as the original protocol, depending on the clarity of the protocol and the fluency of the subject. In psychological research, in principle everything may be relevant and therefore basically everything should be typed out [SBS94]. So, generally speaking, the typist should try to type it out as faithfully as possible. For instance the transcription should include:

- Utterance by the subject which have no bearing on the problem-solving process at all. For example, asking for a glass of water, remarks about any interruption during the session (someone came into the room), etc.
- Any subject's humming and hawing should by transcribed like 'Er, I er....'. Stammering as well should be typed out just as it occurred.
- Within the transcription special marks as '...' are conventionally used to note recognizable pauses and unusual silences between two words.

3. **Segmenting the Protocol:** The protocol is segmented, that is divided into segments (Figure 7.3, Activity A), each of which is given an identification number. Although there is no standard definition of a segment, most segmentation schemes are broadly similar, being based upon boundaries of phrases marked by pauses, an identifiable single unit such as a reference or assertion, etc. In general, the combination of these pauses and the linguistic structure provide a natural and general method to segment a think aloud protocol [SBS94]. In the analysis, segments are often combined into episodes. An episode is a sequence of segments that corresponds to a single element in the model. However if the task analysis and the psychological model are very detailed then certain elements of the psychological model may correspond directly to segments in the protocol.

4. **Constructing a Coding Scheme:** The coding scheme specifies how the elements of a cognitive model can be identified in the data and is based on our knowledge of the way in which cognitive process will be verbalized. Codes must be developed to correspond to a formalism which will be used to represent the knowledge [Chi97]. This scheme is useful when comparing the protocol and the psychological model. Every process or component in the psychological or cognitive model is stated as how we expect that these processes or components will appear in the protocols. For example, a model containing a guessing process produce that the coding scheme includes a coding category 'guessing' to this process, and one would expect to find statements in the protocol indicating a solution with a certain uncertainty. The subject will for example say: 'Maybe it is X', 'Could it be X?', etc. This would appear in a coding scheme as:

CODING PROCESS	Description
Guessing	'Maybe it is X' or 'Could it be X?'

Categories in the coding scheme can be described in general terms but it is usually very helpful to give some examples of prototypical statements for each category. If two or more categories are similar it helps to emphasize the difference.

There are some categories in the coding scheme which are not directly derived from the model. These are the verbalizations which are not covered by the model, but may still be anticipated in the protocols. Sometimes the content of these categories in task performance is not relevant, but the moment at which they occur is. For example:

(a) Talking about not-task related issues ('Oh, I must not forget to call my friend'),

(b) Evaluation of the task or task-situation at a meta-level ('It is tiring to talk so much'),

(c) Comments about oneself ('I am thirsty'),

(d) Silent periods. At times people will briefly stop verbalizing and they may be prompted to continue.

(e) Actions: The subject performs an action (for example, writes a note or manipulates a device).

The result of applying the coding scheme to a segmented protocol is a coded protocol (Figure 7.3, Activity B).

5. **Verifying intercoder reliability:** Whatever the origin of the coding scheme is, there should be confidence in the reliability of a scheme's application to a set of protocols. The consistency and reliability should be assessed, typically by means of a second encoder. The original encoding can be compared with that of the second encoder in order to obtain either the percentage agreement or the coefficient of agreement (Cohen's K). The accepted percentage figures are typically above 75% or above 0.8 K. The Kappa makes a correction for the correspondence that can be expected from the marginal frequencies. This is a conservative estimate of intercoder reliability [SBS94].

6. **Analysing the code patterns:** The encoded protocol is analysed in order to answer the research questions. The number and type of analyses used typically reflect the number and type of research questions being asked. Examples include: enumeration of specific categories, relative distribution of various activity categories identified, analysis of the pattern of activities, for example switching between two types of activity and even verbalization rates.

7.2.2 A Think Aloud Experiment

We have carried out a think aloud experiment through which to test a categorical model[5] of the cognitive process of modelers dealing with OCL expressions. The

[5]Descriptions of cognitive processes can take different forms. The most important forms are dimensional models, categorical models and procedural models.

categories concerned whether the utterances obtained by the students pertained to different coupling aspects: problem objects, relation between problem objects and reified objects. However we included other categories to describe cognitive factors such as: task planning, which involve how subject plan to tackle the comprehension process, the activity of chunking the whole expression, etc.

As we have previously explained the think aloud method involves the analysis of recorded verbal protocols that resulted from asking subjects to voice their thoughts when executing particular problem-solving tasks. Within the experiment the tasks we asked the subject to perform were to comprehend an OCL expression, and to express the meaning of the OCL expression. Expression consists of suitable short assertions that are not always easy to understand, specially when a lot of objects are coupled through the expression.

We carried out a thinking aloud experiment in order to test the following research questions:

- Does the modelers apply tracing and chunking cognitive techniques while they comprehend the OCL expression?
- Does the subjects make an attempt at a broad comprehension of the class diagram before the understanding of the OCL expression?
- Reified objects, the relation between objects and problem objects are the most important categories of the model of the subject' cognitive process when they deal with the OCL expression comprehension.

We gave a training course about OCL language where 10 subjects took part. The subjects were ten graduate and undergraduate students of the Department of Computer Science at the University of Castilla La-Mancha in Spain. We divided the course into three groups according to their experience, and then three (one from each group) were chosen to participate in the think aloud experiment. Their utterances were taped and video-recorded. The use of a relatively small number of subjects is typical of studies that collect and analyze verbal protocol data [HHC04]. Think-aloud protocols were collected while each subject comprehended each of the three tests. The three tests had different coupling complexity. The first test was the easiest test. The second and third tests present different kinds of difficulties. The difficulty of the second test was due to the use of different rolenames whereas the difficulty of the third was due to the use of collection operations. Nevertheless we believed that the last test would be more difficult to solve than the second test.

In order to set the experiment appropriately we took several factors into account such as: we chose a comfortable and quiet room, we provided the subject with a glass of water, etc. In order to start the experiment and to give the subject an

Table 7.4: Intercoder Agreement

	Subject 1			**Subject 2**			**Subject 3**		
	Protocol #			**Protocol #**			**Protocol #**		
	1	2	3	1	2	3	1	2	3
Percentage of Agreement	0.75	0.85	0.81	0.88	0.75	0.81	0.85	0.86	0.85
Coehn' K (Kappa)	0.75	0.85	0.81	0.86	0.69	0.76	0.82	0.83	0.81

opportunity to practice thinking aloud, a warming up session took place before the real experiment.

Participants verbalizations were transcribed, and nine protocols were obtained, three protocols from each subject. A researcher checked each transcription against the video tape in order to make any necessary corrections or to adjust phrasing. The raw protocols were then broken up into segments which represented the subject' utterances. Once the verbal protocols were segmented, they were ready to be coded.

These coding categories consisted of whether the explanations and utterances used by the subjects pertained to reified objects (RO), the Relation between problem objects (RBPO), the Problem Objects (PO), etc. Table 7.5 shows the coding categories and a description of each one.

To control the coding reliability, each verbal protocol was coded by two independent coders. Table 7.4 shows the Coehn' K across the nine protocols. As we explained in the previous section the values of Coehn' K are acceptable due to the fact that they are greater than 0.8 K.

The RO category was coded as being those utterances which were concerned with the collection of objects, the collection operation and explicit iterator of collections. Within the RBPO category we included related problem objects such as the understanding of a relationship within the class diagram or the use of OCL navigations. Within the PO category concepts such as the contextual type, the contextual objects of the expression (or even their attributes) were included.

The SP category was used when the subjects explanations were related to a statement of the problem whereas the TP category was related to how the subjects planned to perform the task. The TE category was used when the subject explained the OCL expression according to its type (all the expression in the experiment were invariant expressions). The CWE category was used when the subject chunked the whole OCL expression and described its meaning. We also used special coding categories (SCC categories) to code some utterance that were not covered by the model as we have described in the previous section (see the last five entries of Table 7.5.

The number of utterances ranged from 20 to 46. Table 7.6 contains a coded protocol from a subject to illustrate the coding. The utterances belong to a warm up session

Table 7.5: Coded Categories

Coding Categories	Description
RO-CO	Reified Object: collection manipulation, collection operations, etc.
RO-EI	Reified Object: iterator variable within a collection operation.
RO-LOO	Looking for the definition of an operation collection in a list of operations
RBPO-CD	Relation between problem object: after reading the class diagram
RBPO-NR	Relation between problem object: Navigated Relationship
PO-CD	Problem Object(s) after reading the class diagram
PO-CC	Problem Object(s) a concept of a class from the OCL expression
PO-ACO	Problem Object(s): attribute belonging to contextual objects
PO-CA	Context Analysis and evaluation
EO	Elementary Operation
TE	Type of Expression
CWE	Chunking the whole OCL expression
TP	Task Planning
SP	Statement of Problem
SCC-NT	Special coding categories: Talking about not task related issues
SCC-EM	Special coding categories: Evaluation of the task or task-situation at a meta-level
SCC-CO	Special coding categories: Comments on oneself
SCC-SP	Special coding categories: Silent periods.
SCC-A	Special coding categories: Actions

using the expression and the class diagram of Figure 7.3.

We did not aggregate the protocol segments into episodes because the grain size of segments is suitable for testing the hypotheses. Although we coded all the segments according the coding categories of Table 7.3 we were only interested in RO, RBPO, PO categories that are related to coupling concepts.

Although we are only interested in some specific process in the protocols, and in this case it may be more efficient to encode protocols directly from the audio-tape (i.e. direct encoding) instead of transcribing and coding the transcription it is often not a method to be recommended [SBS94].

The rest of the categories were used to describe a general process of comprehension used by the subjects. Special coding categories were also included because these remarks might be an indication of the level of difficulty of a sub-task or of the cognitive load of the subject.

7.2.2.1 Analysis

The encoded protocols were analysed to answer the research questions. We performed an analysis of each subject performing the tasks. For each task of each

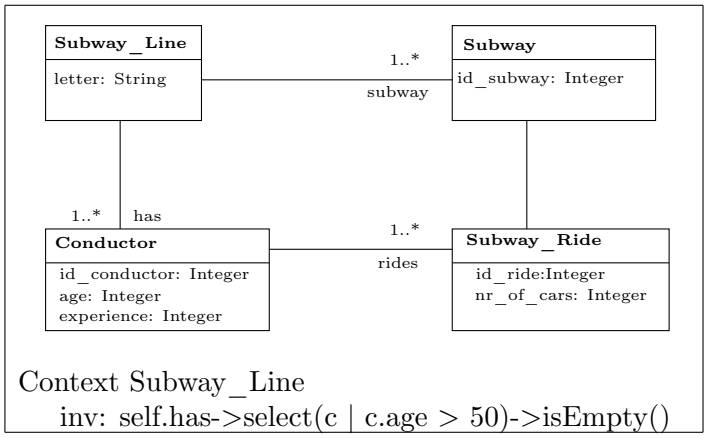


Figure 7.3: Sample of an OCL Expression used in a Warm Up Session

subject we depicted the coded protocols. Two types of figures were obtained to facilitate the analysis. First, the sequence of coded utterances was represented in a graph that summarizes each subject's protocol. In the graph, each dot represents a subject' utterance and the coding categories assigned to the utterance. Second, we depict the number of related coded protocols for each protocol. Second, In the following we give the analysis for each subject. General conclusions are described in the next subsection.

7.2.2.1.1 Subject 1 The number of utterances of each protocol obtained from subject 1 were: 22 (protocol 1), 20 (protocol 2), 33 (protocol 3). The time (in minutes and seconds) spent in each protocol was: 02:16, 02:24, 5:36.

From Figure 7.4, a graph of the subject' protocols, we can observe that the subject does not attempt to understand the class diagram before starting to comprehend the OCL expression.

The technique applied by subject 1 was the same in the tree protocols: after reading the statement of problem (SP) he analyze the context of the expression and started to understand its meaning. Nevertheless, he uses the class diagram only to focus on those relationships and classes (and attributes) of the class diagrams when they appear within the OCL expression. We believe that the subject takes an as-needed strategy of the class diagram. So, the understanding of the expression and the class diagrams is product of an intertwining activity. The OCL expression is comprehended from left to right, the subject started to capture the meaning obtaining different assertions or utterance of the meaning, until the subject chunk the whole expression. After the meaning is obtained the subject review the expression to be sure of the meaning. In several segments of protocol it is clear that the subject

uses many expression such as 'now we are in the X class', 'now we are going to Y class', etc. to reveal the OO-perspective taken by the subject in understanding the expression. This cognitive perspective is similar to the identification of deictic words such as *here* and *there* in the work of Hutchis et al. to reveal the perspective taken by the subject in solving the missionary and cannibals problem [HL81].

Table 7.6: Example of a Verbal Protocol from a Pilot Subject

Id	Code	Utterance
1	SP	... what I'm going to do first is take a look at the class diagram
2	PO-CD	Subway Line ...
3	PO-ACO	I guess the identification would be to say which is the line
4	RBPO-CD	Associated with each subway line there are many subway train, one or more
5	PO-ACO	Yes, it is correct, the diagram show the year in which the train was bought...
6	PO-ACO	An identification for the subway train and another for the motor...
7	RBPO-CD	A subway train operates on more than one subway line ....
8	RBPO-CD	Of course, in a subway line there are many subway trains
9	RBPO-CD	A subway train can do many trip
10	RBPO-CD	Each trip with a number of coachs
11	RBPO-CD	Err ...a trip is conducted obviously by a real conductor, and a real subway train
12	RBPO-CD	The subway line has many conductors
13	RBPO-CD	A conductor drives in many subways lines
14	PO-ACO	A conductor has an identification, a age, and year of experience
15	RBPO-CD	A conductor drives in different trip
16	RBPO-CD	Within each trip there is no information about how many conductors there are?
17	TP	Well, having analysed the diagram I will start to understand the expression
18	TE	The context is Subway Line, an invariant expression
19	PO-CA	That means that the expression represents a constraint that always must be true
20	RBPO-NR	Self dot tiene (self.tiene) is talking about the drivers that the subway line has,
21	RO-CO	then we have a selection with a c variable of conductor
22	RO-EI	that means that c select the conductors of each subway line
23	RO-CO	c dot edad (c.edad) greater than fifty, that is select the conductors which have an age greater than fifty, and then the boolean operation 'is empty' is applied
24	EO	that is the operation should be true
25	CHU	The meaning of the expression is ... [writing] each subway line should verify that ...
26	CHU	[writing] none of its associated drivers are more than fifty years old.
27	SCC-NT	I've finished...

A blue dashed rectangle in Figure 7.4 was depicted to highlight the RO, RBPO, PO and EO coded utterances during the process of the OCL expression comprehension. As we can see few dots remain outside the limit of blue rectangle. During task 3

he had doubts about the meaning of a collection operation that appear in the OCL expression and he spent a time using the list of collection operations in order to look for the meaning. In task 3 the subject made different intentions of chunking the whole expression before giving the final result.

The composition of the coded utterances for each protocol is depicted in Table 7.7. The coded utterances related to coupling were grouped into three group RO, RBPO and PO (reified objects, problem objects and the relationship between them). We also grouped the special coding categories codes in the SCC group. TP and SP were depicted together due to the fact that they refer to the problem or the the way the subject plan to tackle it.

From Table 7.7 we see that the quantity of coupling coded categories increases at the same time as the import-coupling within the expression increases. Whether we subtract from the total number of utterance (of each protocol) the number of SCC coded utterances, which are special coded categories not relevant from the model itself, the coupling coding categories (RO, RBPO, PO and EO) represents the 0.65%, 0.77%, and the 0.83% of protocol 1, 2 and 3 respectively. That means that coupling coding categories seems to be a significant proportion of the cognitive process of subject 1. We conclude that RO category represents a significant proportion of the utterance in the three tasks; the quantity of RBPO and PO coding categories is higher in the last two protocols.

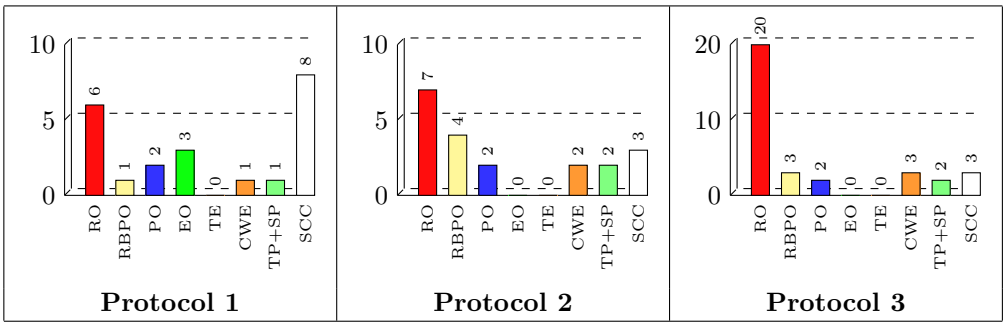


Table 7.7: Coding Categories from Subject 1

7.2.2.1.2 Subject 2 The number of utterances of each protocol obtained from subject 2 are: 26 (protocol 1), 45 (protocol 2), 36 (protocol 3). The time (in minutes and seconds) spent in each protocol (1, 2 and 3) was: 02:52, 08:04, 05:55. The fact that in protocol 2 the subject spent the longest time understanding a expression is depicted in the graph of the subject' protocols (see Figure 7.5). We believe that the second test was more difficult to verbalize for the subject than the third one.

From Figure 7.5 we can also observe that the subject attempt to understand the class diagram before starting to comprehend the OCL expression. The same tech-

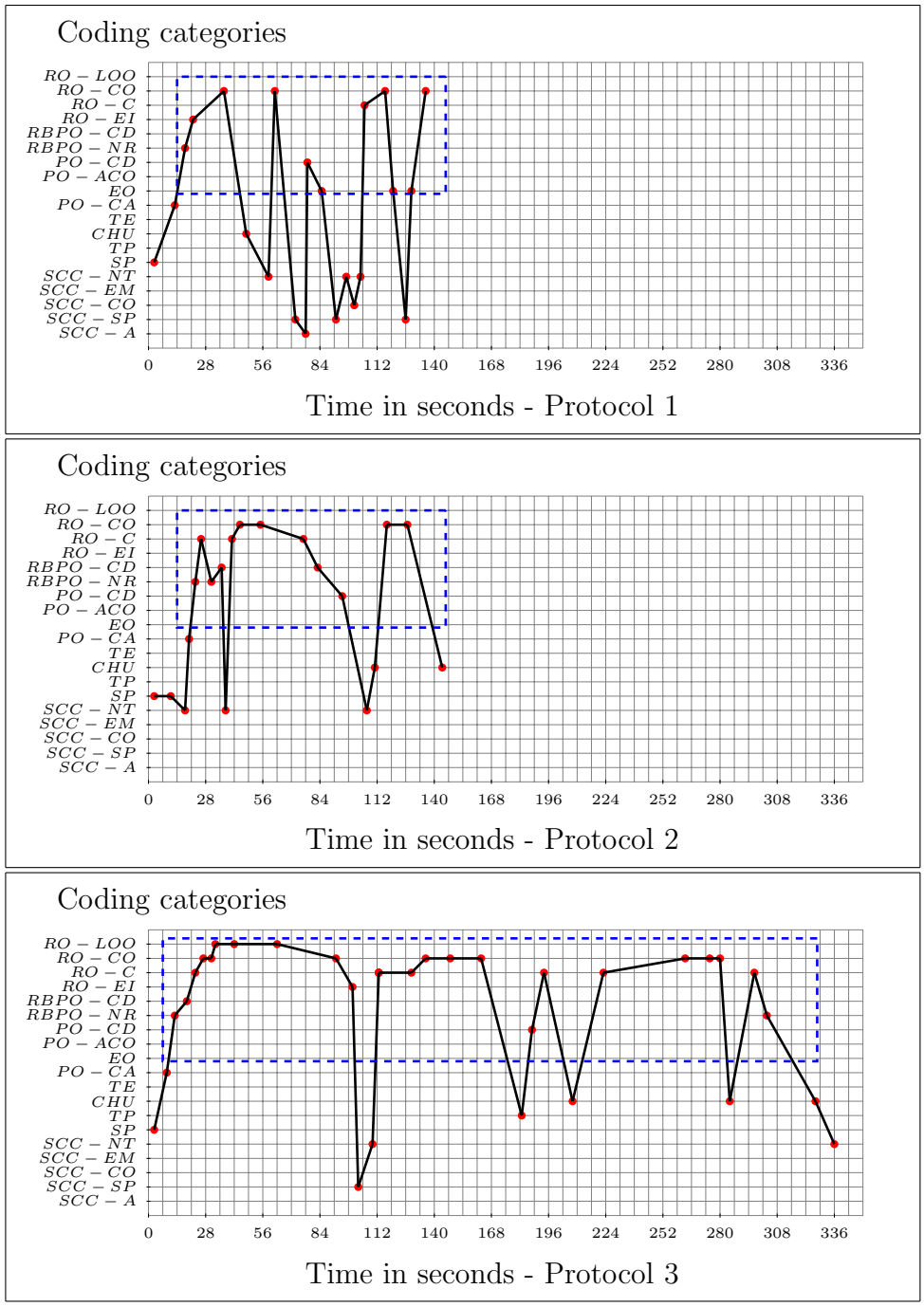


Figure 7.4: Coded Protocols of Subject 1

nique was applied win the tree protocols: the subject focuses on the relationship between the classes while chunking the diagram. Moreover, in the last two protocols the subject look at the contextual type of the expression in order to start to see the relationships that this contextual type has in the diagram before starting to comprehend the expression. We depicted this regular situation with a the green rectangles which contains all the coded utterances that corresponds to the comprehension of the class diagram. The subject only focused on relations between classes, he said nothing about the classes' attributes.

A blue dashed rectangle in Figure 7.4 was depicted to highlight the RO, RBPO, PO and EO coded utterances during the process of the OCL expression comprehension.

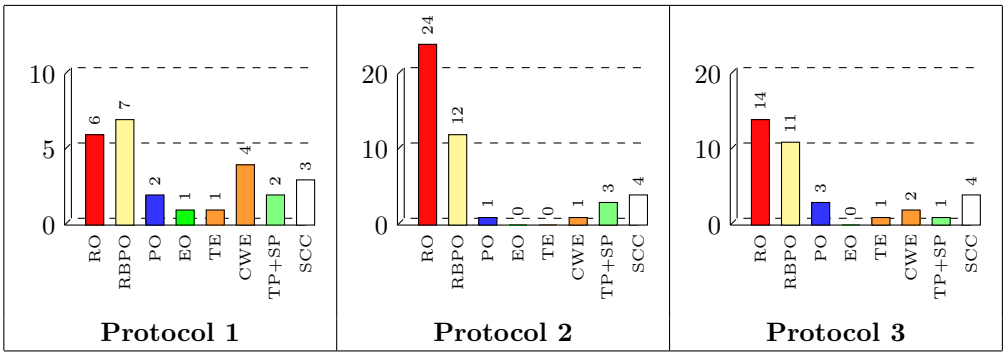


Table 7.8: Coding Categories from Subject 2

The composition of the coded utterances is shown in Table 7.8. From Table 7.8 it is clear that RO and RBPO coding categories are a significant portions in protocol 2 and 3 respectively, representing almost half of the coupling categories. We observe that the quantity of coupling coded categories is higher in protocol 2 than protocol 3. Nevertheless, its ratio (number of coupling categories divided by the number of the protocol utterance) are 0.82 and 0.77. If we do not consider the SCC coded utterance from the total number of protocol utterance, this ratio is 0.90 and 0.87 respectively. The ratio of protocol 1 is 0.58 (considering the total number of utterances) and 0.66 (without the consideration of SCC coded protocols). That means that coupling coding categories seem to be a significant proportion of the cognitive process of subject 2.

7.2.2.1.3 Subject 3 The number of utterances of each protocol are: 28 (protocol 1), 43 (protocol 2), 47 (protocol 3). These quantities include SCC coding categories. The time (in minutes and seconds) spent in each protocol (1, 2 and 3) was: 03:07, 06:41, 06:27. From Figure 7.7 we can also observe that the subject does attempt a broad understanding of the class diagram before starting to comprehend the OCL expression. The subject applied the same technique in the tree protocols:

he focused on the relationship between the classes while reading the diagram, he

also focused on the classes involved and their attributes. We depicted this regular situation with a the green rectangle including the coded utterances related to the understanding of the relationship between classes in the diagrams as well as the understanding of the classes and their attributes.

The composition of the coded utterances for each protocol is depicted in Table 7.2.2.1.3. The ratio of coupling coding categories from the total number of utterance are 0.79, 0.79 and 0.83. Similar values are obtained if we subtract from the total number of utterances the SCC coded protocols: 0.81, 0.88, 0.89. That means that coupling coding categories seem to be a significant proportion of the cognitive process of subject 3. The quantity of RBPO category is similar from protocol 2 and 3, nonetheless the quantity of PO is slightly higher from protocol 2 than 3 (it should be remembered that the second protocol uses a expression where the complexity relies in using different rolenames). The third protocol according to the complexity of the third OCL expression demands more RO utterances from the subjects than the first and second protocols.

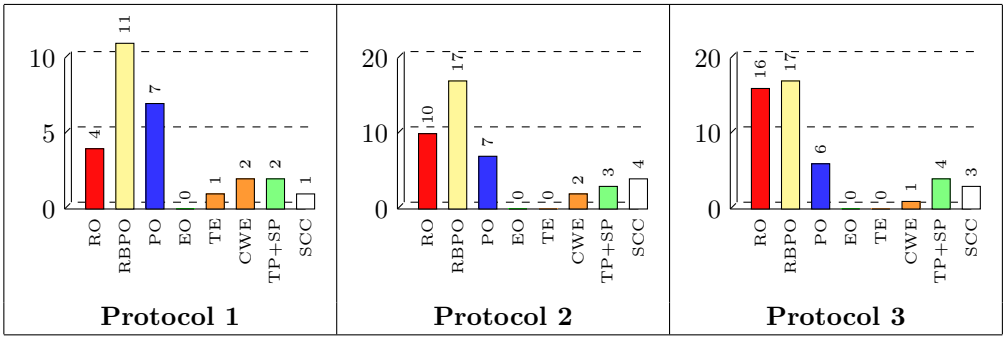


Table 7.9: Coding Categories from Subject 3

7.2.2.2 Results of the Verbal Protocol

According to each hypothesis, our conclusion are:

1. The scope of comprehension of the class diagram taken by the subjects was different from each other. Nevertheless, the technique that each subject applied was the same in the three protocols. Subject 1 could understand the OCL expression correctly without much understanding of the class diagram before he started to comprehend the OCL expression. However subject 2 focuses on the relationship between classes in the diagram before he attempt to comprehend the OCL expression, whereas the breadth of familiarity with the class diagram of subject 3 is the widest. Subject 3 focussed on relationships, classes and attributes before he started to comprehend the diagram. This difference was depicted using a green rectangle in the graph of the coded utterances (see Figures 7.4 and 7.5). According to the situation of subject 1 the green rectangle is missing from Figure 7.7 due to the fact that there is no comprehension of the class diagram at all before the subject started to chunk the expression.

2. Tracing and chunking techniques are applied during the comprehension of OCL expression. Nonetheless, it is not possible to produce a landscape model from the verbal protocol data. In order to model a landscape model we need to use a coarser-grain information during comprehension such us that likened to tracking eye movements from the UML diagram to the OCL expression (and the other way around). The eye fixations can support us to indicate what information is being heeded at any moment in time [Chi97]. Nevertheless, the verbal protocol provides us with empirical evidence that tracing and chunking are applied:

   - During the comprehension of the OCL expression the subject traces to the class diagrams, even those subjects who before chunking the expression take the more systematic comprehension of the class diagram (for example subject 3). This is confirmed through different verbal utterances related to the relationships (RBPO-CD), problem objects (PO-CD) which includes aspects of multiplicity of relationships, etc.
   - From the recorded video it is possible to visualize that sometimes the subject follow some navigation using a pencil which is moved over between classes of diagram.

3. RO, RBPO and PO are significant part of the mental model of subjects when they deal with the OCL expression comprehension. This was measured according to the ratio of coupling coded utterances by the number of utterances of each protocols. The range of the ratio for all the subjects were between 0.7 and 0.9.

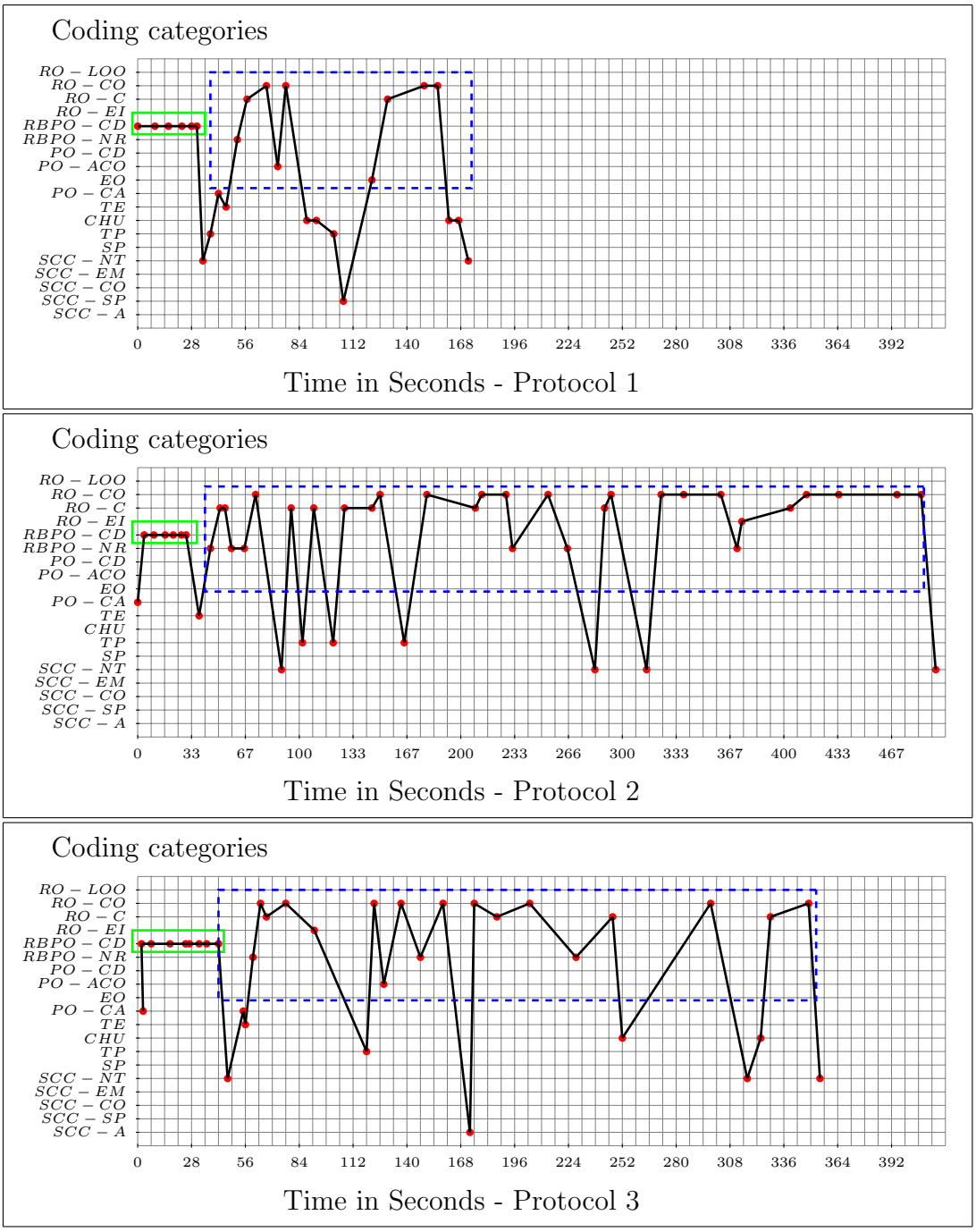


Figure 7.5: Coded Protocols of Subject 2

7.3 Contribution to the Dissertation

In this chapter we focused on a psychological explanation of how the modelers deal with OCL expressions. We showed that is possible to reason about the OCL expression comprehension according to the underlying theory of cognitive and mental models.

The most important conclusions are:

- Tracing and chunking cognitive process are concurrently applied during OCL expression comprehension. We obtained empirical evidence of their application through a thinking aloud protocol. We described which proposed measures are related to each cognitive technique in turn.
- Landscape models were used to explain how the cognitive process of tracing and chunking are ideally applied, however landscape models can not be depicted from verbal protocol data due to the fact that a more coarser grained information is needed such us eye movements from OCL expressions to class diagrams or the other way around.
- Preliminary findings from a thinking aloud experiment show that subjects take different scope of comprehension of the class diagram before starting to comprehend an OCL expression. The scope varies in a continuous that runs from those subjects who did not attempt to comprehend the diagram at all to those who attempt to systematically comprehend the diagram before chunking the expression.
- Due to the fact that recently a number of studies [BDW98b], [BDW02], [CW99b], [CW00] have tested elements of mental model structure supporting the evidence of a separate situation and program model we focus on the Burkhardt mental model and we described its main structural components [GEMM00]. We provide a mapping between the proposed measures and the structural component of the Burkhardt model.
- We obtain empirical evidence that problem objects, relations between problem objects and reified objects are a significant part of the mental model of subjects when they deal with the OCL expression comprehension.

We think that OCL expression *modifications* demands a wide scope of comprehension, and also a high *cognitive flexibility* from the modelers [CMF04], and they should apply a *systematic* comprehension strategy of the surrounding coupled objects to the contextual type in order to modify the OCL expression. However, we think that we must obtain empirical evidence of this, and we plan to tackle this issue in a future work.

Broadly speaking, we believe that an OCL expression is a key facilitator to the construction of chunks and also compound chunks, whereas OCL navigation is a key facilitator for tracing. OCL navigations help the modeler to collect information about the coupled objects through the expression, and as Davis suggest [Dav95] the information gathering process is significant in forming a mental representation (Figure 7.6).

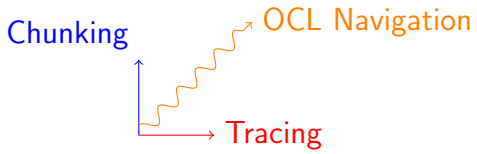


Figure 7.6: Tracing and Chunking Techniques

We hypothesize that the ensuring disruption of chunking OCL expression in order to trace to coupled UML artifacts, leads to comprehension difficulties, and therefore to a low maintainability. Furthermore, one can argue that many interacting and coupled objects could overflow short-term memory, would influence maintainability. The interaction of coupled object is usually manipulated though reified objects such as collection operations and its operations. So import-coupling seems to affect the cognitive complexity and external quality attributes of the product. Nevertheless, more empirical evidence is crucial to support this belief.

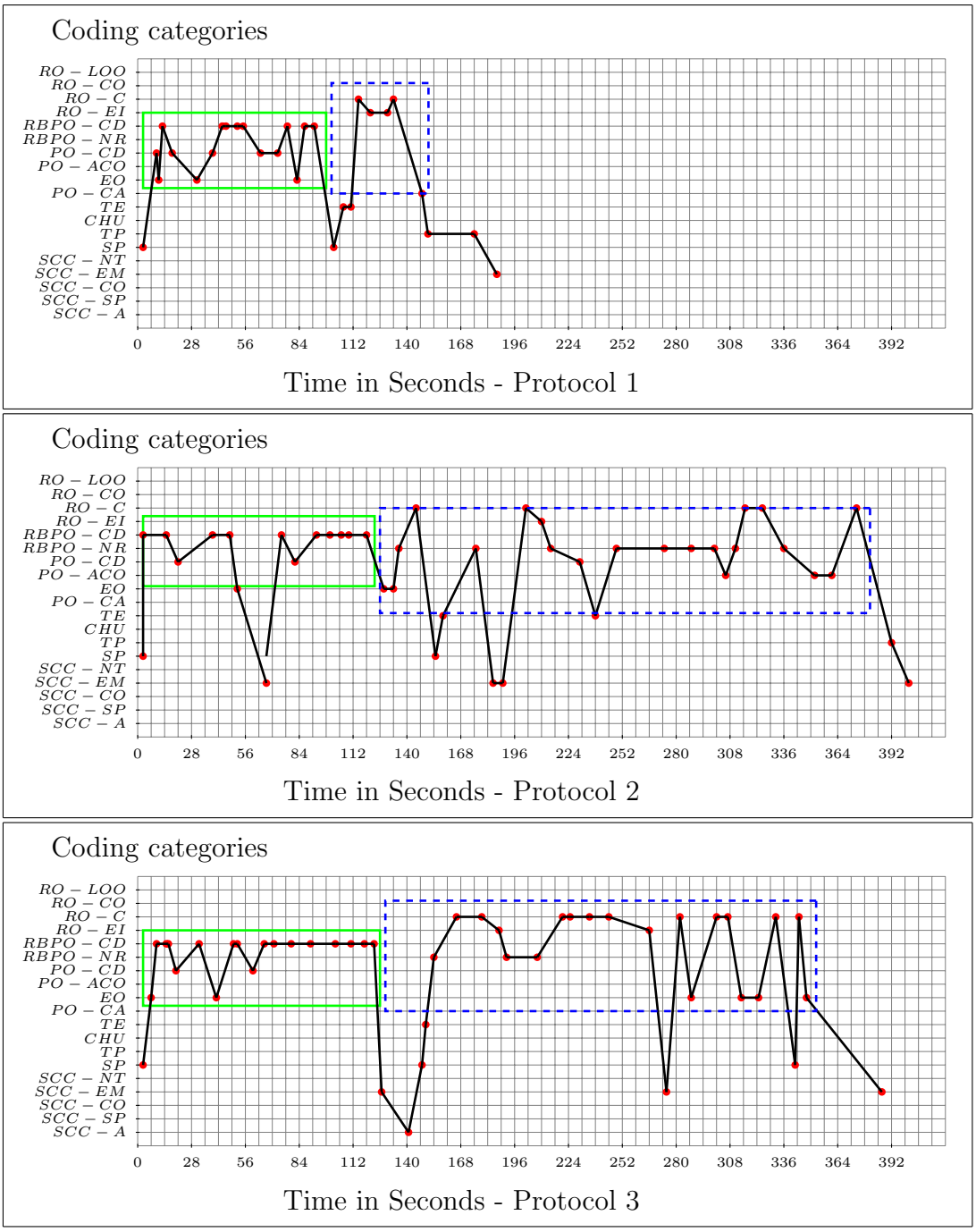


Figure 7.7: Coded Protocols of Subject 3

Chapter 8

Empirical Validation

Product measures are of little value by themselves unless there is empirical evidence that they are associated with external attributes [BDW99]. So, in this chapter we describe nine experiments we have carried out to empirically validate the proposed measures as early comprehensibility and modifiability indicators of OCL expressions within UML/OCL models.

Due to experiments can be viewed as part of common families of studies, rather than being isolated events, eight of the experiments presented in this chapter are part of three families of experiments. Figure 8.1 shows a chronological schedule of the conducted experiments. Building up a body of knowledge from families of experiments has many benefits for software engineering (see section 2.3.4.1). Moreover, common families of studies can contribute to important and relevant hypothesis that may not be suggested by individual experiments.

We have followed some recommendations provided by Ciolkowski et al. [CSB02] on how to perform family of experiments and the experimental process of Wohlin et al. [WRH$^+$00] on conducting controlled experiments. Nevertheless we also take into account the experience and theory of Briand et al. [BAC$^+$99], Perry et al. [PPV00] and Juristo and Moreno [JM01].

The structure of the chapter is the following: next section, section 8.1, describes the most important aspects related to the experimental objects and experimental subjects we stated before running any experiment. Section 8.2 includes the experimental process of a first experiment whereas each of the last sections, 8.3, 8.4, 8.5 describe three families of experiments, respectively. Finally, section 8.6 includes conclusions as part of the contribution to the dissertation. All the experiments described in this chapter were run applying the activities of the method for measure definition described in chapter 2. So, we titled the main sections accordingly to the activity they represent and we included the activity number between parenthesis as a reference.

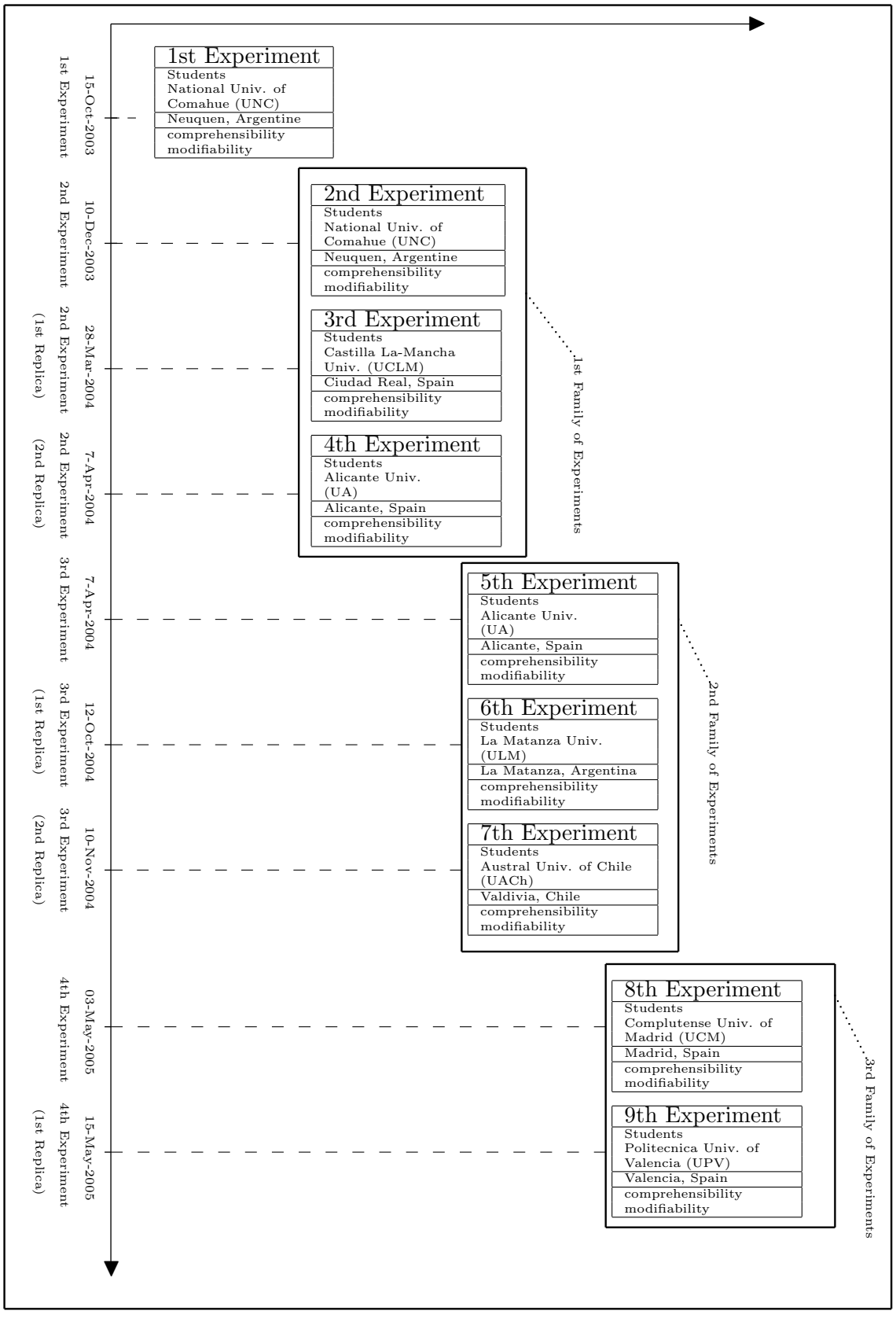


Figure 8.1: Overview of the Experiments Conducted

8.1 How the Experiments were Conducted

The design of experiments for OCL expressions within UML models is a challenging activity due to many factors arise:

- **We focus on OCL expressions:** OCL is not a stand-alone language [CBC05] and must be used along with UML diagrams. We used UML class diagrams as the experimental material where OCL expressions were attached, and we took into account that the complexity of these provided models do not bias our results. So, we tried to see that the designed UML diagrams were not confusing, cluttered, and unwieldy [MM04]. Regarding the modeling of clear UML diagrams we included only four to five classes. We adopt this limit as a cognitive threshold of unfamiliar concepts without affecting cognitive burden, taking into account the studies of Broadbent [Bro75] and Miller [Mil56].

- **Type of OCL expressions used:** Two of the most important OCL expressions used in class diagrams are invariants and pre- or post-conditions. The former are attached to classes, the latter are associated to methods. However as Briand et al. argue [BWL99] in OO system often methods just invoke other methods, thus passing through the request to another class, and we have good reasons to assume that a good deal of the cognitive complexity of an OO system lies in the way the system objects collaborate, and less in the implementation of individual methods. Moreover, from the coupling measures surveyed in the literature (see chapter 3) mostly were defined at the class level [BWL99]. So, we decided to focus on expressions attached to classes, i.e. on OCL expression invariants. Furthermore, many research studies had argue the importance of constraints [SSR04], [VS02], [VS02] as invariant expressions.

- **Experimental context and subject training:** Most of the times it is extremely difficult to access to professional subjects, due to the important cost in time, effort and resources. That is mainly why researchers carry out their studies with students in academic environments. Under certain circumstances, the differences between the students and the professionals are not very important, provided that the tasks to perform do not require an excessive experience, sometimes empirical experiments with students shows many advantage (see [Ver05], [DBM+96]). However, as Briand et al. argue [BLYP04], in general students are probably better trained at OCL and software modeling with the UML than most software professionals. So, regarding external validity we did not evaluate if the subjects are representative of software professionals. So, our families of experiments were launched off-line (not in an industrial software development environment).

  Nevertheless, in many universities where we run experiments the students were not prepared enough to perform our experiment, and in other cases students

had no training at all in OCL. Actually, OCL will rapidly become a standard part of many Computer Science curricula, nevertheless we found that the training session for student was an important aspect of our experiments planning and should be carefully evaluated. The training was tailored specifically for the experiments, ranging between 5, 10 and 20 hours. Despite this, we think that the fact that students have little (or none) pre-existing knowledge of OCL could improve the assimilation of concepts [RW02], but maturation process could not be obtained appropriately in those reduced training session (for instance, at a session of 5 hours). In some experiments we could not extend the training session because we were offered to use only this period of time. In one case we cancelled a replica of an experiment we plan to do (that was the case of the course in the JCC Jornadas Chilenas de Computación in Chile in 2005) because the training session was less than 5 hours and we only teach a course. Nonetheless we think that we had good opportunities to do replicas of the experiments in different universities and many researchers and teachers had collaborated with the purpose of our experimentation.

8.2 A First Experiment

We were conscious that it is impossible to conduct an experiment in which all the OCL concepts being used at the same time in expressions. In fact, it is also impractical because such expressions would be quite long probably having an unwieldy structure (with *if* expression, *let* expression, etc) and operating in a large context. OCL expressions tend to be short, and use different portions of the set of OCL concepts. So, we selected the more commonly used OCL concepts from the more relevant literature about OCL.

The first experiment was designed with the intention to obtain an exploratory explanation of the influence of coupling, measured by a set of OCL expression measures, on comprehensibility and modifiability. Through this experiment we were also interested to see in this influence if the measures related to tracing -which was described as a fundamental activity in comprehension- had a more important contribution rather than chunking measures.

8.2.1 Definition (EF_1)

Using the GQM template for goal definition, the goal pursued in this experiment is shown in Table 8.1.

Analyze	measures for OCL expressions within UML/OCL models
for the purpose of	Evaluating
with respect to	The capability to be used as comprehensibility and modifiability indicators of OCL expressions
from the point of view of	OO Software modelers
in the context of	Undergraduate Computer Science enrolled in a course related to OCL, of the Department of Computer Science at the National University of Comahue

Table 8.1: Goal of the First Experiment

8.2.2 Planning (EF_2)

After the definition of the experiment, the planning phase took place. It prepares for how the experiment is conducted, including the following six steps:

1. **Context selection**. The context of the experiment is a group of undergraduate students who had agreed to take part in a course on OCL, and hence the experiment is run off-line. The subjects were twenty-nine students enrolled in the third and fourth-year of Computer Science at the Department of Computer Science at the National University of Comahue in Argentina.

   The experiment is specific since it is focused on twelve measures for OCL expression within UML/OCL combined models. The experiment addresses a real problem, i.e., which indicators can be used for the comprehensibility and modifiability of OCL expressions? With this end in view it investigates the relationship between measures and the time spent on comprehensibility and modifiability tasks.

2. **Selection of subjects**. According to [WRH+00] we have applied a probability sampling technique: a convenience sampling. The nearest subjects we could choose were undergraduate students who had, in average, one year of experience in the development of OO systems using UML, and by the time the experiment took place they were taking a course of OCL we specially prepared for them.

3. **Variables selection**. The independent variable (IV) is import-coupling of an OCL expression. The dependent variables (DVs) are the comprehensibility and mo-difiability of OCL expressions.

4. **Instrumentation**. The objects were four UML/OCL combined models, having each of them only one OCL expression. The independent variable was measured through the a set of measures which are related to the most commonly used OCL concepts:

- measures related to chunking: NKW (Number of OCL Keywords), NES (Number of Explicit Self), NBO (Number of Boolean Operators), NCO (Number of Comparison Operators), NEI (Number of Explicit Iterator variables), NAS (Number of Attributes belonging to the classifier that Self represents).
- measures related to tracing: NNR (Number of Navigated Relationships), NAN (Number of Attributes referred through Navigations), NNC (Number of Navigated Classes), WNN (Weighted Number of Navigations), DN (Depth of Navigations), WNCO (Weighted Number of Collection Operations).

The dependent variables (DVs) were measured according to:

- The time each subject carried out the comprehensibility and modifiability tasks, defined as COM Time and MOD Time respectively.
- The subjects´ ratings of comprehensibility or modifiability. We call this measure COM or MOD SubComp (comprehensibility or modifiability subjective complexity).

We have also used as indicators of comprehensibility and modifiability the following measures:

$$COM\ Correctness = \frac{Number\ of\ correct\ answers}{Number\ of\ questions\ answered} \tag{2.1}$$

$$MOD\ Correctness = \frac{Number\ of\ correct\ modifications}{Number\ of\ modifications\ applied} \tag{2.2}$$

The number of correct answers represents the correctness of the understanding the questionnaire, i.e. the number of questions correctly answered. The number of correct answers is a reasonable measure of the understanding since all the tests have the same design, it has the same quantity of questions.

$$COM\ Completeness = \frac{Number\ of\ correct\ answers}{Number\ of\ questions\ required} \tag{2.3}$$

$$MOD\ Completeness = \frac{Number\ of\ correct\ modifications}{Number\ of\ modifications\ required} \tag{2.4}$$

5. **Hypothesis formulation**. We wish to test the following hypothesis (two hypothesis for each measure for the dependent variables)

   (a) $H_{0,1}$: There is no significant correlation between the OCL measures and the COM and MOD Time. $H_{1,1} : \neg H_{0,1}$

Test	NNR	NAN	WNN	WNCO	NAS	NEI	NCO	NBO	NES	NKW	NNC	DN
1	2	1	2	1	1	0	3	2	4	4	2	1
2	4	2	5	4	0	0	3	4	5	6	4	1
3	3	1	4	3	0	1	2	3	5	5	2	3
4	4	2	2	3	2	0	3	2	4	4	4	3

Table 8.2: Measure Values for each UML/OCL Model (1^{st} Experiment)

(b) $H_{0,2}$: There is no significant correlation between the OCL measures and COM/MOD correctness/completeness. $H_{1,2} : \neg H_{0,2}$

(c) $H_{0,3}$: There is no significant correlation between the OCL measures and the COM/MOD SubComp. $H_{1,3} : \neg H_{0,3}$

(d) $H_{0,4}$: There is no significant correlation between the COM/MOD Time and the COM/MOD SubComp. $H_{1,4} : \neg H_{0,4}$

6. **Experiment design.** We selected a within-subject and balanced design, i.e., all the tests (experimental tasks) had to be solved by each of the subjects. The tests were put in a different order for each subject for alleviating learning effects.

8.2.3 Operation (EF_3)

The operational phase is divided into three steps: preparation, execution and data validation.

- **Preparation**. We have selected as experimental subjects a group of students who have taken a semester class on System Analysis. In this course the student had learnt the use of UML. The students were motivated to take a course on OCL, they were informed that OCL is an expressive language used for formally expressing additional and necessary information about a model specified in UML. Later, the students were asked to participate in the course, 29 subjects agreed to take part, so they were volunteers. They were motivated to take a training session on OCL language and to do some practical exercises as part of the session, but it was not mentioned that these exercises are constituent of an experiment. The subjects were not aware of what aspects we intended to study. Neither were they aware of the actual hypothesis stated.

  We prepared the material handed to the subjects, consisting of four UML/OCL models. The experiment material is included in appendix D.1. These diagrams were related to different universes of discourse that were easy enough to be

understood by each of the subjects, and some of them were obtained from the existent OCL literature. The structural properties of each model is different as it is revealed from the measures values of each UCL/OCL model (see Table 8.2). Before running the experiment we performed a pilot experiment. We asked a researcher who has experience on OCL to carry out the experimental tasks. All the modifications she suggested were considered. Each UML/OCL model had a test enclosed that included different type of tasks:

- **Comprehensibility Tasks (COM Tasks)**:
  - The subjects had to answer four questions about the meaning of the OCL expression. These questions had the purpose to test if the subjects had understood each expression. The first question was related to navigations concepts, meanwhile the last three questions were a multiple choice about the meaning of the OCL expression. Each question has three options, being only one option the correct answer. They also had to note how long it took to answer the questions. The COM Time, expressed in minutes and seconds, was obtained from that.
- **Modifiability Tasks (MOD Tasks)**:
  - Each UML/OCL model used by the subjects in the COM Task had also enclosed three new requirements for the OCL expression. Each subject had to modify the OCL expression according to the new requirements. The modifications to each test were similar, including defining new navigations, using attributes referred through navigations, etc.
  - They also had to write down the time when they started to do the modifications and when they finished. This time was called MOD Time.
- **Rating Tasks**:
  - The subjects had to rate the COM tasks using a scale consisting of five linguistic labels (Very difficult to comprehend, A bit difficult to comprehend, Neither difficult nor easy to comprehend, A bit easy to comprehend and Very easy to comprehend). As we described previously we called this measure comprehension Subjective Complexity (COM SubComp).
  - We have also used a scale consisting of five linguistic labels similar to the one used for comprehensibility, so that the subject could rate the modifiability tasks. We called this measure MOD SubComp.

Moreover, we prepared a debriefing questionnaire. This questionnaire included personal details and experience (see the questionnaire used in this experiment in section D.5).

- **Execution**. In the lecture before the experiment was carried out, the subjects were asked to bring a watch in the next lecture. Those subjects who did not bring a watch were able to use a clock rendered with a multimedia projector. The subjects were given all the materials described in the previous paragraph. We explained to them how to carry out the test, asking for carrying out the test alone, and using unlimited time to solve it. There was an instructor who supervised the experiment and any doubt could be asked to him. We collected all the data, including subjects' rating obtained from the responses of the experiment.

- **Data validation**. Analyzing the debriefing questionnaire, we can corroborate that the subjects had approximately the same degree of experience in modelling with UML, the profile of the subject is the following: their average age is 24 years old, they have an average of 4 years programming experience, and one year in modelling UML class diagrams. Taking into account their profile, we consider their subjective evaluation reliable. However, most of the answers for the modifiability part of the four tests were not correctly answered, only the comprehensibility part of the four tests had an optimal rate of answers. We think that the reason is that the experiment was carried out after two lectures of 2 hours each, and this period of time was enough for the students to understand OCL expressions but they did not have enough practice in modifying OCL expressions. We think we exposed the students prematurely to do OCL expression modification. For that reason we consider only the COM tasks of the experiment. Regarding this part, three tests were studied as outliers and 12 tests were separated because they have a correctness below 75%. Finally, we had 101 data sets to be analyzed.

8.2.4 Analysis and Interpretation (EF_4)

We had analysed the experiment data in order to test the hypothesis formulated in section 8.2.2. For this purpose we used the Statistical Package for Social Science (SPSS) [SPS02]. First we had to check the normality of the data obtained. If the data was normal, the best option in our case was to use parametric tests because they are more efficient than non-parametric tests. We applied the Kolmogorov-Smirnov test to ascertain if the distribution of the data collected was normal. As the data was non-normal we decided to use a non-parametric test like Spearman´s correlation coefficient, with a level of significance $\alpha = 0.05$, which means the level of confidence is 95 % (i.e. the probability that we reject H_0 when H_0 is false is of at least 95 %, which is statistically acceptable). Each of the measures was correlated separately to the mean of the subjects' COM Time (see Table 8.3).

All the required tasks were answered so completeness and correctness have the same

		NNR	**NAN**	**WNN**	**WNCO**	**NAS**	**NEI**
COM. Time	Scc	.162	.020	.207	.223	-.172	.348
	p-value	.105	.840	**.038**	**.025**	.086	**.000**
COM Corr.	Scc	-.073	.016	-.180	-.151	.173	-.229
	p-value	.465	.877	.071	.131	.084	**.021**
COM SubComp	Scc	.093	-.026	.084	.108	-.076	.308
	p-value	.357	.794	.403	.283	.450	**.002**
		NCO	**NBO**	**NES**	**NKW**	**NNC**	**DN**
Und. Time	Scc	-.348	.207	.282	.207	.020	.324
	p-value	**.000**	**.038**	**.004**	**.038**	.840	**.001**
COM Corr.	Scc	.229	-.180	-.224	-.180	.016	-.147
	p-value	**.021**	.071	**.025**	.071	.877	.143
COM SubComp	Scc	-.308	.084	.166	.084	-.026	.326
	p-value	**.002**	.403	.097	.403	.794	**.001**

Table 8.3: Spearman Correlation between Measures and COM Time (1^{st} Experiment)

Table 8.4: Spearman´s Correlation between Measures and COM SubComp (1^{st} Experiment)

		Correctness	Subjective COM
COM Time	Scc	-.102	.349
	p-value	.310	.001

value. For a sample size of 101 and $\alpha = 0.05$, the Spearman cut-off for accepting H_0 is 0.1956. Hence, after analyzing Table 8.3, we can conclude that:

- There is a significant correlation between WNN, WNCO, NEI, NCO, NBO, NES, NKW and DN measures and subjects' COM Time.
- There is a significant correlation between NEI, NCO and NES measures and correctness and completeness.
- There is a significant correlation between the NEI, NCO and DN measures and the COM SubComp.

Moreover, after analyzing Table 8.4 we can conclude that there is a significant correlation between the COM Time and the COM SubComp. Nevertheless, these encouraging findings must be considered as preliminaries. More experimentation would be necessary in order to obtain more conclusive results.

8.2.4.1 Validity Evaluation

Next we will discuss the empirical study´s various threats to validity and the way we attempted to alleviate them:

- **Threats to Conclusion Validity**. The only issue that could affect the statistical validity of this study is the size of the sample data which is perhaps not enough for non-parametric statistic tests. We are aware of this, so we will consider the results of the experiment only as preliminary findings.

- **Threats to Construct Validity**. We proposed an objective measure for the dependent variable, the COM Time, i.e., the time each subject spent answering the questions related to each UML/OCL model, which is considered the time they need to understand the expression. We also proposed measures for the subjective complexity (using linguistic variables) based on the judgment of the subjects. As the subjects involved in this experiment have medium experience in OO system design using UML we think their ratings could be considered significant. The construct validity of the measures used for the independent variables is guaranteed by Briand et al.´s frameworks [BMB99], [BMB96], [BMB97] used to validate them.

- **Threats to Internal Validity**. The analysis performed here is correlational in nature. We have demonstrated that several of the measures investigated had a statistically and practically significant relationship with COM Time, comprehensibility correctness and subjective complexity. Such statistical relationships do not demonstrate per se a causal relationship. They only provide empirical evidence of it. We tried to alleviate some threats: differences among subjects, knowledge of the universe of discourse among UML/OCL combined models, accuracy of subjects responses, learning effects, fatigue effects, subject motivation, plagiarism, etc.

- **Threats to External Validity**. The greater the external validity, the more the results of an empirical study can be generalized to actual software engineering practice. Two threats of validity have been identified which limit the possibility of applying any such generalization:

  - **Materials and tasks used**. In the experiment we have used UML/ OCL models, which can be representative of real cases. Related to the tasks, the judgment of the subjects is to some extent subjective, and does not represent a real task.

  - **Subjects**. See remarks of section 8.1.

8.2.5 Presentation and Package (EF_5)

As we described in section another way to present the findings is through a publication, so, we have published a paper about this experiment in an international workshop (see [RGP04a]).

8.2.6 Conclusions of the First Experiment

The experiment reveals that there is a strong correlation between the subjective comprehensibility rating and the COM Time. The findings we have obtained are: (1) only the set of measures composed of WNN (Weighted Number of Navigations), WNCO (Weighted Number of Collection Operations), NEI (Number of Explicit Iterator variables), NCO (Number of Comparison Operators), NBO (Number of Boolean Operators), NES (Number of Explicit Self), NKW (Number of OCL Keywords) and DN (Depth of Navigations) is related with the COM Time. (2) A subset of this set of measures, that is composed of NEI, NCO and DN has an impact on the subjective complexity of subjects, and, although the rating is subjective we have also corroborated that almost the same subset of measures (with exception of NES although it has a low p-value) is also correlated to the completeness and correctness of the experimental tests, giving the second finding more significance.

Some aspects of a lesson learned after conducting this experiment are:

- We realized that in order to obtain more solid findings we should conduct a family of experiments.

- Either a experiment or a family of experiments, we should take care in the experimental training of the subjects with the purpose of prepare them appropriately for doing MOD tasks, otherwise we could exposed the subject prematurely to do MOD tasks as it happens in this experiment.

- Regarding the result of the experiment we believe that both cognitive techniques (tracing and chunking) are part of the influence of coupling on comprehensibility. However, we think that the two important concepts regarding the two cognitive aspects involved with coupling are the quantity of concepts involved (which is related to chunking) and the depth of coupling (related to tracing) and we plan a controlled experiment to study their influence in comprehensibility and modifiability. We intended to reveal if one of these factors (or its interaction) influences on comprehensibility/modifiability. And this is the focus on conducting the following family of experiments.

8.3 First Family of Experiments

The main goal of this section is to carefully describe the experimental process we followed in order to corroborate if the depth of coupling and quantity of coupled objects influence on the comprehension and modifiability of OCL expressions. The original experiment was carried out at the National University of Comahue in Argentina (UNC experiment) and its replicas at University of Castilla La-Mancha (UCLM experiment) and University of Alicante (UA experiment) in Spain, respectively.

8.3.1 Experiment Preparation (F_1)

We believe that two important factors of the import-coupling are the number of different objects which are coupled to the contextual instance and the depth of navigation. For measuring these factors we have selected two measures: the number of navigated classes (NNC) which represents the quantity of coupled objects and the depth of navigation (DN) which stands for the distance of the farthest coupled object from the contextual instance. So, the goal pursed of the first family of experiment is defined using the GQM template [BR98], [CPG01] : Analyze <<length and coupling, measured by DN and NNC respectively >>; for the purpose of <<Evaluating>>; with respect to <<The capability to be used as comprehensibility and modifiability indicators of OCL expressions>>; from the point of view of <<researchers>>: in the context of <<Undergraduate Computer Science students >>.

8.3.2 Context Definition (F_2)

We obtained a profile of the subjects who participated in each experiment using a debriefing questionnaire similar to the previous experiment and including personal details and experience. A short description of the subjects, how they were invited to take part of the experiment and the inducement of the experiment are detailed in the following:

- **UNC Experiment (December 2003)**: In order to select the subjects we motivated a group of students who had taken a semester on System Analysis to take an additional and intensive course in the OCL language. The course consisted of two sessions of 10 hours each one. Fifteen students who attended the second session participated in the last module. In this session they had to do some practical exercises, but it was not mentioned that these exercises were related to any experiment whatsoever. The subjects were not aware of what aspects we intended to study.

- **UCLM Experiment (March 2004)**: The subjects were twelve undergraduate students enrolled in the fifth-year of Computer Science in the Department of Computer Science at the University of Castilla La-Mancha in Spain. The experiment was run in a practical session of the Software Engineering II course. Before the experiment was run the subjects participated in three lectures about OCL of one hour each one. These lectures were given by the same professor who supervised the experiment. The subjects were motivated to participate in the practical session because we told them that similar exercises could be included in the final exam. However, not all the students who participated in the experiment attended all the lectures, having the subjects different backgrounds in the OCL language. This issue could be the most important threat to the internal validity of this replica.

- **UA Experiment (April 2004)**: The subjects were twenty nine students enrolled in the third-year of Computer Science at the Department of Computer Science at the University of Alicante (UA), Spain. They were students of the first Software Engineering course. We invited the students to participate in a short seminar about OCL and to do a test as part of the seminar. The subjects were motivated to participate in the experiment because they could obtain an extra point in the final score of the Software Engineering course if and only if they completed all the tests. The extra point we gave them was only dependent on finishing the exercise, not on how the exercise was done.

8.3.3 Design Framework of the First Family of Experiments (F_3)

Hereafter, we will summarize the main experimental process steps common to the original experiment and its replicas.

The planning phase deals with how the experiment is conducted, including the

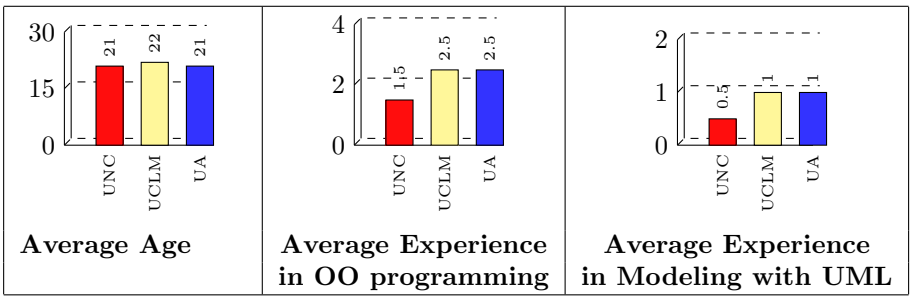


Table 8.5: Subject Profile (1^{st} Family of Experiments)

following steps:

- **Variables selection.** The independent variables (IVs) are the quantity of coupled objects and the depth of coupling. The dependent variables (DVs) are the comprehensibility and modifiability of OCL expressions.

- **Instrumentation.** The material to be handed to the subjects consists of four UML/OCL models with one OCL expression in each one. As we proceed in the previous experiment we evaluate that the complexity of the universes of discourse of the models do not bias the results and we also did a pilot experiment.

  The quantity of coupled objects were measured by the Number of Navigated Classes (NNC) and the depth of coupling by the Depth of Navigations (DN) measures. The DVs were measured according to the COM Time and MOD Time and also to Comprehensibility Efficiency (COM Eff) and the Modifiability Efficiency (MOD Eff) defined as:

$$COM\ Efficiency = \frac{Number\ of\ correct\ answers}{COM\ Time} \tag{3.5}$$

$$MOD\ Efficiency = \frac{Number\ of\ correct\ modifications}{MOD\ Time} \tag{3.6}$$

- **Hypothesis formulation**. We wished to test the following hypotheses:

  - Hypotheses 1, $H_{0,1}$: There is no effect of the depth of coupling (measured by DN) on the COM Time of OCL expressions // $H_{1,1}$: $\neg$ $H_{0,1}$
  - Hypotheses 2, $H_{0,2}$: There is no effect of the quantity of coupled objects (measured by NNC) on COM Time of OCL expressions. // $H_{1,2}$: $\neg$ $H_{0,2}$
  - Hypotheses 3, $H_{0,3}$: There is no interaction effect between depth of coupling (measured by DN) and the quantity of coupled-objects (measured by NNC) on COM Time of OCL expressions. // $H_{1,3}$: $\neg$ $H_{0,3}$

  Similar hypotheses were defined for MOD Time ($H_{i,j}$, i=0,1 ; j=4,5,6), COM Eff ($H_{i,j}$, i=0,1 ; j=7,8,9) and MOD Eff ($H_{i,j}$, i=0,1 ; j=10,11,12).

  Besides, we wish to test the following hypothesis:

  - Hypotheses 13, $H_{0,13}$: The COM subjective complexity distribution is the same for all the treatments. $H_{1,13}$: $\neg$ $H_{0,13}$
  - Hypotheses 14, $H_{0,14}$: The MOD subjective complexity distribution is the same for all the treatments. $H_{1,14}$: $\neg$ $H_{0,14}$

  Table 8.6 explains the hypotheses and the tests used to verify them.

Table 8.6: Synopsis of Hypotheses and the Statistical Test Applied (1^{st} Family of Experiments)

	Time		**Efficiency**	
Measures	**COM Time**	**MOD Time**	**COM Eff**	**MOD Eff**
DN	Hypotheses 1	Hypotheses 4	Hypotheses 7	Hypotheses 10
	Test: ANOVA			
NNC	Hypotheses 2	Hypotheses 5	Hypotheses 8	Hypotheses 11
	Test: ANOVA			
DN and NNC interaction	Hypotheses 3	Hypotheses 6	Hypotheses 9	Hypotheses 12
	Test: ANOVA			

	Subjective Complexity	
	COM SubComp	**MOD SubComp**
	Hypotheses 13	Hypotheses 14
	Test: W of Kendall	

Table 8.7: A 2 × 2 Factorial Design (1^{st} Family of Experiments)

		DN	
		low	**high**
NNC	**low**	**2, 1** (group 1)	**2, 3** (group 3)
	high	**4, 1** (group 2)	**4, 3** (group 4)

- **Experiment Design**. Taking the hypotheses into account, we considered two factors: NNC and DN with two levels each one (low, high), 1 and 3 for DN, and 2 and 4 for NNC. The 2x2 crossed factorial design is shown in Table 8.7. For each group we designed one UML/OCL model composed of one class diagram with one OCL expression. We selected a within-subject design experiment, i.e. all the tasks of the four models had to be solved by each of the subjects. The four models were randomly assigned in different order to the subjects to avoid learning effects.

  Experiment Tasks. Each UML/OCL model had an enclosed test that included the same types of tasks as the first experiment described. The tests used in the experiment are included in the appendix B, see D.1. Each UML/OCL model had an enclosed test that included two types of tasks: COM and MOD Tasks as we used in the previous experiment.

Experimental Execution. The experiments were run in one session. We explained to them how to carry out the tests. A same instructor supervised each of the experiments.

Data validation. Once the data were collected, we checked them and noted down the different times and the number of answered (right and wrong) questions. Their

COM and MOD Eff was calculated.

From these values, we calculated four measures of the dependent variable, two of them are COM and MOD Correctness defined in the previous experiment and the other two are new, defined as COM and MOD Eff (see their definition in Instrumentation item).

8.3.4 Data Analysis and Interpretation ($\mathbf{F}_5$)

8.3.4.1 Descriptive Analysis

Fig. 8.2 shows a descriptive plot of the time spent by the subject in comprehending and modifying OCL expressions whereas Fig. 8.3 shows the subject' efficiency. Both tables shows the time and the efficiency for each group (G1, G2, G3 and G4). The more time consuming COM task for UA and UCLM experiment' subject was the G2 group. However they were more efficient in this group rather than the other. It seems that G3 group was the more time consuming MOD task for the UA and UCLM experiment' subjects. From Table 8.3 we can see that subjects of UA experiment were less efficient in modifying G3 and G2 tasks.

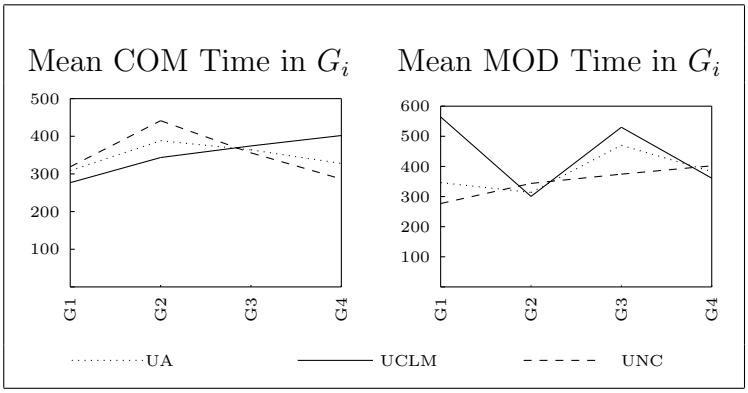


Figure 8.2: Mean COM and MOD Time (1^{st} Family of Experiments)

8.3.4.2 Testing the Hypotheses

Two different analysis were performed for each experiment of the family. The analysis only differs in the instrumentation for measuring the DVs. The results of the first analysis were already published in [RGP04b] and [RGP04c], however the last analysis is new. In the first analysis we discarded those tests that were incomplete (completeness = 0) and also tests having a correctness than 0.75. We believe that discarding tests according to the correctness we loose valuable information, and

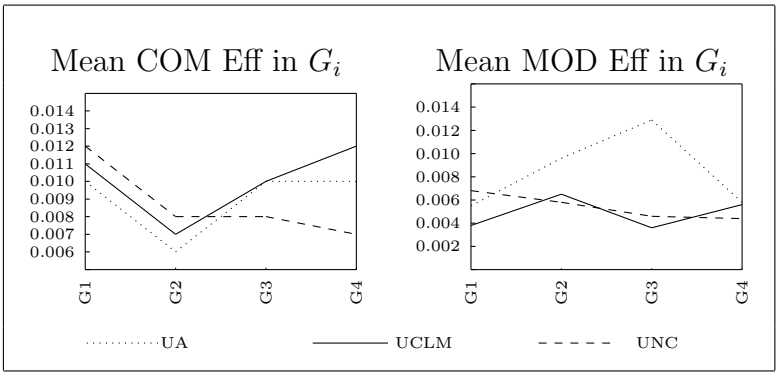


Figure 8.3: Mean COM and MOD Efficiency (1^{st} Family of Experiments)

Table 8.8: Shapiro Wilk Normality Test for COM and MOD Time (1^{st} Family of Experiments)

Group	**Shapiro-Wilk significance level**			
	G1	**G2**	**G3**	**G4**
COM Time	0.200	0.555	0.530	0.682
MOD Time	**0.093**	0.237	0.734	**0.060**

we decided to include those tests discarded in the second analysis. In the second analysis we use as DV the COM/MOD Eff instead of the COM/MOD Time.

We selected the best test to apply to our data for obtaining results susceptible of being interpreted. For the three analysis we performed similar tests. Before testing the formulated hypothesis we evaluated if the data follow a normal distribution or not using the Shapiro-Wilk test, with α =0.10 which means a 90% level of confidence.

8.3.4.3 Analysis of COM/MOD Time in the UNC Experiment

Shapiro-Wilk tests results are shown in Table 8.8 for COM and MOD Time. Only two groups were not normal (G1 and G4) in the MOD Time and we decided to carry out an ANOVA with repeated measures because this type of analysis allow us to analyse interaction between the independent variables under study and the measurement of the dependent variable is repeated.

All the data were complete. There were no outliers in the comprehensibility part, however in the case of the modifiability part the tests belonging to eleven subjects were discarded as outliers. Due to the number of outliers were high in the modifiability part, we decided to tests the hypothesis twice, with outliers and without them.

Table 8.9 (a) shows the ANOVA results for the COM Time. The eighth column of Table 8.9 (a) represents the level of significance which will allow us to reject or accept the hypothesis we have formulated, significant coefficient at level 0.10 are shown in bold font. In each row of the table we have the factors (DN, and NNC), their interaction, and their errors. The last column of Table 8.9 (a) shows the observed power. Analyzing Table 8.9 (a) we can conclude that:

- The value of the DN and NNC measures affect the COM Time of an OCL expression within UML/OCL models, but there is no effect of their interaction.
- The power of the length (measured by DN) main effect was .996, which indicates that there is 99.6% chance of detecting a genuine effect. The power of the coupling (measured by NNC) main effect was 0.535, which indicates that the chance of detecting a genuine main effect is fairly low, 53.5%. Finally, the power of the length by coupling interaction was 0.276, which indicates that the chance of detecting a genuine interaction effect is low, 27.6%.

We found many outliers in the MOD tasks and few subjects remains (only four), so we did two ANOVAS, one where outliers were removed and another including the whole data. Table 8.9 (b) and (c) shows the ANOVAs for MOD Time for the whole set of data of UNC experiment. Table 8.9 (b) shows the results after removing the outliers whereas Table 8.9 (c) shows the results including outliers. Nevertheless the results from Table 8.9 (b) and 8.9 (c) are the same.

The conclusion is the DN and NNC factors affect the COM Time and the factor that affects the MOD Time is DN.

8.3.4.4 Analysis of COM/MOD Time in the Experiment Replicas

As most of the details of the analysis of the replicas were very similar to the first experiment, we will only focus on different aspects. Tables 8.10, 8.11 and 8.12 include their principal results, the first table shows the number of tests that were discarded (completeness = 0) and the results of the Shapiro Wilk test. The second table, table 8.11 includes the quantity of outliers we found. Table 8.12 contains the significance level and observed power for ANOVA Tests respectively. Table 8.13 summarize the results for the three experiments. The main conclusions are:

- The COM Time is affected by DN and the interaction of DN and NNC for UCLM experiment. However in UA experiment only the interaction between DN and NNC affects the COM Time. That means that individually each main factor (DN or NNC) does not affects the COM Time but their interaction does. In order to try to understand the interaction, we need to look at the means

a. COM Time								
Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		91182.0167	1	91182.0167	20.5736	**0.0004**	.996
Error(DN)	L		62047.7333	14	4431.98095			
NNC		L	33464.8167	1	33464.8167	3.3151	**0.0900**	.535
Error(NNC)		L	141324.933	14	10094.6381			
DN * NNC	L	L	5782.01667	1	5782.01667	1.1923	0.2932	.276
Error(DN*NNC)	L	L	67889.7333	14	4849.26667			
b. MOD Time without Outliers								
Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		63504	1	63504	12.3137	**0.0392**	0.829
Error(DN)	L		15471.5	3	5157.1666			
NNC		L	4624	1	4624	1.3056	0.3361	0.233
Error(NNC)		L	10624.5	3	3541.5			
DN * NNC	L	L	992.25	1	992.25	0.2405	0.6574	0.125
Error(DN*NNC)	L	L	12377.25	3	4125.75			
c. MOD Time including Outliers								
Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		115720.417	1	115720.417	10.805	**.005**	.930
Error(DN)	L		149939.333	14	10709.952			
NNC		L	1050.017	1	1050.017	.115	.740	.118
Error(NNC)		L	128080.733	14	9148.624			
DN * NNC	L	L	277.350	1	277.350	.020	.891	.103
Error(DN*NNC)	L	L	198479.400	14	14177.100			
L means Lineal								

Table 8.9: ANOVA with Repeated Measures for the UNC Experiment (1^{st} Family of Experiments, 1^{st} Experiment)

for the interaction. Figure 8.4 show the estimated marginal means in a profile plot for each experiment. The vertical axis of a profile plot represents the dependent variable, the COM Time. DN factor was selected to be represented as the horizontal axes, whereas NNC factors will be displayed as separate lines or plots. A simple overview of the plot clearly shows that the two lines are not parallel. The fact that the lines are not parallel is the defining feature of an interaction, due parallel lines would indicate no interaction. The line representing the NNC = 2 is relatively flat for UCLM experiment, whereas it has a positive slope in UA experiment. In the last case it means that the larger the DN factor the worse the COM Time the subjects spent on the model. The line representing the NNC = 4 has a negative slop in both experiments, the larger the DN factor the better the COM Time. Moreover, we can say that if the depth of the navigation is low, better COM Time are obtained if the quantities of classes involved are low (NNC= 2) than high

Table 8.10: Analysis of the Shapiro Wilk Normality and Number of Test Discarded for Incompleteness (1^{st} Family of Experiments)

Completeness		**UNC**	**UCLM**	**UA**	**UNC+UCLM+UA**
# COM comp = 0		0	0	0	0
# MOD comp = 0		0	0	3	3
Dependent Variable		**UNC**	**UCLM**	**UA**	**UNC+UCLM+UA**
COM Time	**group1**	.200	.224	.464	**.036**
	group2	.555	.130	**.000**	**.000**
	group3	.530	**.090**	.159	**.032**
	group4	.682	.477	.373	.199
MOD Time	**group1**	**.093**	.481	.373	**.000**
	group2	.237	.681	**.005**	**.039**
	group3	.734	.246	.133	**.003**
	group4	**.060**	.012	**.056**	**.001**

Table 8.11: Quantity of Outliers Detected for COM/MOD Time (1^{st} Family of Experiments)

	UNC	**UCLM**	**UA**	**UNC+UCLM+UA**
COM Time	0	2	3	7
MOD Time	9	4	8	19

(NNC= 4). However if the depth of the navigation is high, better COM Time are obtained if the quantities of classes involved are high (NNC= 4) than low (NNC= 2). The last case, DN= 3 and NNC = 2 represents navigation using reflexive relationships which seems to be more difficult to understand than simple navigations involving two different classes.

- The MOD Time is affected by the NNC in the UCLM experiment, and by DN and NNC in the UA experiment. The interaction does not affect the MOD Time for both experiments.

Regarding the results for the three experiments we decided to analyze the whole data for the family, we add a precedence factor (PR) as we did in analysing COM and MOD Time. We obtain as the results that DN, NNC and their interaction affect the COM Time (see Table 8.14 (a)), and DN and NNC affect the MOD Time but not their interaction (see Table 8.14 (b)). However if we consider the precedence factor of the data neither factors nor their interaction affect COM or MOD Time.

Table 8.12: The Analysis Results of ANOVAs COM/MOD Time (1^{st} Family of Experiments)

Dependent Variable		**UCLM**			**UA**		
		# of Subject	**p-value p-value**	**Observed Power**	**# of Subject**	**p-value p-value**	**Observed Power**
COM Time	**DN**	10	**.058**	**.640**	27	.434	.200
	NNC	10	.376	.223	27	.666	.131
	Interaction	10	**.027**	**.784**	27	**.004**	**.923**
MOD Time	**DN**	8	.393	.213	21	**.000**	**.996**
	NNC	8	**.026**	**.812**	21	**.006**	**.911**
	Interaction	8	.295	.271	21	.121	.469

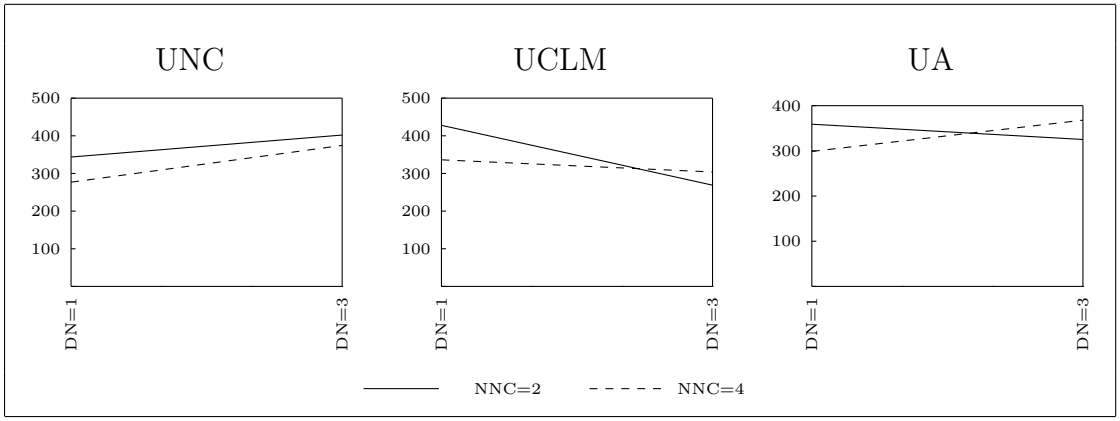


Figure 8.4: Estimated Marginal Means in a Profile Plot for COM Time (1^{st} Family of Experiments)

Table 8.13: Summary of the Results (1^{st} Family of Experiments)

Experiments	**COM Time**			**MOD Time**		
	DN	NNC	Interaction	DN	NNC	Interaction
Experiment at UNC	Yes	Yes		Yes		
Experiment at UCLM	Yes		Yes		Yes	
Experiment at UA			Yes	Yes	Yes	

a. COM Time								
Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		38086.598	1	38086.598	3.600	**.064**	.589
DN * PR	L		47629.698	2	23814.849	2.251	.117	.568
Error(DN)	L		486715.118	46	10580.763			
NNC		L	25707.735	1	25707.735	3.216	**.080**	.549
NNC * PR		L	9823.327	2	4911.663	.614	.545	.238
Error(NNC)		L	367763.347	46	7994.855			
DN * NNC	L	L	80007.813	1	80007.813	12.186	**.001**	.964
DN * NNC * PR	L	L	13955.422	2	6977.711	1.063	.354	.337
Error(DN*NNC)	L	L	302009.272	46	6565.419			
b. MOD Time								
Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		252680.667	1	252680.667	23.152	**.000**	.999
DN * PR	L		704.709	2	352.355	.032	.968	.107
Error(DN)	L		371070.210	34	10913.830			
NNC		L	84567.366	1	84567.366	8.437	**.006**	.885
NNC * PR		L	36205.754	2	18102.877	1.806	.180	.482
Error(NNC)		L	340804.543	34	10023.663			
DN * NNC	L	L	6685.417	1	6685.417	.923	.343	.246
DN * NNC * PR	L	L	4617.133	2	2308.566	.319	.729	.170
Error(DN*NNC)	L	L	246246.543	34	7242.545			
L means Lineal								

Table 8.14: ANOVA with Repeated Measures of MOD Time (1^{st} Family of Experiments)

8.3.4.5 Analysis of the COM/MOD Efficiency in the UNC Experiment

Table 8.15 shows the normality of the data for the COM and MOD Eff. However, analysing the data obtained three outliers were identified in the COM part. In the case of the MOD part, the tests belonging to five subjects were discarded as outliers. Analyzing Table 8.16 we can conclude that:

- the value of the DN measure, NNC measure, and they interactions affect the COM Eff of an OCL expression. The value of DN affects the MOD Eff.

8.3.4.6 Analysis of the COM/MOD Efficiency in the Experiment Replicas

As we proceed previously, in Table 8.17 we include the results of the normality analysis.

Table 8.15: Shapiro Wilk Normality Test Results for COM and MOD Efficiency (1^{st} Family of Experiments, 1^{st} Experiment)

Group	**Shapiro-Wilk significance level**			
	G1	**G2**	**G3**	**G4**
COM Eff	**0.060**	0.651	0.840	0.921
MOD Eff	0.149	0.553	0.929	0.895

a. COM Efficiency

Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		6.3599E-05	1	6.3599E-05	7.310	**0.020**	0.813
Error(DN)	L		9.5698E-05	11	8.6998E-06			
NNC		L	0.00013692	1	0.00013692	23.793	**0.000**	0.998
Error(NNC)		L	6.33E-05	11	5.7546E-06			
DN * NNC	L	L	2.9885E-05	1	2.9885E-05	5.716	**0.035**	0.724
Error(DN*NNC)	L	L	5.751E-05	11	5.2282E-06			

b. MOD Efficiency

Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		1.6296E-05	1	1.6296E-05	3.590	**0.090**	
Error(DN)	L		4.0849E-05	9	4.5388E-06			
NNC		L	2.8348E-06	1	2.8348E-06	0.523	0.487	
Error(NNC)		L	4.8748E-05	9	5.4164E-06			
DN * NNC	L	L	1.4308E-06	1	1.4308E-06	0.632	0.446	
Error(DN*NNC)	L	L	2.0355E-05	9	2.2616E-06			

L means Lineal

Table 8.16: ANOVA with Repeated Measures (1^{st} Family of Experiments, 1^{st} Experiment)

Table 8.17: Shapiro Wilk Normality Test Results for COM and MOD Efficiency (1^{st} Family of Experiments)

Completeness		**UNC**	**UCLM**	**UA**	**UNC+UCLM+UA**
# COM comp = 0		0	0	0	0
# MOD comp = 0		0	0	3	3
Dependent Variable		**UNC**	**UCLM**	**UA**	**UNC+UCLM+UA**
COM Time	**group1**	.060	.802	.579	.962
	group2	.651	.256	.643	.791
	group3	.840	.430	.278	.385
	group4	.921	.607	.808	.054
MOD Time	**group1**	.149	.415	.098	.187
	group2	.553	.320	.467	.809
	group3	.929	.497	.275	.409
	group4	.895	.286	.965	.658

Table 8.18: Quantity of Outliers Found in Each Experiment (1^{st} Family of Experiments)

	UNC	**UCLM**	**UA**	**UNC+UCLM+UA**
COM outliers	3	1	1	3
MOD outliers	5	4	2	7

Table 8.19: Analysis Results of ANOVAs COM/MOD Efficiency (1^{st} Family of Experiments)

Dependent Variable		**UCLM**			**UA**		
		# of Subject	**p-value p-value**	**Observed Power**	**# of Subject**	**p-value p-value**	**Observed Power**
COM Time	**DN**	11	**.079**	.569	25	**.004**	.918
	NNC	11	.479	.179	25	.173	.391
	Interaction	11	.103	.511	25	**.003**	.939
MOD Time	**DN**	8	.873	.104	24	**.001**	.985
	NNC	8	**.002**	.995	24	**.002**	.967
	Interaction	8	.786	.111	24	.287	.323

We run different ANOVAs according to the three analysis applied and the three experiment launched, their results are included in Table 8.18, 8.19 and 8.20. Through experimentation we can conclude that:

- DN measure affects the COM Eff of the three experiments, the interaction of DN and NNC affects the COM Eff in the UNC and UA experiments, and COM Eff is also affected by NNC in UNC experiment.
- In the case of MOD Eff is affected by DN in UNC and UA experiments and also for NNC in the UCLM and UA experiments. The interaction of the factor does not affect the MOD Eff.

Regarding the results for the three experiments we decided to analyze the whole data for the family, we add a precedence factor (PR) when we gather all the information

Table 8.20: Summary of the Results of Hypothesis (1^{st} Family of Experiments)

Experiments	**COM Eff**			**MOD Eff**		
	DN	NNC	Interaction	DN	NNC	Interaction
Experiment at UNC	Yes	Yes	Yes	Yes		
Experiment at UCLM	Yes				Yes	
Experiment at UA	Yes		Yes	Yes	Yes	

a. COM Time								
Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		3.7027E-05	1	3.7027E-05	2.45011122	0.125	.459
DN * PR	L		0.00014466	2	7.233E-05	4.78613491	**0.013**	.857
Error(DN)	L		0.00063473	42	1.5113E-05			
NNC		L	0.00010965	1	0.00010965	8.0955151	**0.006**	.876
NNC * PR		L	3.4223E-05	2	1.7111E-05	1.26334196	0.293	.378
Error(NNC)		L	0.00056887	42	1.3545E-05			
DN * NNC	L	L	0.00017744	1	0.00017744	21.1263877	**.000**	.998
DN * NNC * PR	L	L	2.8715E-05	2	1.4358E-05	1.70941261	0.193	.468
Error(DN*NNC)	L	L	0.00035276	42	8.3991E-06			
b. MOD Time								
Source	**DN**	**NNC**	**Sum of squared**	**df**	**mean squared**	**F-ratio**	**Sig. level**	**Observed Power**
DN	L		9.8859E-05	1	9.8859E-05	13.359565	**0.000**	.975
DN * PR	L		2.0653E-05	2	1.0326E-05	1.39549725	0.258	.406
Error(DN)	L		0.00031819	43	7.3998E-06			
NNC		L	6.6571E-05	1	6.6571E-05	11.3090939	**0.001**	.952
NNC * PR		L	6.9431E-05	2	3.4716E-05	5.89750125	**0.005**	.917
Error(NNC)		L	0.00025312	43	5.8865E-06			
DN * NNC	L	L	3.671E-06	1	3.671E-06	1.04285488	0.312	.265
DN * NNC * PR	L	L	1.7683E-06	2	8.8414E-07	0.2511656	0.779	.156
Error(DN*NNC)	L	L	0.00015137	43	3.5201E-06			
L means Lineal								

Table 8.21: ANOVA with Repeated Measures (1^{st} Family of Experiments)

(PR = 1 for UNC, PR = 2 for UCLM, and PR = 3 for UA). We obtain as the results: that DN, NNC and their interactions affects the COM Eff (see table 8.21 (a)), and DN and NNC affect the MOD Eff but not their interaction (see table 8.21 (b)). However if we consider the precedence factor of the data DN only affects COM Eff and NNC affects the MOD Eff.

Figure 8.5 depict the estimated marginal means in a profile plot for COM and MOD Eff. The vertical axis of a profile plot represents the dependent variable (the COM or MOD Eff). As we proceed in the plots of Figure 8.4, DN factor was selected to be represented as the horizontal axes, whereas NNC factors will be displayed as separate plots. A simple overview of the plot clearly shows that the two lines are not parallel in COM Eff (interaction of DN and NNC seems to affects COM Eff), and indeed they are parallel in the MOD Eff (indicating no interaction of DN and NNC). Here we summarize the conclusions:

- In the COM Eff we can say that if the depth of the navigation is low, better COM Eff is obtained if the quantities of classes involved are low (NNC= 2) than high (NNC= 4). However if the depth of the navigation is high, better

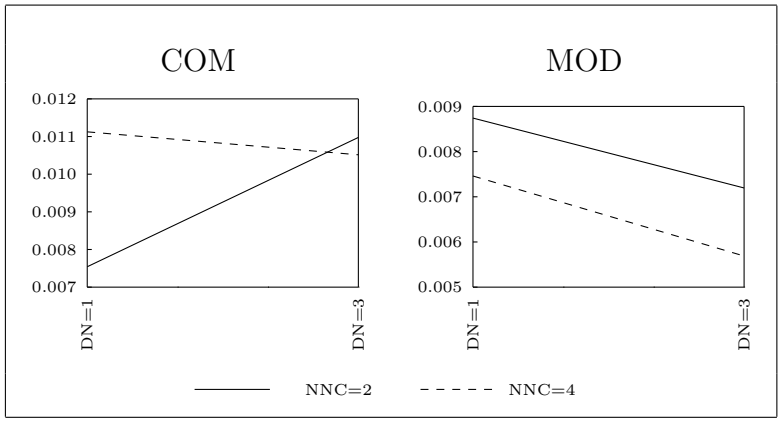


Figure 8.5: Estimated Marginal Means in a Profile Plot (1^{st} Family of Experiments)

COM Eff are obtained if the quantities of classes involved are high (NNC= 4) than low (NNC= 2). The last case happens when DN= 3 and NNC = 2, and represent navigations using reflexive relationships in a class which seems to be more difficult to understand than simple navigations involving two different classes (see a similar conclusion in section 8.3.4.4).

- In the MOD Eff whichever the plots (representing the NNC = 4 or NNC =2) they have a negative slop in both experiments, also their slop seem to be proportional. Moreover, considering the plot, the larger the DN factor the worse the MOD Eff. This reveals an important aspect, despite of the quantity of classes being high (NNC = 4) or low (NNC = 2), if they are loosely coupled (DN is low) the better the MOD Eff, if they are highly coupled the worse MOD Eff.

8.3.4.7 Analysis of the COM/MOD Subjective Complexity

The average range of the Rating for the four models is shown in Table 8.22 and Figure 8.6. The easiest models were when DN is low (G1 and G2), the more difficult was G3 (DN is high and NNC is low), the models which includes reflexive navigations. According to the results shown in Table 8.23, we reject hypothesis $H_{0,7}$ for UNC and UA experiment, and we reject $H_{0,8}$ for the three experiments, meaning that for most of the cases, the rating distribution is not the same for all the treatments.

Looking for an answer of how the rating distributes along with the treatments, we study the rating depending of the combination of treatments. Due to the number of treatments are four we have to study six cases (G1-G2, G1-G3, G1-G4, G2-G3, G2-G4 and G3-G4). The results included in Table 8.24 allow us to conclude:

Table 8.22: Summary of the Average Range of COM and MOD Subjective Complexity (1^{st} Family of Experiments)

Experiment	COM SubComp			MOD SubComp		
	UNC Avg. Range	UCLM Avg. Range	UA Avg. Range	UNC Avg. Range	UCLM Avg. Range	UA Avg. Range
G1	1.80	2.33	1.90	1.80	2.88	2.29
G2	2.70	2.29	2.60	2.63	1.67	2.00
G3	3.23	2.71	3.02	2.73	3.33	3.17
G4	2.27	2.67	2.48	2.83	2.13	2.54

Table 8.23: Summary of the Average Range (W of Kendall) of COM and MOD Subjective Complexity (1^{st} Family of Experiments)

Experiment	COM SubComp			MOD SubComp		
	UNC Avg. Range	UCLM Avg. Range	UA Avg. Range	UNC Avg. Range	UCLM Avg. Range	UA Avg. Range
N	15	12	26	15	12	24
W of Kendall	.383	.042	.205	.230	.521	.232
Chi-square	17.216	1.5	15.988	10.330	18.740	16.706
Gl	3	3	3	3	3	3
Sig.	**.001**	.682	**.001**	**.016**	**.000**	**.001**

Table 8.24: Summary of the W of Kendall (1^{st} Family of Experiments)

DV	Exp	G1 G2	G1 G3	G1 G4	G2 G3	G2 G4	G3 G4
COM SubComp	UNC	**.008**	**.001**	.317	.059	.102	**.014**
	UCLM	1.0	.480	.480	.414	.180	.705
	UA	**.012**	**.003**	.46	**.035**	.763	**.033**
MOD SubComp	UNC	.58	.20	**.011**	.655	.705	.655
	UCLM	**.02**	.317	**.102**	.002	.083	**.005**
	UA	.317	**.012**	.366	**.000**	.134	.052

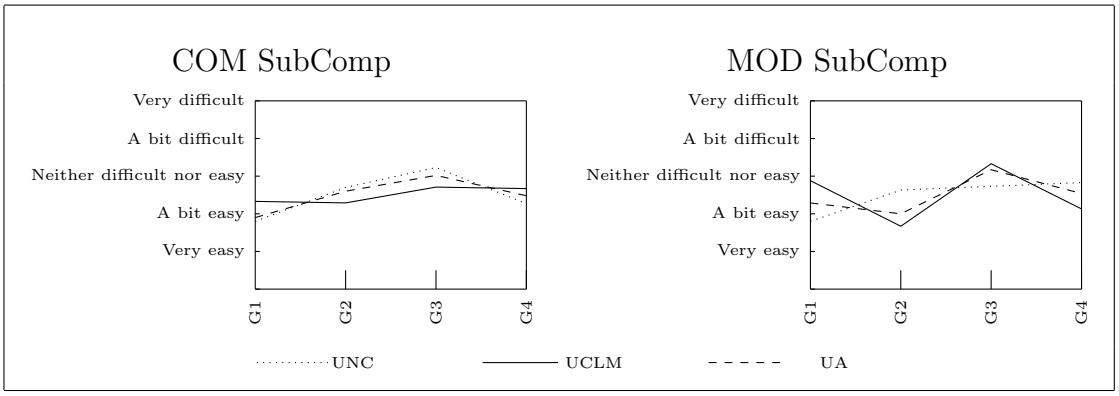


Figure 8.6: Average Range' Plot of COM and MOD Subjective Complexity (1^{st} Family of Experiments)

- Either DN is high (G3-G4) or low (G1-G2), the COM rating depends of NNC for UNC and UA experiments. For these two experiment the rating also depends of DN when NNC is low (G1-G3). The rating also depends of G2-G3 for the UA experiment, that situation happens when NNC is high and DN is low. In all the other combinations, there is no relation between treatments and rating.

- In the MOD case, different cases were found, for example, in the UCLM experiment the rating depends of NNC for DN high or low, in the UNC depends of DN when NNC is low. Three significant results were found when NNC and DN has different values (one is high and the other is low, or the other way around), these are the case of G1-G4 in the UNC experiment, and G2-G3 in the UCLM and UA experiments.

8.3.4.8 Validity Evaluation

Next we will discuss the means by which we attempted to alleviate the issues that could threaten the validity of the family experiments:

- **Threats to Conclusion Validity**. In the conclusion validity we want to make sure that there is a statistical relationship, i.e. that our conclusions are statistically valid.

  - **Low statistical power**. The power of a statistical test is the ability of the test to reveal a true pattern in the data. If the power is low, there is a high risk that an erroneous conclusion will be drawn. We have considered the observed power of the ANOVAs in our analysis, and when

we evaluate the results for the whole family we also add a PR factor as an inter-subject factor.

- **Violated assumption of statistical tests**. Violating the assumptions of a test may lead to wrong conclusions. In our case we used a robust test, the ANOVA.
- **The error rate**. The error rate is concerned with the actual significance level. In order to improve the power of the test we have selected $\alpha = 0.10$ significance level which is common in ANOVA tests.

- **Threats to Construct Validity**. In this family we proposed an objective measure for the variables used in the hypothesis: (1) for the dependent variable we use a measure of how precise the subjects answering tasks per time are (the COM and MOD Eff) as well as the time the subject spent on different tasks (the COM and MOD Time)[1]. (2) for those hypotheses related to cognitive aspects of the subjects we have used a qualitative and objective measure of the subject's subjective opinion, and we use linguistic labels, providing a scale to rate tasks. (3) for the independent variables, their validity is guaranteed by Briand et al.'s framework which was used to validate them [BMB99], [BMB96], [BMB97].

  An issue we also dealt with is:

  - **Confounding constructs and levels of constructs**. In some relations they are not primarily related to the presence or absence of a construct, but to the level of the construct which is of importance to the outcome [WRH+00]. We selected subjects having knowledge in UML language, but the years of experience with UML is different: UNC students have two years of experience, UCLM students have one year of experience, and UA students have less than a year.

- **Threats to Internal Validity**. We have dealt with the following issues:

  - **History**. An existent risk is that history affects the experimental results, since the circumstances are not the same on the experiments [WRH+00]. This is the situation in the UA experiment, in which case we ran the experiment in the final lecture before Easter Holidays, and we realized that during the experiment they were eager to finish the exercise. We also became aware of that because, before the seminar started, many students had asked us about the ending time of the seminar. We think that this could have affected the results obtained in this replica because in general, the tests' correctness was low.

[1] The first time we used efficiency was in this family. We believe that efficiency is a more accurate indicator of the DVs

  - **Maturation**. Subjects may react differently as time passes by. For example, the subjects are affected negatively during the experiment if they get tired or bored [WRH+00]. We dealt with this issue and we ran a pilot experiment in order to estimate the average time the subject will spend performing the four tests. The estimation was of one hour and we believe that it would not produce a boredom effect [JM01]. In fact, after running the experiment we proved that the estimated time was right.

  - **Selection**. This is the effect of natural variation in human performance. Depending on how the subjects are selected from a larger group, the selection effects may vary [WRH+00]. The selection of subjects was not the same in the three experiments as described in section 8.3.2. Here, a short description of subjects' selection. The UNC experiment was composed of volunteers, who decided to take part of an OCL course, and as Wohlin [WRH+00] argues the effect of letting volunteers may influence the results because their motivation and adequacy for a new task. In the case of UA experiments all the students of a software engineering's course decided to participate in a seminar where the experiment was run. In the UCLM experiment only those subjects who participate in a some lectures of a software engineering's course did the experiment.

- **Threats to External Validity**. The greater the external validity, the more the results of an empirical study can be generalized to actual software engineering practice. Three threats of validity have been identified which limit the possibility of applying any such generalization:

  - **Interaction of selection and Treatment**. See the evaluation of this issue in section 8.1.

  - **Interaction of setting and treatments**. This is the effect of not having representative experimental setting or material. In the experiment we used UML/OCL models which can be representative of real cases. Moreover, we give a course on OCL using the same terminology as its last version (2.0)

  - **Interaction of history and treatment**. The aforementioned issue related to history in the internal validity could affect the result of UA experiments. Students were eager to finish the exercises in the last day before Easter Holidays, and by finishing them they could obtain an extra point and this point was only dependent on finishing the exercise, not on how the exercise was done.

8.3.5 Conclusions of the First Family of Experiments (F_6)

We presented the analysis of a family of experiments (a experiment and its two replicas) to ascertain if any relations exists between the navigation depth (measured by DN) and the quantity of different object coupled (NNC) of an OCL expression and its comprehensibility and maintainability. Both measures constitute new kinds of coupling, and were defined in terms of a fundamental OCL concept related to it: 'the navigation'. Whenever a navigation occurs in an OCL expression the modeler should apply a tracing cognitive technique interrupting the chunking of an OCL expression in order to find out other chunks through the relationships which are navigated. High values of DN indicates that the objects of a class where the OCL expression is specified, is highly coupled of distant objects in the diagrams. In [WK03] is recommended to limit the object's knowledge to only its direct surroundings. That means, DN should be kept as lower as possible. Through experimentation we obtained that OCL expression comprehensibility and modifiability are dependent on how far objects coupled to the contextual instance are (DN), and also how many different objects are coupled to the contextual instance (NNC). Moreover, the interaction of DN and NNC affects the comprehensibility of OCL expressions but not the modifiability where each principal factors affects but separately.

Through experimentation (see summary results in Table 8.25) we can conclude that both measures (DN and NNC) affect the COM and MOD Time and efficiency of OCL expressions. The interaction of DN and NNC affects the COM Time and COM efficiency, but not the MOD Time and MOD efficiency.

As Cant et al. states [CHSJ94], [CJHS92] tracing and chunking techniques are synergically and concurrently applied. We conducted a new family of experiment in order to go further studying the influence of coupling on OCL expression comprehensibility and modifiability. We design a family of experiments, similar to the first one explained in section 8.2, but improving its design (mainly its experimental objects).

8.4 Second Family of Experiments

8.4.1 Experiment Preparation (F_1)

The purpose of this family of experiment is to ascertain if any relationship exists between OCL expression import-coupling and two maintainability sub-characteristics: comprehensibility and modifiability of OCL expressions. Our belief is that the inner definition of OCL as a textual add-on to UML models makes that within an expression being possible to refer to UML artifacts but not the other way around, so, the coupling locus of impact of an OCL expression is primary of import-coupling. As

Table 8.25: Summary of the Results (1^{st} Family of Experiments)

Experiment	**COM Time**			**MOD Time**		
	DN	NNC	Interaction	DN	NNC	Interaction
Experiment at UNC	Yes			Yes		
Experiment at UCLM			Yes		Yes	
Experiment at UA			Yes	Yes	Yes	
All the Experiments	Yes	Yes	Yes	Yes	Yes	
Experiment	**COM Eff**			**MOD Eff**		
	DN	NNC	Interaction	DN	NNC	Interaction
Experiment at UNC	Yes	Yes	Yes	Yes		
Experiment at UCLM					Yes	
Experiment at UA	Yes		Yes	Yes	Yes	
All the Experiments	Yes	Yes	Yes	Yes	Yes	

we defined in section 3.2.1.2 import-coupling a criterion defined in [BDW99] which focus on the client entity in the client-supplier relationship defining a coupling connection. Within this context, modelers should be aware that OCL expressions with high import-coupling operate in large context and to comprehend the meaning of the expression modelers need to know all the services an OCL expression relies on. We believe that high import-coupling (artifacts) can influence on the cognitive complexity of modelers (subjects), and high cognitive complexity leads to OCL expressions will exhibit undesirable external qualities, such as less comprehensibility or reduced modifiability.

Within import-coupling, object coupling -the more complex software attribute in OO systems- is an important aspect that should be reduced as a good practice of any design of high quality [BBD01]. Usually these objects are linked in OCL expressions through the use of navigations. A navigation creates coupling between the objects involved and the coupled objects are usually manipulated through collections and its collection operations (to handle its elements). So three important aspect of the import-coupling in OCL expressions is specified in terms of navigation, collection and collection operations concepts.

8.4.2 Context Definition ($\mathbf{F}_2$)

The first experiment took place in April 2004 and it was replicated twice in October and November 2004 respectively.

- **UA Experiment (April 2004)**: We invited the third-year students of Computer Science at the University of Alicante (UA, Spain) to do a short seminar

Table 8.26: Measures Used for Measuring IV (2^{nd} Family of Experiments)

Measure	Measure Description
NNR	Number of Navigated Relationships
NAN	Number of Attributes referred through Navigations
NNC	Number of Navigated Classes
WNCO	Weighted Number of Collection Operations
DN	Depth of Navigations
WNN	Weighted Number of Navigations
NEI	Number of Explicit Iterator variables
NKW	Number of OCL KeyWords
NES	Number of Explicit Self
NCO	Number of Comparison Operators

about OCL (only 5 hours) and to do an experiment as part of the seminar. Sixty undergraduate students agreed to take part in a course. They were motivated to participate in the experiment because they would be able to obtain an extra point in the final score of the Software Engineering course if and only if they completed a test. The extra point we gave them was only dependent on finishing the exercise, not on how the exercise was done. The collected data was called 'UAE'.

- **ULM Experiment (October 2004)**: Twenty six students who participate in a course of the Eighth International School of Computer Science (celebrated in Argentina in La Matanza University) were the subjects of the first replica. The duration of the course was 20 hours and during the last two hours we ran the experiment replica. The subjects were undergraduate students of different universities, graduate students and teachers. The data obtained in this replication, was called 'ULME' data.

- **UACh Experiment (November 2004)**: Twenty nine students of fifth year enrolled on a Software Engineering course of the Austral University of Chile participated in a course of 20 hours about OCL. As an inducement to do the course, students were informed that they would do a test and its result would be considered as a point of a the course of Software Engineering. The collected data was called 'UAChE'.

Table 8.36 shows a brief description of the profile of the subjects, all the quantities are in years. We think that the course duration, the inducement for students to take the course, and their profile could have affected the experimental results.

Table 8.27: Measure Values for each Model (2^{nd} Family of Experiments)

object	Tracing						Chunking			
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO	ObjClass
Model1	1	1	1	1	2	1	1	1	0	LC
Model2	1	1	1	1	2	1	1	1	1	LC
Model3	2	2	2	1	2	0	0	2	1	LC
Model4	3	2	6	4	3	0	1	2	0	MC
Model5	3	2	5	4	1	0	1	2	1	MC
Model6	3	2	6	4	3	0	1	2	0	MC
Model7	2	2	3	4	7	2	2	2	1	HC
Model8	3	3	3	3	5	2	2	1	1	HC
Model9	3	3	6	3	8	1	3	1	1	HC

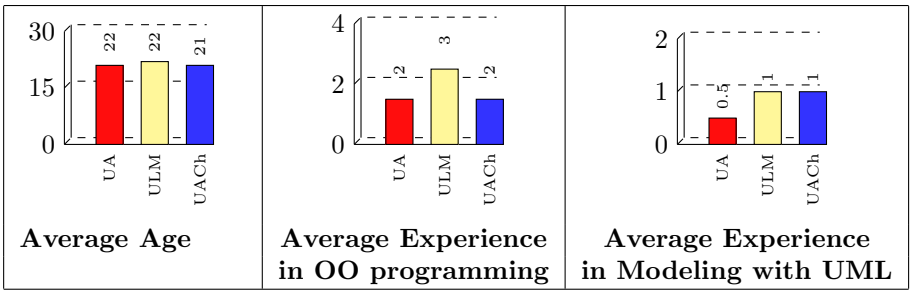


Table 8.28: Subject Profile (2^{nd} Family of Experiments)

8.4.3 Design Framework of the Second Family of Experiments (F_3)

In this section we will summarize the main experimental process steps common to the three experiments.

- **Independent and dependent variables**: The independent variable (IV) is the coupling defined in OCL expressions. It was measured through the measures shown in Table 8.26. We used NNR , NNC, WNN, DN, WNCO, NES and NAN measures, because in all of them an aspect of the navigation concept is captured in its intent. We also used the NEI measure which is related to the collection operation iterator variables, and allows us to define the context inside the collection operations. The rest of the measures NWK (number of keywords) and NCO (number of comparison operators) were not related to collection operations but they are needed to define simple OCL expressions. Because we are not interested in studying the last two measures we try to keep their value as constant as possible. For example all the OCL expressions used in experimental object are defined with three OCL keywords. The dependent

variables (DVs) are two maintainability sub-characteristics: comprehensibility and modifiability.

- **Experimental Material:** The experimental objects were nine UML/OCL combined models, each model having an OCL expression. Since we wanted to have objects of different complexity we designed them covering a wide range of the measure values (except in the case of NES, NWK, and NCO). But in reality, it is impossible to cover all of the possible combination of measures values. Fifteen model were initially designed, but we thought that some models were quite similar, and the fact of having many models of the same complexity could bias the experiment result. For that reason we carried out a hierarchical clustering of the 15 models to group them into three groups: Low, Medium or High Complexity (we identify each complexity by using the acronyms LC, MC, HC respectively). This clustering was run using the measure values. Finally, we obtained three models of each group (see Table 8.27). The clustering provided us with an objective classification of the UML/OCL models, which we called ObjClass.

- **Tasks:** During the test each subject had to perform three tests. The tests have the following required tasks:

  - **COM-Tasks**: The subjects had to answer a questionnaire consisting of 4 questions that reflected whether or not they had understood the OCL expression attached to the class diagram.
  - Modifiability tasks (**MOD-Tasks**): The subjects had to modify the OCL expressions according to a new requirement expressed in natural language.
  - Rating Tasks: After finishing any task the subject uses the same scale of five linguistic labels used in the experiment of section 8.2. This rate indicates the perception of the subjects of how complex it was for them to do COM-Tasks or MOD-Tasks. The collected data was called **COM SubComp** or **MOD SubComp**. This information is vital to estimate the cognitive load of subjects dealing with artifacts.

  All three tests assigned to any subject had three different complexities, ie. HC, MC or LC, which means there is no subject doing two tests of the same complexity. However, the tests were randomly assigned to the subjects. We identify as C_1 the collection of the first tests performed by all the subjects, C_2 the second collection, and C_3 the third one.

- **Experiment Hypotheses**: We formulated different hypotheses along with distinct beliefs:

Table 8.29: Synopsis of Hypotheses and the Statistical Test Applied (2^{nd} Family of Experiments)

Relation between	**Efficiency** **COM Eff** **MOD Eff**	**Time** **COM Time** **MOD Time**	**Subjective Complexity** **COM SubComp** **MOD SubComp**
OCL expr. measures	Hypotheses 1		Hypotheses 2
	Test: Spearman		Test: Spearman
COM SubComp **MOD SubComp**	Hypotheses 4	Hypotheses 3	
	Test: τ Kendall	Test: τ Kendall	

- **Belief 1**: The structural properties related to import-coupling in OCL expressions influences the degree of correctness of the performed Tasks per time, i.e. the subject's efficiency (COM or MOD Eff). The hypotheses is:
  Hypotheses 1: $H_{0,1}$ There is no significant correlation between the OCL expression measures related to import-coupling and their COM Eff /MOD Eff. $H_{1,1} = \neg H_{0,1}$

- **Belief 2**: The structural properties related to import-coupling in OCL expressions influences the subjective rate provided by subjects (COM SubComp or MOD SubComp) tasks. If so, we will be able to find an early indicator of the subject's cognitive load. The hypotheses is:
  Hypotheses 2: $H_{0,2}$ There is no significant correlation between the OCL expression measures related to import-coupling and the SubComp Eff. $H_{1,2}= \neg H_{0,2}$

- **Belief 3**: The COM (or MOD) Time is a valued factor that influences the subjective criteria of subjects when they have to rate tasks. For example, we expect subjects to rate time-consuming comprehensibility tasks as 'quite difficult to understand' or 'barely understable'.
  Hypotheses 3: $H_{0,3}$ The subjective complexity (SubComp) is not correlated with the COM and MOD Time; otherwise $H_{1,3}$: $\neg H_{0,3}$

- **Belief 4**: We believe the degree of correctness of the tasks performed per time, i.e. the COM or MOD Eff, could be an indicator of the subjective complexity of subjects when they have to rate tasks.
  Hypotheses 4: $H_{0,4}$ The subjective complexity (COM or MOD SubComp) is not correlated with the COM and MOD Eff; otherwise $H_{1,4}$: $\neg H_{0,4}$

8.4.4 Data Analysis and Interpretation (F_5)

In this section we will summarize the main aspects of the analysis. As previously mentioned we have three different observations for each subject, these three observation for each subject have a different complexity (HC, MC or LC). C_i represents the collection of the i-tests performed by all the experimental subjects. Now we will describe which statistic test we used. Because all the hypotheses defined in the last section are concerned with dependency degree between two variables, a correlation coefficient can be used. Coefficients such as Spearman or Tau of Kendall, work with pairs of observation, (X_i, Y_j), over n-objects (in our case 9 diagrams), but observations must be independent. That means for example, if we study a dependent variable, said COM Eff, of the subject 'j' in the i-diagram we are not allowed to consider any other observation of the same j-subject. So, the correlations of the formulated hypothesis are tested for each C_i. In same way, studying the correlation for each C_i will indicate whether our hypotheses are dependent on the learning curve of subjects during the experiment. The analysis of the empirical data is laid out as follows. First we will make a descriptive and exploratory study (section 5.1). In section 5.2, we will study hypotheses 1, the correlation between the proposed measures and the dependent variable is studied, in order to discover whether the former could predict the latter. In this section we also study the correlation between the cognitive aspects of subjects (SubComp) and the dependent variable. Afterwards (section 5.3) we study whether the time has influenced the students to rate the OCL expressions within UML/OCL models, or if their efficiency has a correlation with the SubComp (section 4.3).

8.4.4.1 Descriptive and Explanatory Studies

The fact that the dependent variables do not follow a normal distribution was corroborated using the Shapiro Wilk tests. Table 8.30 shows some descriptive statistics. At the top of Figure 8.7 we depict the COM and MOD Time as time passed. As previously described, the set of C_i represents the order of the performed tasks, which allows us to show how the time spent on each task decreases as new tasks are solved by subjects. COM and MOD Time decrease during the experiment's execution. In the case of COM Eff and MOD Eff, we expected the subject rump up efficiency but it does not improve as time goes on, except in the UA experiment for COM Eff. However if we arrange the collected data according to the objective classification (see bottom of Figure 8.7) the COM Time and COM Eff improves as we diminish the complexity. This is not the case for MOD Time and MOD Eff because the Medium Complexity (MC) tasks were more difficult to modify than the tasks corresponding to High Complexity. This situation occurs in the three experiments. The main difference between MC and HC models is that in the former the complexity is mainly based on combined navigations, (see the value of WNN) whereas in the latter the

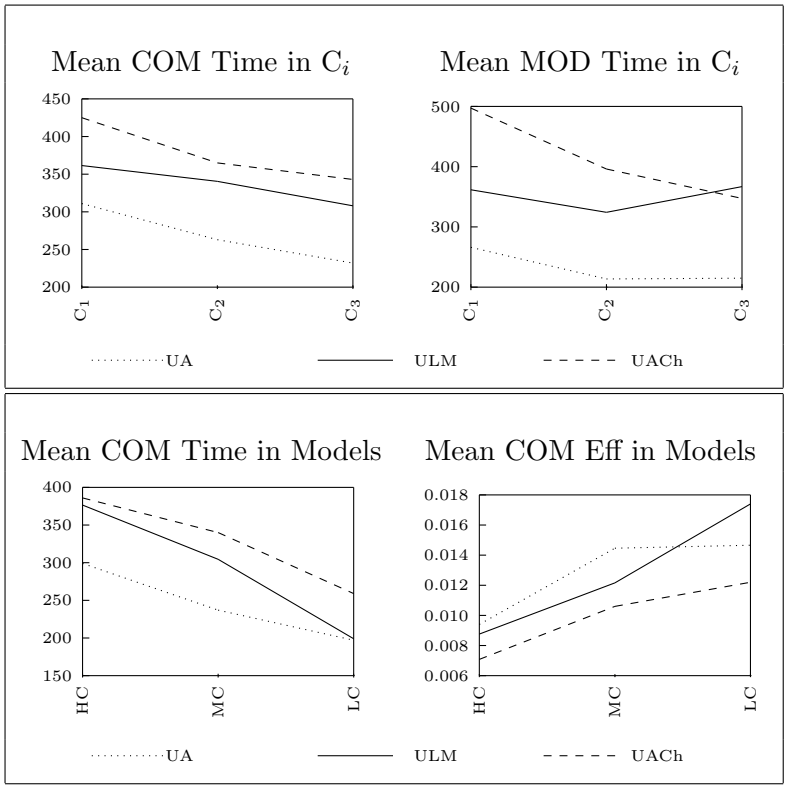


Figure 8.7: Descriptive Analysis (2^{nd} Family of Experiments)

complexity is mainly based on an intertwining collection operations (see the value of WNCO). We believe that for the subjects it was more difficult to identify and trace which relationships they should use (its rolename, attribute name, etc.) in MOD Tasks, instead of identifying which operation collections should be used to modify the expression. The descriptive statistics for mean COM Time and mean MOD Time have higher values in UAChE compared with ULME and UAE, and between the last two, the smallest mean values are from UAE. Chilean students have low experience in UML, so they required a certain amount of necessary extra time to undertake any task. Although UAE presents a higher mean COM Time than ULME their COM Eff are similar, if we compare the C_i.

8.4.4.2 Testing Hypotheses 1 and 2

To test the first two hypotheses, a correlation analysis was performed using Spearman's correlation coefficient with a level of significance $\alpha = 0.05$, which means the level of confidence is 95% (i.e. the probability that we reject H_0 when H_0 is false is at least 95%, which is statistically acceptable). Tables 8.31 and 8.33 show the Spear-

man coefficient rho between measures and efficiency' DVs, and between measures and SubComp variables respectively. The conclusions are:

- Hypotheses 1:
  - According to the Table 8.31 we draw the following conclusions:
    - The NNC, WNCO and NEI measures have several correlations with the COM Eff in the UAE and UAChE. This is logical, meaning that the number of classes (NNC), the number of collection operation (WNCO) and the number of collection operation's iterator variables (NEI) influences the subjects' efficiency. This influence seems to be independent of the order of the tasks performed for UAE because we find a correlation for all C_i.
    - The stronger correlation is between WNN and the Eff MOD in the UA experiment. This correlation reveals that UA students were less efficient when values of the weighted number of navigation WNN were high. Another relevant situation is how NNR become more correlated with the MOD Eff according to the order of performed tasks (Ci).
  - According to Table 8.32, which show the analysis for the family, the conclusions are:
    - NNR, NNC, WNN, WNCO, NAN and NEI have several correlations with the COM Eff.
    - NNR, WNN, DN, NES and NCO have several correlations with the MOD Eff.
- Hypotheses 2:
  - According to Table 8.33 the conclusions are:
    - There are few correlations in UAE and UACh. From the set of measures that present a correlation none of them are correlated in more than one C_i.
    - NNR, WNN and DN are correlated with the subjective complexity of the subject for MOD tasks. DN has the stronger correlation in UAE, independent of the tasks' order. However in this experiment, the correlation of NNR and WNN is stronger as time goes on.
  - From the analysis of the family, see Table 8.34, we can draw the following conclusions:
    - There are no significant correlations between measures and COM SubComp.
    - NNR, WNN and DN are correlated with the SubComp of the subject for MOD tasks.

Table 8.30: Mean COM/MOD Eff and COM/MOD Time during the Time (2^{nd} Family of Experiments)

	UAE			ULME			UAChE		
	IQR	Mean	SE	IQR	Mean	SE	IQR	Mean	SE
COM Eff C_1	0.035	0.012	0.007	0.026	0.013	0.008	0.05	0.011	0.01
COM Eff C_2	0.027	0.014	0.007	0.045	0.015	0.012	0.059	0.012	0.01
COM Eff C_3	0.059	0.017	0.01	0.034	0.014	0.008	0.037	0.012	0.008
MOD Eff C_1	0.03	0.007	0.009	0.021	0.006	0.006	0.03	0.007	0.008
MOD Eff C_2	0.05	0.006	0.011	0.05	0.008	0.011	0.05	0.006	0.01
MOD Eff C_3	0.033	0.006	0.008	0.033	0.007	0.009	0.031	0.006	0.009
COM Time C_1	822	311.88	152.92	567	361.57	188.84	1038	425.538	255.15
COM Time C_2	455	263.08	103.41	644	340.88	182.29	956	365.89	208.02
COM Time C_3	703	232.15	112.08	505	308.73	150.70	871	343.28	180.64
MOD Time C_1	749	266.1	162.93	998	361.61	222.79	1775	497.25	415.22
MOD Time C_2	611	213.6	125.74	807	324.23	210.22	1198	396.10	306.53
MOD Time C_3	843	214.96	130.36	729	366.92	210.58	1823	356.83	336.00

8.4.4.3 Testing Hypotheses 3 and 4

In order to test the 3^{rd} and 4^{th} hypotheses, we study the correlation using measures for ordinal data. We transform the involved variables in the following way:

- Subjective complexity (SubComp): For carrying out the data analysis of linguistic labels we assigned numbers to each label: ranging from 1 (assigned to 'Easily understandable/modifiable') to 5 (which correspond with 'Barely understandable/ modifiable').
- COM and MOD Time: In order to relate these variables of ratio scale we have changed the ratio scale to an ordinal scale.

After the data was transformed we used a Kendall's tau coefficient to analyze the correlation of $H_{0,3}$ and $H_{0,4}$. The statistics for ordinal measures are summarized in Table 8.35 which allow us to conclude the following:

- COM SubComp and COM Time: In the UA and UACh experiments there is a statistically significant relationship between the SubComp variable and the COM Time. However in the ULME we only found correlation in one trial (C_2).
- MOD SubComp and MOD Time: Regarding the MOD Time, almost the same results as the previous case are obtained. Only in UAE and UAChE is there a statistically significant relationship between the SubComp variable and the MOD Time.

Table 8.31: Correlation between Measures and COM/MOD Eff (2^{nd} Family of Experiments)

Spearman correlation between measures and COM Eff									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
UAE C_1	0.058	**0.001**	0.2	0.103	**0.011**	0.155	0.128	0.179	0.579
UAE C_2	0.034	**0.041**	**0.026**	0.418	**0.026**	0.646	**0.029**	0.202	**0.047**
UAE C_3	0.174	**0.000**	0.409	0.686	**0.000**	**0.000**	**0.000**	0.082	0.127
UAChE C_1	0.094	**0.000**	0.552	0.999	**0.010**	**0.004**	**0.001**	**0.035**	**0.051**
UAChE C_2	0.161	0.062	0.083	0.239	**0.002**	0.173	**0.010**	0.500	0.532
UAChE C_3	**0.025**	**0.04**	0.187	0.103	0.322	0.624	0.297	0.616	0.538
ULME C_1	0.081	0.053	0.076	0.128	**0.021**	0.103	**0.013**	0.451	0.952
ULME C_2	0.063	0.393	0.115	0.112	0.488	0.894	0.197	0.781	**0.02**
ULME C_3	0.595	0.375	0.312	0.093	0.225	**0.012**	0.065	**0.004**	0.057
Spearman correlation between measures and MOD Eff									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
UAE C_1	0.201	0.403	**0.061**	0.000	0.329	0.061	0.316	**0.000**	**0.015**
UAE C_2	0.129	0.86	**0.112**	0.004	0.329	0.453	0.075	0.77	0.336
UAE C_3	0.36	0.322	0.154	**0.001**	0.184	**0.015**	0.146	**0.000**	**0.003**
UAChE C_1	0.117	0.364	0.685	0.413	0.532	0.907	0.953	0.954	0.751
UAChE C_2	**0.013**	0.497	**0.013**	**0.004**	0.418	0.286	0.387	0.355	**0.024**
UAChE C_3	**0.003**	0.089	0.115	0.246	0.559	0.446	0.868	0.918	0.201
ULME C_1	0.166	0.374	0.479	0.057	0.903	0.68	0.977	0.241	0.831
ULME C_2	**0.023**	0.559	0.057	**0.004**	0.684	0.104	0.909	0.07	0.122
ULME C_3	0.07	0.403	0.05	0.225	0.929	**0.02**	0.34	0.101	0.824

- SubComp and COM/MOD Eff: There is a statistically significant relationship between SubComp and COM Eff and, between SubComp and MOD Eff variables, in the case of UA and UACh experiments. In the ULME we found that SubComp is correlated with the MOD Eff.

8.4.4.4 Validity Evaluation

A fundamental question concerning any experimental results is how valid they are. We had considered a number of validity issues inherent to this family of experiments. However for the sake of brevity we only describe the more important threats.

- **Threats to the External Validity**: The remarks of section 8.1 are valid for this family. Besides, we believe that subjects of the ULM experiment were not homogeneous. In fact, they were students coming from different universities, professionals and teachers also participated in the course. The subjects'

Table 8.32: Correlation between Measures and COM/MOD Eff for the Family (2^{nd} Family of Experiments)

Spearman correlation between measures and UAE + UAChE + ULME COM Eff									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
C_1	**0.002**	**0.000**	**0.038**	0.081	**0.000**	**0.000**	**0.000**	0.416	0.163
C_2	**0.002**	**0.006**	**0.002**	0.066	**0.001**	0.388	**0.000**	0.160	**0.004**
C_3	0.069	**0.000**	0.447	0.672	0.000	**0.000**	**0.000**	**0.040**	0.090

Spearman correlation between measures and UAE + UAChE + ULME MOD Eff									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
C_1	**0.015**	0.854	**0.046**	**0.000**	0.332	0.252	0.423	**0.001**	**0.030**
C_2	**0.000**	0.460	**0.000**	**0.000**	0.336	0.499	0.078	0.125	**0.008**
C_3	**0.002**	0.521	**0.004**	**0.000**	0.229	**0.001**	0.184	**0.000**	**0.004**

Table 8.33: Correlation between Measures and COM/MOD SubComp (2^{nd} Family of Experiments)

Spearman correlation between measures and COM SubComp									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
UAE C_1	0.175	**0.007**	0.296	0.362	0.062	0.439	0.212	0.260	0.061
UAE C_2	**0.038**	0.203	**0.04**	0.221	0.141	0.877	0.256	0.588	**0.006**
UAE C_3	0.126	0.058	0.061	0.072	**0.002**	0.127	**0.007**	0.931	0.291
UAChE C_1	0.506	0.100	0.817	0.728	0.175	0.053	0.179	0.203	0.834
UAChE C_2	0.663	0.914	0.651	0.53	0.608	0.832	0.580	0.763	0.053
UAChE C_3	**0.014**	0.162	0.069	**0.022**	0.726	0.728	0.615	0.310	0.379
ULME C_1	0.222	0.132	0.264	0.51	0.343	0.709	0.187	0.909	0.591
ULME C_2	0.133	0.608	0.395	0.123	0.954	0.831	0.222	0.76	0.278
ULME C_3	0.790	0.439	0.511	0.328	0.229	0.018	0.048	0.017	0.096

Spearman correlation between measures and MOD SubComp									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
UAE C_1	0.064	0.749	0.053	**0.000**	0.676	0.388	0.802	**0.008**	0.421
UAE C_2	**0.012**	0.372	**0.025**	**0.000**	0.291	0.639	0.055	0.704	**0.048**
UAE C_3	**0.003**	0.317	**0.009**	**0.008**	0.446	**0.015**	0.586	0.064	0.136
UAChE C_1	0.356	0.255	0.977	0.716	0.352	**0.047**	0.167	0.159	0.424
UAChE C_2	0.051	0.635	0.146	**0.019**	0.960	0.222	0.859	0.122	0.088
UAChE C_3	0.016	0.530	**0.032**	**0.035**	0.892	0.239	0.873	0.427	**0.011**
ULME C_1	**0.070**	0.437	**0.03**	**0.015**	0.396	0.917	0.461	0.289	0.084
ULME C_2	**0.027**	0.534	**0.041**	0.000	0.643	0.532	0.301	0.169	0.466
ULME C_3	0.693	0.921	0.756	0.685	0.235	0.839	0.741	0.850	0.641

Table 8.34: Correlation between Measures and COM/MOD SubComp for the Family (2^{nd} Family of Experiments)

Spearman correlation between measures and UAE + UAChE + ULME COM SubComp									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
c_1	0.088	**0.000**	0.377	0.500	**0.014**	0.054	**0.030**	0.972	0.087
c_2	0.818	0.585	0.845	0.243	0.832	0.766	0.680	0.280	0.160
c_3	0.486	0.148	0.499	0.875	0.714	0.514	0.981	0.254	0.122

Spearman correlation between measures and UAE + UAChE + ULME MOD SubComp									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
c_1	**0.000**	0.120	**0.015**	**0.009**	0.653	0.542	0.513	0.325	0.138
c_2	0.357	0.707	0.834	0.927	0.323	0.319	0.398	0.852	0.851
c_3	**0.000**	0.111	**0.000**	**0.000**	0.919	**0.011**	0.740	0.051	**0.001**

heterogeneity could explain why the results obtained in ULME were quite different from the other two experiments in most of the hypothesis. We have carefully considered other factors such as the knowledge of the universe of discourse among the material used, learning effects as well as subject motivation and other factors (plagiarism, fatigue effects).

- **Threats to the Conclusion Validity**: In the conclusion validity we want to make sure that our conclusions are statistically valid. Two threats can be described. Firstly, it was not possible for us to plan the selection of a population sample by using any of the common sampling techniques, so we decided to take the whole population of the available classes in software engineering courses of universities that collaborate with our research. Secondly, the quantity and quality of the data collected and the data analysis were enough to support our conclusion, mainly as described in previous sections, concerning the existence of a statistical relationship between independent and dependent variables.

- **Threats to Construct Validity**: Idem the remarks of the first family for this criterion.

- **Threats to Internal Validity**: The internal validity is the degree of confidence in a cause-effect relationship between factors of interest and the observed results. We had alleviated some issues: knowledge of the universe of discourse among the material used, accuracy of response, learning effects as well as subject motivation and other factors (plagiarism, fatigue effects).

Table 8.35: Correlation between Subjective Complexity (SubComp) and COM/MOD Time, COM/MOD Eff (2^{nd} Family of Experiments)

	UAE			ULME			UAChE		
	Coef.	p-value	size	coef.	p-value	size	coef.	p-value	size
COM SubComp-COM Time C_1	**.243**	.015	60	.136	.406	26	**.439**	.000	39
COM SubComp-COM Time C_2	**.269**	.007	60	**.401**	.010	26	**.430**	.001	39
COM SubComp-COM Time C_3	**.376**	.000	60	.060	.702	26	**.471**	.000	39
MOD SubComp-MOD Time C_1	**.366**	.000	60	**.438**	.005	26	**.289**	.018	39
MOD SubComp-MOD Time C_2	**.277**	.005	60	.251	.103	26	**.296**	.018	38
MOD SubComp-MOD Time C_3	.172	.086	58	.105	.503	26	.281	.025	37
COM SubComp-COM Eff C_1	**-.317**	.001	60	-.111	.497	26	**-.458**	.000	39
COM SubComp-COM Eff C_2	**-.300**	.002	60	**-.401**	.010	26	**-.519**	.000	39
COM SubComp-COM Eff C_3	**-.411**	.000	60	-.165	.293	26	**-.452**	.000	39
MOD SubComp-MOD Eff C_1	**-.423**	.000	60	**.436**	.007	26	**-.544**	.000	39
MOD SubComp-MOD Eff C_2	**-.439**	.000	60	**-.544**	.001	26	**-.428**	.002	38
MOD SubComp-MOD Eff C_3	**-413**	.000	58	**-.355**	.030	26	**-5.12**	.000	37

8.4.5 Conclusions of the Second Family of Experiments (F_6)

We launched a family of experiments in order to ascertain whether any relationship exists between the coupling defined in OCL expressions and the comprehensibility and modifiability of OCL expressions. The experiment was run at the University of Alicante (UA) with undergraduate students, and was replicated twice at the University of La Matanza (ULM) and Austral University of Chile (UACh). In order to study the comprehensibility and modifiability of the OCL expression we have considered not only the time subjects spent on tasks related to this activities, but also their efficiency and their subjective perception of their activities. We think that quantitative (COM and MOD Eff) and qualitative (subject's rating of their cognitive load) information is important to obtain an empirical validation. Through a thorough analysis of the collected data of the experiment and its two replicas we can summarize the obtained results as follows:

- We have used as experimental objects nine models consisting of a UML class diagram and one attached OCL expression. We obtain an objective classification of them according to a hierarchical cluster in order to obtain balanced models of different complexity.

- From descriptive analysis, we can conclude that Mean COM Time diminishes as time passes. Moreover, the mean COM and MOD Time is different if we arrange the collected data according to their complexity. The higher the complexity of the OCL expression, the greater is the COM Time spent by subjects and the lower their efficiency. However, models of Medium complexity

(MC) were more difficult to modify for the subjects than models of High complexity (HC). The former (MC models) were models in which students should have traced from different navigations. In the latter (HC models), students should identify which collection operations they have to use and how to combine them. Their experience in collection operations and the difficulty of tracing in class diagrams influences this situation.

- We found that there is a statistically significant relationship between many measures, especially those related to tracing, and the efficiency of comprehensibility and modifiability. For example, the number of classes (NNC) used in navigations, the number of collection operations (WNCO) and the number of collection operation's iterator variables (NEI) influences the subjects' COM Eff. The number of navigations used in navigation (NNR), the weighted number of navigations (WNN) and more importantly the length of navigations (DN) has a correlation with the cognitive load when subjects rate MOD Tasks.

- We realize that we are not able to conclude that only the measures related to tracing (or chunking) are correlated with efficiency, due to the fact only some of the measures related to tracing (or chunking) present a significant correlation with the efficiency. We think that this is valid because chunking and tracing are concurrently and synergically applied [CHSJ94]. So, we believe that in order to go further in a more clear explanation we should carry out an analysis in the context of mental model approach to comprehension So, next family of experiments, which indeed improve the second family of experiments, is described using mental model's theory.

- Finally, in the UA and UACh experiments the subjects' subjective ratings (comprehensibility or modifiability rating) is influenced by the time they used to understand or modify the OCL expression, ie. both times seems to affect their appreciation of the level of complexity of an OCL expression. In these two experiment the COM or MOD Eff is also correlated with the subjective complexity, in stronger way.

The MOD tasks correctness still reveals that this kind of task is difficult to perform by subjects. MOD tasks require that a subject to modify a expression according to new requirements. This activity demands more cognitive flexibility [CMF04] than comprehension tasks, and require a wide knowledge of the language.

Besides, the required modification of an OCL expression can be solved in different ways because OCL is too expressive and allows the modelers to specify in different manner the same meaning. In this way, two modelers can provide (each of them) a correct answer but the complexity of the answer of the OCL expression each provide can vary significantly, i.e. the structural properties of their correct answer could be too different. We believe that we must control this during experimentation, and we

opted to redesign MOD tasks in a new family. We think that instead of asking the subject to specify the modification of an expression according a new requirement, we could ask the subject to select from three expressions that expression which represents the modification according to the new requirement. So, we conducted a new family of experiments which is described in the following section.

8.5 Third Family of Experiments

We believe that our conclusion regarding MOD tasks of the second family of experiments were not clear enough because the correctness of the performed MOD tasks was low. Probably in the second family of experiments we expected from the subjects to write different OCL expressions (in MOD Tasks) and they were not sufficiently trained for writing expressions from scratch. This fact motivated us to undertake a new family of experiments using the material of the second family but changing their MOD tasks. The new family intends to strength the conclusions and external validity of the second family of experiments, trying to confirm if import-coupling (defined through navigations and collection operations) has really influence on the COM and MOD of OCL expressions.

8.5.1 Experiment Preparation ($\mathbf{F}_1$)

The experiment goal was to confirm if import-coupling (defined through navigations and collection operations) has really influence on the comprehensibility and modifiability of OCL expressions.

8.5.2 Context Definition ($\mathbf{F}_2$)

In order to fulfill the experiment goal we conducted a family of experiments, composed of two experiments, executed in two Spanish universities:

- **UPV Experiment (May 2005)**: Forty six students who were enrolled in a fifth-year course of Software Engineering at the Technical University of Valencia (UPV) were invited to participate in a seminar of 10 hours about OCL. As an inducement to do the course, students were informed that they would do an assessment and its result would be considered to obtain an extra credit as part of its course passing process. The collected data was called 'UPVE'.
- **UCM Experiment (June 2005)**: Third-year students of Computer Science at the Complutense University of Madrid (UCM, Spain) were invited to take

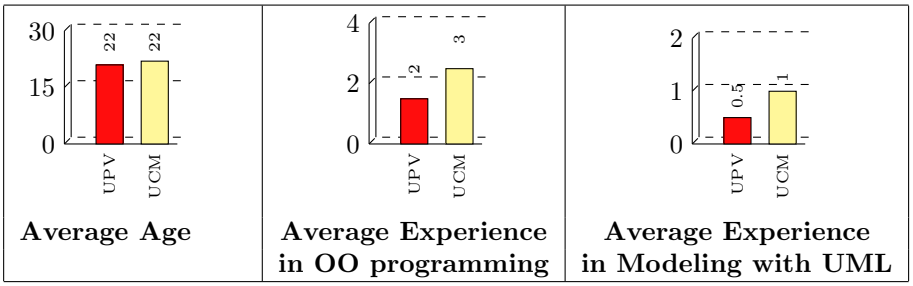


Table 8.36: Subject Profile (3^{rd} Family of Experiments)

an elective seminar course about OCL (only 10 hours). A number of elective seminar courses in different subjects are offered each semester in UCM. The participants were notified that they must do an assessment as part of the seminar course. They were motivated to participate in the assessment because they could obtain a credit of free configuration if and only if they passed a test. The collected data was called 'UCME'.

8.5.3 Design Framework of the Third Family of Experiments (F_3)

Although we followed the experimental process suggested by Ciolkowski et al. and Wohlin et al. [WRH+00], due to the experimental design is similar to the second family of experiments we will specifically focus in the main aspects of the family.

- **Independent and Dependent Variables**: The same of the previous family, see the corresponding item in section 8.4.
- **Experimental Material**: The experimental objects were six UML/OCL combined models, each model having one OCL expression. This models were six models of the nine models used in the second family of experiments. We carried out a hierarchical clustering of the six models to group them into three groups according to their measures values: Low, Medium or High Complexity (we identify each complexity by using the acronyms LC, MC, HC respectively). Finally, we obtained two models of each group, i.e. six models.

  Each model had a test enclosed including the following types of tasks:

  - **COM Tasks**: The subjects had to answer a questionnaire consisting of 4 questions that reflected whether or not they had understood the OCL expression attached to the class diagram.

- **MOD Tasks**: These tasks were different from the previous family of experiments where the subjects had to write a new OCL expression according to a new requirement. In this family, two different modifications were asked, in the form of new requirements expressed in natural language. For each modification, the subjects had to select one of three OCL expressions which represent the modification of the original OCL expressions (the one associated to the model) according to the new requirement (this is a multiple choice task). The correct OCL expression which should be selected by the subjects had the same measures' values as the original expression associated to the model, i.e. present the same structural properties. MOD Tasks of this family are included in appendix D.
- **Rating Tasks**: The subject uses a scale of five linguistic labels to rate the tasks. The scale is the same as the previous family of experiments.

We assigned to each subject the six tests. The first three tests (and the second three ones) assigned to the subjects had a model of different complexity, i.e. HC, MC or LC models. However, the tests were assigned to the subjects in such a way that there were no two subjects doing the six tests in the same order. We identify as C_1 the collection of the first tests performed by all the subjects, C_2 the second collection, and so on. It is important to notice, that our purpose is that the six models were examined by the same number of subjects in each C_i. That was possible in the UPV experiments but in the UCM there were not enough subjects for balancing the models.

The independent variable was measured through the measures used in the previous family, i.e. NNR, NNC, WNN, DN, WNCO, NES, NEI, NKW, NCO and NAN measures.

We think that the time each subject spent doing each required tasks (i.e., COM Time and MOD Time) is not the most accurate measure for studying their relationships with measures (as IVs). Therefore we used the same two measures of the DVs as the one defined in the previous family (COM and MOD Eff).

Moreover, through the rating tasks we obtained two subjective measures called COM Subjective Complexity (COM SubComp) and MOD Subjective Complexity (MOD SubComp), respectively. These measures are essential to estimate the cognitive load of subjects when dealing with UML/OCL combined models.

- **Experiment Hypotheses**: We formulated different hypotheses along with distinct beliefs. We explain them in the following and we summarized them in Table 8.37:

- **Belief 1**: The Efficiency of the subjects would be different for the models they should perform.
  Hypotheses 1: $H_{0,1}$ The ranks of the (COM or MOD) Eff do not differ from their expected value, i.e. the mean efficiency is the same for all the models. $H_{1,1} = \neg H_{0,1}$
- **Belief 2**: The import-coupling in OCL expressions influences the degree of correctness of the performed tasks per time, i.e. the subject's efficiency (COM or MOD Eff). The greater the import-coupling the lower the subjects' efficiency.
  Hypotheses 2: $H_{0,2}$ There is no significant correlation between the measures defined for OCL expressions (related to import-coupling) and their (COM or MOD) Eff. $H_{1,2} = \neg H_{0,2}$
- **Belief 3**: The import-coupling in OCL expressions influences the subjective rate provided by subjects (COM SubComp or MOD SubComp) tasks. This means that if the import-coupling increases the subjects perceive the tasks as more difficult.
  Hypotheses 3: $H_{0,3}$ There is no significant correlation between the OCL expression measures related to import-coupling and the (COM or MOD) SubComp. $H_{1,3} = \neg H_{0,3}$
- **Belief 4**: The subjective criteria of subjects when they have to rate tasks has been influenced by the COM (or MOD) Time. For example, we expect subjects to rate time-consuming COM tasks as 'quite difficult to understand' or 'barely understandable'.
  Hypotheses 4: $H_{0,4}$ The COM or MOD SubComp are not correlated with the COM or MOD Time. $H_{1,4}$: $\neg H_{0,4}$
- **Belief 5**: We believe the degree of correctness of the tasks performed per time, i.e. the COM or MOD Eff, could be an indicator of the subjective rating given by the subjects about the complexity of the required tasks. This means that the perception of the subjects about the complexity of the tasks is influenced by their efficiency when performing such tasks.
  Hypotheses 5: $H_{0,5}$ The COM or MOD SubComp are not correlated with the COM or MOD Eff . $H_{1,5}$:$\neg H_{0,5}$

8.5.4 Data Analysis and Interpretation (F_5)

First we will carry out a descriptive and an exploratory study. Later on, we will test the formulated hypotheses. As all the formulated hypotheses are concerned with dependency degree between two variables, a bi-variate correlation analysis was used. In Table 8.37 we also explain the hypotheses and the correlation test used, the tests are similar to those applied in the the second family (see section 8.4.4).

Table 8.37: Synopsis of Hypotheses and the Statistical Test Applied (3^{rd} Family of Experiments)

	Efficiency	**Time**	**Subjective Complexity**
Relation between	**COM Eff MOD Eff**	**COM Time MOD Time**	**COM SubComp MOD SubComp**
	Hypotheses 1		
	Test: Chi-square Test		
OCL expr. measures	Hypotheses 2		Hypotheses 3
	Test: Spearman		Test: Spearman
COM SubComp MOD SubComp	Hypotheses 5	Hypotheses 4	
	Test: τ Kendall	Test: τ Kendall	

So, the descriptive analysis and the correlations of the formulated hypotheses are tested for each C_i (the *i*-tests performed by all the experimental subjects), ranging i from 1 to 6. Studying the correlation for each C_i was not possible for the UCM experiment, due the small size of the population and consequently the small number of tests analyzed. Moreover, in UCM the assigned tests in each C_i were not correctly balanced. So we only test the five hypotheses for the UPV experiment and we use only the UCME data for validating the results obtained in the UPV experiment. For the validation we will use the whole set of data (UPVE and UCME) and we test the five hypotheses again. Previous to doing the analysis for the whole set of data obtained in both experiment we carried out statistical tests to prove that this was possible.

8.5.4.1 Descriptive and Exploratory Studies for UPVE Experiment

We depict in left-top side of Figure 8.8 the subjects' efficiency along with each C_i. In this figure we show that subjects are more efficient in MOD tasks (dash line) than in COM tasks (solid one). Indeed, the subjects were more efficient than in the previous family. As it was previously explained we change MOD tasks, instead of writing a new OCL expression according to a new requirement, the subjects should select one of three proposed OCL expressions which represented an OCL expression modeling the new requirement. However in both kinds of tasks subjects improved their efficiency as time goes on. In relation to the subjective complexity (SubComp) during the time, see right-top size of Figure 8.8, it seems that subjects rated MOD tasks as more difficult than COM tasks. The collected data grouped by the model complexity (LC, MC, HC) is depicted at the bottom of Figure 8.8, according to the subject efficiency and subjective complexity respectively. The COM Eff is decreasing as the complexity of models increase (note that the horizontal axis shows the two models of each complexity, from low to high complexity). The same was expected

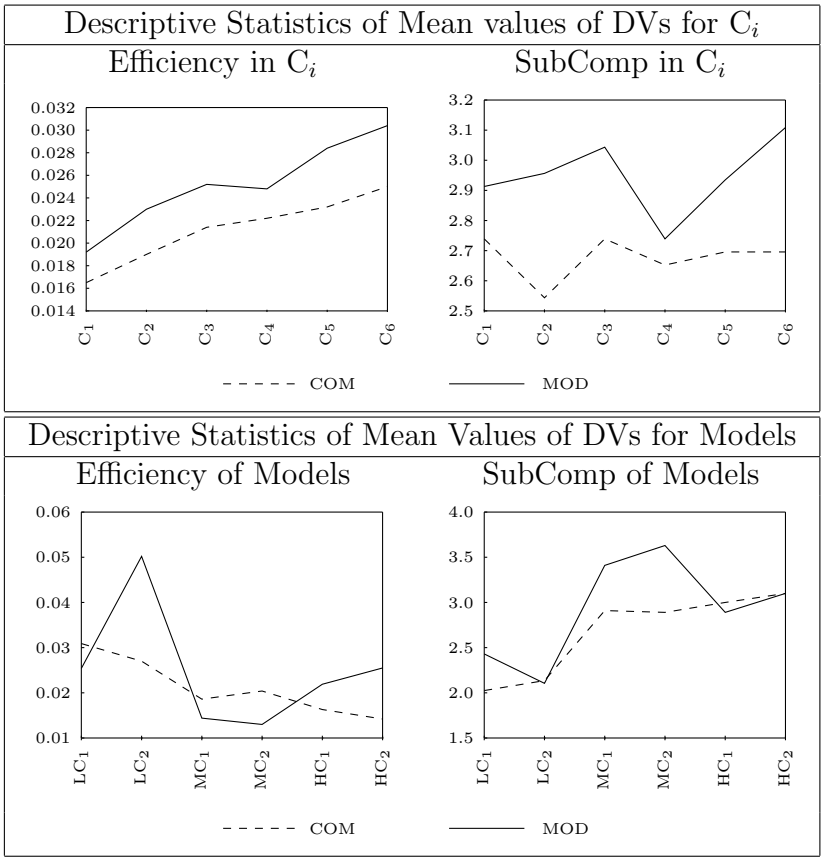


Figure 8.8: Descriptive Statistics of Mean Values of DVs (3^{rd} Family of Experiments)

to prove in MOD Eff, however as we found in a previous study 8.4, the subjects were less efficient performing MC models than HC models. The findings regarding the subjects' efficiency (grouped by the model complexity) is similar to the findings of their subjective complexity. An increasing subjective complexity is observed in COM Tasks as model complexity increase, however in MOD Tasks, MC models are rated as more difficult than HC models. The main difference between MC and HC models is that in the former the complexity is mainly based on combined navigations, (see the value of WNN) whereas in the latter the complexity is mainly based on an intertwining collection operations (see the value of WNCO). We believe that for the subjects it was more difficult to identify and trace which relationships they should use (its rolename, attribute name, etc) in MOD Tasks, instead of identifying which operation collections should be used to modify the expression. Finally, through Shapiro-Wilk tests, we found that the dependent variables do not follow a normal distribution.

Table 8.38: Mean COM/MOD Eff and COM/MOD Time (3^{rd} Family of Experiments)

	COM Eff		MOD Eff	
	UPVE	UCME	UPVE	UCME
N	46	14	46	14
Chi-square	105.035	50.122	114.091	44.531
gl.	5	5	5	5
sign.	.000	.000	.000	.000

8.5.4.2 Testing Hypotheses for UPVE Experiment

In this section we will describe how the hypotheses formulated in section 8.5.3 were tests along with its psychological explanations.

8.5.4.2.1 Testing Hypothesis for UPVE Experiment To test the first hypotheses, we used Friedman chi-square test (a non-parametric test for multiple related samples) which tests the null hypothesis that mean Eff is the same in all the considered models. The results were significant (p-values were equal to 0.000, less than 0.05), i.e. the mean efficiency of subjects when performing COM and MOD tasks is different according the complexity of the diagrams. The results of chi-square tests for UPVE are shown in the second and fourth columns of Table 8.38.

8.5.4.2.2 Testing Hypotheses 2 and 3 for UPVE Experiment To test the hypotheses 2 and 3, a correlation analysis was performed using Spearman's correlation coefficient due to this statistics measure the rank-order association between two scale or ordinal variables. We used a level of significance $\alpha = 0.05$, which means the level of confidence is 95% (i.e. the probability that we accept H_0 when H_0 is true is 0.95). We studied the correlation for independent observations, i.e. for each C_i, as it was justified at the beginning of this section. Table 8.39 summarizes the significant coefficients we found at level 0.05 between measures and DVs for the C_i. For example, reading the intersection between COM Eff's rows and NNR column, five C_i present a significant correlation in $C_2 = 0.0018$, $C_3 = 0.0000$, $C_4 = 0.0158$, $C_5 = 0.0011$ and $C_6 = 0.0022$. In relation to the second hypotheses we concluded that: (1) the NNR, NNC, WNN, DN, WNCO and NEI measures have a strong correlation with the COM Eff for almost the six models; (2) the NNR, WNN, DN and NCO have a strong correlation with the MOD Eff for almost the six models. It seems that many factors influences the efficiency of COM tasks, as classes, relationships, the navigations, the collection operations and the iterators variables, but only relationships, collection operations and the depth of navigations influences the efficiency of MOD tasks. These results are similar to the one obtained in the previous family,

Table 8.39: Correlation between Measures and COM/MOD Eff (3^{rd} Family of Experiments, 1^{st} Experiment)

Spearman correlation between measures and UPVE COM Eff									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
C_1	0.2543	**0.0388**	0.0928	**0.0438**	**0.0003**	**0.0069**	**0.0002**	0.5852	0.5248
C_2	**0.0018**	**0.0006**	**0.0002**	**0.0001**	**0.0000**	0.1938	**0.0001**	0.3604	0.2885
C_3	**0.0000**	**0.0033**	**0.0000**	**0.0000**	**0.0001**	0.4702	**0.0126**	0.2179	**0.0004**
C_4	**0.0158**	**0.0001**	**0.0037**	**0.0195**	**0.0000**	**0.0230**	**0.0000**	0.6745	0.6215
C_5	**0.0011**	**0.0000**	**0.0004**	0.0583	**0.0000**	0.2601	**0.0002**	0.3791	0.8205
C_6	**0.0022**	**0.0000**	**0.0007**	**0.0036**	**0.0000**	0.4339	**0.0047**	0.2755	0.5546
Spearman correlation between measures and UPVE MOD Eff									
	Measures for import-coupling								
C_1	0.0553	0.3154	**0.0213**	**0.0002**	0.5806	0.3639	0.7056	0.2123	**0.0000**
C_2	**0.0013**	0.7019	**0.0001**	**0.0000**	**0.0295**	0.6063	0.0864	0.2049	**0.0000**
C_3	**0.0002**	0.8617	**0.0000**	**0.0000**	0.2342	0.0615	0.5163	0.155	**0.0000**
C_4	**0.0253**	0.9343	**0.0077**	**0.0001**	0.1503	0.6771	0.3236	0.1323	**0.0005**
C_5	**0.0058**	0.8213	**0.0014**	**0.0000**	0.1236	0.5381	0.1745	0.4385	**0.0000**
C_6	**0.0000**	0.1425	**0.0000**	**0.0000**	**0.0009**	0.7194	**0.0083**	0.4455	**0.0000**

nevertheless in the results of the current experiment we found a bigger quantity of significant coefficients for the C_i than in the previous family.

Regarding the third hypotheses (see Table 8.40): the NNR, NNC, WNN, DN, WNCO and NEI measures are correlated with the subjective complexity of the subject for COM tasks in more than four C_i (more than the half of the C_i-s they are six in total) and NNR, WNN, DN, WNCO and NCO measures are correlated with the subjective complexity of the subject for MOD tasks in more than four C_i-s.

8.5.4.2.3 Psychological Explanation of Testing Hypotheses 2 and 3. Regarding the conclusions and the distinct measures affecting COM and MOD tasks, we can add the following:

1. Problem objects (NNC, NEI), relation of problems objects (NNR, WNN, DN) and reified objects (WNCO) seems to affect the COM Eff (Hypothesis 2). However only Relation of problems objects (NNR, WNN, DN) affects MOD Eff. Almost the same set of groups affects the COM and MOD SubComp (Hypothesis 3). We think that during comprehension a broad familiarity of the expression and its contextual information should be gathered by the subjects, however during modification, the focus of the comprehension was only in relationship between problem objects.

2. Measures related to chunking and tracing affects COM tasks, and mainly trac-

Table 8.40: Spearman Correlations between Measures and COM/MOD SubComp (3^{rd} Family of Experiments, 1^{st} Experiment)

Spearman correlation between measures and UPVE COM SubComp									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
c_1	0.2168	0.369	0.0756	**0.0005**	**0.0214**	0.2122	0.0806	0.062	0.2587
c_2	**0.0223**	0.0711	**0.008**	**0.0017**	**0.0069**	0.6988	0.0507	0.1929	0.1202
c_3	**0.0013**	**0.0181**	**0.0002**	**0.0003**	**0.0006**	0.7989	**0.0097**	0.3693	0.036
c_4	**0.0049**	**0.0047**	**0.0025**	**0.0451**	**0.0018**	0.6767	**0.0119**	0.831	0.375
c_5	**0.0466**	**0.0138**	**0.043**	0.5306	**0.0211**	0.6685	**0.0420**	0.2894	0.941
c_6	**0.0042**	**0.0025**	**0.0013**	**0.0056**	**0.0002**	0.488	**0.0067**	0.5813	0.3315

Spearman correlation between measures and UPVE MOD SubComp									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
c_1	**0.0363**	0.6885	**0.012**	**0.0002**	0.2679	0.5982	0.3771	0.2865	**0.0001**
c_2	**0.0119**	0.2705	**0.0045**	**0.0048**	**0.0362**	0.8769	0.0823	0.712	**0.0188**
c_3	**0.0001**	0.3085	**0.0000**	**0.0000**	**0.0236**	0.2788	0.14	0.1273	**0.000**
c_4	**0.0024**	0.1901	**0.0011**	**0.0041**	**0.0474**	0.4929	0.1424	0.6767	**0.0041**
c_5	**0.0000**	0.1637	**0.0000**	**0.0010**	0.0856	**0.0395**	0.333	0.5398	**0.000**
c_6	**0.0000**	0.009	**0.0000**	**0.0004**	**0.0006**	0.8451	**0.0111**	0.656	**0.0123**

ing affect the MOD tasks.

8.5.4.2.4 Testing hypotheses 4 and 5 for UPVE Experiment. In order to test the 4^{th} and 5^{th} hypotheses, we transformed the variables COM SubComp and MOD SubComp, assigning numbers to the linguistic labels: ranging from 1 (assigned to 'Easily understandable/modifiable') to 5 (which correspond with 'Barely understandable/modifiable'). After the data was transformed we used a Kendall's τ coefficient to contrast the hypotheses $H_{0,4}$ and $H_{0,5}$. As the results were significant (p-values ranged from 0.000 and 0.018), see Table 8.41, we can conclude that there seem to exist a statistically significant correlation between the COM/ MOD SubComp variable and the COM/MOD Time, and between COM/MOD SubComp and COM/MOD Eff. Coefficients are negative for the efficiency and positive for Time, i.e. those tasks rated as difficult were time-consuming tasks and the subjects were less efficient. Moreover, the observed p-values of COM/MOD Eff are smaller (and coefficients are greater) than COM/MOD Time, meaning that the relationship of efficiency and the subjective complexity is stronger than the influence between time and subjective complexity.

8.5.4.2.5 Psychological Explanation of Testing hypotheses 4 and 5 The fact that the instrumentation we used for the OCL expression maintainability is correlated with the subjective complexity means that the perception of the subjects

Table 8.41: Correlation between SubComp and COM/MOD Time, and between SubComp and COM/MOD Eff (3^{rd} Family of Experiments, 1^{st} Experiment)

H4: τ Kendall	UPVE			H5: τ Kendall	UPVE		
	coef	p-value	size		coef	p-value	size
COM SubComp-COM Time C_1	.170	.072	46	COM SubComp-COM Eff C_1	-.186	.054	46
COM SubComp-COM Time C_2	**.373**	**.001**	46	COM SubComp-COM Eff C_2	**-.443**	**.000**	46
COM SubComp-COM Time C_3	**.529**	**.000**	46	COM SubComp-COM Eff C_3	**-.597**	**.000**	46
COM SubComp-COM Time C_4	**.411**	**.000**	46	COM SubComp-COM Eff C_4	**-.384**	**.000**	46
COM SubComp-COM Time C_5	**.398**	**.000**	46	COM SubComp-COM Eff C_5	**-.374**	**.001**	46
COM SubComp-COM Time C_6	**.403**	**.000**	46	COM SubComp-COM Eff C_6	**-.423**	**.000**	46
MOD SubComp-MOD Time C_1	**.358**	**.002**	46	MOD SubComp-MOD Eff C_1	**-.350**	**.003**	46
MOD SubComp-MOD Time C_2	**.304**	**.008**	46	MOD SubComp-MOD Eff C_2	**-.324**	**.005**	46
MOD SubComp-MOD Time C_3	**.539**	**.000**	46	MOD SubComp-MOD Eff C_3	**-.511**	**.000**	46
MOD SubComp-MOD Time C_4	**.271**	**.018**	46	MOD SubComp-MOD Eff C_4	**-.390**	**.001**	46
MOD SubComp-MOD Time C_5	**.392**	**.001**	46	MOD SubComp-MOD Eff C_5	**-.464**	**.000**	46
MOD SubComp-MOD Time C_6	**.426**	**.000**	46	MOD SubComp-MOD Eff C_6	**-.448**	**.000**	46

about the complexity of the tasks is influenced by the COM (or MOD) Time and also by their COM (or MOD) Eff when performing such tasks.

8.5.4.3 Testing Hypotheses for UPVE + UCME Experiments.

As we expressed in the beginning of this section the UPVE and UCME data were analysed together, because the UCME data by itself could not be analysed due to the size of the population and the unbalanced order of the assigned tests to the subjects. Before analysing the results for both experiments (UPVE+UCME) we studied if it is possible to study them together, we compare the mean Efficiency and the Mean SubComp for each model in the UPVE and UCME separately. We used the T-student Test for non-ordinal (Eff) data and Mann-Whitney Test for ordinal data (SubComp). Results were not significant, so we gather the whole data from both experiments and we test the formulated hypotheses.

8.5.4.3.1 Testing hypotheses 1 for UPVE + UCME Experiments. The Friedman test was carried out in the same way as we did when we test hypotheses 1 for UPVE only. The result was reaffirmed, i.e. the mean efficiency of subjects when performing COM and MOD tasks is different according the complexity of the diagrams.

8.5.4.3.2 Testing hypotheses 2 and 3 for UPVE + UCME Experiments. In order to test the 2^{nd} and 3^{rd} hypotheses, a Spearman correlation analysis was

Table 8.42: Correlation between Measures of Import-coupling and COM/MOD Eff (3^{rd} Family of Experiments)

Spearman correlation between measures and UPVE + UCME COM Eff									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
C_1	**0.0275**	**0.0004**	**0.0063**	**0.0036**	**0.0000**	**0.0112**	**0.0000**	0.6766	0.6551
C_2	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	0.6105	**0.0002**	0.1319	0.0579
C_3	**0.0000**	**0.0001**	**0.0000**	**0.0000**	**0.0000**	0.4913	**0.0073**	**0.0318**	**0.001**
C_4	**0.0056**	**0.0000**	**0.0003**	**0.0010**	**0.0000**	**0.0005**	**0.0000**	0.2794	0.6785
C_5	**0.0017**	**0.0000**	**0.0004**	**0.0363**	**0.0000**	0.0736	**0.0000**	0.3135	0.9025
C_6	**0.0049**	**0.0000**	**0.0009**	**0.0044**	**0.0000**	0.0775	**0.0001**	0.8546	0.8723
Spearman correlation between measures and UPVE + UCME MOD Eff									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
C_1	**0.0044**	0.954	**0.0012**	**0.0000**	0.3620	0.1335	0.7648	0.0947	**0.0000**
C_2	**0.0003**	0.5768	**0.0000**	**0.0000**	**0.0311**	0.2676	0.1417	0.1075	**0.0000**
C_3	**0.0000**	0.8288	**0.0000**	**0.0000**	0.1325	**0.0155**	0.6603	**0.0091**	**0.0000**
C_4	**0.0103**	0.5085	**0.0014**	**0.0000**	0.1513	0.7143	0.2693	0.0672	**0.0000**
C_5	**0.0003**	0.8778	**0.0000**	**0.0000**	**0.0169**	0.716	**0.0275**	0.4016	**0.0000**
C_6	**0.0000**	0.1223	**0.0000**	**0.0000**	**0.0000**	0.9437	**0.0005**	0.8235	**0.0000**

performed as we did for UPVE (see previous section). Results are reaffirmed and more cohesive, compare Tables 8.39 and 8.42, and Tables 8.40 and 8.43.

- From table 8.42, we conclude that NNR, NNC, WNN, DN, WNCO and NEI measures are significant correlated with the COM Eff for all the C_i, and the NNR, WNN, DN and NCO are significant correlated with the MOD Eff for all the C_i.

- Table 8.43 shows The NNR, NNC, WNN, WNCO and NEI measures are significant correlated with the subjective complexity of the subject for COM tasks in all the C_i (DN is correlated in five C_i) and NNR, WNN, DN and NCO measures are significant correlated with the subjective complexity of the subject for MOD tasks in all the C_i (WNCO is correlated in five C_i).

8.5.4.3.3 Testing hypotheses 4 and 5 for UPVE + UCME Experiments.

In order to test the 4^{th} and 5^{th} hypotheses, we transform the data in the same way we did for UPVE (see previous section). Results of UPVE were confirmed, we conclude that there seems to exist a statistically significant correlation between the COM / MOD SubComp variable and the COM/MOD Time, and between COM/MOD SubComp and COM/MOD Eff.

Table 8.43: Correlation between Measures of Import-coupling and COM/MOD SubComp (3^{rd} Family of Experiments)

Spearman correlation between measures and UPVE + UCME COM SubComp									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
C_1	**0.0312**	**0.0061**	**0.0096**	**0.0004**	**0.0002**	0.1363	**0.0063**	0.0533	0.6050
C_2	**0.0002**	**0.0068**	**0.0000**	**0.0000**	**0.0005**	0.7401	**0.0347**	**0.0232**	**0.0080**
C_3	**0.0066**	**0.0314**	**0.0015**	**0.0003**	**0.0004**	0.4421	**0.0035**	0.3693	0.0833
C_4	**0.0043**	**0.0050**	**0.0008**	**0.0072**	**0.0000**	0.1304	**0.0003**	0.4385	0.4205
C_5	**0.0284**	**0.0175**	**0.0180**	0.3092	**0.0037**	0.2506	**0.0030**	0.0503	0.7979
C_6	**0.0134**	**0.0005**	**0.0041**	**0.0138**	**0.0000**	0.1687	**0.0022**	0.7266	0.8604
Spearman correlation between measures and UPVE + UCME MOD SubComp									
	Measures for import-coupling								
	NNR	NNC	WNN	DN	WNCO	NAN	NEI	NES	NCO
C_1	**0.0300**	0.9940	**0.0100**	**0.0004**	0.2574	0.5269	0.4587	0.2011	**0.0004**
C_2	**0.0072**	0.1216	**0.0029**	**0.0052**	**0.0161**	0.8495	0.0535	0.6631	**0.0273**
C_3	**0.0000**	0.2528	**0.0000**	**0.0000**	**0.0189**	0.1583	0.1605	0.0725	**0.0000**
C_4	**0.0037**	0.4635	**0.0007**	**0.0001**	**0.0278**	0.9904	0.0564	0.6653	**0.0014**
C_5	**0.0000**	0.0731	**0.0000**	**0.0000**	**0.0037**	0.1982	**0.0327**	0.5908	**0.0000**
C_6	**0.0023**	0.0801	**0.0004**	**0.0002**	**0.0011**	0.5466	**0.0056**	0.7529	**0.0237**

8.5.5 Conclusions of the Third Family of Experiments (F_6)

Regarding its application in an empirical work, we described a family of experiments run in May and June (2005) at the Technical University of Valencia (UPV) and Complutense University of Madrid (UCM) respectively. In order to study the COM and MOD of the OCL expressions we have considered not only the time subjects spent on the COM and MOD tasks required in the experimental material, but also their efficiency and their subjective perception of the difficulty when carrying out these tasks. We think that, both quantitative (COM and MOD Eff) and qualitative (subject's rating or SubComp) information is important to obtain more solid findings. The results reveal that there is empirical evidence that import-coupling defined in an OCL expression is strongly correlated with the maintainability of OCL expressions (hypothesis 2), but this finding is strengthened by the following facts:

1. From the descriptive and exploratory analyses, we can conclude that the efficiency of subjects when they had to understand and modify the OCL expressions, increased as time passed. We obtained a considerable improvement of subjects' efficiency in MOD tasks. In this new family MOD tasks differ considerably from those in the previous experiments where subjects' MOD Eff was lower. Moreover, subjects were more efficient in MOD tasks than in COM tasks. COM and MOD efficiency are different if we arrange the collected data according to the model complexity. The higher the complexity of the OCL ex-

pression, the lower is the COM Eff of the subjects. However, models of Medium Complexity (MC) were more difficult to modify for the subjects than models of High Complexity (HC). The former (MC models) were models in which students should have traced from different navigations. In the latter (HC models) subjects should identify which collection operations they have to use and how to combine them. We think that their experience in collection operations and the activities of tracing relationships between classes could have produced this situation. Due to the size of the population from the UCM experiment the formulated hypotheses of the experiments could not be tested in the same way as we analyzed the hypotheses for the UPV experiment. However, we used the UCM data to confirm the findings obtained in UPV gathering the whole set of data from both experiments (after using statistical tests needed to realize if this is possible and enable us to collect all the data). We found different results for the hypotheses that were confirmed once we gathered the whole data set. We summarize the findings for each hypothesis in Table 8.44. The results reveal again that there is empirical evidence that import-coupling defined in an OCL expression through navigations and collection operations is strongly correlated with the maintainability of OCL expressions.

2. The main goal of the experiment of explaining the influence of import-coupling on the maintainability of OCL expressions although it is verified through hypothesis 2, it was reaffirmed using a triangulation of hypothesis relating the import-coupling and cognitive complexity (hypothesis 3), and between the cognitive complexity and the maintainability of OCL expressions (hypothesis 4 and 5). We found different results for the formulated hypotheses which we summarize in Table 8.44.

3. Considering the COM and MOD tasks, the import-coupling structural property of OCL expressions impact in the cognitive complexity in different way (hypothesis 3), and this effect is explained as follow using the cognitive theory:

   (a) We observed that MOD tasks were more localized than COM tasks, see that a subset of the measures which influences in COM Eff also influence in MOD Eff, possibly indicating that an as-needed comprehension strategy was applied.

   (b) The subjects in COM tasks gained a broad understanding of OCL expressions through Problem objects (NNC, NEI), Relation of problems objects (NNR, WNN, DN) and Reified objects (WNCO). However once this breath of familiarity with the OCL expressions was gained the modelers mainly concentrate during MOD tasks in Relation of Problems Objects (NNR, WNN, DN).

   (c) With regard to the cognitive process of Cant et al. modelers applied chunking and tracing cognitive process in COM tasks whereas in MOD

tasks tracing was the more relevant process applied. Notice that mainly relations (NNR) and navigations (WNN and DN) affect MOD Eff.

(d) Measures related to chunking and tracing affect COM tasks, and mainly tracing affect the MOD tasks.

4. COM and MOD SubComp is correlated to the COM and MOD Time (hypothesis 4) and also with COM and MOD Eff (hypothesis 5).

5. The set of measures that are correlated with the maintainability of OCL expression (hypothesis 2) is almost the same as the set of measure correlated with COM and MOD SubComp (hypothesis 3).

Table 8.44: Synopsis of Hypotheses and the Statistical Tests Applied (3^{rd} Family of Experiments)

	Efficiency	**Time**	**Subjective Complexity**
Relation between	**COM Eff** **MOD Eff**	**COM Time** **MOD Time**	**COM SubComp** **MOD SubComp**
	Hypotheses 1		
	Test: Chi-square Test		
	The mean efficiency of COM and MOD tasks is different according to the complexity of diagrams		
OCL expressions measures	**Hypotheses 2**		**Hypotheses 3**
	Test: Spearman		Test: Spearman
	(1) the NNR, NNC, WNN, DN, WNCO and NEI measures have are significant correlated with the COM Eff (2) the NNR, WNN, DN and NCO have significant correlated with the MOD Eff		The NNR, NNC, WNN, DN, WNCO and NEI measures are significant correlated with the COM subjective complexity; and NNR, WNN, DN, NCO and WNCO measures are significant correlated with the MOD subjective complexity
COM MOD SubComp	**Hypotheses 5**	**Hypotheses 4**	
	Test: τ Kendall	Test: τ Kendall	
	The subjective ratings are influenced by the COM and MOD Eff	The subjects' subjective ratings (COM or MOD rating) are influenced by the time they spent understanding or modifying the OCL Expressions, i.e. both times seem to affect their perception of the level of complexity of an OCL expression	

8.6 Contribution to the Dissertation

In order to empirically validate the proposed measures this chapter carefully describes three families of experiments. The obtained findings shows that the structural properties that capture import-coupling seems to exert a significant influence on the comprehensibility and modifiability of OCL expressions.

According to the results we claim that a high number of coupled objectgs, navigations and collection operations will increase the import-coupling in an OCL expression, and also bring serious difficulties in OCL expression comprehension and correspondingly in comprehension-related tasks, such as OCL expression modifiability. As guidelines we suggest that navigations should be used moderately during modeling taking these concerns into account:

- You should try to reduce the number of import-coupling in an OCL expression. The larger the set of coupled objects (and its properties) to be chunked the greater the context to be understood.
- You should try to reduce the number of combined navigations to improve the comprehension of OCL expressions. An intertwining specification of relationships could reduce the comprehension of an OCL expression and make the contextual instance know details of distant objects. Whenever possible, limit the knowledge of coupled objects to the immediate surroundings of the contextual instance.
- The number of collection operations should be as low as possible in order to obtain OCL expressions more easily to comprehend.
- The relationship between the coupled objects could seriously affect the modifiability of OCL expressions. Try to use the lower number of relationships in order to reduce the tracing during comprehension and modification. As a future work we plan to study how the subject has reasoned while solving the COM and MOD tasks. We will record students while they reason aloud to discover more about the relationship between OCL expressions and cognitive complexity [Hen02].

We are aware that the next step is to obtain a multivariate regression analysis in order to continue interpreting the collected data obtained, and to run new experimental replicas. In this way we will strengthen the conclusion and external validity respectively. The empirical validation of the rest of the measures is also pending. Moreover, we will work in a generalization of the benefits of the set of measures defined for OCL expressions, trying to obtain a global complexity of UML/OCL models (note that all the proposed measures are defined in terms of a single OCL expression).

Chapter 9

Conclusions

This chapter analyses the achievement of the objectives outlined at the beginning of the Ph.D. thesis, then the main contributions and conclusions of this thesis are presented and the principal publications are listed. Finally, the lines of work which are left open are described.

9.1 Analysis of Achievement of Objectives

At the beginning of this document, chapter one shows the partial objectives pursued, which led us to the achievement of the main goal of this Ph.D. Thesis, which is:

> ASSESSING THE INFLUENCE OF IMPORT-COUPLING ON THE MAINTAINABILITY OF OCL EXPRESSIONS THROUGH A MEASUREMENT-BASED APPROACH

And, on the basis of the main objective, a series of partial objectives have arisen:

Obj-1. Analyse the existing measures for UML models and measures for coupling: After a thorough review of the existing literature related to measures for UML diagrams which we presented in the State of The Art (see chapter 3), we have shown measures that can be applied to use case diagrams, class diagrams and statecharts diagrams. Although various proposed measures for different kinds of diagrams do exist, and have been theoretically and empirically validated there are no measures for UML diagrams when they are complemented by OCL expressions.

In the State of the Art we also showed that the quantity of measures for

coupling is vast and numerous empirical studies suggest that coupling has an impact on several external quality attributes. Nevertheless, the coupling defined through OCL expressions was not analyzed in none of the existing studies from the literature.

Furthermore after analysing the literature related of OO measures, we found that the majority of existing measures have been designed to be applied at an advanced design level and not for a PIM model.

Obj-2. Extend and refine the method for the definition of valid measures: We have refined and extended the method for measure definition specified by Calero et al. [CPG01] which pursues three main goals and phases: measure definition, theoretical validation and empirical validation (see chapter 2). The enhancement of the method, in terms of refinement and extensions, is described as part of the main contribution of this dissertation (see section 9.2).

Obj-3. Propose a set of measures for measuring the structural properties of OCL expressions within UML/OCL models: A proposal of measures for structural properties of OCL expressions within UML/OCL combined models is presented in chapter 4. We consider OCL expressions that are commonly written for UML class diagrams but without loss of generality many of these measures can be applied for OCL expressions used in other kinds of diagrams. In this way 21 measures for OCL expressions have been defined considering the main elements of the OCL Metamodel that most frequently appear in OCL expressions.

Obj-4. Carry out the formal definition of the proposed measures: In order to avoid misunderstanding and misinterpretation with the definition of measures in natural language we have presented in chapter 5 the formal specification of the measures (proposed in chapter 4) using OCL upon the OCL metamodel.

Obj-5. Perform the theoretical validation of the proposed measures using the most suitable frameworks: In chapter 5 we have theoretically validated the proposed measures according to the formal framework of Briand et al. [BMB96], [BMB97], its adaptation for interaction-based measures for coupling and cohesion [BMB99] and the Poels and Dedene´s framework [PD99]. We theoretically validated fifteen measures as interaction-based measures for coupling based on context-dependent properties for import-coupling.

Obj-6. Describe the rationale of the measures using a psychological explanation from a cognitive point of view: The rationale behind the software measures, that is the cognitive complexity of modelers dealing with OCL expressions was described from a cognitive point of view. Mental models and cognitive models were used to obtain a clear goal definition and also to interpret the experimental findings in the empirical validation. Using verbal protocols we prove that coupling

cognitive categories such as: problem objects, relationships between problem objects and reified objects are important components of the mental burden of modelers dealing with OCL expressions. These categories are inherently related to the influence of import-coupling on cognitive complexity.

Obj-7. Perform the empirical validation of the proposed measures to find early indicators of OCL maintainability: Three families of experiments have been carried out for validating the proposed measures as early indicators of the maintainability of OCL expressions. The findings have been shown in chapter 8. We empirically validated that the structural properties that capture import-coupling dependencies exert significant influence on the comprehensibility and modifiability of OCL expressions. More experimentation would be necessary in order to obtain more conclusive results.

As we have shown the achievement of all the objectives, we believe that we have successfully come to achieve our primary objective 'ASSESSING THE INFLUENCE OF IMPORT-COUPLING ON THE MAINTAINABILITY OF OCL EXPRESSIONS THROUGH A MEASUREMENT-BASED APPROACH'. This lead us to conclude that the initial hypothesis has been satisfied and we can assert that:

IT IS FEASIBLE TO ASSESS THE INFLUENCE OF IMPORT-COUPLING ON THE MAINTAINABILITY OF OCL EXPRESSIONS THROUGH A MEASUREMENT-BASED APPROACH

9.2 Main Contributions and Conclusions

A systematic use of models as primary engineering artifacts throughout a model-driven engineering will increase the importance of their maintainability [AWÁF02]. However, maintainability has been and will continue to be an expensive and challenging task poorly managed [DJ03] unless proven measures for software maintainability would be used.

It is widely believed that predicting the maintenance at early stages (such as during the specification of a platform independent-model) will help software designers and maintainers to alter the architecture of the software system for better quality that leads to the overall reduction of maintenance costs [MPKS00]. However, what makes a model be of better quality than another is a subject that should be carefully analysed, due to the fact that quality is a composite of many internal attributes. We believe that during the maintainability of models, maintainers should apply a

principle which many text books admonish as a good practice, they should reduce coupling among software artifacts and improve cohesion within them.

However, in an OO setting, there are many forms of coupling that can arise in systems, and establishing what the different forms of coupling are and which are the most harmful are open research questions [BAC+99]. This thesis is focused on this direction. We studied the influence of import-coupling on OCL expression maintainability. We take a measurement-based approach in the study. We theoretically and empirically validated that the structural properties that capture dependencies among OCL expressions and its associated system (in our case UML models), exert significant influence on the comprehensibility and modifiability of OCL expressions.

At the same time we discovered new forms of coupling, we defined a set of measure to deal with it. These measures may be useful to (1) optimize OCL expressions in case of arbitrary model changes [AHK07], [CW99a] and transformations [AWÁF02] (a relevant aspect within the newly emerged modeling technologies -MDE, MDA, MDD-), (2) to improve constraint specification, behaviour specification of query operation and initial and derived property values.

The assessment of the impact of coupling on OCL expression maintainability was a challenging activity due to the many issues that appeared:

- Language characteristics: As was mentioned in chapter 3 the language itself has many roots: (1) set theory [Baa00] (represented by the OCL collections and many collections operations) (2) predicate logic and (3) operational semantics: iterate construct, the basis of most of the collection operations [Baa00].
- Graphical and Textual information: OCL should be used along with graphical information (the textual specification of OCL expressions complements the information provided by graphical information [CCBC04]). OCL is a clear integration of formal and informal aspects.
- Domain of the Measure: Although the expression can be seen as short textual description many times its extent involves many coupled objects of the model.

The diversity and complexity of the aforementioned aspects makes the assessment a difficult activity. To tackle our goal in a proper manner we based the assessment following a method for measure definition. In fact, we also focus on the redefinition of the method itself due to the fact that the method applied is based on a previous work which was refined and extended.

The main contributions of this Ph.D. thesis are:

- **Related to the Method for Measure Definition:** The extension and refinement of the method was modelled in chapter 2 using UML activity diagrams. The method had been strengthening not only in the order of its

activities but also identifying object flows between activities and important decisions that should be evaluated during the activities. The application of the new method will help the researcher to methodically obtain reliable and consistent measures.

The refinements of the method were introduced in:

- Identification step (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 that questions fit the goals. New activities were added such as the identification of abstractions for measuring structural properties, the statement of general hypotheses, etc.
- Creation step (C_i activities, i=1, .., 4): Although the main steps of the creation activity were already defined in the previous methods, we refined important node decisions and object flows between subactivities.
- Empirical Validation Step (E_i, F_j and EF_k activities, i=1,2, j=1,..,6, k=1,..,5): We identified the more relevant activities in carrying out families of experiments and isolated experiments.

The extension of the method were focused on:

- Definition in Natural Language (N_i activities, i=1,..,4): We used a template to define the measures which is 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 identify the more important activities that should be performed in a formal definition of measures.
- 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.
- Psychological Explanation (PE_i activities, i=1,..,3): Three relevant activities were detected in a psychological explanation of how the subjects deal with the software artifact which is measured.

- **Related to the Definition of Measures**: We defined 21 measures for structural properties of OCL expressions: 15 measures for measuring import-coupling concepts of OCL expressions, 1 measure for length and 9 measures to control the size. During the definition we analyze all the concepts mentioned in the OCL metamodel which are related to coupling, length and size attributes. Originally, the measures were introduced in natural language. The rationale and goal of each measure are carefully explained.

- **Related to the Formal Specification of the Measures**: Measures were defined using OCL upon the OCL metamodel (OCL^2). The definition of the measures was not an easy activity due to the fact that the concepts involved in their definition are related to many OCL metaclasses. The number of OCL metaclasses involved is considerably big (more than fifty metaclasses). We had selected a strategy so as not to clutter the OCL metaclasses and we decided to used a pattern-based approach for the formal definition of the measures.

- **Related to the Theoretical Validation:** One important contribution in the theoretical validation of the measures was the definition of context-dependent properties using the property-based framework of Briand et al. [BMB99]. Its application will allow us to prove that the 15 measures are interaction-based measures for coupling. We did not find in the literature about measurement, any application of context-dependent properties with exception of the definition and application provided by its authors [BMB99].

- **Related to the Psychological Explanation:** The definition of measures and their empirical validation were enriched through a plausible explanation of the cognitive complexity of modellers dealing with OCL expressions. We based our reasoning using mental models and cognitive models, and we related these cognitive theories to explain the rationale behind the measure and how the modellers deal with OCL expressions. Another important contribution of this thesis is the application of qualitative methods which help us to delve the complexity of human role in software engineering tasks [Sea99]. In our case, we run a think aloud experiment to explain the more important categories that compose the mental model of modellers during comprehension activities. We also found that modellers use different attempt to comprehend the class diagram associated to the OCL expressions.

  Related to the psychological explanation we can draw the following conclusions:

  - In order to comprehend and modify an OCL expression modelers use two important cognitive techniques: tracing and chunking. These techniques defined in the CCM Model [CHSJ94] constitute relevant aspect in OCL expression comprehension. On one hand, OCL subexpressions are suitable mechanisms which facilitates the chunking activities due to the fact any OCL expression can be read and evaluated from left to right combining different properties, similarly to the composition of functions. Properties can be combined to make more complicated expressions through the use of chunking cognitive technique. On the other hand, tracing technique is guided by OCL navigations through a UML model.
  - When modelers deal with OCL expressions, the three more important categories of the modelers' mental model are (see Figure 9.1): objects

from the application domain (i.e. problem objects), relationships between problem objects and reified objects (they are not problem domain objects per se) such as collections. Regarding the last component, OCL provides the modellers with a rich set of built-in collections and collections operations. Probably, this issue of the language is one the most important similarity of OCL with OO programming languages due to the fact that dealing with collection is an important features of OO languages [1]. We evaluate the contribution of collection and collection operations during import-coupling through the definition of the WNCO measure and the consideration of reified objects within a plausible explanation of cognitive complexity of modellers.

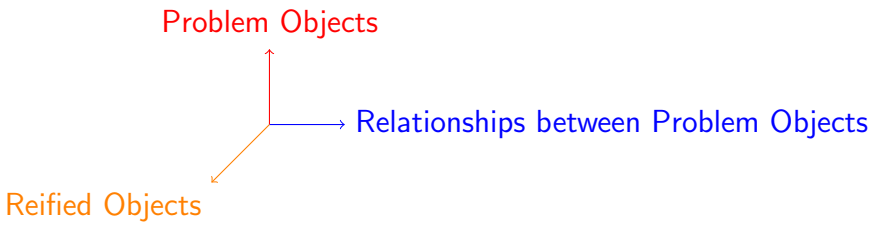


Figure 9.1: Main Categories of the Mental Model of Subjects dealing with OCL Expressions

- **Related to the Empirical Validation:** We carried out three families of experiments across seven universities of Spain, Argentina and Chile.

  We can draw several conclusions from the empirical validation:

  – Through a specific family of experiment (the 1^{st} family of experiments of chapter 8) we evaluated whether the number of classes (NNC, a measure of the quantity of problem objects), the depth of navigations (as a measure of the length of coupling, DN), and its interactions affect the OCL expression maintainability. Both aspects and their interactions seem to affect the impact of import-coupling on OCL expressions maintainability. We found through experimentation that reflexive relationships are more subtle and difficult to understand than simple relationships, i.e. even when a less quantity of objects are coupled in an expression if the depth of navigation is high (this happens when reflexive navigation is used) the OCL expression comprehension could be deeply decreased.

[1] The notation is also recognized as a similarity of OCL with OO programming language, having a very positive psychological effect [Baa00].

– Modifications of OCL expressions are more difficult to deal with than comprehension (see 2^{nd} family of experiments). Although chunking and tracing cognitive techniques influence on OCL expressions comprehension, the modification of an OCL expression demands that these techniques are applied in an intertwining way. Modelers during OCL modifications should identify new problem objects, new relationships between objects and collections operations to specify the modifications. The identification of these aspects during modification and how they must be combined demands from modelers to be more cognitively flexible [CMF04] than when they are comprehending OCL comprehensions.

– We also observed (see 2^{nd} and 3^{rd} families of experiments) that the OCL learning curve of subjects is an important issue that should be considered, we showed that the efficiency of subjects is higher as time passes during experimentation.

– We also found that the coupling' complexity which is based on the manipulation of collection operations was less difficult to deal with than the coupling' complexity which is based on dealing with different problem objects and relationships. Probably this happens due to the fact that the experimental objects we used in all the experiments were from different application domain, and the recurrent aspect was dealing with OCL expressions, i.e. the recurrent aspect was the language knowledge rather than the domain knowledge. Empirical results would be different whether the experimental objects belong to the same application domain. So, we think that the efficiency depends not only of the language knowledge but also of domain knowledge (see Figure 9.2). Nevertheless we think that more experimentation would be necessary in this topic to obtain more conclusive results.

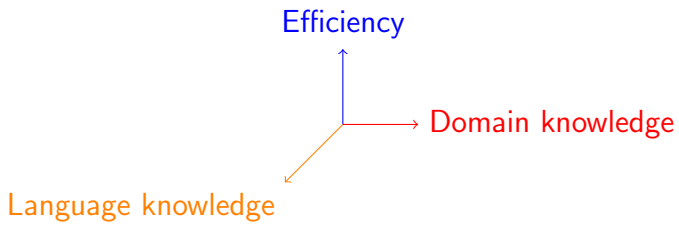


Figure 9.2: Language and Domain Knowledge Affect Subject Efficiency Dealing with OCL Expressions

9.3 Contrast of Results

The partial results obtained during the investigation have been published and presented in different forums, some of which will be presented next.

9.3.1 Book Chapter

1- L. Reynoso, M. Genero and M. Piattini, *Measuring OCL Expressions: An Approach Based on Cognitive Techniques*. Chapter 5 in "Metrics for Software Conceptual Models". M. Genero, M. Piattini and C. Calero Editors. pp: 161-206. **Imperial College Press**, UK. 2005.

This chapter presents, the definition in a methodological way of a set of measures for structural properties of OCL expressions, considering those OCL concepts specified in its metamodel which involves the use of two cognitive techniques, tracing and chunking. The chapter also shows the theoretical validation of the measures using the property-based framework proposed by Briand et al [BMB99].

9.3.2 International Journals

2- L. Reynoso and M. Genero and M. Piattini, *Towards a metric suite for OCL Expressions expressed within UML/OCL models*. Journal of Computer Science and Technology. Journal of Computer Science and Technology **JCS&T**. Issue Volume 4 Number 1, pp: 38-44. ISSN 1666-6038. April 2004.

The definition of a measure suite for OCL expressions is presented in this paper. Within the definition we evaluated whether each measure is related to either tracing or chunking cognitive techniques. The theoretical validation of the measures using property-based frameworks is outlined.

3- M. Genero, M. Piattini, J. A. Cruz-Lemus and L. Reynoso, *Metrics for UML Models*. **UPGRADE**. The European Journal for the Informatics Professional. ISSN 1684-5285. Volume 5 (2). pp: 43-48. http://www.upgrade.- cepis.org/. 2004. Also published as:
M. Genero, M. Piattini, J. A. Cruz-Lemus and L. Reynoso, *Métricas para Modelos UML*. **Revista Novática**. Revista de la Asociación de Técnicos de Informática. ISSN 0211-2124. http://www.ati.es/novatica. Volume 170, pp: 61-65. Spain. 2004.

These are informative articles that describe a thorough state of the art of the relevant literature related to measures for UML models. Different set of closed-ended measures for class diagrams, statechart diagrams, use case diagrams and OCL expressions were included.

4- L. Reynoso, M. Genero, M. Piattini and E. Manso, *El Efecto del Acoplamiento en la Comprensibilidad y Modificabilidad de Expresiones OCL: Un Anlisis Experimental*. **IEEE Latin America**. Vol 4, Issue 2. pp: 62-67. April, 2006.

This paper includes an improved and expanded spanish version of a paper published in JISBD 2005. It was selected from the JISBD 2005 Proceeding.

9.3.3 International Conferences

5- L. Reynoso and M. Genero and M. Piattini, *A Controlled Experiment for Validating Metrics for OCL Expressions*. ACM-IEEE International Symposium on Empirical Software Engineering. **ISESE 2004**. pp. 15-16. 19-20 August 2004 Redondo Beach, CA, USA., 2004.

This short paper describes the first experiment of the first family of experiments. The main goal of the paper is to ascertain if any relation exists between the measures DN and NNC and the comprehensibility and modifiability of OCL expressions.

6- L. Reynoso and M. Genero, M. Piattini and E. Manso, *Assessing the Impact of Coupling on the Understandability and Modifiability of OCL Expressions within UML/OCL Combined models*. 11^{th} IEEE International Software Metrics Symposium **IEEE METRICS 2005**. 19-22 September. Como, Italy, 2005. pp:14.

This paper presents the second family of experiments. The results reveal that there is empirical evidence that import-coupling defined in an OCL expression through navigations and collection operations is significant correlated with the maintainability of OCL expressions. The findings are preliminary and more experimentation is needed.

7- L. Reynoso and M. Genero, M. Piattini and E. Manso, *Does Object Coupling Really Affect the Understandability and Modifiability of OCL Expressions?*. 21^{st} ACM Symposium on Applied Computing. **ACM SAC-SE 2006**. Dijon, France. April 2006. pp: 1721-1727.

This paper presents the first experiment of the third family of experiments. The experiment confirms the results of the second family of experiments: object coupling affects the maintainability of OCL expressions. In this paper we show that we obtained a considerable improvement of the subject's efficiency in modifying OCL expressions, which was the reason to start the third family of experiments.

8- L. Reynoso and M. Genero and E. Manso, *Measuring Object Coupling in OCL Expressions: A Cognitive Theory-Based Approach*. IEEE Instrumentation and Measurement Technology Conference. **IEEE IMTC 2006**. pp: 1087-1092. Sorrento, Italy. April 2006.

This paper presents the third family of experiments, the experiment and its replica. An analysis of the whole data for the family is performed. The results of the experiment are explained using a cognitive theory-based approach using a relationship between the main components of the mental model of Burkhardt et al. [BDW02] and the concepts captured by the measures for OCL expressions.

9- L. Reynoso and M. Genero and M. Piattini, *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.

This paper shows the use of verbal protocols to explain the main categories in the mental model of the subjects dealing with OCL expressions. It also describes the different approaches that the modellers attempt in order to comprehend the class diagrams associated to the OCL expression. We detail the more important steps using verbal protocols and we applied them in a think-aloud experiment carried out at the Castilla La-Mancha University.

10- L. Reynoso, J. A. Cruz-Lemus, M. Genero and M. Piattini, *Formal Definition of Measures for UML Statechart Diagrams Using OCL*. 23^{rd} ACM Symposium on Applied Computing. **ACM SAC-SE 2008**. Fortaleza, Ceará, Brazil. March 16-20, 2008. (to appear)

In this paper we show the formal definition of measures for UML statechart diagrams using OCL, upon the UML statechart metamodel. Originaly the measures were defined by Cruz-Lemus [CL07] and its formal definition is included in this paper. The use of a formal definition upon a metamodel assure that measures capture the concepts they intend for.

9.3.4 National Conferences

11- L. Reynoso, M. Genero and M. Piattini. *Una Propuesta de Métricas para Expresiones OCL basada en "Tracing"*. Jornadas sobre Innovación y Calidad del Software. **JICS 2003**. Universitat Politecnica de Catalunya. June 2003. Barcelona, Spain.

A set of open-ended measures for OCL expressions are presented in this paper. This document shows the first attempt of defining a set of measures focused on tracing as the main factor that affects the comprehension of OCL expressions.

12- L. Reynoso and M. Genero, M. Piattini and E. Manso, *Validating OCL Metrics through a Family of Experiments*. IX Jornadas de Ingeniería del Software y Base de Datos. **JISBD 2004**. November 10-12. Málaga, Spain, 2004.

This paper presents a controlled experiment and two replicas (the first family of experiments) for assessing if two of the proposed measures, DN and NNC, are related with to the comprehensibility and modifiability of OCL expressions.

13- L. Reynoso and M. Genero, M. Piattini and E. Manso, *The Effect of Coupling on Understanding and Modifying OCL Expressions: An Experimental Analysis*. X Jornadas de Ingeniería del Software y Base de Datos. **JISBD 2005**. pp: 139-146. September 14-16. Granada, Spain, 2005.

In this paper we present the second family of experiments in order to analyse the effect of coupling on the understandability and modifiability of OCL expressions. We found a statistically significant correlation between many measures, specially those related to tracing, and the understandability and modifiability efficiency.

9.3.5 International Workshops

14- L. Reynoso, M. Genero and M. Piattini. *Measuring OCL Expressions: a Tracing-based Approach*. Workshop on Quantitative Approaches in Object-Oriented Software Engineering. **QAOOSE 2003**. 21-25 July 2003. Germany.

A preliminary work of the measures for OCL expressions were presented in this paper. We explain the goal of the measures using cognitive techniques of Cant et al. [CHSJ94].

15- L. Reynoso, M. Genero and M. Piattini, *Validating Metrics for OCL Expressions Expressed within UML/OCL Models*, International Workshop on Software Audits and Metrics. ISBN 972-8865-04-X. **SAM 2004**. April 13-14, 2004 - Porto, Portugal, 2004. pp: 59-68.

In this paper we show the first experiment we carried out. Although this experiment is an isolated experiment which does not belong to any family, it was useful to comprehend the different phases in the experimental process. Preliminary findings were obtained.

16- L. Reynoso, M. Genero and M. Piattini, *OCL^2: Using OCL in the Formal De- finition of OCL Expression Measures*, 1st Workshop on Quality in Modeling **QIM 2006** co-located with the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MODELs 2006). 1st. October, 2006. Genova, Italy. pp: 95-107.

In this paper the measures proposed for OCL expressions were formally defined using OCL upon the OCL metamodel. We explain the strategy we used and we exemplify how the value of a measure is obtained using abstract syntax trees for OCL expressions.

9.3.6 Latinoamerican Conferences

17- L. Reynoso, M. Genero and Piattini M. *Definición de métricas para la Complejidad de Expresiones OCL de Forma Metodológica*. Workshop on Computer Science 2003. **WICC 2003**. 22-23 May 2003. Tandil, Argentina. pp: 461-465.

This paper presents the initial method we used for the definition of valid measures. The creation step is carefully explained.

18- L. Reynoso, M. Genero and Piattini M. *Métricas para Expresiones OCL Relacionadas con la Técnica Cognitiva de Chunking*. XXIX Conferencia Latinoamericana de Informática. **CLEI 2003**. September 29- October 2, 2003. La Paz, Bolivia.

A set of open-ended measures for OCL expressions are presented in this paper. This document shows the first attempt of defining measures focused on chunking as an important factor that affects the comprehension.

19- L. Reynoso, M. Genero and M. Piattini. *Métricas para Propiedades Estructurales de Expresiones OCL Relacionadas con la Técnica de Chunking*. 6-10 October 2003. IX Argentinian Congress on Computer Science. **CACIC 2003**. La Plata, Argentina. pp: 1099-1111.

This paper presents a set of open-ended measures for OCL expressions. The definition is focused on the analysis of the structural properties of OCL ex-

pressions related to chunking cognitive technique.

20- L. Reynoso, M. Genero and Piattini M. *An Experiment Family for OCL Expressions within UML/OCL Model*. Argentine Symposium on Software Engineering. **ASSE 2004**. September 20-22, 2004. Cordoba, Argentina.

The first family of experiments is presented in this paper. However we only show preliminary analysis of the data. We only include the time that subjects spent on tasks as the dependent variables for measuring comprehensibility and modifiability of OCL expressions.

9.4 Future Research Lines

There are many more directions this work could take in the future. Possible topics for further research that will provide valuable information include:

1. **Specify the Measures using Maude:** The proposed measures can be defined using Maude formal language [CDE+02], [CDE+00]. Maude [CDE+98] may offer a reasoning power over the specifications that OCL does not, allowing in this way the chance of identifying interesting transformation to apply on a model or comparing several specifications for semantic equivalences.

   Another important difference between OCL and Maude is related to their execution capacity, Maude allows executing its specifications. Although there are tools that allow executing OCL restrictions on models, these have not reached the same development level as Maude yet [CL07].

   Once the measures are defined in Maude, a comparison between both specifications (Maude and OCL) will be done.

2. **Delve into the Psychological Explanation Step:** The cognitive complexity of modelers dealing with OCL expressions is an aspect that needs further studies and analyses. It is important to run a family of qualitative methods such as thinking aloud [SBS94] experiments or even interviews in order to delve the way the different coupling categories affect the maintainability of OCL expressions. Moreover, the inclusion of qualitative methods within quantitative empirical validation is also possible and should be carefully considered (e.g. [Sea99]).

3. **Use Data Analysis Techniques:** The analysis of the families of the experiments may be further analysed using different techniques:

- **Analyse Principal Component Analysis:** Part of the information that these proposed measures provide might be redundant, which in statistical terms is equivalent to saying that measures might be correlated. This justifies the interest of analyzing the information that each measure captures to eliminate such redundancy. We should better understand the underlying and orthogonal dimensions captured by the measures, which in turn will help us to interpret subsequent results. For that purpose, we should perform a principal component analysis (PCA) [Dun89] on our coupling measures. Nevertheless, the principal component analysis should be performed carefully because it is heavily affected by the scaling of the variables and the identity of the original variables is lost, so interpretation can be more difficult.
- **Build Prediction Models:** Another goal of a future work is to build prediction models for understandability and modifiability of OCL expressions. These prediction models can be validated by using experimental data from the experiments we run or even from data obtained in real projects. We plan to use Multivariate Regression Models [KKM88] However, the prediction models obtained can be compared using several techniques such as: rough sets, bayesien nets, genetic algorithms, etc.
- **Apply Meta-Analysis:** According to research activity in Software Engineering, rarely a single experiment or study provides sufficiently definitive answers. However, when several studies address a set of related research hypotheses, a meta-analysis may combine the results from them. The meta-analysis makes good use of independent studies with the purpose of integrating the findings. It has been used in different research areas such as psychology, medicine, etc. So, the analysis of the results from a group of studies can allow more accurate data analysis [PJ97], [Mil00], [LEH01].

4. **Extend and Refine the Acceptation, Application and Accreditation Activities:** The last three high level activities (activities M_3, M_4 and M_5 of Figure 2.1) of the method for measures definition were neither extended nor refined. Due to the fact that more experimentation would be necessary in real environments and projects, we think that at the same time they are carried out we would be able to study the aforementioned activities of the method.

5. **Study the Maintainability of Expressions in Different UML models:** We should study the proposed measures in the maintainability of UML statechart diagrams and component diagrams, which are, after the class diagrams, important diagrams where OCL expressions are more widely used:

   - Within a statechart diagram, OCL expressions can be used to specify guard, specific target of actions, actual parameter values, change events

and state constraints [WK03]. We believe that OCL facilities within statechart diagrams can change substantially the comprehensibility and expressiveness of these models.

- Regarding component diagrams OCL expressions are used to specify the pre and post conditions of component interfaces [WK03]. Having a clear specification of the component interface is the first step in comprehending and correctly using a component.

6. **Study the Maintainability of Different Types of OCL Expressions:** We should study the maintainability of different types of OCL expressions (for instance, pre- and post-condition, query expressions, etc.) in UML/OCL models. We believe that results obtained for invariant expression of the Ph.D. thesis are sufficiently generalizable so that they can also be used on other kinds of expressions. However, empirical validation of their application is needed.

7. **Define Cohesion Measures:** We plan to assess the influence of cohesion on the maintainability of OCL expressions, through the definition of cohesion measures. Although many empirical studies found no relationship between cohesion and several external quality attributes (for instance in fault proneness [BWDP00], [BWL01], and maintainability [DJ03]) other authors argue that coupling and cohesion are dimensions that should be jointly analysed [DS05]. Nevertheless, many time authors could not apply cohesion measures to the analysed systems in their empirical validation because they did not have enough information about the method implementation at early stages of software development. However, at design level stage using UML/OCL models, we have declarative semantic descriptions about the contract of interfaces and many cohesion measures could be computed in order to reason about their impact on external quality attributes.

   In order to compute cohesion measures, most of the coupling measures defined in this thesis could be redefined in such a way they could obtain the set of imported features of an OCL expression. For instance instead of NAN measures (number of navigation referred through navigation), we should use a function to obtain the set of attributes referred through navigations (not the quantity). Once the redefinition of many measures was performed, the intersection of sets of imported features of two OCL expressions could be used to obtain cohesion measures.

8. **Scale the Measures up to Class Granularity:** We should scale the coupling and cohesion measures up to the granularity of classes. We think this is an important step in obtaining useful measures for UML/OCL models. Nevertheless it is important to take into account other measures which were defined at a class-level in order to derive reliable measures.

Bibliography

[Aag98] J. Aagedal. Towards an ODP-Compliant Object Definition Language with QoS-Support. In *IDMS '98: Proceedings of the 5th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 181–194, London, UK, 1998. Springer-Verlag.

[ADSJ01] B. Anda, H. Dreiem, D. I. K. Sjoberg, and M. Jorgensen. Estimating Software Development Effort Based on Use Cases-Experiences from Industry. In *UML '01: Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 487–502, London, UK, 2001. Springer-Verlag.

[AHK07] M. Altenholfen, T. Hettel, and S. Kusterer. Ocl support in an industrial environment. In *Proceedings of MoDELS 2006 Workshops. LNCS 4364*, pages 169–178, 2007.

[AK03] C. Atkinson and T. Kuhne. Model Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5):36–41, 2003.

[Ale01] I. F. Alexander. Capturing use cases with doors. In *RE '01: 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada*, page 264. IEEE Computer Society, 2001.

[APA84] APA. *Publication Manual of the American Psychological Association*. American Psychological Association, Washington, DC, USA, third edition, 1984.

[Ari02] E. Arisholm. Dynamic Coupling Measures for Object-Oriented Software. In *METRICS '02: Proceedings of the 8th International Symposium on Software Metrics*, page 33, Washington, DC, USA, 2002. IEEE Computer Society.

[AWÁF02] J. Araújo, J. Whittle, J. A. Toval Álvarez, and R. B. France. Integration and transformation of uml models. In *ECOOP Workshops*, volume 2548 of *LNCS*, pages 184–191. Springer, 2002.

[Baa00] T. Baar. Experiences with the UML/OCL-approach in practice and strategies to overcome deficiencies. In Net.ObjectDays-Forum, editor, *Proc. Net.ObjectDays2000, Erfurt, Germany*, pages 192–201, 2000.

[Bab90] E. Babbie. *Survey Research Methods*. (2nd Ed.) Belmont, CA: Wadsworth Publishing Company, 1990.

[BAC+99] L. C. Briand, E. Arisholm, S. Counsell, F. Houdek, and P. Thévenod-Fosse. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions. *Empirical Software Engineering*, 4(4):387–404, 1999.

[Bar02] A. L. Baroni. Formal Definition of Object-Oriented Design Metrics. *Master of Science in Computer Science Thesis, Vrije Universiteit Brussel, Belgium*, 2002.

[BBD01] L. C. Briand, C. Bunse, and J. W. Daly. A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs. *IEEE Transaction on Software Engineering*, 27(6):513–530, 2001.

[BBDV03] J. Bezivin, E. Breton, G. Dupe, and P. Valduriez. The ATL Transformation-based Model Management Framework. Technical Report 03.08, IRIN Universite de Nantes: Nantes, 2003.

[BBeA02] A. L. Baroni, S. Braz, and F. Brito e Abreu. Using OCL to Formalize Object-Oriented Design Metrics Definitions. *In Proc. of QUAOOSE'2002, Malaga, Spain*, 2002.

[BBK78] B. W. Boehm, J. R. Brown, and J. R. Kaspar. *Characteristic of Software Quality*. TRW Series of Software Technology, Amsterdam, North Holland, 1978.

[BBM96] V. R. Basili, L. C. Briand, and W. L. Melo. Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transaction on Software Engineering*, 22(10):751–761, 1996.

[BBM98] V. R. Basili, L. C. Briand, and S. Morasca. Defining and Validating Measures for Object-Based High-Level Design. Technical Report IESE-Report No. 018.98/E, Fraunhofer Institute for Experimental Software Engineering, 1998.

[BD02] J. Bansiya and C. G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Transaction on Software Engineering*, 28(1):4–17, 2002.

[BDG04] B. Bernandez, A. Duran, and M. Genero. Empirical Evaluation, Review of a Metric-Based Approach for Use Case Verification. *Journal of Research, Practice in Information Technology, Special Collection on Requirements Engineering*, 36(4):247–258, 2004.

[BDM97] L. C. Briand, P. Devanbu, and W. Melo. An Investigation into Coupling Measures for C++. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 412–421, New York, NY, USA, 1997. ACM Press.

[BDR97] L. C. Briand, C. Differding, and D. Rombach. Practical Guidelines for Measurement-Based Process Improvement. *Software Process Improvement and Practice Journal*, 2(4):253–280, 1997.

[BDV04] B. Du Bois, S. Demeyer, and J. Verelst. Refactoring Improving Coupling and Cohesion of Existing Code. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)*, pages 144–151, Washington, DC, USA, 2004. IEEE Computer Society.

[BDW98a] L. C. Briand, J. W. Daly, and J. Wüst. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering*, 3(1):65–117, 1998.

[BDW98b] J. M. Burkhardt, F. Detienne, and S. Wiedenbeck. The Effect of Object-Oriented Programming Expertise in Several Dimensions of Comprehension Strategies. In *IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension*, pages 82–89, Washington, DC, USA, 1998. IEEE Computer Society.

[BDW99] L. C. Briand, J. W. Daly, and J. Wüst. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999.

[BDW02] J. M. Burkhardt, F. Detienne, and S. Wiedenbeck. Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase. *Empirical Software Engineering*, 7(2):115–156, 2002.

[BeA02] A. L. Baroni and F. Brito e Abreu. Formalizing Object-Oriented Design Metrics upon the UML Meta-Model. In *Proceedings of the Brazilian Symposium on Software Engineering, Gramado, Brazil*, 2002.

[BeA03a] A. L. Baroni and F. Brito e Abreu. A Formal Library for Aiding Metrics Extraction. In *International Workshop on Object-Oriented Re-Engineering at ECOOP'2003. Darmstadt, Germany*, 2003.

[BeA03b] A. L. Baroni and F. Brito e Abreu. An OCL-Based Formalization of the MOOSE Metric Suite. In *QUAOOSE' 03: Proceedings of the 7th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Darmstadt, Germany*, 2003.

[BEDL99] J. Bansiya, L. Etzkorn, C. Davis, and W. Li. A Class Cohesion Metric For Object-Oriented Designs. *The Journal of Object-Oriented Programming*, 11(8):47–52, 1999.

[BEGR00] S. Benlardi, K. El Eman, N. Goel, and S. Rai. Thresholds for Object Oriented Measures. In *ISSRE '00: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE'00)*, pages 24–39, Washington, DC, USA, 2000. IEEE Computer Society.

[BEM95] L. C. Briand, K. El Emam, and S. Morasca. Theoretical and Empirical Validation of Software Product Measures. Technical Report ISERN-95-03, ISERN: International Software Engineering Research Network, 1995.

[BH91] D. Bergantz and J. Hassell. Information Relationships in PROLOG Programs: How do Programmers Comprehend Functionality? *International Journal of Man-Machine Studies*, 35(3):313–328, 1991.

[BKW04] H. Baumeister, A. Knapp, and M. Wirsing. Property-Driven Development. In *SEFM '04: Proceedings of the Second International Conference on Software Engineering and Formal Methods*, pages 96–102, Washington, DC, USA, 2004. IEEE Computer Society.

[BLM03] L. C. Briand, Y. Labiche, and Y. Miao. Towards the Reverse Engineering of UML Sequence Diagrams. In *WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering*, pages 57–66, Washington, DC, USA, 2003. IEEE Computer Society.

[BLYP04] L. C. Briand, Y. Labiche, H. D. Yan, and M. Di Penta. A Controlled Experiment on the Impact of the Object Constraint Language in UML-based Maintenance. *Proceeding of the 20th IEEE Int. Conference on Software Maintenance*, pages 380–389, 2004.

[BMB96] L. C. Briand, S. Morasca, and V. R. Basili. Property-Based Software Engineering Measurement. *IEEE Transaction on Software Engeneering*, 22(1):68–86, 1996.

[BMB97] L. C. Briand, S. Morasca, and V. R. Basili. Response to: Comments on "Property-Based Software Engineering Measurement: Refining the Additivity Properties". *IEEE Transactions on Software Engineering*, 23(3):196–197, 1997.

[BMB99] L. C. Briand, S. Morasca, and V. R. Basili. Defining and Validating Measures for Object-Based High-Level Design. *IEEE Transactions on Software Engineering*, 25(5):722–743, 1999.

[BMB02] L. C. Briand, S. Morasca, and V. R. Basili. An Operational Process for Goal-Driven Definition of Measures. *IEEE Transaction on Software Engineering*, 28(12):1106–1125, 2002.

[BP02] G. A. Bunde and A. Pedersen. Defect Reduction by Improving Inspection of UML Diagrams in the GPRS Project. *Master Thesis. Siv.ing. Degree. Information and Communication Technology*, 2002.

[BR98] V. R. Basili and H. D. Rombach. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transaction on Software Engineering*, 14(6):758–773, 1998.

[Bro75] D. E. Broadbent. The Magical Number Seven after Fifteen Years. *In A. Kennedy and A. Wilkes (eds.), Studies in Long-Term Memory, New York: Wiley*, pages 3–18, 1975.

[Bro83] R. Brooks. Towards a Theory of Comprehension of Computer Programs. *International Journal of Man Machine Studies*, 18(6):543–554, 1983.

[BS98] A. B. Binkley and S. R. Schach. Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures. In *ICSE '98: Proceedings of the 20th International Conference on Software Engineering*, pages 452–455, Washington, DC, USA, 1998. IEEE Computer Society.

[BSH86] V. R. Basili, R. W. Selby, and D. H. Hutchens. Experimentation in Software Engineering. *IEEE Transactions on Software Engineering (TSE)*, 12(7):733–743, 1986.

[BSL99] V. R. Basili, F. Shull, and F. Lanubile. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering (TSE)*, 25(4):456–473, 1999.

[BVT03] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk. Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics. *IEEE Transaction on Software Engineering*, 29(1):77–87, 2003.

[BW84] V. R. Basili and D. M. Weiss. A Methodology for Collecting Valid Software Engineering Data. *IEEE Transaction on Software Engineering (TSE)*, 10(6):728–738, 1984.

[BW01] L. C. Briand and J. Wüst. Modeling Development Effort in Object-Oriented Systems Using Design Properties. *IEEE Transactions on Software Engineering*, 27(11):963–986, 2001.

[BWDP00] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter. Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. *Journal of Systems and Software*, 51(0):245–273, 2000.

[BWIL99] L. C. Briand, J. Wüst, S. Ikonomovski, and H. Lounis. A Comprehensive Investigation of Quality Factors in Object-Oriented Designs. In Editor, editor, *IEEE ICSE '99: International Conference on Software Engineering*. Publisher, 1999.

[BWL99] L. C. Briand, J. Wüst, and H. Lounis. Using Coupling Measurement for Impact Analysis in Object-Oriented Systems. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, pages 475–482, Washington, DC, USA, 1999. IEEE Computer Society.

[BWL01] L. C. Briand, J. Wüst, and H. Lounis. Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs. *Empirical Software Engineering*, 6(1):11–58, 2001.

[BWSL99] L. C. Briand, J. Wüst, V. Ikonomovski S, and H. Lounis. Investigating Quality Factors in Object-Oriented Designs: an Industrial Case Study. In *ICSE '99: Proceedings of the 21st International Conference on Software Engineering*, pages 345–354, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[Car93] D. N. Card. What Makes for Effective Measurement? *IEEE Software*, 10(6):94–95, 1993.

[Car98] M. Cartwright. An Empirical View of Inheritance. *Information and Software Technology*, 40(4):795–799, 1998.

[CBC05] D. Chiorean, M. Bortes, and D. Corutiu. Proposals for a Widespread Use of OCL. In Thomas Baar, editor, *MoDELS '05: Proceedings of the Conference Workshop on Tool Support for OCL and Related Formalisms - Needs and Trends, Montego Bay, Jamaica, October 4, 2005*, Technical Report LGL-REPORT-2005-001, pages 68–82. EPFL, 2005.

[CBCS04] D. Chiorean, M. Bortes, D. Corutiu, and R. Sparleanu. UML/OCL Tools - Objectives, Requirements, State of the Art - the OCLE Experience. In *NWPER '04: Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques. Turku, Finland, August 17-20,*, pages 163–180, 2004.

[CC79] T. D. Cook and D. T. Campbell. *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin Company, 1979.

[CCBC04] D. Chiorean, D. Corutiu, M. Bortes, and I. Chiorean. Good Practices for Creating Correct, Clear and Efficient OCL Specifications. In *NWUML '04: Proceedings of the 2nd Nordic Workshop on the Unified Modeling Language*, pages 127–142. TUCS General Publication, 2004.

[CD99] G. Cantone and P. Donzelli. Goal Oriented Software Measurement Models. In *ESCOM-ENCRESS '98: European Software Control and Metrics Conference, Herstmonceux Castle, East Sussex, UK*, 1999.

[CD00] G. Cantone and P. Donzelli. Production and Maintenance of Software Measurement Models. *Journal of Software Engineering and Knowledge Engineering*, 5:605–626, 2000.

[CDE+98] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude as a Metalanguage. In *WRLA '98: 2nd International Workshop on Rewriting Logic and its Applications. Electronic Notes in Theoretical Computer Science*, volume 15. Elsevier, 1998.

[CDE+00] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Towards Maude 2.0. In *Electronic Notes in Theoretical Computer Science*, volume 36, 2000.

[CDE+02] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and Programming in Rewriting Logic. In *Electronic Notes in Theoretical Computer Science*, volume 285, pages 187–243, 2002.

[CDK98] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer. Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis. *IEEE Transactions on Software Engineering*, 24(8):629–639, 1998.

[CES01] D. N. Card, K. El Emam, and B. Scalzo. Measurement of Object Oriented Software Development Projects. *Software Development Consortium*, 2001.

[Chi97] M. T. H. Chi. Quantifying Qualitative Analyses of Verbal Data: a Practical Guide. *Journal of the Learning Sciences*, 6(3):271–315, 1997.

[CHSJ94] S. N. Cant, B. Henderson-Sellers, and D. R. Jeffery. Application of Cognitive Complexity Metrics to Object-Oriented Programs. *Journal of Object-Oriented Programming (JOOP)*, 7(4):52–63, 1994.

[CJHS92] S. N. Cant, D. R. Jeffery, and B. Henderson-Seller. A Conceptual Model of Cognitive Complexity of Elements of the Programming Process. *Information and Software Technology*, 37(7):351–362, 1992.

[CK91] S. R. Chidamber and C. F. Kemerer. Towards a Metrics Suite for Object Oriented Design. In *OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications*, pages 197–211, New York, NY, USA, 1991. ACM Press.

[CK94] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[CKM+02] S. Cook, A. Kleepe, R. Mitchell, B. Rumpe, J. Warmer, and A. Wills. The Amsterdam Manifesto on OCL. In *Object Modeling with the OCL, The Rationale behind the Object Constraint Language*, pages 115–149, London, UK, 2002. Springer-Verlag.

[CL07] J. A. Cruz-Lemus. *A Measurement-based Approach for Assessing UML Statechart*. PhD thesis, PhD Thesis. Universidad de Castilla La Mancha, 2007.

[CLGO+04] J. A. Cruz-Lemus, M. Genero, J. A. Olivas, F. P. Romero, and M. Piattini. Predicting UML Statechart Diagrams Understandability Using Fuzzy Logic-Based Techniques. *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004), Banff, Alberta, Canada, June 20-24, 2004*, pages 238–245, 2004.

[CLGO+05] J. A. Cruz-Lemus, M. Genero, J. A. Olivas, F. P. Romero, and M. Piattini. Evaluating the Effect of Composite States on the Understandability of UML States on Understandability of UML Statechard Diagrams. In *MODELS '05: Model Driven Engineering Languages and Systems, 8th International Conference*, pages 113–125, 2005.

[CLGP05] J. A. Cruz-Lemus, M. Genero, and M. Piattini. Metrics for UML Statechart Diagrams. In *Metrics for Software Conceptual Models. Genero, Piattini and Calero (eds.)*. Imperial College Press, UK., 2005.

[CLGPM06] J. A. Cruz-Lemus, M. Genero, M. Piattini, and S. Morasca. Improving the Experimentation for Evaluating the Effect of Composite States on the Understandability of UML Statechart Diagrams. In *ISESE '06: Proceedings of the 2006 ACM/IEEE International Symposium on International Symposium on Empirical Software Engineering*, pages 9–11, New York, NY, USA, 2006. ACM Press.

[CLGPT05] J. A. Cruz-Lemus, M. Genero, M. Piattini, and J. A. Toval. An Empirical Study of the Nesting Level of Composite States Within UML Statechar Diagrams. In *BP-UML '05: 1st International Workshop on Best Practices of UML, ER (Workshops)*, pages 12–22. Springer, 2005.

[CMF04] V. M. Chieu, E. Milgrom, and M. Frenay. Constructivist Learning: Operational Criteria for Cognitive Flexibility. In *ICALT '04: Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT'04)*, pages 221–225, Washington, DC, USA, 2004. IEEE Computer Society.

[CMSD04] E. Cariou, R. Marvie, L. Seinturier, and L. Duchien. OCL for the Specification of Model Transformation Contracts. In Octavian Patrascoiu, editor, *UML '04: OCL and Model Driven Engineering Workshop of the Seventh International Conference on UML Modeling Languages and Applications Conference, Lisbon, Portugal*, pages 69–83. University of Kent, 2004.

[CPG01] C. Calero, M. Piattini, and M. Genero. Method for Obtaining Correct Metrics. In *ICEIS '01: Proceedings of the 3rd International Conference on Enterprise and Information Systems*, volume 2, pages 779–784, 2001.

[CS63] D. T. Campbell and J. Stanley. Experimental and Quasi-Experimental Designs for Research on Teaching. In *Handbook of research on teaching, Rand McNally, Chicago*, pages 171–246. Houghton Mifflin Co, 1963.

[CS02] M. Carbone and G. Santucci. Fast and Serious: a UML Based Metric for Effort Estimation. In *QAOOSE '02: Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Malaga, Spain*, pages 35–44, 2002.

[CSB02] M. Ciolkowski, F. Shull, and S. Biffl. A Family of Experiments to Investigate the Influence of Context on the Effect of Inspection Techniques. In *EASE '02: Proceedings of the 6th International Conference on Empirical Assesment in Software Engineering, Keele, UK*, pages 48–60, 2002.

[CW99a] A. L. Correa and C. M. L. Werner. Applying Refactoring Techniques to UML/OCL Models. In T. Baar et al., editor, *UML'04: The Unified Modeling Language*, volume 3273 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 1999.

[CW99b] C. L. Corritore and S. Wiedenbeck. Mental Representations of Expert Procedural and Object-Oriented Programmers in a Software Maintenance Task. *International Journal of Human-Computer Studies*, 50(1):61–83, 1999.

[CW00] C. L. Corritore and S. Wiedenbeck. Direction and Scope of Comprehension-Related Activities by Procedural and Object-Oriented Programmers: An Empirical Study. In *IWPC '00: Proceedings of the 8th International Workshop on Program Comprehension*, page 139, Washington, DC, USA, 2000. IEEE Computer Society.

[CW04] A. L. Correa and C. M. L. Werner. Precise Specification and Validation of Transactional Business Software. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International (RE'04)*, pages 16–25, Washington, DC, USA, 2004. IEEE Computer Society.

[CWD00] M. Casanova, T. Wallet, and M. D'Hondt. Adaptations to OCL for Ensuring Quality of Geographic Data (Poster Session). *OOPSLA '00: Addendum to the 2000 proceedings of the conference on Object-oriented programming, systems, languages, and applications (Addendum)*, pages 69–70, 2000.

[Dav95] J. S. Davis. A Guessing Measure of Program Comprehension. *International Journal of Human-Computer Studies*, 42(3):245–263, 1995.

[DBM+96] J. Daly, A. Brooks, J. Miller, M. Roper, and M. Wood. An Empirical Study Evaluating Depth of Inheritance on the Maintainability of Object-Oriented Software. *Empirical Software Engineering*, 1(2):109–132, 1996.

[Der95] K. Derr. *Applying OMT: A Practical Step-by-Step Guide to Using the Object Modeling Technique.* SIGS Publications, Inc., New York, NY, USA, 1995.

[DJ03] M. Dagpinar and J. H. Jahnke. Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison. In *WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering*, page 155, Washington, DC, USA, 2003. IEEE Computer Society.

[DS05] D. P. Darcy and S. A. Slaughter. The Structural Complexity of Software: An Experimental Test. *IEEE Transactions on Software Engineering*, 31(11):982–995, 2005. Member-Chris F. Kemerer and Member-James E. Tomayko.

[Dun89] G. Dunteman. *Principal Component Analysis*. Sage University Paper 07-69, Thousand Oaks, CA, USA, 1989.

[eAC94] F. Brito e Abreu and R. Carapuga. Object-Oriented Software Engineering: Measuring and Controlling the Development Process. In *ICSQ '94: Proceedings of the 4th International Conference on Software Quality, Mc Lean, VA, USA*, pages 3–5, 1994.

[eAEG96] F. Brito e Abreu, R. Esteves, and M. Goulao. The Design of Eiffel Programs: Quantitative Evaluation Using the MOOD Metrics. In *TOOLS USA '96: Proceedings of the Technology of Object Oriented Languages and Systems, Santa Barbara, California, USA*, 1996.

[eAGE95] F. Brito e Abreu, M. Goulao, and R. Esteves. Towards the Design Quality Evaluation of Object-Oriented Software System. In *ICSQ '95: Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA*, 1995.

[eAM96] F. Brito e Abreu and W. Melo. Evaluating the Impact of Object-Oriented Design on Software Quality. In *METRICS '96: Proceedings of the 3rd International Symposium on Software Metrics*, page 90, Washington, DC, USA, 1996. IEEE Computer Society.

[EBG+02] K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S. N. Rai. The Optimal Class Size for Object-Oriented Software. *IEEE Transaction on Software Engineering*, 28(5):494–509, 2002.

[EBGR99] K. El Emam, S. Benlarbi, N. Goel, and S. Rai. A Validation of Object-Oriented Metrics. Technical Report NRC/ERB-1063, National Research Council of Canada, 1999.

[EBGR01] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai. The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. *IEEE Transaction on Software Engineering*, 27(7):630–650, 2001.

[Ecl00] Eclipse. *Eclipse Fundation Inc.* Ottawa, Ontario, Canada. www.eclipse.org, 2000.

[EK95] K. A. Ericsson and W. Kintsch. Long Term Working Memory. *Psychological Review 102*, pages 211–245, 1995.

[EKS94] J. Eder, G. Kappel, and M. Schrefl. Coupling and Cohesion in object-oriented systems. Technical report, Technical Report, University of Klagenfurt, Austria, 1994.

[Ema01] K. El Emam. Object-Oriented Metrics: A Review of Theory and Practice. Technical Report NRC 44190, National Research Council Canada. Institute for Information Technology, 2001.

[Ema02] K. El Emam. Object-Oriented Metrics: A Review of Theory and Practice. *Advances in Software Engineering*, pages 23–50, 2002.

[EMM01] K. El Emam, W. Melo, and J. C. Machado. The Prediction of Faulty Classes using Object-Oriented Design Metrics. *Journal of Systems and Software*, 56(1):63–75, 2001.

[ES93] K. A. Ericsson and H. Simon. *Protocol Analysis: Verbal Reports as Data*. MIT Press, Cambridge, MA, 1993.

[ES98] K. Erdös and H. M. Sneed. Partial Comprehension of Complex Programs (Enough to Perform Maintenance). In *IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension*, pages 98–107, Washington, DC, USA, 1998. IEEE Computer Society.

[FCTJ01] A. Friis-Christensen, N. Tryfona, and C. S. Jensen. Requirements and Research Issues in Geographic Data Modeling. *GIS '01: Proceedings of the 9th ACM international symposium on Advances in geographic information systems*, pages 2–8, 2001.

[Fel00] P. Feldt. *Requirements Metrics Based on Use Cases. Master's Thesis.* PhD thesis, Department of Communication Systems, Lund Institute of Technology, Lund University, Box 118, S-221 00 Lund, Sweden, 2000.

[Fla02] S. Flake. Real-time Constraints with the OCL. In *ISORC '02: Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 425–426, Washington, DC, USA, 2002. IEEE Computer Society.

[FM02] S. Flake and W. Mueller. Specification of Real-Time Properties for UML Models. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9*, page 277, Washington, DC, USA, 2002. IEEE Computer Society.

[FP98] N. E. Fenton and S. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 1998.

[FS00] M. Fowler and K. Scott. *UML Distilled. Second Edition.* Addison-Wesley, 2000.

[GB01] N. Gold and K. Bennett. A Flexible Method for Segmentation in Concept Assignment. In *IWPC '01: Proceedings of the 9th International Workshop on Program Comprehension*, pages 135–144, Washington, DC, USA, 2001. IEEE Computer Society.

[GEMM00] D. Glasberg, K. El Emam, W. Melo, and N. Madhavji. Validating Object-Oriented Design Metrics on a Commercial Java Application. Technical Report NRC/ERB-1080, National Research Council of Canada, 2000.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Longman Publishing Co., Inc., Boston, MA, 1995.

[GJG04] P. Van Gorp, D. Janssens, and T. Gardner. Write Once, Deploy N: A Performance Oriented MDA Case Study. In *EDOC '04: Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04)*, pages 123–134, Washington, DC, USA, 2004. IEEE Computer Society.

[GL05] M. Giese and D. Larsson. Simplifying Transformation of OCL Constraints. In L. Briand and C. Williams, editors, *MoDELS '05: Proceedings of the 8th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, October 4, 2005*, volume 3713 of *LNCS*, pages 309–323. EPFL, 2005.

[GO05] R. Gronmo and J. Oldevik. An Empirical Study of the UML Model Transformation Tool (UMT). In *The First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA), Geneva, Switzerland*, 2005.

[Gop91] R. Gopal. Dynamic Program Slicing Based on Dependence Relations. In *ICSM '91: Proceedings of the Conference on Software Maintenance, Sorrento, Italy*, pages 191–200, 1991.

[GPC00] M. Genero, M. Piattini, and C. Calero. Early Measures for UML Class Diagrams. *L'Objet: Software, Databases, Networks*, 6(4):495–515, 2000.

[GPE05] M. Genero, M. Piattini, and M. Calero (Eds.). *Metrics For Software Conceptual Models*. Imperial College Press, UK, 2005.

[Ham99] A. Hamie. Enhancing the Object Constraint Language for More Expressive Specifications. In *APSEC '99: Proceedings of the Sixth Asia Pacific Software Engineering Conference*, pages 376–383, Washington, DC, USA, 1999. IEEE Computer Society.

[Ham04] A. Hamie. Translating the Object Constraint Language into the Java Modelling Language. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1531–1535, New York, NY, USA, 2004. ACM Press.

[Hay05] A. F. Hayes. *Statistical Methods for Communication Science*. Lawrence Erlbaum Assoc Inc, 2005.

[HCN98a] R. Harrison, S. Counsell, and R. Nithi. Coupling Metrics for Object-Oriented Design. In *METRICS '98: Proceedings of the 5th International Symposium on Software Metrics*, pages 150–156, Washington, DC, USA, 1998. IEEE Computer Society.

[HCN98b] R. Harrison, S. J. Counsell, and R. V. Nithi. An Evaluation of the MOOD Set of Object-Oriented Software Metrics. *IEEE Transaction on Software Engineering*, 24(6):491–496, 1998.

[HCN00] R. Harrison, S. Counsell, and R. Nithi. Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems. *Journal of Systems and Software*, 52(2-3):173–179, 2000.

[Hen02] J. B. Van Der Henstl. Mental Model Theory Versus the Inference Rule Approach in Relational Reasoning. *Thinking and Reasoning*, 8:193–205, 2002.

[HHC04] B. C. Hungerford, A. R. Hevner, and R. W. Collins. Reviewing Software Diagrams: A Cognitive Study. *IEEE Transaction on Software Engineering*, 30(2):82–96, 2004.

[HHW04] B. Hofreiter, C. Huemer, and W. Winiwarter. OCL-Constraints for UMM Business Collaborations. In *EC-Web '04: Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies, Zaragoza, Spain*, volume 3182 of *LNCS*, pages 174–185, 2004.

[HK99] J. Hahn and J. Kim. Why are Some Representations (Sometimes) More Effective? In *Proceeding of the 20th international conference on Information Systems*, pages 245–259. Association for Information Systems, 1999.

[HL81] E. L. Hutchins and J. A. Levin. Point of View in Problem Solving. Technical Report CHIP TR-105, California University at San Dingo, 1981.

[HM95] M. Hitz and B. Montazeri. Measuring Coupling and Cohesion in Object-Oriented Systems. In *ISACC '95: Proceedings of the 3rd International Symposium on Applied Corporate Computing, Monterrey, Mexico*, 1995.

[HP03] J. Huges and S. Parkes. Trends in the Use of Verbal Protocol Analysis in Software Engineering Research. *Behaviour and Information Technology, BIT*, 22(2):127–140, 2003.

[HS96] B. Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[HSZKP02] B. Henderson-Sellers, D. Zowghi, T. Klemola, and S. Parasuram. Sizing Use Cases: How to Create a Standard Metrical Approach. In *OOIS '02: Proceedings of the 8th International Conference on Object-Oriented. Information Systems*, pages 409–421, London, UK, 2002. Springer-Verlag.

[HWT05] M. Host, C. Wohlin, and T. Thelin. Experimental Context Classification: Incentives and Experience of Subjects. In *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*, pages 470–478, New York, NY, USA, 2005. ACM Press.

[HZ04] H. Hussmann and S. Zschaler. The Object Constraint Language for UML 2.0 Overview and ASsessment. *UPGRADE: Digital Journal of CEPIS (Council of European Professional Informatics Societies*, V:25–28, 2004.

[IKB03] P. In, S. Kim, and M. Barry. UML-Based Object-Oriented Metrics for Architecture Complexity Analysis. In *GSAW '03: Proceedings of Ground System Architectures Workshop. El Segundo, CA, The Aerospace Corporation*, 2003.

[ISO01] ISO. *IEC 9126-1 Information Technology - Software Product Quality - Part 1: Quality Model.* ISO, 2001.

[JA97] J. P. Jacquet and A. Abran. From Software Metrics to Software Measurement Methods:. In *ISESS '97: Proceedings of the 3rd International Software Engineering Standards Symposium (ISESS '97)*, pages 128–135, Washington, DC, USA, 1997. IEEE Computer Society.

[JCF03] S. R. Judson, D. L. Carver, and R. B. France. A Metamodeling Approach to Model Transformation. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 326–327, New York, NY, USA, 2003. ACM Press.

[JCJO92] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1992.

[JM01] N. Juristo and A. Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001.

[JP05] A. Jedlitschka and D. Pfahl. Reporting Guidelines for Controlled Experiments in Software Engineering. In *ISESE '05: International Symposium on Empirical Software Engineering (ISESE 2005), 17-18 November 2005, Noosa Heads, Australia*, pages 95–104, 2005.

[KAKB+06] B. Kitchenham, H. Al-Khilidar, M. Ali Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu. 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*, pages 38–47, New York, NY, USA, 2006. ACM Press.

[Kan04] D. Kang. A Complexity Measure for Ontology Based on UML. In *FTDCS '04: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)*, pages 222–228, Washington, DC, USA, 2004. IEEE Computer Society.

[Kar93] G. Karner. Metrics for objectory. *Master Thesis, Linköping University, Linköping, Sweden*, LiTH-IDA-Ex-9344:21, 1993.

[KB02] H. Kim and C. Boldyreff. Developing Software Metrics Applicable to UML Models. In *QAOOSE '02: 6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering. Málaga, Spain*, pages 67–76, 2002.

[KDST06] D. H. Krantz, D. R. Duncan, P. Suppes, and A. Tversky. *Foundations of Measurement Volume II: Geometrical, Threshold, and Probabilistic Representations (Foundations of Measurement)*. Dover Publications, 2006.

[Kim99] H. Kim. *Representing and Reasoning about Quality using Enterprise Models*. PhD thesis, Dept. Mechanical and Industrial Engineering, University of Toronto, Canada, 1999.

[KKM88] D. G. Kleinbaum, L. L. Kupper, and K. E. Muller. *Applied Regression Analysis and other Multivariable Methods*. PWS Publishing Co., Boston, MA, USA, 1988.

[Kle00] T. Klemola. 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.

[KLST06] D. H. Krantz, D. R. Luce, P. Suppes, and A. Tversky. *Foundations of Measurement Volume I: Additive and Polynomial Representations (Foundations of Measurement)*. Dover Publications, 2006.

[KM02] C. Knight and M. Munro. Program Comprehension Experiences with GXL; Comprehension for Comprehension. In *IWPC '02: Proceedings of the 10th International Workshop on Program Comprehension*, page 147, Washington, DC, USA, 2002. IEEE Computer Society.

[KPF95] B. Kitchenham, S. L. Pfleeger, and N. Fenton. Towards a Framework for Software Measurement Validation. *IEEE Transaction on Software Engineering*, 21(12):929–944, 1995.

[KR02] T. Klemola and J. Rilling. Modeling Comprehension Processes in Software Development. In *ICCI '02: Proceedings of the 1st IEEE International Conference on Cognitive Informatics*, pages 329–339, Washington, DC, USA, 2002. IEEE Computer Society.

[KS97] B. Kitchenham and J. Stell. The Danger of Using Axioms in Software Metrics. In *IEEE Proceedings on Software Engineering*, volume 144, pages 279–285, 1997.

[LC94] A. Lake and C. Cook. Use of Factor Analysis to Develop OOP Software Complexity Metrics. Technical report, Oregon State University, Corvallis, OR, USA, 1994.

[LEH01] O. Laitenberger, K. El Emam, and T. G. Harbich. An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-based Reading of Code Documents. Technical Report IESE-Report, 006.99/E, Fraunhofer IESE, 2001.

[LH93] W. Li and S. Henry. Object-Oriented Metrics that Predict Maintainability. *Journal of Systems and Software*, 23(2):111–122, 1993.

[Lia04] W. Liang. UML Object Constraint Language in Meta-Modeling. Technical report, School of Computer Science. McGill University, 2004.

[LK94] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics: A Practical Guide*. Prentice-Hall, Inc, Upper Saddle River, NJ, USA, 1994.

[LLWW95] Y. S. Lee, B. S. Liang, S. F. Wu, and F. J. Wang. Measuring the Coupling and Cohesion of an Object Oriented Program Based on Information Flow. In *Proceedings of International Conference Software Quality*, pages 81–90, 1995.

[LPLS87] D. C. Littman, J. Pinto, S. Letovsky, and E. Soloway. Mental Models and Software Maintenance. *Journal of Systems and Software*, 7(4):341–355, 1987.

[LR96] C. M. Lott and H. D. Rombach. Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques. *Empirical Software Engineering: An International Journal*, 1(3):241–277, 1996.

[LS87] J. Larkin and H. Simon. Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, 11:65–99, 1987.

[Mar98] M. Marchesi. OOA Metrics for the Unified Modeling Language. *2nd Euromicro Conference on Software Maintenance and Reengineering*, pages 67–73, 1998.

[MB97] S. Morasca and L. C. Briand. Towards a Theoretical Framework for Measuring Software Attributes. In *METRICS '97: Proceedings of the 4th International Symposium on Software Metrics*, pages 119–126, Washington, DC, USA, 1997. IEEE Computer Society.

[MB00] M. G. Mendonca and V. R. Basili. Validation of an Approach for Improving Existing Measurement Frameworks. *IEEE Transaction on Software Engineering*, 28(6):484–509, 2000.

[MEJ+03] G. Miller, A. Evans, I. Jacobson, H. Jondell, A. Kennedy, S. Mellor, and D. Thomas. Model Driven Architecture: How Far Have We Come, How Far Can We Go? In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 273–274, New York, NY, USA, 2003. ACM Press.

[MG04] A. Mohan and N. Gold. Programming Style Changes in Evolving Source Code. In *IWPC '04: Proceedings of the 12th IEEE International Workshop on Program Comprehension*, pages 236–240, Washington, DC, USA, 2004. IEEE Computer Society.

[MGBB90] A. C. Melton, D. A. Gustafson, J. M. Bieman, and A. L. Baker. A Mathematical Perspective for Software Measures Research. *Software Engineering Journal*, 5(5):246–254, 1990.

[MGP03] D. Miranda, M. Genero, and M. Piattini. Empirical Validation of Metrics for UML Statechart Diagrams. In *ICEIS '03: Fifth International Conference on Enterprise Information Systems*, pages 87–95, 2003.

[Mil56] G. A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limtis on our Capacity for Processing Information. *Psycological Review*, 63:81–97, 1956.

[Mil00] J. Miller. Applying Meta-analytical Procedures to Software Engineering Experiments. *J. Syst. Softw.*, 54(1):29–39, 2000.

[MM04] N. MacKinnon and S. Murphy. Designing UML Diagrams for Technical Documentation: Continuing the Collaborative Approach to Publishing Class Diagrams. In *SIGDOC '04: Proceedings of the 2nd Annual International Conference on Design of Communication*, pages 120–127, New York, NY, USA, 2004. ACM Press.

[MPKS00] S. Muthanna, K. Ponnambalam, K. Kontogiannis, and B. Stacey. A Maintainability Model for Industrial Software Systems Using Design Level Metrics. In *WCRE '00: Proceedings of the Seventh Working Conference on Reverse Engineering*, page 248, Washington, DC, USA, 2000. IEEE Computer Society.

[MRRR02] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins. Modeling Software Architectures in the Unified Modeling Language. *Transactions on Software Engineering and Methodology*, 11(1):2–57, 2002.

[MRW77] J. A. McCall, P. K. Richards, and G. F. Walters. 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.

[MTH04] S. Murphy, S. Tilley, and S. Huang. 4th Workshop on Graphical Documentation: UML Style Guidelines. In *SIGDOC '04: Proceedings of the 22nd annual international conference on Design of communication*, pages 118–119, New York, NY, USA, 2004. ACM Press.

[Mut00] D. Muthiayen. *Real Time Reactive System Development - A Formal Approach Based on UML and PVS.* PhD thesis, Deparment of Computer Science at Concordia University, Montreal, Canada., 2000.

[MV96] A. V. Mayrhauser and A. M. Vans. Identification of Dynamic Comprehension Processes During Large Scale Maintenance. In *IEEE Transaction on Software Engineering*, volume 22, pages 424–437, Piscataway, NJ, USA, 1996. IEEE Press.

[MV04] B. A. Malloy and J. M. Voas. Programming with Assertions: a Prospectus. *IT Professional*, 6(5):53–59, 2004.

[MW01] R. Mosemann and S. Wiedenbeck. Navigation and Comprehension of Programs by Novice Programmers. In *IWPC '01: Proceedings of the 9th International Workshop on Program Comprehension*, page 79, Washington, DC, USA, 2001. IEEE Computer Society.

[NF06] C. Nebut and F. Fleurey. Automatic Test Generation: A Use Case Driven Approach. *IEEE Transactions on Software Engineering*, 32(3):140–155, 2006.

[Nun03] Isabel Nunes. An OCL Extension for Low-Coupling Preserving Contracts. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, volume 2863 of *LNCS*, pages 310–324. Springer, 2003.

[Obj00] Klasse Objecten. *OCTUPUS: OCL Tool for Precise UML Specification*. http://www.klasse.nl/octopus/index.html, 2000.

[OMG03a] Object Management Group OMG. *MDA - The OMG Model Driven Architecture*. OMG, 2003.

[OMG03b] Object Management Group OMG. UML 2.0 OCL Final Adopted Specification. OMG Document ad/2003-01-07, 2003.

[OMG03c] Object Management Group OMG. UML Specification. OMG Document formal/03-03-01, 2003.

[OMG05a] Object Management Group OMG. MOF QVT Final Adopted Specification. OMG Document ptc/05-11-01, 2005.

[OMG05b] Object Management Group OMG. UML 2.0 OCL Available Specification (FTF Report). OMG Document ptc/2005-06-06, 2005.

[oT01] Software Solutions on Time. A Fresh and Innovative Approach to Systems Development and Software Project Management. *Available in http://www.tassc-solutions.com/omx/pages/metric_ data.htm#usecase-metrics*, 2001.

[Pat04] O. Patrascoiu. YATL: Yet Another Transformation Language. In *MDA-IA: Proceedings of the First European Workshop on Model Driven Architecture with Emphasis on Industrial Application. University of Twente, Enschede, the Netherlands*, pages 83–90, 2004.

[PD97] G. Poels and G. Dedene. Comments on "Property-Based Software Engineering Measurement: Refining the Additivity Properties". *IEEE Transaction on Software Engineering*, 23(3):190–195, 1997.

[PD99] G. Poels and G. Dedene. DISTANCE: A Framework for Software Measure Construction. Technical Report DTEW9937, Dept. Applied Economics, Katholieke Universiteit Leuven, Belgium, 46 p., 1999.

[PD00] G. Poels and G. Dedene. Distance-based Software Measurement: Necessary and Sufficient Properties for Software Measures. *Information and Software Technology*, 42(1):35–46, 2000.

[PD01] G. Poels and G. Dedene. Evaluating the Effect of Inheritance on the Modifiability of Object-Oriented Business Domain Models. In *CSMR '01: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 20–29, Washington, DC, USA, 2001. IEEE Computer Society.

[Pen87] N. Pennington. Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs. *Cognitive Psychology, 19*, pages 295–341, 1987.

[PJ97] A. A. Porter and P. M. Johnson. Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies. *IEEE Trans. Softw. Eng.*, 23(3):129–145, 1997.

[PJCK97] S. L. Pfleeger, R. Jeffery, B. Curtis, and B. Kitchenham. Status Report on Software Measurement. *IEEE Software*, 14(2):33–43, 1997.

[Poe99] G. Poels. *On the Formal Aspects of the Measurement of Object-Oriented Software Specifications*. PhD thesis, Faculty of Economics and Business Administration. Katholieke Universiteit Leuven, Belgium, 1999.

[PPV00] D. E. Perry, A. A. Porter, and L. G. Votta. Empirical Studies of Software Engineering: a Roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 345–355, New York, NY, USA, 2000. ACM Press.

[QaMKI04] L. M. Quiroga and M. E. Crosby adn M. K. Iding. Reducing Cognitive Load. In *HICSS'04: Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, volume 5, page 50131, Washington, DC, USA, 2004. IEEE Computer Society.

[QT06] A. Queralt and E. Teniente. Reasoning on UML Class Diagrams with OCL Constraints. In *ER '06: Proceedings of the 25th International Conference on Conceptual Modeling. LNCS 4215*, pages 497–512, 2006.

[RG98] M. Richters and M. Gogolla. On Formalizing the UML Object Constraint Language OCL. In Tok-Wang Ling, Sudha Ram, and Mong Li Lee, editors, *ER '98: Proc. 17th International Conference on Conceptual Modeling*, volume 1507 of *LNCS*, pages 449–464. Springer-Verlag, 1998.

[RGP04a] L. Reynoso, M. Genero, and M. Piattini. Validating Metrics for OCL Expressions Expressed within UML/OCL Models. In *SAM '2004: Proceedings of the 1st International Workshop on Software Audits and Metrics, April 13-14, 2004 - Porto, Portugal*, pages 59–68, 2004.

[RGP04b] L. Reynoso, M. Genero, and M. Piattini. Validating Metrics for OCL Expressions Expressed within UML/OCL Models. In *ISESE '04: ACM-IEEE International Symposium on Empirical Software Engineering. 19-20 August 2004 Redondo Beach CA, USA*, pages 15–16, 2004.

[RGP04c] L. Reynoso, M. Genero, and M. Piattini. Validating OCL Metrics through a Family of Experiments. In *JISBD '04: IX Jornadas de Ingenieria del Software y Base de Datos. 10-12 November 2004*, pages 475–482, 2004.

[Ric02] M. Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, Logos Verlag, Berlin, BISS Monographs, No. 14, 2002.

[RK03] J. Rilling and T. Klemola. Identifying Comprehension Bottlenecks Using Program Slicing and Cognitive Complexity Metrics. In *IWPC '03: Proceedings of the 11th IEEE International Workshop on Program Comprehension*, pages 115–119, Washington, DC, USA, 2003. IEEE Computer Society.

[RLW04] V. Ramalingam, D. L., and S. Wiedenbeck. Self-efficacy and Mental Models in Learning to Program. In *ITiCSE '04: Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 171–175, New York, NY, USA, 2004. ACM Press.

[RM00] L. Reynoso and R. Moore. GoF Behavioural Patterns: A Formal Specification. Technical Report 201, International Institute for Software Technology, United Nations University, UNU/IIST, P.O.Box 3058, Macau, 2000.

[Rob79] F. S. Roberts. *Measurement Theory with Applications to Decisionmaking. Utility and the Social Sciences.* Addison-Wesley, 1979.

[Rob93] C. Robson. Real World Research: A Resource for Social Scientists and Practioners-Researchers. *Blackwell, Oxford*, 1993.

[Rou03] B. Roussev. Generating OCL Specifications and Class Diagrams from Use Cases: A Newtonian Approach. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences - Track 9*, page 321.2, Washington, DC, USA, 2003. IEEE Computer Society.

[RW02] V. Rajlich and N. Wilde. The Role of Concepts in Program Comprehension. In *IWPC '02: Proceedings of the 10th International Workshop on Program Comprehension*, page 271, Washington, DC, USA, 2002. IEEE Computer Society.

[Sae03] M. Saeki. Embedding Metrics into Information System Development Methods: An Application of Method Engineering Technique. *Lecture Notes in Computer Science 2681*, pages 374–389, 2003.

[SB82] M. Sebrechts and J. Black. Software Psychology: A Rich New Domain for Applied Psychology. *Applied Psycholinguistics*, 3:223–232, 1982.

[SB99] R. V. Solingen and E. Berghout. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development.* McGraw-Hill, 1999.

[SB01] R. V. Solingen and E. Berghout. 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*, pages 246–259, Washington, DC, USA, 2001. IEEE Computer Society.

[SBGE82] E. Soloway, J. Bonar, J. Greenspan, and K. Ehrlich. What Do Novices Know About Programming? In *Directions in Human-Computer Interactions*. B. Shneiderman and A. Badre, Ablex Publishing Company, 1982.

[SBS94] M. W. V. Someren, Y. F. Barnard, and J. Sandberg. *The Think Aloud Method: a Practical Guide to Modelling Cognitive Processes.* Academic Press, London, 1994.

[SC02] J. L. Sourrouille and G. Caplat. Constraint Checking in UML Modeling. In *SEKE '02: Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering*, pages 217–224, New York, NY, USA, 2002. ACM Press.

[Sch92] N. F. Schneidewind. Methodology for Validating Software Metrics. *IEEE Transactions on Software Engineering*, 18(5):410–422, 1992.

[SCT+03] F. Shull, J. Carver, G. H. Travassos, J. C. Maldonado, R. Conradi, and V. R. Basili. Replicated Studies: Building a Body of Knowledge about Software Reading Techniques. *Lecture Notes on Empirical Software Engineering*, pages 39–84, 2003.

[Sea99] C. B. Seaman. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999.

[Sel03] B. Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5):19–25, 2003.

[Sen03] S. Sendall. Supporting Model-to-Model Transformations: The VMT Approach. Technical Report TR-CTIT-03-27, Workshop on Model Driven Architecture: Foundations and Applications. University of Twente, The Netherlands, 2003.

[SFM99] M. A. D. Storey, F. D. Fracchia, and H. A. Müller. Cognitive Design Elements to Support the Construction of a Mental Model during Software Exploration. *The Journal of Systems and Software*, 44(3):171–185, 1999.

[SHH+05] D. Sjoberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasanovic, N. Liborg, and A. Rekdal. A Survey of Controlled Experiments in Software Engineering. In *IEEE Transactions on Software Engineering*, volume 31, pages 733–753, 2005.

[Sia99] K. Siau. Information Modeling and Method Engineering: a Psychological Perspective. *Journal of Database Management*, 10(4):44–50, 1999.

[Sin99] J. Singer. Using the American Psychological Association (APA) Style Guideline to Report Experimental Results. In *WESS '99: Proceedings of Workshop on Empirical Studies in Software Maintenance, Oxford. England*, pages 71–75, 1999.

[SK03] R. Subramanyam and M. S. Krishnan. Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software

Defects. *IEEE Transaction on Software Engineering*, 29(4):297–310, 2003.

[SM79] B. Shneiderman and R. Mayer. Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results. *International Journal of Computer and Information Services, 7*, pages 219–239, 1979.

[SMC99] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured Design. *IBM Systems Journal*, 38(2-3):231–256, 1999.

[Smi99] J. Smith. The Estimation of Effort based on Use Cases. *(Rational Software white paper). Rational Software.Available in http://www.rational.com/media/whitepapers/finalTP171.PDF*, 1999.

[SPS02] SPSS. *Syntax Reference Guide, SPSS version 13*. SPSS Inc. Chicago, 2002.

[SSR04] M. Satpathy, N. T. Siebel, and D. Rodriguez. Assertions in Object Oriented Software Maintenance: Analysis and Case Study. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 124–135, Washington, DC, USA, 2004. IEEE Computer Society.

[Sto05] M. A. Storey. Theories, Methods and Tools in Program Comprehension: Past, Present and Future. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, pages 181–191, Washington, DC, USA, 2005. IEEE Computer Society.

[SV98] T. M. Shaft and I. Vessey. The Relevance of Application Domain Knowledge: The Case of Computer Program Comprehension. *Journal on Management of Information Systems*, 15(1):51–78, 1998.

[SW98] G. Schneider and J. P. Winters. *Applying Use Cases: A Practical Guide*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1998.

[SW01] A. Schleicher and B. Westfechtel. Beyond Stereotyping: Metamodeling Approaches for the UML. In *HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 3*, page 3051, Washington, DC, USA, 2001. IEEE Computer Society.

[TB84] S.J. Taylor and R. Bogdan. *Introduction to Qualitative Research Methods*. New York: John Wiley and Sons, 1984.

[Tch02] A. Tchertchago. *Analysis of the Metamodeling Semantics for OCL.* Masther Thesis. Department of Computer Science. Dresden University of Technology, 2002.

[TFS06] C. Tibermacine, R. Fleurquin, and S. Sadou. Simplifying Transformation of Software Architecture Constraints. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1240–1244, New York, NY, USA, 2006. ACM Press.

[TH03] S. Tilley and S. Huang. A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding. In *SIGDOC '03: Proceedings of the 21st annual international conference on Documentation*, pages 184–191, New York, NY, USA, 2003. ACM Press.

[TKC99] M. Tang, M. Kao, and M. Chen. An Empirical Study on Object-Oriented Metrics. In *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, pages 242–249, Washington, DC, USA, 1999. IEEE press.

[Tor04] M. Torchiano. Empirical Assessment of UML Static Object Diagrams. In *IWPC '04: Proceedings of the 12th IEEE International Workshop on Program Comprehension*, pages 226–230, Washington, DC, USA, 2004. IEEE Computer Society.

[UPP98] B. Unger, L. Prechelt, and M. Philippsen. The Impact of Inheritance Depth on Maintenance Tasks: Detailed Description and Evaluation of Two Experiment Replications. *Technical Report, Karlsruhe University: Karlsruhe, Germany*, 1998.

[Ver05] J. Verelst. The Influence of the Level of Abstraction on the Evolvability of Conceptual Models of Information Systems. *Empirical Software Engineering*, 10(4):467–494, 2005.

[VS02] B. Verheecke and R. Van Der Straeten. Specifying and Implementing the Operational Use of Constraints in Object-Oriented Applications. In *CRPITS '02: Proceedings of the Fortieth International Conference on Tools Pacific*, pages 23–32, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[WCH02] X. Wang, C. W. Chan, and H. J. Hamilton. Design of Knowledge-based Systems with the Ontology-domain-system Approach. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 233–236, New York, NY, USA, 2002. ACM Press.

[Wey88] E. J. Weyuker. Evaluating Software Complexity Measures. *IEEE Transaction on Software Engineering*, 14(9):1357–1365, 1988.

[Whi97] S. A. Whitmire. *Object Oriented Design Measurement*. John Wiley and Sons, Inc., New York, NY, USA, 1997.

[WK99] J. Warmer and A. Kleppe. *The Object Constraint Language. Precise Modeling with UML*. Object Technology Series. Addison Wesley, 1999.

[WK03] J. Warmer and A. Kleppe. *The Object Constraint Language. Second Edition. Getting Your Models Ready for MDA*. Object Technology Series. Addison-Wesley, 2003.

[WRH+00] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: an Introduction*. Kluwer Academic Publishers, 2000.

[YTB05] H. Y. Yang, E. Tempero, and R. Berrigan. Detecting Indirect Coupling. In *ASWEC '05: Proceedings of the 2005 Australian conference on Software Engineering*, pages 212–221, Washington, DC, USA, 2005. IEEE Computer Society.

[YWG04] T. Yi, F. Wu, and C. Gan. A Comparison of Metrics for UML Class Diagrams. *SIGSOFT Software Engineering Notes*, 29(5):1–6, 2004.

[Zho03] Y. Zhou. Measuring Structure Complexity of UML Class Diagrams. *Journal of Electronics (China)*, 20(3):227–231, 2003.

[Zus97] H. Zuse. *A Framework of Software Measurement*. Walter de Gruyter & Co. Hawthorne, NJ, USA, 1997.

Appendix A

Acronyms and Definitions

A.1 Acronyms

Acronym	Definition
CCM	Cognitive Complexity Model
DN	Depth of Navigations
GQM	Goal-Question-Metric
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organisation for Standardisation
LTM	Long-Term Memory
LTWM	long-Term Working Memory
MMLC	Measurement Model Life Cycle
N@P	Number of properties postfixed by @ Pre
NAN	Number of Attributes referred through Navigations
NAS	Number of Attributes belonging to the classifier that Self represents
NCO	Number of Comparison Operators
NEI	Number of Explicit Iterator variables
NES	Number of Explicit Self
NIE	Number of If Expressions
NII	Number of Explicit Iterator variables
NIS	Number of Implicit Self
NOS	Number of Operations belonging to the classifier that Self represents
NKW	Number of OCL KeyWords
NNC	Number of Navigated Classes
NNR	Number of Navigated Relationships

NPT	Number of Parameters whose Types are classes defined in a class diagram
NUDTA	Number of User-Defined Data Type Attributes
NUDTO	Number of User-Defined Data Type Operations
NVL	Number of Variables defined by Let expressions
OMG	Object Management Group
OCL	Object Constraint Language
OO	Object Oriented
SE	Software Engineering
SPSS	Statistical Package for Social Science
STM	Short-Term Memory
TQM	Total Quality Management
UA	Universidad de Alicante
UACh	Universidad Austral de Chile
UCLM	Universidad de Castilla La Mancha
UCM	Universidad Complutense de Madrid
ULM	Universidad de La Matanza
UML	Unified Modeling Language
UNC	University of Comahue
UoD	Universe of Discourse
UPV	Universidad Politecnica de Valencia
WNN	Weighted Number of Navigations
WNCO	Weighted Number of Collection Operations
NON	Number of referred Operations through Navigations
XML	Extensive Markup Language

A.2 Important Definitions

Cognitive Complexity: The mental burden of the individuals (modelers, developers, testers, etc.) who have to deal with software artefacts.

Dynamic Multiplicity: When the multiplicity of the association is not fixed, but should be determined based on another value in the system, this is called dynamic multiplicity [WK03].

Import-Coupling: The extent to which a software part depends on the rest of the software system.

Structural Complexity: The organization of the program elements within a program.

Appendix B

Frameworks for the Theoretical Validation

In this appendix we detail the main concepts of the frameworks used for the theoretical validation:

- Ther Briand et al ´s Frameworks [BMB96], [BMB99], [BBM98] representing the Property-based approaches, explained in section B.1
- The DISTANCE Framework [PD00], [PD99] representing an approach based on the Measurement Theory, explained in section B.2.

B.1 Briand et al.´s Frameworks

Property-based approaches (also called axiomatic approaches) such as the Briand et al.´s frameworks, formally define desirable properties of the measures for a given software attribute. These properties are properties of the numerical relation system of measures. They aim to formalize the empirical properties that a generic attribute of software or a system (e.g. the length or size) must satisfy in order for it to be used in the analysis of any measurement proposed for that attribute. The two best known approaches were proposed by Weyuker [Wey88] and Briand et al.[BMB96].

B.1.1 The Original Framework of Briand et al. [BMB96]

Briand et al. [BMB96] have provided a set of mathematical properties that characterize and formalize several important measurement concepts such as size, length, complexity, cohesion and coupling, related to internal software attributes. After

some criticisms made by Poels and Dedene [PD97], Briand et al. [BMB97] made some modifications to the definition of the properties they had initially proposed. Hereafter we refer to the final framework, i.e. the modified version. This framework is based on a graph-theoretic model of a software artifact, which is seen as a set of elements linked by relationships. The idea is to characterize the properties for measurement of a given software attribute via a set of mathematical properties, based on this graph-theoretic model. The properties they provide are generally enough to be applied not only to code, but also to other artifacts produced during the software process, for example for OCL expression measures.

Systems, Modules and Modular Systems. In this framework a system is characterized by its elements and the relationships between them. The authors want the properties they define to be as independent as possible of any product abstraction. Thus, the framework does not reduce the number of possible system representations, as elements and relationships can be defined according to the needs. A software artifact is modelled by a graph S = < E, R >, called system, where E is the set of elements of S, and R is a binary relation among the elements of E (R $\subseteq$ E $\times$ E). From this point, we say that m is a module of S if and only if $E_m \subseteq E$, $R_m \subseteq E_m \times E_m$ and $R_m \subseteq R$.

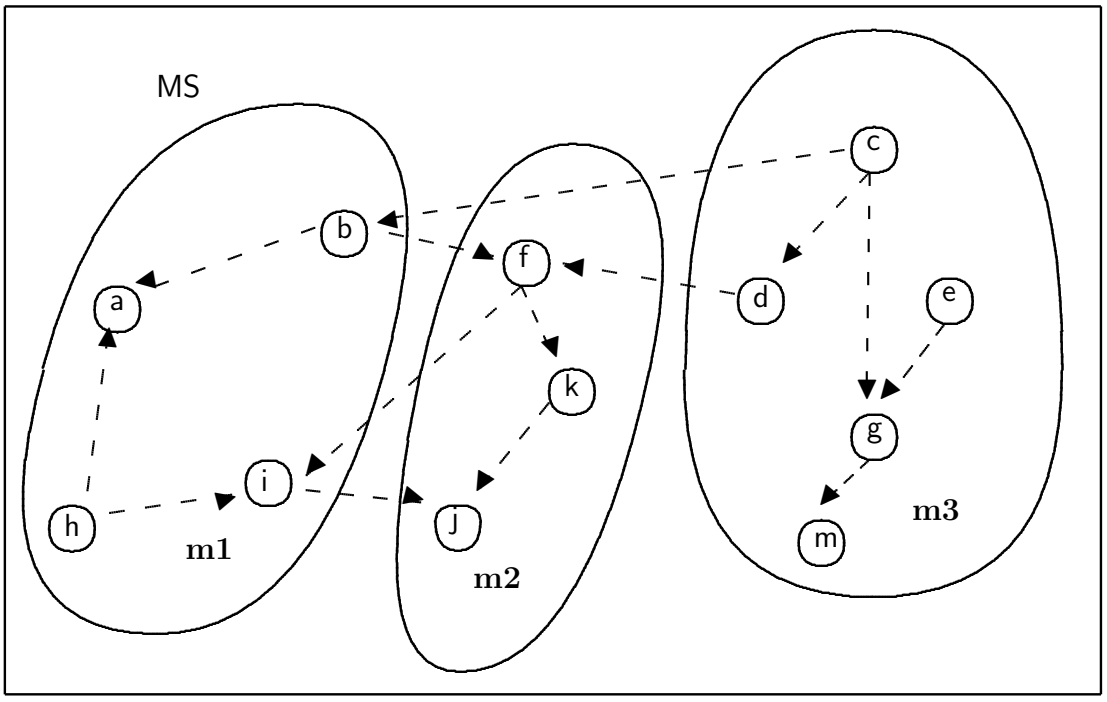


Figure B.1: A Modular System

A module is connected to the rest of the system by external relationships, whose set is defined as

OuterR(m) = { <e_1,e_2 > | ($e_1 \in E_m \wedge e_2 \notin E_m$) $\vee$ ($e_1 \notin E_m \wedge e_2 \in E_m$)}. A modular system is one where all the elements of the system have been partitioned into different modules. Therefore the modules of a modular system do not share elements, but there may be relationships across modules.

Example B.1.1 *Fig. B.1 shows a modular system, called* MS*, with three modules* $\mathbf{m}_1$, $\mathbf{m}_2$ *and* $\mathbf{m}_3$*. Also we can add that* $\mathbf{m}_1$ *contains four elements* a, b, h *and* i.

We will introduce inclusion, union, intersection operations for modules and the definitions of empty and disjoint modules.

- **Inclusion**. Module m_i = <E_{mi}, R_{mi}> is said to be included in module m_j = <E_{mj}, R_{mj}> (notation: $m_i \subseteq m_j$) if $E_{mi} \subseteq E_{mj}$ y $R_{mi} \subseteq R_{mj}$.
- **Union**. The union of modules m_i = <E_{mi}, R_{mi}> and m_j = <E_{mj}, R_{mj}> (notation: $m_i \cup m_j$) is the module < $E_{mi} \cup E_{mj}$, $R_{mi} \cup R_{mj}$>.
- **Intersection**. The intersection of modules m_i = <E_{mi}, R_{mi}> and m_j = <E_{mj}, R_{mj}> (notation: $m_i \cap m_j$) is the module < $E_{mi} \cap E_{mj}$, $R_{mi} \cap R_{mj}$>.
- **Empty module**. Module <$\emptyset$,$\emptyset$> (denoted by $\emptyset$) is the empty module.
- **Disjoint modules**. Modules m_i and m_j are said to be disjoint if $m_i \cap m_j = \emptyset$.

Since in this framework modules are just subsystems, all systems can theoretically be decomposed into modules. The definition of a module for a particular measure in a specific context is just a matter of convenience.

B.1.1.1 Properties for Size and Length

We will describe in this section only the properties for the internal attributes size and length due to the fact these two properties were applied in the theoretical validation. Both properties may be defined for entire systems or modules of entire systems.

- **Size**. The basic idea is that size depends on the elements of the system. The size of a system S = <E, R> is a function Size(S) that is characterized by the following properties:
  1. **Property 1**. Nonnegativity. Size(S) $\geq$ 0
  2. **Property 2**. Null value. The size of a system S is null if E is empty: E = $\emptyset$ $\Rightarrow$ Size(S) = 0

3. **Property 3**. Module additivity. The size of a system S is equal to the sum of the sizes of two of its modules $m_1 = <E_{m1}, R_{m1}>$ and $m_2 = <E_{m2}, R_{m2}>$ such that any element of S is an element of either m_1 or m_2:

   ($m_1 \subseteq S$ and $m_2 \subseteq S$ and $E = E_{m1} \cup E_{m2}$ and $E_{m1} \cap E_{m2} = \emptyset$) $\Rightarrow$
   $Size(S) = Size(m_1) + Size(m_2)$

- **Length**. The length of a system $S = <E, R>$ is a function Length(S) characterized by the following properties:

  1. **Property 1**. Nonnegativity. $Length(S) \geq 0$
  2. **Property 2**. Null value. The length of a system S is null if E is empty: $E = \emptyset \Rightarrow Length(S) = 0$
  3. **Property 3**. Nonincreasing monotonicity for connected components. Let S be a system and m be a module of S such that m is represented by a connected component of the graph representing S. Adding relationships between elements of m does not increase the length of S:
     ($S=<E, R>$ and $m = <E_m, R_m>$ y $m \subseteq S$ y m "is a connected component of S" and $S' = <E, R'>$ and $R' = R \cup \{<e_1,e_2>\}$ and $<e_1,e_2> \notin R$ and $e_1 \in E_{m1}$, and $e_2 \in E_{m1}$) $\Rightarrow Length(S) \geq Length(S')$
  4. **Property 4**. Nondecreasing monotonicity for non-connected components. Let S be a system and m_1 and m_2 be two modules of S such that m_1 and m_2 are represented by two separate connected components of the graph representing S. Adding relationships from elements of m_1 to elements of m_2 does not decrease the length of S.
     ($S=<E, R>$ and $m_1 = < E_{m1}, R_{m1}>$ y $m_2 = < E_{m2}, R_{m2}>$
     and $m_1 \subseteq S$ and $m_2 \subseteq S$ "are separate connected components of S" and
     $S'=<E, R'>$ and $R'= R \cup \{<e_1,e_2>\}$ and $<e_1,e_2> \notin R$
     and $e_1 \in E_{m1}$, and $e_2 \in E_{m2}$) $\Rightarrow$
     $Length(S') \geq Length(S)$
  5. **Property 5**. Disjoint modules. The length of a system S made of two disjoint modules m_1, m_2 is equal to the maximum of the lengths of m_1 and m_2.
     ($S = m_1 \cup m_2$ and $m_1 \cap m_2 = \emptyset$ and $E = E_{m1} \cup E_{m2}$) $\Rightarrow$
     $Length(S) = \max\{Length(m_1), Length(m_2)\}$

B.1.2 An Adaptation of Briand et al.´s Framework

In [BMB99], [BBM98] a precise study of how measures for a high level design is defined and validated. Although the definition of the measures are applied to a

specific context, a high-level designs written in Ada83, the defined concepts *can be applied to other object-based design methods and formalisms* [BMB99]. A careful explanation of this study is described in this subsection, providing us an overview of how interaction-based measures are defined, and how the initial framework of Briand [BMB96] is instantiated for this specific context. A clear comprehension of this adaptation framework will allows us to take into account all the considerations to validate in our specific context a set of interaction-based measures.

B.1.2.1 Basic Definitions

Briand et al. define in [BMB99] some basic terminology prior to the definition of high-level design measures for object-based software systems. They define *interactions* and *data dependency links* on which the cohesion and coupling measures are based. This subsection introduce these concepts.

Modules and High Level Design

The elementary components in the study of [BMB99] are software modules. The authors mentioned that in the literature, there are two commonly accepted definitions of modules. The first one sees a module as a subroutine and has been used in most of the design publications. The second definition, which takes an OO perspective, sees a module as a collection of type, data and subroutine definitions, i.e. a provider of computational services. In this view, a module is the implementation of an Abstract Data Type (ADT), for example a class in C++. In [BMB99] the subroutine term is used for the first category whereas the term module is reserved for the second category. Modules are composed of two parts: interface and body (which may be empty). The interface contains the computational resources that the module makes visible for use to other modules. The body contains the implementation details that are not to be exported. Modules and subroutines may be related to each other by IS_COMPONENT_OF and USES relationships.

In general modules/subroutines A is related to module/subroutine B by an IS_COMPONENT_OF relationship if A is defined within B. Modules/subroutines A is related to module/subroutine B by an USES relationship if A uses computational services that B makes available.

Modules and subroutines can be seen as the components of higher level aggregations, as defined below.

Definition B.1.2 *Library Module Hierarchy (LMH). A library module hierarchy is a hierarchy where nodes are modules and subroutines, arcs between nodes are*

IS_ COMPONENT_ OF relationships, and there is exactly one top level node, which is a module.

The term software part (sp) denotes either a module or a LMH. In the high-level design phase of their context (Ada) [BMB99], only module and subroutine interfaces and their relationships are defined. Therefore, the definition of a high-level design of a software systems for the context is as follow.

Definition B.1.3 *The high-level design of a software system is a collection of module and subroutine interfaces related to each other by means of USES and IS_ COMPONENT_ OF relationships. Precise and formalized information on module or subroutine bodies is not yet available at this stage.*

Interactions

Special focus is specifically paid in [BMB99] on the dependencies that can propagate inconsistencies from data declarations to data declarations or subroutines when a new software part is integrated in a system. Those relationships are called *interactions* and will be used to define measures capturing cohesion and coupling within and between software parts, respectively.

Definition B.1.4 *Data declaration-Data declaration (DD) Interaction. A data declaration A DD-interacts with another data declaration B if a change in A´s declaration or use may cause the need for a change in B´s declaration or use.*

The DD-interaction relationship is transitive. If A DD-interacts with B, and B DD-interacts with C, then a change in A may cause a change in C, i.e., A DD-interacts with C. Data declarations can DD-interact with each other regardless of their location in the designed system. Therefore, the DD-interaction relationship can link data declarations belonging to the same software part or to different software parts. The DD-interaction relationships can be defined in terms of the basic relationships between data declarations allowed by the language, which represent direct DD-interactions (i.e., not obtained by virtue of the transitivity of interaction relationships). Data declaration A directly DD-interacts with data declaration B if A is used in B´s declaration or in a statement where B is assigned a value. As a consequence, as bodies are not available at high-level design time, we will only consider interactions detectable from the interfaces. DD-interactions provide a means to represent the dependency relationships between individual data declarations. Yet, DD-interactions per se are not able to capture the relationships between individual data declarations and subroutines, which are useful to understand whether data declarations and subroutines are related to each other and therefore should be encapsulated into the same module.

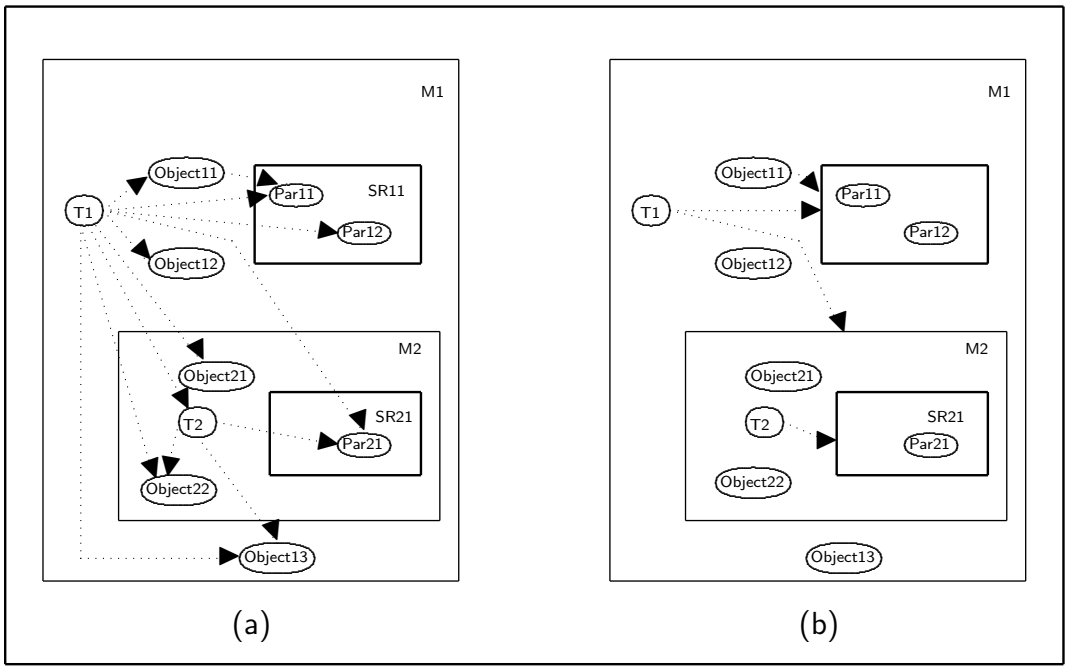


Figure B.2: (a) DD-interaction (b) DS-interaction

Definition B.1.5 *Data declaration-Subroutine (DS) Interaction. A data declaration DS-interacts with a subroutine if it DD-interacts with at least one of its data declarations.*

Whenever a data declaration DD-interacts with at least one of the data declarations contained in a subroutine interface, the DS-interaction relationship between the data declaration and the subroutine can be detected by examining the high-level design.

Example B.1.6 *For instance, from the following Ada-like code fragment:*

```
Package M1 is
    ...
    type T1 is ... ;
    Object11, Object12: T1 := ...
    procedure SR11(Par11: in T1 := Object11, Par12: in T1 );
    ...
    Package M2 is
        ...
        Object21: T1;
        type T2 is array (1 ... 100) of T1;
```

```
        Object22: T2;
        procedure SR21(Par21: in out T2 );
        ...
    end M2;     ...
    Object13: M2.T2;
    ...
    end M1;
```

it is apparent that both type T1 and object Object11 DS-interact with procedure SR11, since they both DD-interact with parameter Par11, procedure SR11´s interface data declaration. Figure B.2 depicts the two kinds of defined interaction for this specific context (Ada).

B.1.2.2 Interaction-Based Measures for Coupling

In the Briand et al.´s framework adaptation for interaction-based measures [BMB99] coupling is defined as *a relation between an individual software part and its associated software system, rather than as a relation between two software parts*. The interactions described in [BMB99] are directly related with the definition of a high level design.
Coupling can be divided into two parts:

1. *import-coupling*, i.e., the extent to which a software part depends on the rest of the software system, and

2. *export-coupling*, i.e., the extent to which the rest of the software system depends on the software part.

[BMB99] is focused only on import-coupling. The experimental hypothesis of the work is defined as follow:
Hypothesis H-IC: The more dependent a software part on external data declarations, the more external information needs to be known in order to make the software part consistent with the rest of the system. In other words, the larger the amount of external data declarations, the more incomplete the local description of the software part interface, the more spread the information necessary to integrate a software part in a system. Thus, the software part becomes more fault-prone.

The properties the authors [BMB99] believe should be satisfied by interaction-based import-coupling measures are defined below. These properties are instantiated for their specific Ada context, of the properties defined in [BMB96] for coupling:

- **Property AdaCoupling1. Nonnegativity.** Given a software part *sp*, the measure import_coupling_measure(*sp*) ≥ 0. Import_coupling_ measure(*sp*) $= 0$ if *sp* does not have import interactions with other software parts.

- **Property AdaCoupling2. Monotonicity.** Let m_1 be a module and II(m_1), its set of import interactions. If m_2 is a modified version of m_1 with the same sets of data and subroutine declarations and one more import interaction so that II(m_2) $\supseteq$ II(m_1), then import_coupling_measure(m_2) $\geq$ import_coupling_measure(m_1)[1].

- **Property AdaCoupling3. Merging of Modules.** The sum of the import-couplings of two modules is no less than the coupling of the module which is composed of the data declarations of the two modules.

An interaction-based coupling measure is defined, the import-coupling measure. Its definition is:
Import-Coupling: Given a software part *sp*, import-coupling of *sp* (denoted by IC(*sp*) is the number of DD-interactions between data declarations external to *sp* and the data declarations within *sp*

Example B.1.7 *Each box in Fig. B.3 represents a module interface. Module interfaces* m2 *and* m3 *are located in their parents interface* m1. m2 *is assumed to be declared before* m3 *and therefore visible to* m3. Tij *and* Objectij *data declarations represent, respectively, types and objects in module* mi. FP3 *represents a subroutine formal parameter. The IC values for the modules in Fig. B.3 are computed as follows:*
IC(m1) = 0, IC(m2) = 4, IC(m3) = 5, IC(m4) = 2.

B.2 DISTANCE Framework

The DISTANCE framework provides constructive procedures to model software attributes and define the corresponding measures [PD99]. The different procedure steps are inserted into a process model for software measurement that: (i) details for each task the required inputs, underlying assumptions and expected results, (ii) prescribes the order of execution, providing for iterative feedback cycles, and (iii) embeds the measurement procedures into a typical goal-oriented measurement approach such as, GQM. In this section we summarise the procedures for attribute definition and measure construction for ease of reference.

[1] Adding import interactions to a module cannot decrease its import-coupling.

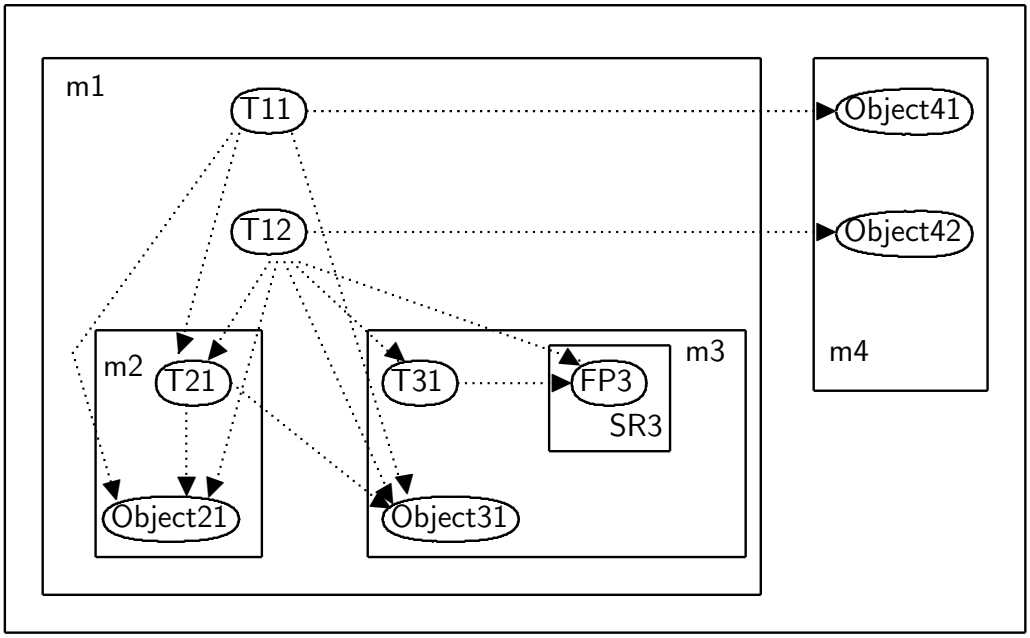


Figure B.3: Calculation of IC with Nongeneric Modules Only

The framework is called DISTANCE as it builds upon the concepts of distance and dissimilarity (i.e. a non-physical or conceptual distance). Software attributes are modelled as conceptual distances between the software entities they characterize and other software entities that serve as reference points or norms for measurement. These distances are then measured by functions that are called "measures" in mathematics. These are functions that satisfy the measure axioms, i.e. a set of axioms that are necessary and sufficient to define measures of distance [PD00].

The measurement theoretic interpretation of the concept of dissimilarity is built into the framework. This ensures that the theoretical validity of the measures obtained with DISTANCE can be formally proven within the framework of measurement theory. A key feature of DISTANCE is that the constructive attribute modelling and measure definition procedures as presented in the process model, hide the complexity of the underlying measurement theoretic constructs from the user. Poels and Dedene [PD99] take full advantage of the intuitiveness and flexibility of the distance concept to arrive at a measure construction framework that is transparent with respect to measurement theory and that is generic, i.e. not limited to the measurement of a specific software attribute.

This framework does not require that the empirical relational system is an extensive structure (leading automatically to ratio scales), nor that a closed binary operation

on the set of software products or software product abstractions is defined.
Another relevant aspect is that the underlying measurement theoretic principles of the DISTANCE framework ensure the construct validity of the resulting measures, i.e. they measures what it is purported to measure. This fact contributes to the validity of the empirical studies we carry out using the measures defined following this framework. We first present a measurement structure from measurement theory that formalizes the idea of "distance measurement" and later we describe each of the steps of the distance-based measure construction process.

B.2.1 Proximity Measurement

In this section we present the measurement theoretic constructs used in the DISTANCE framework.

Proximity Structure

1. Let A be a set. The quaternary relation $\bullet \geq$ on A (i.e., $\bullet \geq \subseteq A^4$) is a binary relation on A × A (i.e., $\bullet \geq \subseteq A^2 \times A^2$) such that ∀ a, b, c, d ∈ A: (a, b) $\bullet \geq$ (c, d) if and only if the dissimilarity between a and b is at least as great as that between c and d.

2. The quaternary relation ≈ on A is an equivalence relation on A × A such that ∀ a, b, c, d ∈ A: (a, b) ≈ (c, d) ⇔ (a, b) $\bullet \geq$ (c, d) and (c, d) $\bullet \geq$ (a, b).

3. The quaternary relation $\bullet >$ on A is a strict ordering relation on A × A such that ∀ a, b, c, d ∈ A: (a, b) $\bullet >$ (c, d) ⇔ (a, b) $\bullet \geq$ (c, d) and not ((c, d) $\bullet \geq$ (a, b)).

4. (A, $\bullet \geq$) is a **proximity structure** if and only if the following axioms hold ∀ a, b ∈ A:

   - P_1. $\bullet \geq$ is a weak order;
   - P_2. (a, b) $\bullet >$ (a, a) whenever a ≠ b (positivity);
   - P_3. (a, a) ≈ (b, b) (minimality);
   - P_4. (a, b) ≈ (b, a) (symmetry).

Part 1 of the proximity structure definition states that dissimilarity (or distance) is an attribute of a pair of elements. Prior to measurement, there exists an order $\bullet \geq$ on the dissimilarities between the pairs of elements of some set A. Parts (2) and (3) derive an equivalence relation ≈ and a strict ordering relation $\bullet >$ from the dissimilarity ordering $\bullet \geq$. Finally, part (4) describes the empirical conditions that

must hold for the pair (A, $\bullet \geq$) to qualify as a proximity structure. Condition P_1 requires the empirical dissimilarity ordering $\bullet \geq$ to be a weak order, i.e., it must be transitive and strongly complete. Conditions P_2, P_3 and P_4 follow from the measure axioms non-negativity, symmetry and identity. The fact that any measure δ satisfies δ(a, b) = δ(b, a) > δ(a, a) = δ (b, b) for all a $\neq$ b requires the empirical conditions of positivity, minimality and symmetry.

The distance-based approach constructs a special kind of proximity structure, i.e., a segmentally additive proximity structure. The representation of such a structure has more favourable properties than the representation of a proximity structure.

Segmentally Additive Proximity Structure

1. Let A be a set. For a, b, c $\in$ A, the ternary relation $\langle$abc$\rangle$ is said to hold if and only if $\forall$ a', b', c' $\in$ A: (a, b) $\bullet \geq$ (a', b') and (a', c') $\bullet \geq$ (a, c) $\Rightarrow$ (b', c') $\bullet \geq$ (b, c)
2. A relational structure (A, $\bullet \geq$) is a **segmentally additive proximity structure** if and only if the following axioms hold for all a, b, c, d $\in$ A:
   - S_1. (A, $\bullet \geq$) is a proximity structure;
   - S_2. If (a, b) $\bullet \geq$ (c, d), then there exists e $\in$ A such that (a, e) $\approx$ (c, d) and $\langle$aeb$\rangle$;
   - S_3. If c $\neq$ d, then there exist e_0',$\cdots$, e_n' $\in$ A such that e_0' = a, e_n' = b and (c, d) $\bullet \geq$ (e_{i-1}', e_i'), for all i = 1,$\cdots$, n.

The ternary relation $\langle$abc$\rangle$ is a collinear betweenness relation. Informally, $\langle$abc$\rangle$ holds if b lies on an additive segment from a to c, i.e., along the segment from a to c distances are additive.
The extra axioms of the segmentally additive proximity structure (i.e., S_2 and S_3) are harder to interpret than those of the proximity structure (i.e., S_1). Axiom S_2 is the segmental solvability condition. It tells us that given two distances (a, b) and (c, d), the former being not less than the latter, we can always find an additive segment from a to e on (a, b) that is exactly equal to the distance between c and d. Axiom S_3 then guarantees that (c, d) cannot be infinitely small compared with (a, b). Informally, S_2 and S_3 were introduced to create a 'unit of distance'. These form the additive segments of any distance between elements of A, and thus allow a quantitative assessment of the relative proportion between distances.

A representation of a segmentally additive proximity structure is called a representation by a measure with additive segments. The following representation and

uniqueness theorem can be found in Suppes et al. [KDST06].

Theorem 1. Representation by a measure with additive segments Let (A, $\bullet \geq$) be a segmentally additive proximity structure. Then there exist a real-valued function δ on A $\times$ A such that, for any a, b, c, d $\in$ A,

- M_1. (A, δ) is a measure space;
- M_2. (a, b) $\bullet \geq$ (c, d) $\Leftrightarrow \delta$(a, b) $\leq \delta$(c, d);
- M_3. $\langle$abc$\rangle \Leftrightarrow \delta$(a, b) + δ(b, c) = δ(a, c);
- M_4. If δ' is another measure on A satisfying the above conditions, then there exist $\beta > 0$ such that $\beta' = \beta\delta$.

The term *measure* is generally understood as referring to a 'measure' of distance. The theorem states the empirical conditions that the attribute of distance must satisfy in order to define a measure as a measure in the sense of measurement theory (i.e., as a homomorphism). These empirical conditions are those associated with the (segmentally additive) proximity structure.

If the measure δ would only satisfy M_1 and M_2, then (A, $\bullet \geq$) must only be a proximity structure (i.e., S1 or P_1 to P_4 must be satisfied). The representation is then essentially an ordinal representation [KDST06].

If in addition M_3 is satisfied, then (A, $\bullet \geq$) must be a segmentally additive proximity structure, implying that distance must also be characterize by S_2 and S_3. The measure d is then called a measure with additive segments. The values returned by d for the additive segments on a distance from a to c must add up to the value that is returned for (a, c) itself. It should be noted that M_3 is similar to, but weaker than the additivity condition of an extensive representation.

Condition M_4 is actually a uniqueness theorem. It tells us that the scale type of a measure with additive segments is ratio (as the class of admissible transformations is that of the similarity transformations). The representation by a measure with additive segments belongs to the class of the unidimensional spatial proximity representations.

B.2.2 A Constructive Measurement Procedure

Here, we detail the distance-based measurement procedure which was briefly presented in section 2.3.2.2. We assume in this subsection that the object of measure-

ment is an object class. Let P be a finite set of object classes. All classes that we wish to measure are included in P.
Five activities are needed to model a software attribute with a segmentally additive proximity structure and to define a software measure using a measure with additive segments:

- **Find a measurement abstraction (MT_1)**

  Measurement abstractions are used in software measurement to emphasize the attribute of interest, while simultaneously de-emphasizing other attributes. For instance, if we wish to measure the size of a class in terms of the numerousness of its variables, then the state vector of the class is a suitable measurement abstraction. However, the state vector is not a suitable abstraction to capture the complexity of the object class methods.
  Whatever attribute of a class needs to be measured, the class must be represented in one way or another. The result of step 1 is therefore a function *abs*: P $\rightarrow$ A that maps object classes into their representations. The range of *abs* is the set A of measurement abstractions of the chosen type. The function *abs* is total, but neither injective, nor surjective.

- **Define Distance between Measurement Abstractions (MT_2)**

  The next step is to model distances between the elements of the set A. To this end we use the construct of an elementary transformation t: A $\rightarrow$ A. For all a $\in$ A, t(a) is the abstraction that results when modifying a in some prescribed way. The modification is assumed to be atomic, i.e. it cannot be subdivided in more elementary changes.

  For a given set A there might exist many elementary transformation functions, each associated with one particular type of atomic change. The set T of elementary transformations on A is required to be constructively complete and inverse constructively complete. This means that:

  - (i) There is an initial abstraction a_i in A;
  - (ii) Each element of A can be built by applying a finite sequence of elementary transformations of T to a_i;
  - (iii) For each elementary transformation t included in T, the inverse of t is also contained in T;

  If conditions (i) to (iii) are satisfied, then any element of A can be transformed into any other element of A. For instance, if object classes are represented by their state vector then the elementary transformations 'add a variable' and 'remove a variable' are sufficient to build a constructively complete and inverse

constructively complete set of elementary transformations on state vectors.

We can now formally define the concept of a distance in A by means of a segmentally additive proximity structure. The set of empirical objects in the relational system is A. We only need to define the dissimilarity relation $\bullet \geq$ on A.

The empirical dissimilarity or conceptual distance ordering $\bullet \geq$ is defined as follows:

Let A be a set and T be a constructively complete and inverse constructively complete set of elementary transformations on A. Then, for a, b, c, d $\in$ A, (a, b) $\bullet \geq$ (c, d) denotes the observation that we need at least as many elementary transformations to transform a into b than to transform c into d.

It can be shown that when the dissimilarity ordering on A is defined as it was defined before, then the axioms of the segmentally additive proximity structure are satisfied. For formal proofs we refer to [PD99].
In our example, A is a set that contains sets of variables as its elements. Some of these sets correspond to state vectors of the classes in P. The initial set ai is the empty set. The shortest sequence(s) of elementary transformations between a pair of elements model(s) the distance between those elements.

- **Quantify Distances between Measurement Abstractions ($\mathbf{MT}_3$)**

  In this step a measure is proposed for the distances in A. The definition of a measure function is relatively straightforward: Let A be a set and T be a constructively complete and inverse constructively complete set of elementary transformations on A. Then, for all a, b $\in$ A:

  δ(a, b) = k_{ab} . c, where

  - k_{ab} = the minimal number of elementary transformations needed to transform a into b;
  - c = any positive real number.

  Hence, the distance between a pair of elements in A is measured by counting the elementary transformations in the shortest sequence(s). This count may be multiplied by any positive real number.

In [PD99] it is formally proven that a function defined by previous definition is a measure with additive segments if (A, $\bullet \geq$) is a segmentally additive proximity structure. The function δ satisfies the measure axioms (i.e. M_1), as well as the representation conditions M_2 and M_3. Multiplying the count of elementary transformations by a positive constant corresponds to an admissible transformation of scale as defined by the uniqueness theorem M_4.

After executing step 3 all distances in A can be modelled and measured. However, we do not need to know all these distances. Only a subset of them is related to the software attribute of interest. In the next steps we determine the distances that are models of the attribute of interest and show how to measure them.

- **Find a Reference Abstractions (MT_4)**

  This is a crucial step in the distance-based measurement procedure as it reflects our understanding of the attribute being measured. We need to carry out a kind of thought experiment.

  Let the object classes in P be characterized by an attribute we wish to measure. The function *abs* is used to map an object class p of P into its representation *abs*(p) of A. We must now imagine what *abs*(p) would look like if p is characterized by the theoretically lowest value of the attribute. This imaginary representation of p is called the reference abstraction of p for the attribute of interest. It can then be argued that the closer *abs*(p) is to this reference abstraction, the lower the value of the attribute. Analogously, the farther *abs*(p) from the reference abstraction, the higher the value of the attribute. Hence, the conceptual distance or dissimilarity between the measurement abstraction and the reference abstraction is used as a model of the attribute of interest. For each class in P it is this particular distance that needs to be measured.
  Formally, the reference abstraction is returned by a function ref: P $\rightarrow$ A. As ref is a deterministic function there is exactly one reference abstraction associated with a class p for the attribute of interest. It is however common (but not required) that ref maps all elements of P into a same abstraction, which is often the initial abstraction ai of A.

  A (common) reference abstraction for class size in terms of the numerousness of the variables is the empty state vector. We can imagine an (artificial) class having no variables. Such a class has the lowest value for the size aspect considered here. Let a class p have the state vector *abs*(p) = a, b, where a and b are variables. The distance between a, b and $\emptyset$ determines the size of this class. Obviously, there are two shortest sequences of elementary transformations to

transform a, b into $\emptyset$.

- **Define a Measure for the Property (MT_5)**

  The fifth and final step is trivial. The desired measurement value is the value returned by the measure d for the pair of abstractions in A consisting of the measurement abstraction of the class and the reference abstraction for that class.

  Then a software measure is defined as follows:
  Let P be a finite set of object classes and let *abs*: P $\rightarrow$ A and ref: P $\rightarrow$ A be functions that return for the classes in P, respectively, the measurement abstraction and the reference abstraction for the attribute of interest. Then, the measure for that attribute is a function μ: P $\rightarrow \Re$ defined such that for all p $\in$ P, μ(p) = δ(*abs*(p), *ref*(p)).

  If μ is a measure of class size (in terms of the numerousness of variables), then m returns for the class p with state vector a, b the count of elementary transformations in the shortest sequence(s) from a, b to $\emptyset$, multiplied by a positive constant. For the constant c =1 we get m(p) = δ(*abs*(p), ref(p)) = δ(a, b, $\emptyset$) = 2 . 1 = 2.

  Table B.1 summarizes the required inputs and expected results of the five steps explained above.

Step	**Inputs**	**Outputs**
Find a measurement abstraction	The attribute of interest *attr*. A set of software entities P	A set of software entities M (to be used as measurement abstractions). A function *abs*: P $\rightarrow$ M
Model distances between measurement abstractions	M	A set of elementary transformation types T_e
Quantify distances between measurement abstractions	M, T_e	A measure δ: M $\times$ M $\rightarrow \Re$
Find a reference abstraction	*Attr*, P, M	A function *ref*: P $\rightarrow$ M (to return a reference abstraction for attr)
Define the software measure	P, *abs*, δ, *ref*	A function μ: P $\rightarrow \Re$

Table B.1: Required Inputs and Expected Results

From a measurement theory point of view, the distance-based software measure construction process results in the definition of an attribute as a segmentally additive proximity structure and in the definition of a measure as a measure

with additive segments [KDST06]. According to the uniqueness theorem associated with the representation theorem for segmentally additive proximity structures the resulting measures are characterized by the ratio scale type.

B.2.3 Template for Measure Definition

For the definition of measures Poels and Dedene [PD99] proposed a template shown in Table B.2. In the top row the original description of the measure is entered. In the second row the name of the attribute of interest attr is entered.

Measure name		(description of the measure)	
Software attribute (*attr*)	(attribute of interest)	**Software entity** (p ∈ P)	(entity characterized by *attr*)
Output of distance-based process		**Underlying measurement theoretic constructs and formal definitions**	
M	(set of measurement abstractions) (abstraction function)		
T_e	(set of elementary transformation types)	• ≥ SAPS	(empirical dissimilarity ordering)) (segmentally additive proximity structure)
δ	(measure function)	MSAS	(measure space with additive segments)
ref	(reference function)	$ST_{abs(p),ref(p)}$ Attribute definition	(shortest sequences of elementary transformations) (formal distance-based measure definition)
μ	(Measure function)	Measure definition	(formal measure-based measure definition)
Remarks:			

Table B.2: Template for Distance-based Definition of Measures

This name describes the conceptual idea that reflects to some extent the empirical understanding of the attribute. However, it is only used as an identifier. The precise meaning of the attribute is contained within its distance-based definition. Finally in the second row, the software entity that is characterized by the attribute of interest is entered.

Next, the output of the distance-based measure construction process is described in the left column of the template. The right column shows the measurement theoretic constructs that underlie the different steps in the distance-based process. In fact, the right column presents the formal, distance-based, definition of the attribute and the formal, measure-based, definition of the corresponding measure. A last row in

the template allows entering additional remarks regarding the choices made when applying the distance-based measure construction process.

Appendix C

Experimental Process

Experimentation is a labor-intensive task. So, in order to properly set up and conduct an experiment, we need a process supporting us in our objectives in doing experiments correctly [WRH+00]. This appendix has three sections: section C.1 explains how the experimental process (briefly described in section 2.3.4) can be depicted as a V model whereas section C.2 describes each of the steps of the experimental process. Section C.3 provides a short introduction to the replication of Experiments.

C.1 A V Model of the Experimental Process

The experimental process is iterative, that means that the activities of Figure 2.13 can not be applied in a secuential order. However there is only one exception, once the operation phase has started it is not possible to go back to previous phases. This phase is in the middle of the experimental process and constitutes a suitable vertex in showing the experimental process using a V model. Meanwhile the prior activities of the Operation phase are mainly concerned with the experiment foundation and its design (cause and effect construct relationship), in the posterior activities of the Operation phase the effort is concerned with the analysis of the effect construct.

The V model is shown in Figure C.1 and uses two different dimension. The horizontal dimension (or edge) shows the experimental process time. At the beginning the researchers have an *experiment construct* and all their ideas are hypothetical, quite subjective. After the experiment has run, their ideas become more objective. The vertical edge shows the relationship between theory and observation, between the abstract world and the concrete one, between analysis and design phases of the experiments. It is clear that some steps of the experimental process are related to 'what is conducted by the experiment?, such as experiment definition and expriment

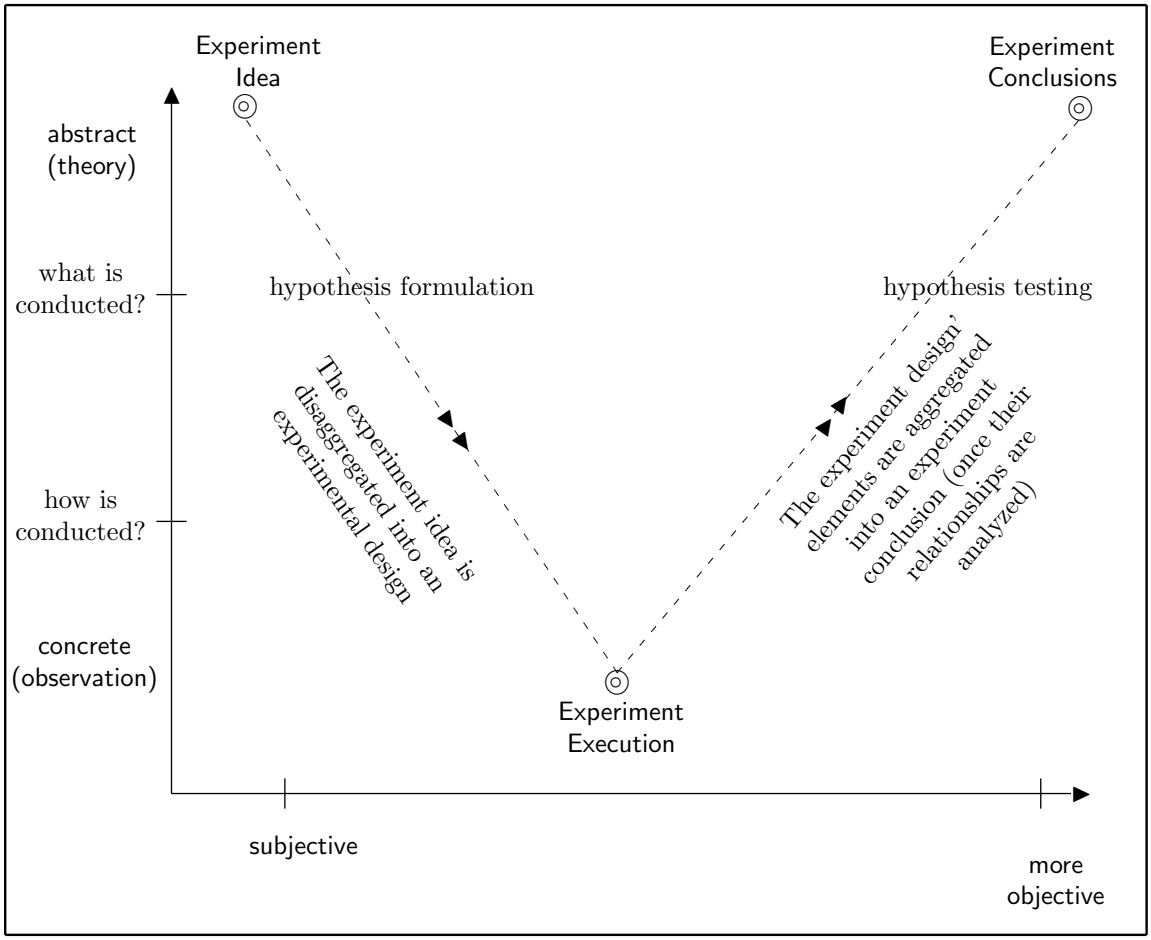


Figure C.1: A Representation of the Experimental Process in a V model

conclusion, also hypothesis formulation and hypothesis testing; meanwhile others, are related to 'how the experiment is conducted?', such as the experimental design and its statistical analysis.

From the *experimental idea* to the *experiment execution* ended points of the V model, the definition, planning and half of the operation phases took place. Similarly, from the *experiment execution* to the *experiment conclusions* the remaining steps of the operations phases, the analysis and presentation phases are carried out.

During the prior activities of the Experiment Execution an Experimental Idea is disaggregated (or decomposed) into a experiment design elements, and its instrumentations (that is the configuration of the experiment elements which will be used during the experiment execution such as objects, subjects, treatments, environment,

etc.) should be clearly defined. The experimental design should successfully measure the theoretical constructs defined in the hypothesis.

After the execution of the experiment there is a tendency to aggregate all the experimental components in terms of a valid conclusion. Figure C.1, graphically show how these two process of desegregation and aggregation are done before and after the experiment execution respectively.

Besides, within any experimental process there are pairs of activities that seem to be symetrically tested (not all are shown in Figure C.1), such as:

- The idea construct and the effect constructs in order to obtain valid conclusions.
- The experiment definition and the experiment conclusion.
- The selection of a set of hypothesis and the hypothesis testing.
- The experimental design and the statistical analysis are closely related [WRH+00].

C.2 Steps of the Experimental Process

As we explain in section 2.3.4 the experimental process can be divided into five main steps, which are shown in Figure C.2 according to Wohlin et al. [WRH+00]. In this section each of the steps conforming the experimental process are explained in details.

C.2.1 Definition

In this phase the foundation of the experiment is specified, determining *why* the experiment is conducted. The purpose of the definition phase is to define the goals of an experiment. In order to capture the goals of the experiment a GQM template for goal definition has been suggested [BDR97], [LR96]. The goal template is shown in table C.1 where each of the elements of the template are described. The template can be filled out with different instances (object of study, purposes, etc.). These instances are shown in angular brackets.

C.2.2 Planning

After the definition of the experiment, the planning takes place. The definition determines *why* the experiment is conducted, whilst the planning prepares for *how*

Figure C.2: Overview of the Experimental Process

Analyse	<Object(s) of study> The **Object of the study** is the entity that is studied in the experiment, which can be products, processes, resources, models, measures or theories.
for the purpose of	<Purpose> The **Purpose** defines what the intention of the experiment is.
with respect to their	<Quality focus> The **Quality focus** is the primary effect under study in the experiment.
from the point of view of the	<Perspective> The **Perspective** tells the viewpoint from which the experiment results are interpreted normally defined in terms of a role, for example, project manager, developer, etc.
in the context of	<Context> The **Context** is the environment in which the experiment is run. The context briefly defines which subjects are involved in the experiment and which software artifacts (objects) are used in the experiment.

Table C.1: Template for Experiment Definition

the experiment is conducted. The planning phase can be divided into six steps, context selection, hypothesis formulation, variables selection, selection of subjects, experiment design and instrumentation.

1. **Context Selection:** The context of the experiment select the environment in which the experiment will be executed [WRH+00], it can be characterised according to four dimensions:

   - **Off-line vs. On-line:** An experiment can be carried out in an off-line situation, for example in a laboratory under controlled conditions, where the events are organized to simulate their appearance in the real world. Experiments may alternatively be carried out on-line, which means that the investigation is executed in the field under normal conditions [Bab90]. The level of control is more difficult in an on-line situation, because it may be possible to control some factors while others may be impossible.
   - **Student vs. Professional:** Experiments should be executed with professional staff or students.
   - **Toy vs. Real problems:** The experiment can address a real problem or a toy situation.
   - **Specific vs. General:** The results of the experiment can be valid in a specific context or valid in the general software engineering domain.

2. **Hypothesis formulation:** The goal for your research can be expressed as a hypothesis you want to test [FP98], in this way the experiment definition

is formalised into hypothesis [WRH+00]. The hypothesis is the tentative theory or supposition that you think explains the behaviour you want to explore [FP98]. Usually the hypothesis that the experimenter wants to reject are specified in a null hypothesis (often denoted as H_0) and an alternative hypothesis (H_1) stated in favor the null hypothesis is rejected. If many alternative hypothesis are specified the rejection of H_0 implies that more experimentation is needed to determine which alternative hypothesis is the best explanation of the observed behaviour [FP98]. Testing the hypothesis involves different types of risks. Accepting a null hypothesis when actually is false is called a *Type II error*. Conversely, incorrectly rejecting the null hypothesis is a *Type I error* [FP98].

The size of the errors depends on different factors. One example is the ability of the statistical test to reveal a true pattern in the collected data. This is referred to as a power of a test [WRH+00]. The power of a statistical test is the probability that the test will reveal a true pattern if H_0 is false. An experimenter should choose a test with as high a power as possible. The power can be expressed as: Power = P(reject H_0 | H_0 false) = 1 - P(type-II-error). All these factors have to be considered when planning an experiment.

3. **Variables selection:** Before any design can start we have to choose the dependent and the independent variables.

   - **Dependent Variable:** The outcome of an experiment is referred to as a dependent variable, also called *response variable* [JM01]. Each dependent variable gathered in an experiment is termed observation, and the analysis of all the observations will decide wheter or not the hypothesis to be tested can be validated [JM01].
   - **Independent Variable:** Those variables that we can control and change in the experiment are independent variables [WRH+00]. They are also called *factors* or *predictor variables*. Juristo et al. [JM01] defines independent variables as each characteristic to be studied that affects the response variable. A particular values of an independent variable is called an *alternative* or *treatment*. Experimentation aims to examine the influence of these alternatives on the value of the response variable [JM01]

   The choice of independent and dependent variables also means that *measures scales* and range for the variables are determined.

4. **Selection of subjects:** The person who applies the methods or techniques to the experimental units is called *experimental subject* [JM01]. The selection of experimental subjects has very important effect on the results of the experiments in SE and its selection should be carefully considered. The selection of subjects is also called a *sample* from population [WRH+00].

		# objects	
		one	**more than one**
# subjects per objects	**one**	single object study	multi-object variation study
	more than one	multi-test within object study	blocked subject-object study

Table C.2: Experiment Context Characterization [WRH+00]

The sample will be closely connected to the generalisation of the results from the experiment. In order to generalise the results to the desired population, the selection must be representative.

Two important factors that have implicances in the generalization are:

- **The way subjects are obtained from the population**: The sampling of the population can be either a probability or non-probability sample. In the former the probability of selection a subject is known, and in the latter is unknown. The most convincing way of obtained subjects is by random sampling [KAKB+06]. If a random sample has been obtained, the method of selection should be specified.
- **The size of the sample** also impacts the results when generalising. The larger the sample is, the lower the error becomes when generalising the results. The sample size is also closely related to the power of the statistical test.

5. **Experiment design:** The design is a crucial step, a poor design may ruin any well-intended study [WRH+00]. To get the most out of the experiment, we must carefully plan and design the experiment. An experiment consists of a series of tests, where each test is a combination of treatment, subject and object. The design of an experiment describes how the tests are organised and run. More formally, we can define an experiment as a set of tests.

   The design and the statistical analysis are closely related. To design the experiment, we have to look at the hypothesis to see which statistical analysis we have to perform to reject the null hypothesis. Based on statistical assumptions, e.g. the measurement scales, and on which objects and subjects we are able to use, we make the experiment design. During the design we determine how many tests the experiment shall have to make sure that the effect of the treatment is visible. A proper design also forms the basis from which to allow for replication [WRH+00].

   The experiment context can be characterized in terms of the number of subjects and objects involved in the study, see table C.2. When a studies are

conducted on a single subject and a single object, we have *single object* studies. *Multi-object variation* studies are conducted on a single subject across a set of object. *Multi-test within object* studies examines a single object a set of subjects. *Blocked subject-object* studies examine a set of subjects and a set of objects.

There are different experimental designs depending on the aim of the experiment, the number of factors, the alternatives of the factors, and the number of undesired variations, etc. Table C.3 (from [JM01]) gives a summary of the most commonly used experimental design. The description of each kind of design is out of the scope of this research work, but is possible to find their complete description in [JM01] and [WRH+00]

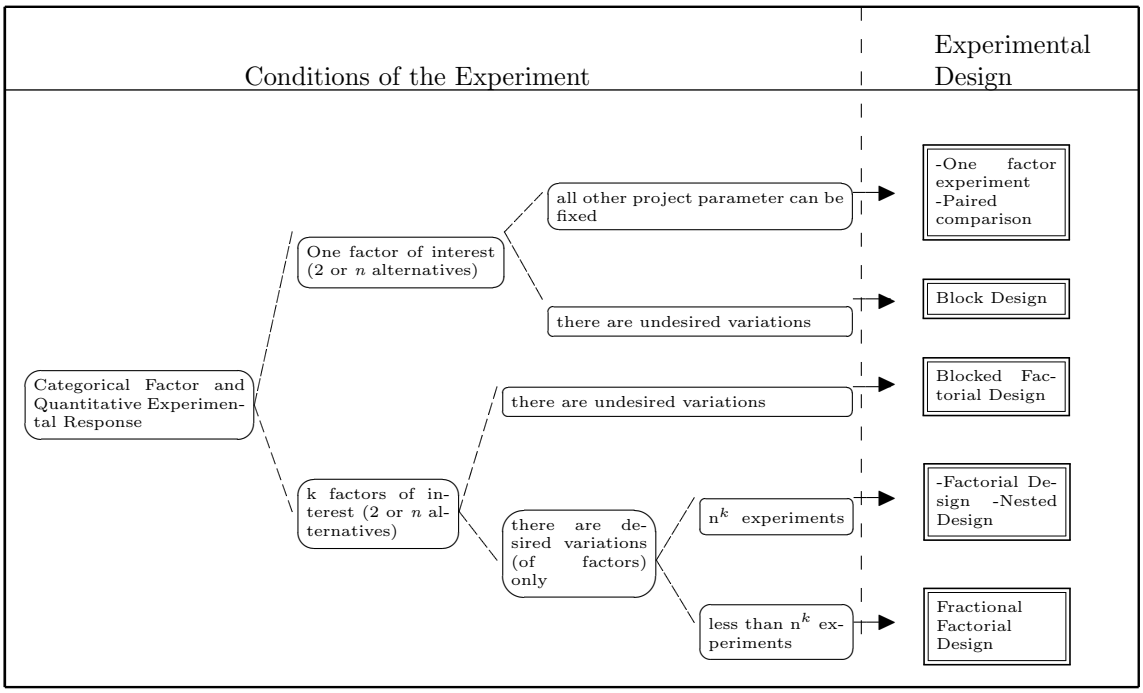


Figure C.3: Different Experimental Designs

6. **Instrumentation:** The instruments for an experiment are of three types, namely *objects*, *guidelines* and *measurement instruments*. In the planning of an experiment, these instruments are chosen and are developed for the specific experiment before its execution.

   - **Experiment objects** may be, for example, specification or code documents. When planning for an experiment, it is important to choose objects that are appropriate.
   - **Guidelines** are needed to guide the participants in the experiment. Guidelines include, for example, process descriptions and checklists.

- **Measurements** in an experiment are conducted via data collection. In human intensive experiments, data is generally collected via manual forms or by interviews.

The overall goal of the instrumentation is to provide means for performing the experiment and to monitor it, without affecting the control of the experiment. The results of the experiment shall be the same independently of how the experiment is instrumented. If the instrumentation affects the outcome of the experiment, the results are invalid.

C.2.2.1 Validity evaluation:

A fundamental question concerning results from an experiment is how valid the results are. It is important to consider the question of validity early on in the planning phase in order to plan for adequate validity of the experiment results [WRH+00]. In an experiment we want to draw conclusions about the theory defined in the hypothesis, based in our observations. In drawing conclusions we have four steps, in each of which there is one type of threat to the validity of the results, see Figure C.4.

Campbell and Stanley [CS63] have defined two types of threats:

- **Internal validity:** To the degree that we are successful in eliminating confounding variables within the study itself is referred to as internal validity [Hay05]. We must make sure that exists a causal relationship between treatment and outcome, and that it is not a result of a factor of which we have no control or have not measured [WRH+00]. See item 2 of Figure C.4. For example, factors that have an impact on the internal validity are: how the subjects are selected and divided into different classes, how the subjects are treated and compensated during the experiment, if special events occur during the experiment, the material used in the experiment, etc.

- **External validity:** The result of the experiment has to be valid not only for the population it is drawn for, but also a bigger population [BP02] -see item 4 of Figure C.4. External validity is the degree to which the results of the research can be generalised to the population under study and other research settings. The greater the external validity, the more the results of an empirical study can be generalised to actual software engineering practice. Threats to the external validity concern the ability to generalise experiments results outside the experiment setting. External validity is not only affected by the experiment design chosen, but also by the objects in the experiments and the subject chosen. There are three main risks: not having the right participants

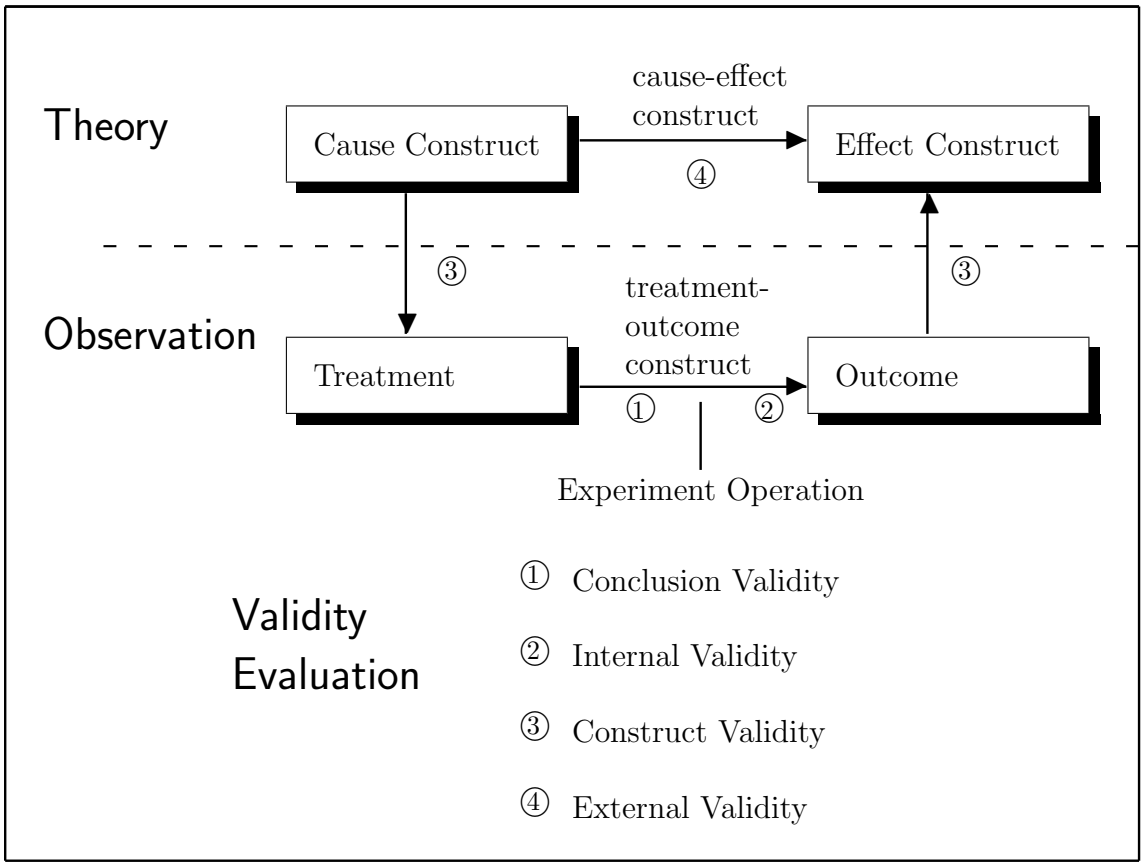


Figure C.4: Experimental Principles

as subjects, conducting the experiment in the wrong environment and performing it with a timing that affects the results.

External and internal validity are not all-or-none, black-and-white, present-or-absent dimensions of an experimental design. Validity varies along a continuum from low to high [Hay05].

Cook and Campbell [CC79] extend the list to four types of threats to the validity of experimental results.

- **Construct validity:** This validity is concerned with the relationship between theory and observation. It defines the extent to which the variables successfully measure the theoretical constructs in the hypothesis. Threats to the construct validity is the lack of theoretical proof whether the measures for the independent and the dependent variables are really measuring the concepts they purport to measure. We must ensure two things: first, that the treatment reflects the construct of the cause well, and second, that the outcome reflects the construct of the effect well [WRH+00], see the two items labeled as 3 in

Figure C.4

- **Conclusion validity:** This validity is concerned with the relationship between the treatment and the outcome -see item 1 of Figure C.4 -. We want to make sure that there is a statistical relationship, ie. that our conclusions are statistically valid.

Sometimes an increase in focus on one threat could decrease focus on one of the other threats. Wohlin suggests to prioritize between threats according to the following order of importance: Internal, external, construct and conclusion validity.

C.2.3 Operation

When an experiment has been designed and planned it must be carried out in order to collect the data that should be analysed. This is what we mean by operation of an experiment. In the operational phase of an experiment, the treatments are applied to the subjects. [WRH+00] This means that this part of the experiment is the part where the experimenter actually meets the subjects. The operational phase consists of three steps: preparation, execution and data validation.

1. **Preparation:** Before an experiment can be started, people who are willing to act as subjects have to be found. It is essential that people are motivated and willing to participate throughout the whole experiment. A well-motivated subject may perform better in an experiment than a poorly motivated subject, as discussed in [HWT05], [BSL99]. The following aspects could be considered when people are participating as subjects:

    - **Obtain consent:** The participants have to agree to the research objectives. It is important to describe how the result of the experiment will be used and published.
    - **Sensitive results:** If the results obtained in the experiment are sensitive [1] for the participants, it is important to assure the participants that the results of their personal performance in the experiment will kept confidential.
    - **Inducements:** One way to attract people to an experiment is to offer some kind of inducement. The APA [APA84] provides severeral recommendations regarding student participants [APA84].

[1] it is quite subjective to judge if a result is sensitive or not, we can generally said that if the result would have a meaning for the participants outside the experiment it is in some way sensitive

- **Deception:** Sometimes the only alternative of conducting the experiment is to deceive or betray the participants (that is called deception). This is generally not favoured. If alternative ways of conducting the experiment are available these methods should be used instead. Otherwise, if deception is the only alternative, it should only be applied if it concerns aspects that are insignificant to the participants and do not affect their willingness to participate in the experiment.

Before the experiment can be executed, all experiment instruments must be ready. This may include the experiment objects, guidelines for the experiment and measurement forms and tools. The required instruments are determined by the design of the experiment and the method that will be used for data collection.

If the subjects themselves should collect the data, this means in most cases that some kind of forms must be handed out to the participants. One thing to determine when forms are constructed is whether they should be personal or the participants should fill them in anonymously.

2. **Execution:** The experiment can be executed in a number of different ways. Some experiments can be carried out at one occasion when all participants are gathered together at, for example, a meeting. The advantage of this is that the results of the data collection can be obtained directly at the meeting and there is no need to contact the participants and ask for their respective results later on. Another advantage is that the experimenter is present during the meeting and if some questions arise they can be resolved directly.

   Some experiments are, however, executed during a much longer time span, and it is impossible for the experimenter to participate in every detail of the experiment and the data collection. Data can be collected either manually by the participants that fill out forms or automatically by tools. The first alternative is probably the most common and the last is probably the least.

3. **Data validation:** When data has been collected, the experimenter must check that the data is reasonable and that it has been collected correctly. This deals with aspects such as whether the participants have understood the forms and therefore filled them out correctly. It is also important to review that the experiment has actually been conducted in the way that was intended.

C.2.4 Analysis and Interpretation

After we have collected the relevant data, we must analyse it in an appropriate way. There are three major items to consider when choosing the analysis techniques:

the nature of the data we have collected, why we performed the experiments, and the type of experimental design. Depending on the purpose of the experiment, different analysis techniques can be used to test the hypothesis. The objective of the hypothesis testing is to see if it is possible to reject a certain null hypothesis, H_0, based on a sample from some statistical distribution.

Pfleeger carried out a detailed study on the different statistical tests applied according to objective sought. In [FP98], Pfleeger presents a *decision tree*, which can be used to choose the best technique. Figure C.5 shows the decision tree only regarding the purpose 'Explore a relationship or Validating software measures', because this is the objective we pursue in all of the experiments we carry out in this PhD Thesis. The decision tree must to be read from top to bottom. Beginning with the objective of our experiment, we move down, along the branch that fits our situation until we reach a leaf node with the appropriate analysis technique.

In this research work we use the Statistical Package For Social Science (SPSS) to automatically apply different statistical tests for analysing the data collected in the empirical studies.

C.2.5 Presentation and Package

When an experiment has been carried out, the intention is often to present the findings. This could for example be done in a paper for a conference, a report for decision-making, a lab package for replication of the experiment or as educational material. It is essential not to forget important aspects or necessary information needed in order to enable others to replicate or take advantage of the experiment and the knowledge gained through the experiment.

C.3 Replication of Experiments

A fundamental strategy for enabling the replication of experiments is to create laboratory packages which contain all the information of an experiment [SCT+03], such as the experimental design, the artifacts, and the processes used. These laboratory packages simplify the experiment replica [BSL99]. There are various types of replicas:

- Replicas which do not vary the hypothesis: They do not vary the dependent variables of the original experiment nor the independent ones. The strict ones duplicate the original experiment and are necessary for increasing the reliability of the conclusions about the validity of the experiment. They are used to demonstrate if the results of the original experiment are repeatable. The

Figure C.5: Decision Tree for Analysis Techniques

replicas that modify the way in which the experiment is made try to increase our confidence in the experimental results by studying the same hypothesis but changing some details of the experiment.

- Replicas which vary the hypothesis: Although these replicas vary some variables they remain at the same level of specificity as the original experiment. They can be:
  - Replicas that vary the independent variables: These kinds of replicas are used for investigating which aspects of the process are important by varying systematically some independent variables and examining the results.
  - Replicas that vary the variables that are intrinsic to the object study: These replicas vary the way in which the effectivity is measured in order to try to understand what dimensions of which tasks are more important and
  - Replicas that vary the context variables in the environment in which the solution is evaluated: These kinds of replica is used for establishing which aspects of this environment are important because they affect the research process results and they allow us to understand the external validity.
- Replicas that extend the theory. These kinds of replica helps us to determine the limits of a process effectivity by making changes in the processes, products and/or models of the context in order to see if the basic principles remain.

Appendix D

Experimental Material

In this appendix we will present the experimental material used in each experiment. Moreover, we will show a debriefing questionnaire we used in each experiment in order to gather personal details of the subjects.

D.1 Material of the 1^{st} Experiment

1^{st} Object, 1^{st} Experiment.

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

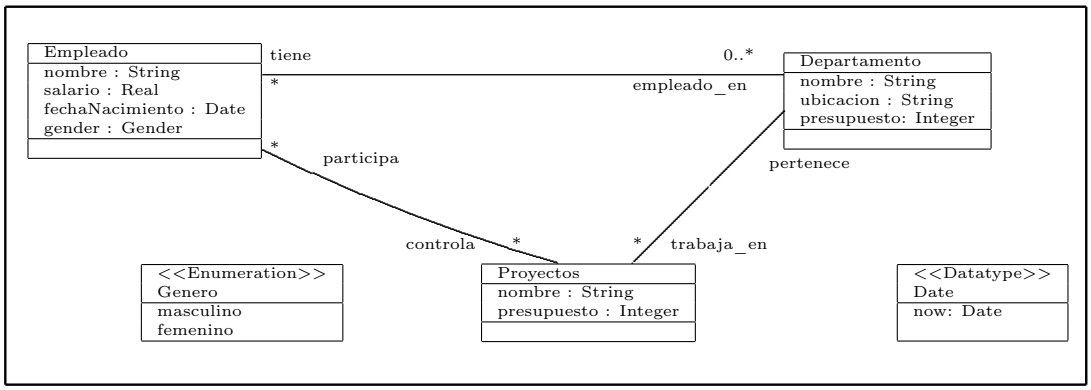


Figure D.1: 1^{st} Object, 1^{st} Experiment

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.1:

```
context Proyectos inv:
        self.presupuesto <= self.pertenece.presupuesto and
        self.presupuesto > 0 and
```

self.participa->size() < 50

1. A cuantas clases (distintas) se ha navegado a partir de la expresión? Indique el número y el nombre de esas clases.
2. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ El presupuesto de un proyecto es mayor que cero y no debe ser mayor que el presupuesto del departamento al cual pertenece el proyecto.

   ☐ El presupuesto de un proyecto es menor que cero y debe pertenecer al presupuesto del departamento al cual pertenece el proyecto.

   ☐ El presupuesto es positivo y debe ser un presupuesto del departamento.

3. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ La cantidad de empleados de los departamentos no es superior a 50

   ☐ La cantidad de empleados que participan en un proyecto no debe ser superior a 50.

   ☐ Los empleados pueden participar en a lo sumo 50 proyectos.

4. Considere la cantidad de veces que aparece escrita la palabra *presupuesto*. Cuál de las siguientes opciones es verdadera?

   ☐ *presupuesto* es siempre una propiedad (atributo) del tipo representado por la instancia contextual self.

   ☐ *presupuesto* aparece dos veces como una propiedad (atributo) del tipo representado por la instancia contextual self, y una vez como una propiedad de la instancia de Departamentos.

   ☐ *presupuesto* aparece una vez como una propiedad (atributo) del tipo representado por la instancia contextual self, y dos veces como una propiedad de la instancia de Departamentos.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

2^{nd} Object, 1^{st} Experiment

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

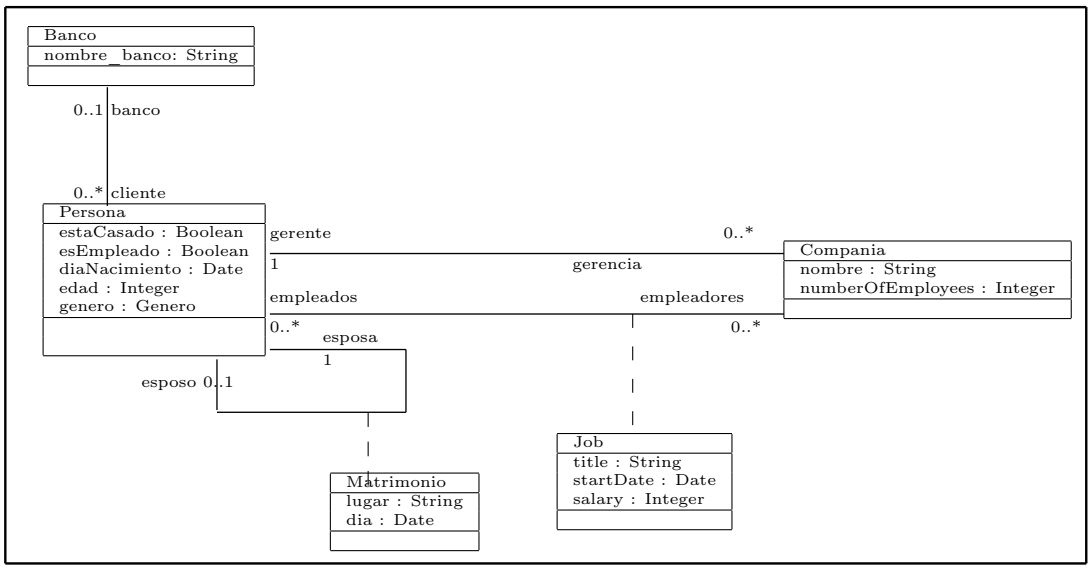


Figure D.2: 2^{nd} Object, 1^{st} Experiment

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.2:

```
context Persona inv:
        self.Trabajo.salario->sum() >= 10000 implies self.banco->notEmpty() and
        self.empleadores-> size() <= 3 and
        self.esposa->notEmpty() implies self.esposa.edad >= 18
```

1. A cuantas clases (distintas) se ha navegado a partir de la expresión? Indique el número y el nombre de esas clases.
2. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Un cliente de un banco no tiene un salario menor a 10000.

   ☐ Si la cantidad de salarios de una persona es superior a 10000 la persona es cliente de un banco.

   ☐ Si la suma de salarios de una persona es superior a 10000 la persona es cliente de un banco.

3. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Si una persona tiene una esposa, su edad es mayor a 18 años.

   ☐ Toda persona tiene una esposa mayor a 18 años.

- ☐ Todas las personas tienen una esposa de 18 años.

4. Considere la navegación expresada como *self.wife*. Cuál de las siguientes opciones es verdadera?

   - ☐ Es una navegación de una relación reflexiva, y su resultado siempre es un objeto Persona (la esposa).
   - ☐ Es una navegación a la clase de asociación y el resultado obtenido son los ma-trimonios de una persona.
   - ☐ Es una navegación de una relación reflexiva y su resultado posiblemente es un objeto Persona (la esposa).

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

3^{rd} Object, 1^{st} Experiment

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

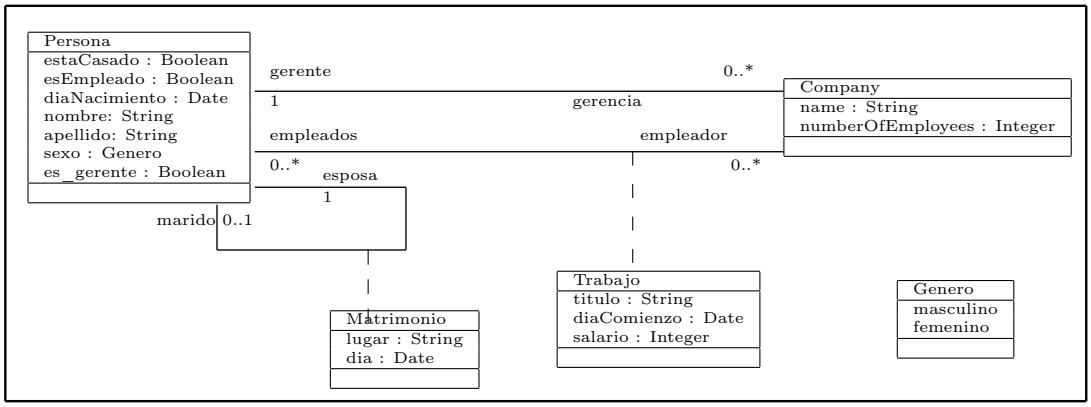


Figure D.3: 3^{rd} Object, 1^{st} Experiment

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.3:

```
context Persona inv:
    (self.esposa->notEmpty() implies
    self.esposa.empleador->forAll(p | p.gerente <> self))
    and (self.marido->notEmpty() implies self.marido.esta_casado = true )
```

1. A cuantas clases (distintas) se ha navegado a partir de la expresión? Indique el número y el nombre de esas clases.
2. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Si dos personas forman un matrimonio (es decir son conyuges) no pueden trabajar en la misma empresa.

   ☐ Una persona (que tiene una esposa) no puede ser gerente de algunas de las empresas empleadoras de su esposa.

   ☐ Una persona (que tiene una esposa) no puede ser gerente de ninguna de las empresas empleadoras de su esposa.

3. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Si una persona tiene un marido, entonces su marido está casado.

   ☐ Si dos personas forman un matrimonio (es decir son cónyuges) no pueden ser solteros.

   ☐ Una persona tiene un único marido, y este es casado.

4. Considere la navegación expresada como *self.esposa.empleador*. Cuál de las siguientes opciones es verdadera?

- ☐ El resultado de la navegación completa es una colección de tipo conjunto debido a que toda navegación resulta en un objeto o un conjunto de objetos.
- ☐ El resultado de la navegación completa es una colección de tipo bag debido a que hemos navegado más de una asociación.
- ☐ El resultado de la navegación completa es una colección de tipo secuencia, en el cual todos los elementos están ordenados.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

4^{th} Object, 1^{st} Experiment

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

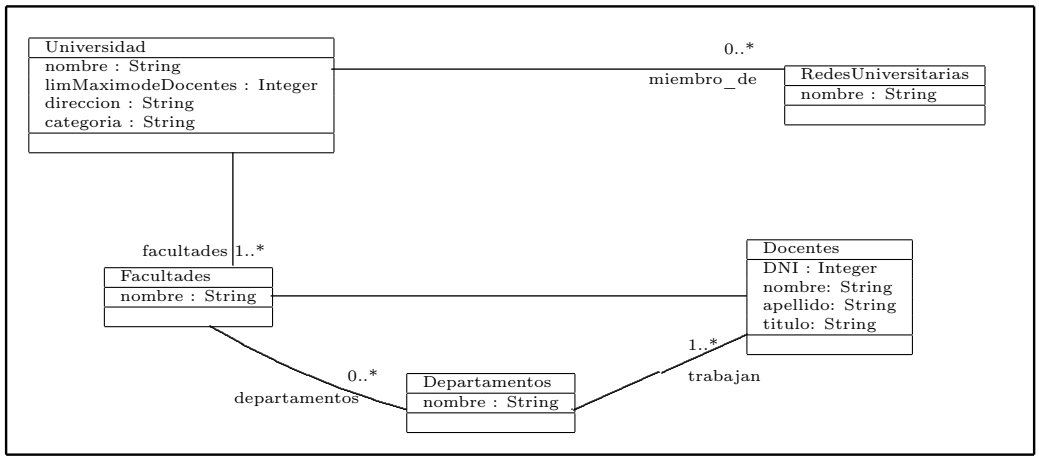


Figure D.4: 4^{th} Object, 1^{st} Experiment

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.4:

```
context Universidad inv:
    (self.miembro_de->size() > 3 implies self.categoria ='alta') and
    (self.limite_max_de_docentes >
    self.facultades.departamentos.trabajan -> asSet()->size() )
```

1. A cuantas clases (distintas) se ha navegado a partir de la expresión? Indique el número y el nombre de esas clases.
2. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ La universidad tiene categoría alta y debe ser miembro de tres redes universitarias.

   ☐ Si una universidad tiene categoría alta es miembro de más de tres redes universitarias.

   ☐ Si una universidad es miembro de más tres redes universitarias, su categoría es alta.

3. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ La cantidad de docentes que trabajan en facultades es inferior al límite máximo de docentes.

   ☐ La cantidad de docentes que trabajan en los departamentos de las facultades de la universidad es inferior al límite máximo de docentes que tiene la universidad.

   ☐ La cantidad de docentes que trabajan en los departamentos de las facultades de la universidad es mayor igual al límite máximo de docentes que tiene la universidad.

4. Considere la siguiente subexpresión perteneciente a la expresión OCL enunciada anteriormente
self.facultades.departamentos.trabajan -> asSet()->size() Cuál de las siguientes opciones es verdadera?

   ☐ Si un docente de la colección self.facultades.departamentos.trabajan figura dos veces, este se cuenta dos veces en la subexpresión.

   ☐ En la subexpresión, se cuentan los docentes de la colección self.facultades.departamentos.trabajan sin repetir.

   ☐ La utilización de la operación de colección asSet() es innecesaria en la subexpresión.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

D.2 Material of the 1^{st} Family of Experiments

1^{st} Object, 1^{st} Family of Experiments

Part I.

Por favor, indique el horario en el cual inicia el ejercicio:

hour..... minute ..... second .....

.

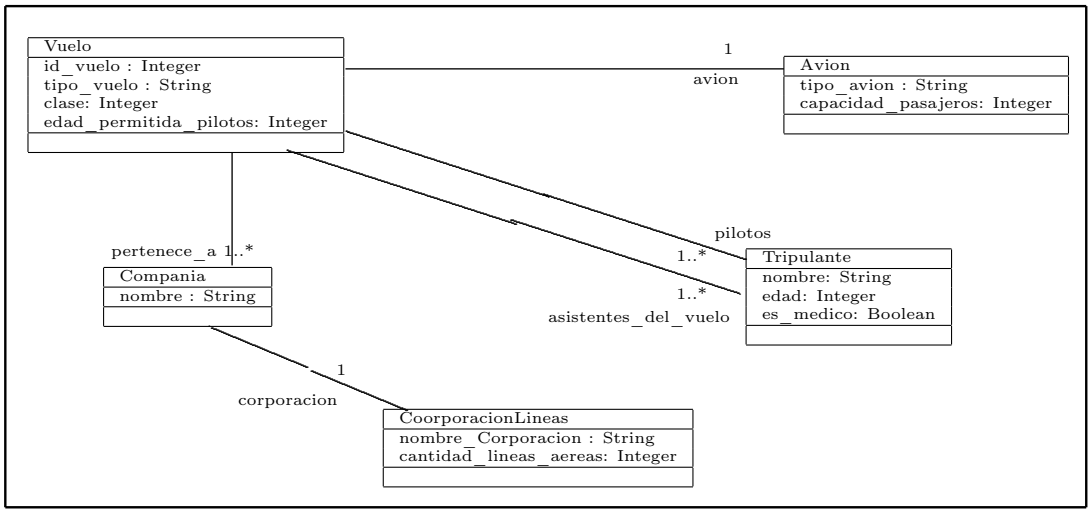


Figure D.5: 1^{st} Object, 1^{st} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.5:

```
context Vuelo
        inv: self.tipo_vuelo = 'trasatlantico' implies
        (self.pilotos-> size() = 3 and self.avion.tipo_avion = 'boing')
        and self.avion.capacidad_pasajeros >= 70
        and self.pilotos ->forAll (h | h.edad >= self.edad_permitida_pilotos)
```

1. A cuantas clases (distintas) se ha navegado a partir de la expresión? Indique el número y el nombre de esas clases.
2. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Si el tipo de avión es boing y la cantidad de pilotos del vuelo es igual a 3, el tipo de vuelo es trasatlántico.

   ☐ Los aviones de tipo boing tienen 3 pilotos y hacen vuelos trasatlánticos.

   ☐ Si el tipo de vuelo es trasatlántico, el tipo de avión es boing y los pilotos del vuelo deben ser 3.

3. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

- ☐ La capacidad de pasajeros del vuelo debe ser siempre superior (o igual) a 70 pasajeros.
- ☐ La capacidad de pasajeros del avión de todo vuelo trasatlántico, es igual (o superior) a 70 pasajeros.
- ☐ La capacidad de pasajeros del avión de un vuelo es igual (o superior) a 70 pasajeros.

4. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   - ☐ Los pilotos de un vuelo deben ser mayores de edad.
   - ☐ La edad de los pilotos de un vuelo no debe ser inferior a la edad permitida para pilotos de ese vuelo.
   - ☐ Los pilotos de un vuelo deben tener una edad mayor que la antigedad mínima del vuelo.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sean válidos los siguientes requerimientos:

   - La capacidad de pasajeros del avión de todo vuelo debe ser inferior a 450 pasajeros.
   - Si el tipo de vuelo es 'cabotaje' entonces la cantidad de 'asistentes del vuelo' debe ser menor a 5 y el tipo de avión no debe ser 'boing'.
   - Una de las personas del conjunto de 'asistentes del vuelo' es médico.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

2^{nd} Object, 1^{st} Family of Experiments

Part I.

Por favor, indique el horario en el cual inicia el ejercicio:

hour..... minute ..... second .....

.

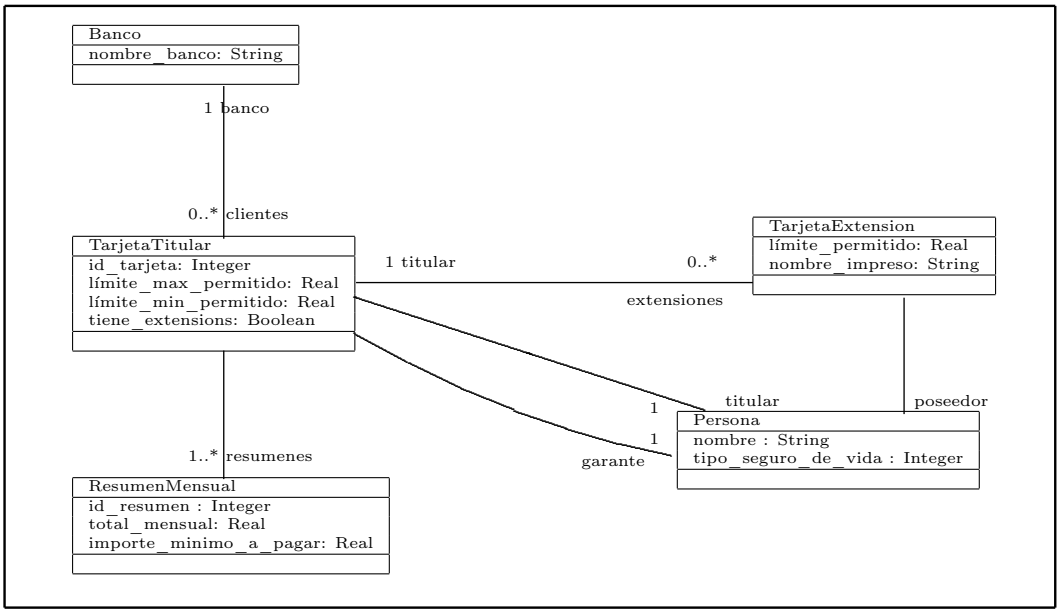


Figure D.6: 2^{nd} Object, 1^{st} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.6:

```
context TarjetaTitular inv:
        ((self.banco.nombre_banco = 'Galicia' and self.tiene_extensiones = true)
        implies (self.extensiones->size() < 6) )
        and self.titular.tipo_seguro_de_vida >= 3
        and self.resumenes ->forAll(a | a.total_mensual <=
        self.limite_max_permitido )
```

1. A cuantas clases (distintas) se ha navegado a partir de la expresión? Indique el número y el nombre de esas clases.

2. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Si la tarjeta de crédito del titular tiene menos de 6 extensiones, la tarjeta pertenece al banco Galicia.

   ☐ Si la tarjeta de crédito del titular tiene extensiones y pertenece al banco Galicia, entonces la cantidad de extensiones no es superior a 5.

   ☐ El banco Galicia permite hasta 6 extensiones de una tarjeta de crédito del titular.

3. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

- ☐ El tipo de seguro de vida es igual o superior a 3.
- ☐ El titular de toda tarjeta de crédito titular tiene un tipo de seguro de vida igual o inferior a 3.
- ☐ El titular de una tarjeta de crédito titular tiene un tipo de seguro de vida superior a 2.

4. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   - ☐ Todos los resúmenes de todas las tarjetas de crédito titulares tienen un importe total que no supera el límite máximo.
   - ☐ Todos los resúmenes de una tarjeta de crédito titular tienen un importe total que no supera el límite máximo permitido para esa tarjeta.
   - ☐ Todos los resúmenes de una tarjeta de crédito titular tienen un importe total que es inferior estrictamente al límite máximo permitido para la tarjeta.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sean válidos los siguientes requerimientos:

   - Si la tarjeta pertenece al banco Galicia y no tiene extensiones, entonces la cantidad de extensiones es cero.
   - El tipo de seguro de vida de la persona que actuó como garantía de una tarjeta titular, es superior a 3.
   - Todos los resúmenes de una tarjeta de crédito tienen un 'importe mínimo a pagar' que es igual a la mitad de su importe total.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

3^{rd} Object, 1^{st} Family of Experiments

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

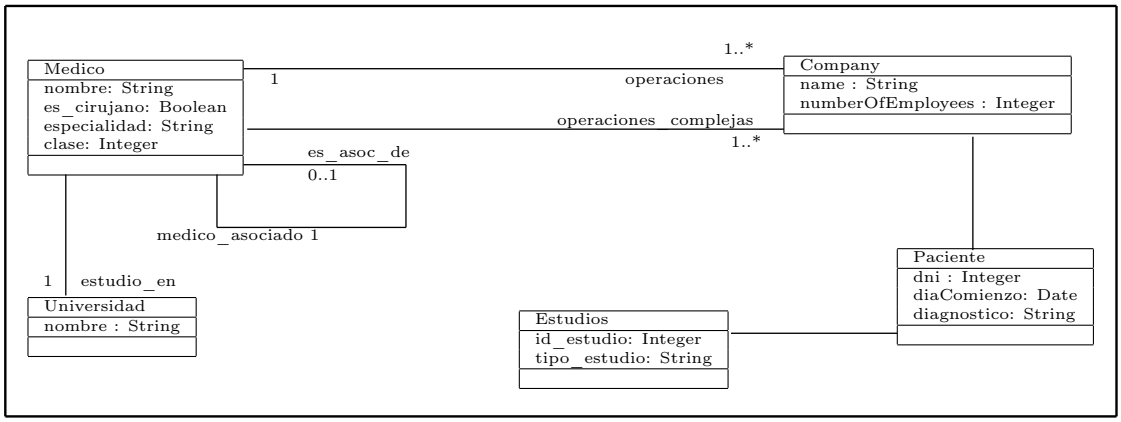


Figure D.7: 3^{rd} Object, 1^{st} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.7:

```
context Operacion inv:
        self.tipo_Operacion > 3 and
        (self.medico_principal.es_cirujano = true implies
        (self.medico_principal.medico_asociado->size() = 1
        and self.medico_principal.medico_asociado.especialidad = 'anestesista' ))
        and self.medico_principal.medico_asociado.operaciones
        ->exists( p | p.id_operacion = self.id_operacion)
```

1. A cuantas clases (distintas) se ha navegado a partir de la expresión? Indique el número y el nombre de esas clases.
2. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Si el médico principal de la operación es cirujano, entonces ese médico principal tiene médicos asociados anestesistas.

   ☐ Si el médico principal de la operación es cirujano, entonces ese médico principal tiene un médico asociado que es anestesista.

   ☐ Todos los anestesistas son los médicos asociados de todo médico cirujano responsable de una operación.

3. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Toda operación no existe en la colección de operaciones que realizó el médico asociado del médico principal de la operación.

- ☐ Toda operación existe en la colección de operaciones que realizó el médico asociado del médico principal de la operación.
- ☐ Toda operación existe en la colección de operaciones que realizó un médico asociado.

4. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   - ☐ El tipo de toda operación es inferior a 3.
   - ☐ 3 es inferior al tipo de toda operación.
   - ☐ El tipo de una operación es superior a 1.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

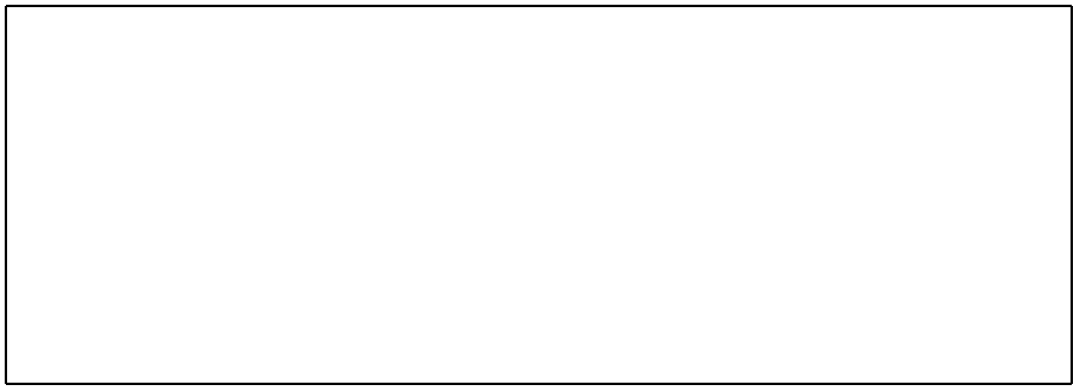
Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sean válidos los siguientes requerimientos:

   - Todos los elementos de la colección de operaciones complejas que realizó el médico asociado del médico principal de una operación, tienen una identificación superior o igual al tipo de esa operación.
   - El tipo de una operación debe ser igual a la clase del médico principal de la asociación.
   - Si el médico principal de una operación no es cirujano entonces ese médico no tiene un médico asociado.



Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

4^{th} Object, 1^{st} Family of Experiment.

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

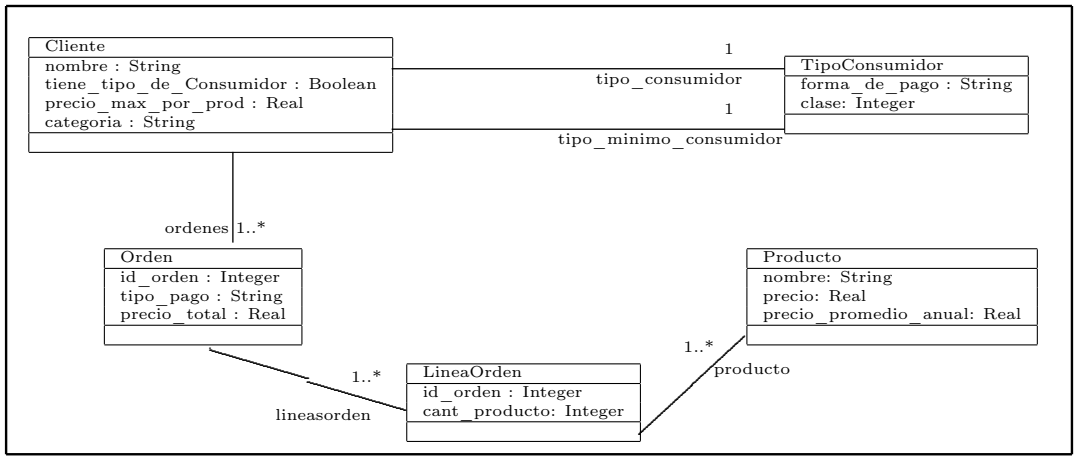


Figure D.8: 4^{th} Object, 1^{st} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.8:

```
context Cliente inv:
    (self.tiene_tipo_de_consumidor = true implies
    (self.tipo_consumidor.forma_de_pago= 'debito' or
    self.tipo_consumidor.clase<= 1))
    and self.ordenes->size() > 50 )
    and self.ordenes.lineasorden.producto ->
    forAll (p | p.precio <= self.precio_max_permitido_de_prod )
```

1. A cuantas clases (distintas) se ha navegado a partir de la expresión? Indique el número y el nombre de esas clases.
2. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ Si el cliente tiene un tipo de consumidor entonces la forma de pago de su tipo de consumidor es debito o la clase del tipo de consumidor es inferior a 1.

   ☐ Si el cliente tiene un tipo de consumidor entonces la forma de pago de su tipo de consumidor es debito o la clase del tipo de consumidor es inferior a 2.

   ☐ Si el tipo de consumidor es de clase 1 o su forma de pago es debito entonces el cliente tiene un tipo de consumidor.

3. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   ☐ La cantidad de ordenes solicitas por un cliente no supera las 50 ordenes.

   ☐ La cantidad de ordenes solicitadas por un cliente no es inferior a 51 ordenes.

- ☐ Todos los clientes han solicitado menos de 51 ordenes.

4. Alguna parte de la expresión OCL enunciada anteriormente significa alguna de las siguientes expresiones escritas en Lenguaje Natural?. Cuál de las siguientes opciones es verdadera?

   - ☐ Todos los productos de las ordenes solicitadas por un cliente tiene un precio, inferior o igual, al precio máximo permitido para el producto.
   - ☐ Todos los productos de las líneas de orden de las ordenes solicitadas por un cliente tiene un precio, inferior o igual, al precio máximo permitido por producto para ese cliente.
   - ☐ Todos los productos de las líneas de orden de las ordenes solicitadas por un cliente tiene un precio inferior al precio máximo permitido por producto.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sean válidos los siguientes requerimientos:

   - Todos los productos de las líneas de orden de las ordenes solicitadas por un cliente tiene un 'precio promedio anual', inferior a la mitad del precio máximo permitido por producto para ese cliente.
   - La cantidad de ordenes solicitadas por un cliente no es superior a 1000.
   - Si el cliente no tiene un tipo de consumidor entonces 'el tipo mínimo de consumidor' tiene una forma de pago que es 'efectivo' y la clase del 'tipo de mínimo de consumidor' es superior a 2.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

D.3 Material of the 2^{nd} Family of Experiments

1^{st} Object, 2^{nd} Family of Experiment.

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

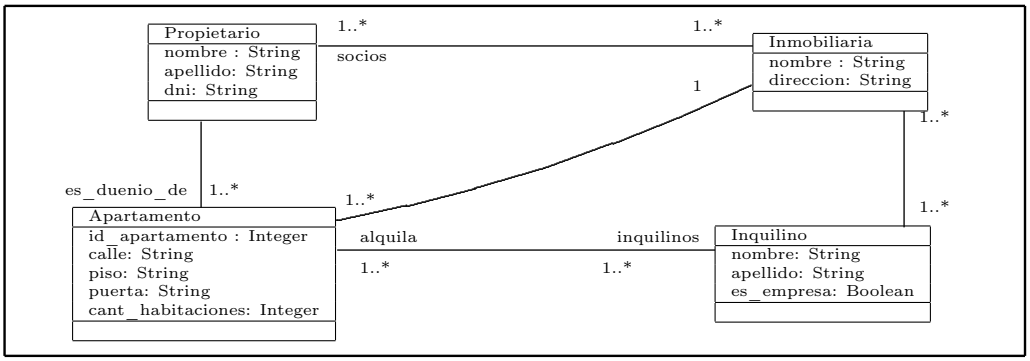


Figure D.9: 1^{st} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.9:

```
context Apartamento inv:
    self.inquilinos ->forAll (i | self.propietario->collect(dni)
    ->select(d:String | d = i.dni)->isEmpty() )
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ Un inquilino de un apartamento no pueden ser propietario de ningún apartamento.

   ☐ Un propietario de un apartamento no puede ser inquilino de su propio apartamento.

   ☐ Todos los inquilinos tienen distintos número de DNI.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo del iterador i es Inquilino.

   ☐ El tipo del iterador i es Apartamento.

   ☐ El tipo del iterador i es Propietario.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:

   - Cada uno de los inquilinos de un apartamento debe tener un DNI distinto.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

2^{nd} Object, 2^{nd} Family of Experiment.

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

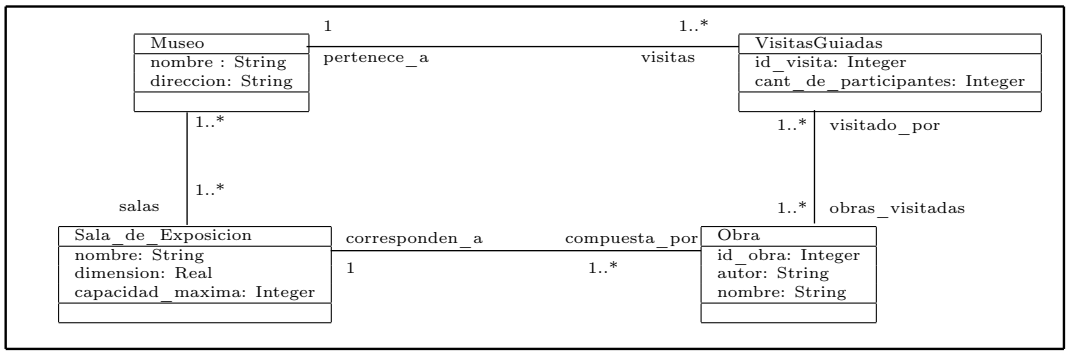


Figure D.10: 2^{nd} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.10:

```
context Museo inv:
        self.visitas -> forAll (v | v.obras_visitadas.corresponden_a
        ->collect(capacidad_maxima)->forAll(c | c > v.cant_de_participantes))
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ Todas las salas de exposición que contienen a las obras que son visitadas por alguna visita guiada de un museo, tienen una capacidad máxima que no supera la cantidad de participantes de esa visita guiada.

   ☐ Todas las capacidades máximas de las salas de exposición son superiores a la capacidad máxima de toda visita guiada.

   ☐ Todas las salas de exposición que contienen a las obras que son visitadas por alguna visita guiada de un museo, tienen una capacidad máxima que no supera la cantidad participantes de toda visita guiada.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo del iterador c es Visita_Guiada.

   ☐ El tipo del iterador c es Integer.

   ☐ El tipo del iterador c es Sala_de_Exposición.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:

   - Todas las obras visitadas en una visita guiada de un museo, son obras que están incluídas en alguna de las salas de exposición del museo cuya dimensión es mayor a 230 m^2.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

3^{rd} Object, 2^{nd} Family of Experiments

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

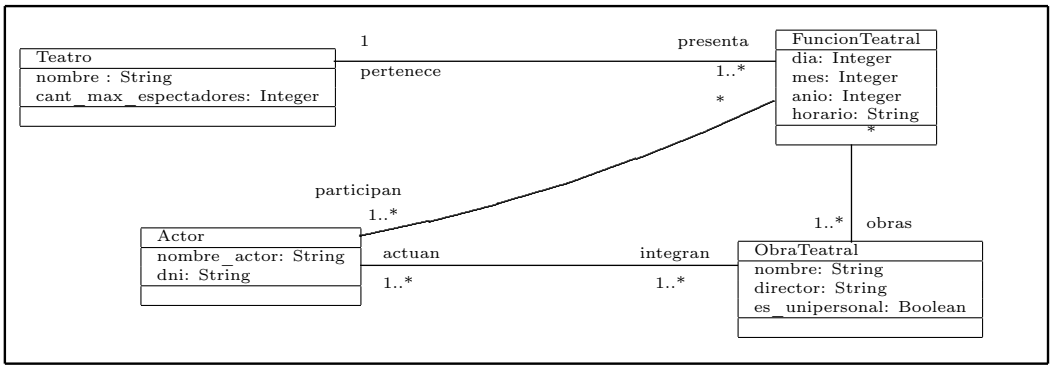


Figure D.11: 3^{rd} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.11:

```
context Teatro inv:
    self.presenta->forAll( f | f.obras->select(o | o.es_unipersonal)->
    forAll(i | i.actuan->size() = 1) )
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ Todas las obras teatrales unipersonales son obras en las cuales actúa un único actor.

   ☐ En todas las funciones teatrales de un teatro, si la obra teatral es unipersonal entonces solo un actor actúa en la obra.

   ☐ Todas las obras teatrales que se realizan en cualquier función teatral de un teatro son obras unipersonales realizadas por un solo actor.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo del iterador i es Obra_Teatral.

   ☐ El tipo del iterador i es Boolean.

   ☐ El tipo del iterador i es Funcion_Teatral.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:

   - Para toda función teatral de un teatro, si la obra teatral no es unipersonal entonces la cantidad de actores que actúan en la obra no superan los 30 actores.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

4^{th} Object, 2^{nd} Family of Experiments

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

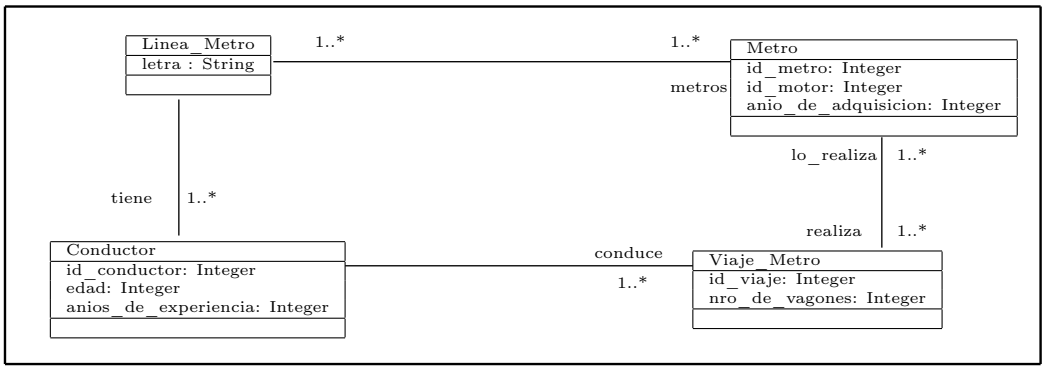


Figure D.12: 4^{th} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.12:

```
context Linea_Metro inv:
    self.tiene-> select(c | c.edad >50)->isEmpty()
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ Todos los conductores de las líneas de metro tienen una edad mayor a 50 años.

   ☐ No hay conductores en ninguna línea de metro cuya edad sea superior a 50 años.

   ☐ La colección de los conductores cuyas edades son inferiores a 50 años, es vacía.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo del iterador c es LineaMetro.

   ☐ El tipo del iterador c es Integer.

   ☐ El tipo del iterador c es Conductor.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:
   - Toda línea de metro tiene al menos un conductor con 10 años de experiencia.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

5^{th} Object, 2^{nd} Family of Experiments

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

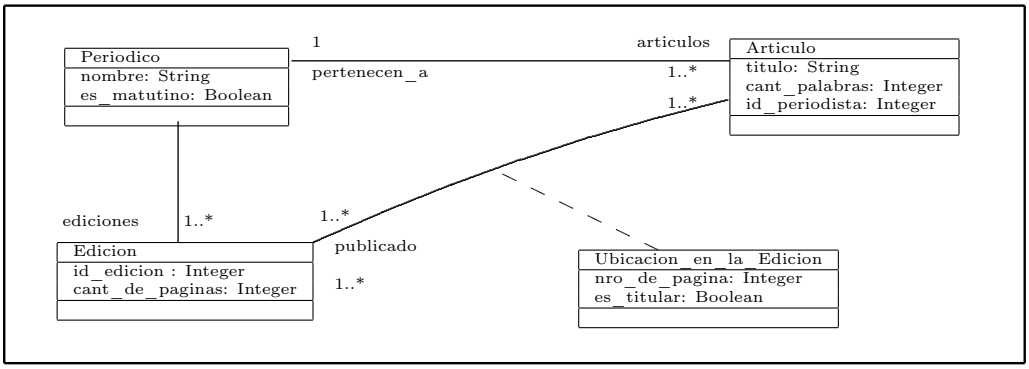


Figure D.13: 5^{th} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.13:

```
context Edicion
        inv: self.Ubicacion_en_la_Edicion -> collect(u | u.es_titular)->notEmpty()
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ La colección de titulares de una edición esta vacía.

   ☐ Una edición siempre contiene artículos que son titulares.

   ☐ Las ediciones solo tienen artículos que son titulares.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo del iterador u es Boolean.

   ☐ El tipo del iterador u es Ubicación_en_la_edición.

   ☐ El tipo del iterador u es Artículo.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:

   - Existen artículos ubicados en la edición que son titulares y ellos se ubican en la página 1.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

6^{th} Object, 2^{nd} Family of Experiments

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

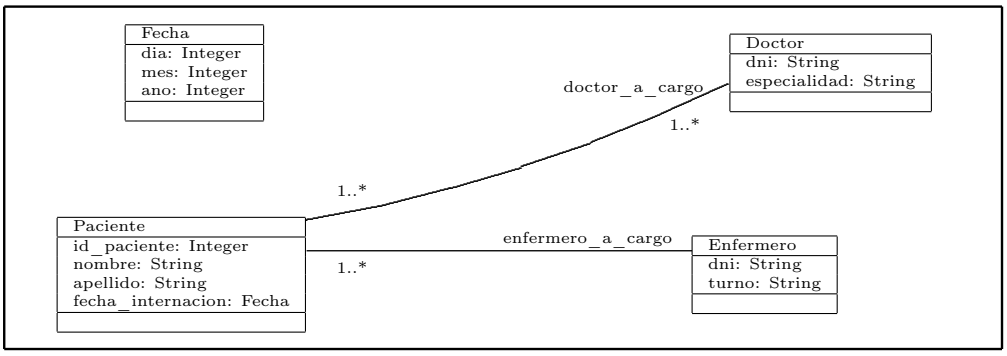


Figure D.14: 6^{th} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.14:

```
context Apartamento inv:
    self.inquilinos ->forAll (i | self.propietario->collect(dni)
    ->select(d:String | d = i.dni)->isEmpty() )
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ Un paciente puede estar a cargo de un doctor y de un enfermero.

   ☐ Si un paciente esta a cargo de un doctor, entonces puede estar a cargo de un enfermero.

   ☐ Un paciente puede estar a cargo de un doctor ó de un enfermero, pero no de ambos, ni de ninguno a la vez.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo de la subexpresión self.doctor_a_cargo es Integer.

   ☐ El tipo de la subexpresión self.doctor_a_cargo es Paciente.

   ☐ El tipo de la subexpresión self.doctor_a_cargo es Doctor.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:
   - Un paciente puede estar a cargo de un doctor o de un enfermero, o de ambos a la vez.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

7^{th} Object, 2^{nd} Family of Experiment.

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

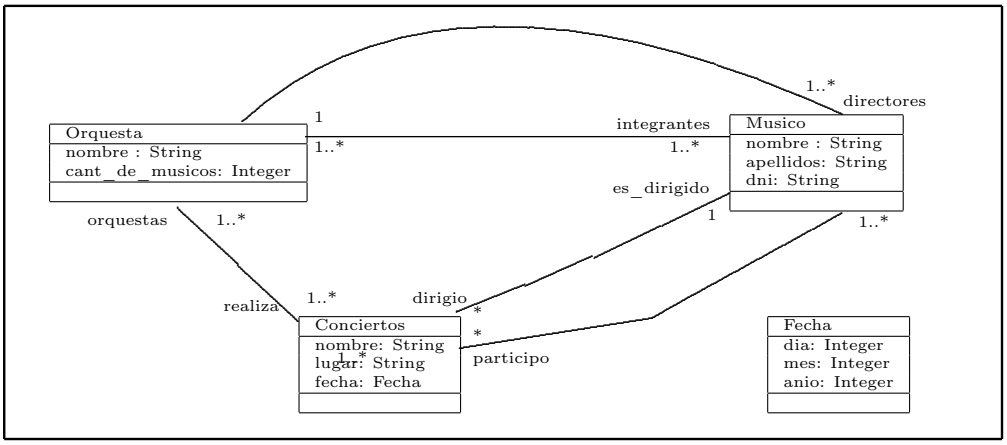


Figure D.15: 7^{th} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.15:

```
context Orquesta inv:
    self.participa -> forAll (c | self.directores -> includes (c.es_dirigido))
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ Todos los directores de una orquesta dirigen todos los conciertos en los cuales participa la orquesta.

   ☐ Los conciertos en los cuales participa una orquesta son dirigidos por algunos de los directores de la orquesta.

   ☐ Los directores de una orquesta incluyen a los directores de los conciertos.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo del iterador c es Orquesta.

   ☐ El tipo del iterador c es Músico.

   ☐ El tipo del iterador c es Conciertos.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:

   - El director de todo concierto en el cual participa una orquesta, es uno de los músicos que integran la orquesta.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

8^{th} Object, 2^{nd} Family of Experiment.

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

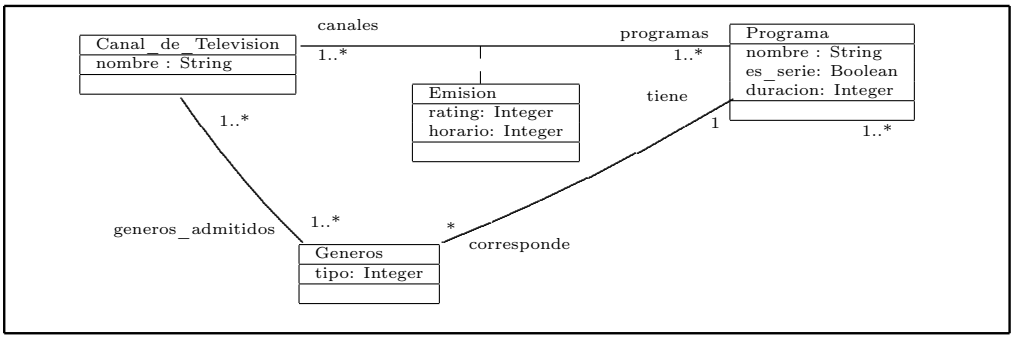


Figure D.16: 8^{th} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.16:

```
context Canal_de_Television
        inv: self.programas->forAll(p | self.generos_admitidos->includes(p.corresponde) )
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ Los géneros de los programas de un canal de televisión no incluyen a los géneros admitidos por dicho canal.

   ☐ Todos los programas emitidos por un canal de televisión tiene un genero que pertenece al conjunto de géneros admitidos por el canal de televisión.

   ☐ Todos los géneros de todos los programas de todos los canales de televisión coinciden con los géneros admitidos por todos los programas.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo del iterador p es Programa.

   ☐ El tipo del iterador p es Integer.

   ☐ El tipo del iterador p es Genero.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:

   - Para todo género admitido de un canal de televisión, se verifica que dicho género pertenece a la colección de géneros de los programas del canal de televisión.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

9^{th} Object, 2^{nd} Family of Experiment.

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

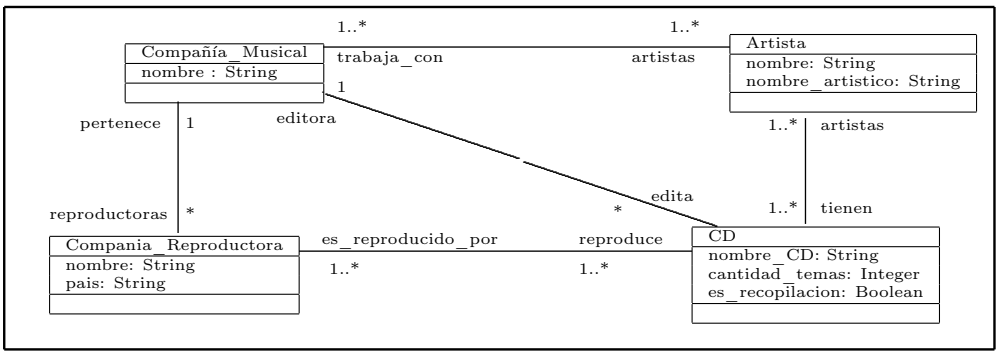


Figure D.17: 9^{th} Object, 2^{nd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.17:

```
context CD inv:
    self.es_reproducido_por->forAll(r| r.pertenece = self.editora)
```

1. Cuantas relaciones distintas ha navegado a partir de la expresión? Indique el número y el nombre de los roles que han utilizado dichas navegaciones.
2. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural se corresponde con el significado de la expresión OCL.

   ☐ La compañía musical de un CD pertenece a la Compañía reproductora que edita el CD.

   ☐ Las compañías reproductoras, como las compañías reproductoras editan todo CD.

   ☐ Las compañías reproductoras de un CD pertenecen a la Compañía musical que edita el CD.

3. Cuantas operaciones de colección se han utilizado en la expresión OCL? Mencione la cantidad y el nombre de cada operación.
4. Marque con una cruz, indicando cual de las siguientes expresiones escritas en Lenguaje Natural es verdadera.

   ☐ El tipo del iterador r es Compañía_Reproductora.

   ☐ El tipo del iterador r es Compañía_Musical.

   ☐ El tipo del iterador r es CD.

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

5. Se requiere rescribir la expresión OCL de tal forma que sea válido el siguiente requerimiento:

   - La compañías reproductoras que reproducen un CD están incluídas en la colección de las compañías reproductoras que pertenecen a la compañía musical que editó el CD.

Rescriba la expresión OCL a continuación:

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

D.4 Material of the 3^{rd} Family of Experiments

1^{st} Object, 3^{rd} Family of Experiment.

Part II Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

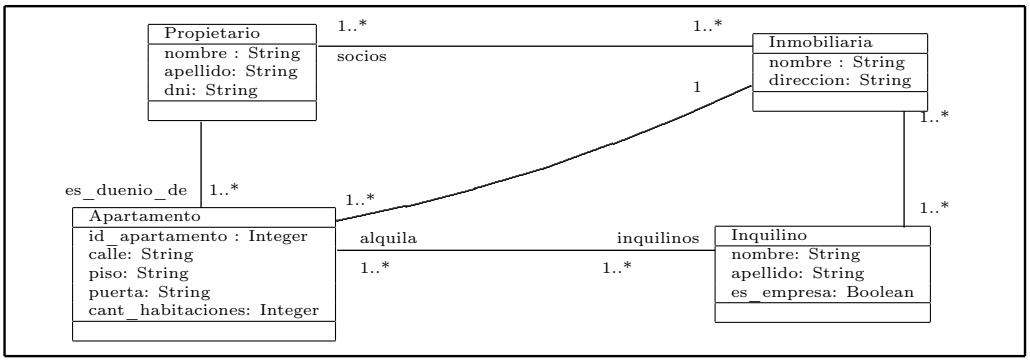


Figure D.18: 1^{st} Object, 3^{rd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.18:

```
context Apartamento inv:
        self.inquilinos ->forAll (i | self.propietario->collect(dni)
        ->select(d:String | d = i.dni)->isEmpty() )
```

Marque con una cruz la única opción que considera que satisface cada requerimento de modificación de la expresión anterior:

Requerimiento: Ningún inquilino de un Apartamento puede tener un nombre que coincida con el nombre de alguna Inmobiliaria que gestiona el alquiler del apartamento.

context Apartamento inv:

- ☐ self.inquilinos-> forAll (i | self.gestiona->collect(nombre)->select(o : String | o = i.nombre)->isEmpty())
- ☐ self.inquilinos-> forAll (i | self.propietario.socios->collect(nombre)->select(o : String | o = i.nombre))
- ☐ self.alquila-> forAll (i | self.gestiona->collect(nombre)->select(o : String | o = i.nombre)->isEmpty())

Requerimiento: El apellido de un propietario de un apartamento no puede coincidir con el apellido de algún inquilino del apartamento.

context Apartamento inv:

- ☐ self.propietarios ->forAll (p | self.inquilinos->collect(nombre)->select(a:String | a = apellido)->isEmpty())
- ☐ self.propietarios ->forAll (p | self.inquilinos->collect(apellido <> p.apellido)->isEmpty())

☐ self.propietarios ->forAll (p | self.inquilinos->collect(apellido)->select(a:String | a = p.apellido)->isEmpty())

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

2^{nd} Object, 3^{rd} Family of Experiment.

Part II Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

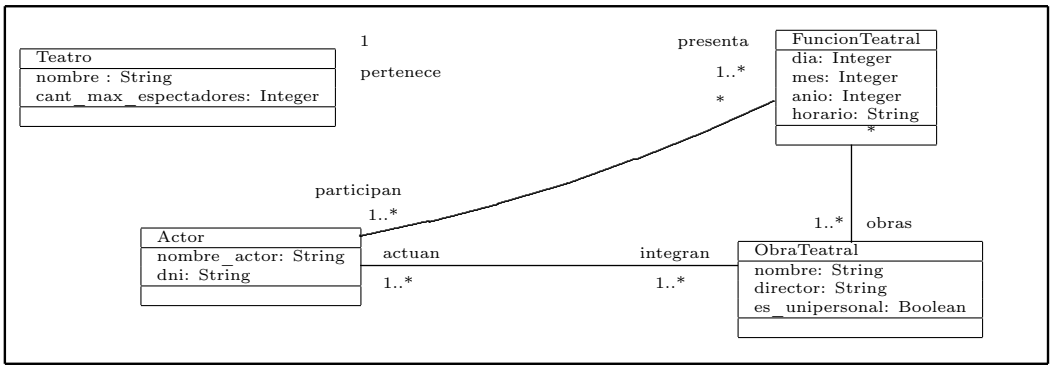


Figure D.19: 2^{nd} Object, 3^{rd} Family of Experiments

Dada la siguiente expresión OCL válida para el diagrama UML de la Figura D.19:

```
context Teatro inv:
        self.presenta->forAll( f | f.obras->select(o | o.es_unipersonal)->
        forAll(i | i.actuan->size() = 1) )
```

Marque con una cruz la única opción que considera que satisface cada requerimento de modificación de la expresión anterior:

Requerimiento 1: En todas las funciones teatrales de un teatro, si la obra teatral no es unipersonal entonces en ella actua más de un actor.

context Teatro inv:

- ☐ self.presenta->forAll(f | f.obras->select(o | o.es_unipersonal = true)->forAll(i | i.actuan->size() > 1))
- ☐ self.presenta->forAll(f | f.obras->reject(o | o.es_unipersonal = true)->forAll(i | i.actuan->size() > 1))
- ☐ self.presenta->forAll(f | f.obras->reject(o | o.es_unipersonal = true)->exists(i | i.actuan->size() > 1))

Requerimiento 2: En todas las funciones teatrales de un teatro, los actores primarios que participan en la función pueden integrar el elenco de a lo sumo 3 obras.

context Teatro inv:

- ☐ self.presenta->forAll(f | f.participan->reject(a | a.es_actor_primario = true)->forAll(a | a.integran->size() < 3)
- ☐ self.presenta->forAll(f | f.participan->select(a | a.es_actor_primario = true)->forAll(a | a.integran->size() <= 3)

☐ self.presenta->forAll(f | f.participan->select(a | a.es_actor_primario = true)->forAll(a | a.integran->size() > 3)

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

3^{rd} Object, 3^{rd} Family of Experiment.

Part II Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

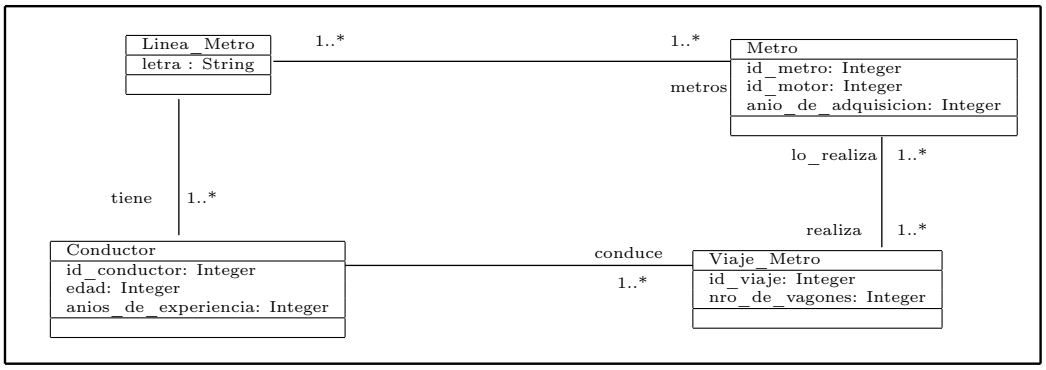


Figure D.20: 3^{rd} Object, 3^{rd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.20:

```
context Linea_Metro inv:
        self.tiene-> select(c | c.edad >50)->isEmpty()
```

Marque con una cruz la única opción que considera que satisface cada requerimento de modificación de la expresión anterior:

Requerimiento 1: En todas las Lineas de Metros deben existir conductores con más de 10 años de experiencia.

context Linea_metro inv:

- ☐ self.tiene->select(c | c.años_de_experiencia < 10)->NotEmpty()
- ☐ self.tiene->reject(c | c.años_de_experiencia > 10)->isEmpty()
- ☐ self.tiene->select(c | c.años_de_experiencia > 10)->NotEmpty()

Requerimiento 2: En todas las Lineas de Metros no deben existir metros cuyos años de adquisición sea menor a 1980.

context Linea_metro inv:

- ☐ self.metros->select(m | m.año_de_adquisición > 1980)->isEmpty()
- ☐ self.metros->reject(m | m.año_de_adquisición >= 1980)->isEmpty()
- ☐ self.metros->reject(m | m.año_de_adquisición < 1980)->isEmpty()

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

4^{th} Object, 3^{rd} Family of Experiment.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

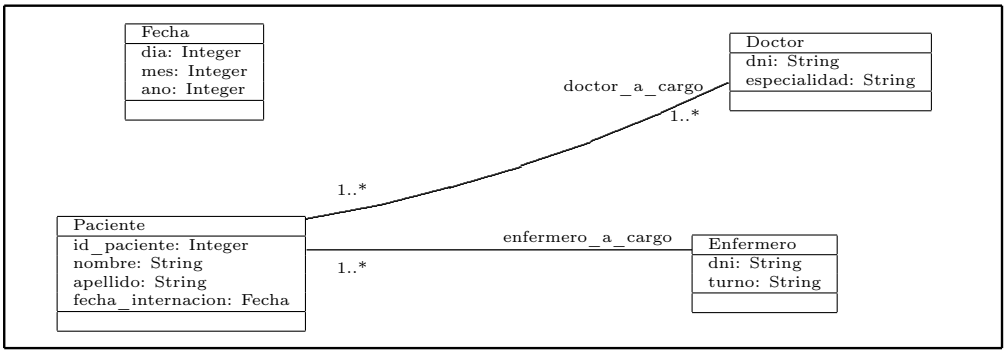


Figure D.21: 4^{th} Object, 3^{rd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.21:

```
context Apartamento inv:
        self.inquilinos ->forAll (i | self.propietario->collect(dni)
        ->select(d:String | d = i.dni)->isEmpty() )
```

Marque con una cruz la única opción que considera que satisface cada requerimento de modificación de la expresión anterior:

Requerimiento 1: Un paciente esta a cargo de un doctor o de un enfermero, o de ninguno a la vez.

context Paciente inv:

- ☐ (self. doctor_a_cargo->size() + self.enfermero_a_cargo->size()) < 1
- ☐ (self. doctor_a_cargo->size() + self.enfermero_a_cargo->size()) <= 1
- ☐ (self. doctor_a_cargo->size() + self.enfermero_a_cargo->size()) > 1

Requerimiento 2: Un paciente esta a cargo de un doctor y de un enfermero.

context Paciente inv:

- ☐ self. doctor_a_cargo->notEmpty() and self.enfermero_a_cargo->isEmpty()
- ☐ self. doctor_a_cargo->notEmpty() and self.enfermero_a_cargo->notEmpty()
- ☐ self. doctor_a_cargo->isEmpty() and self.enfermero_a_cargo->isEmpty()

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

5^{th} Object, 3^{rd} Family of Experiment.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

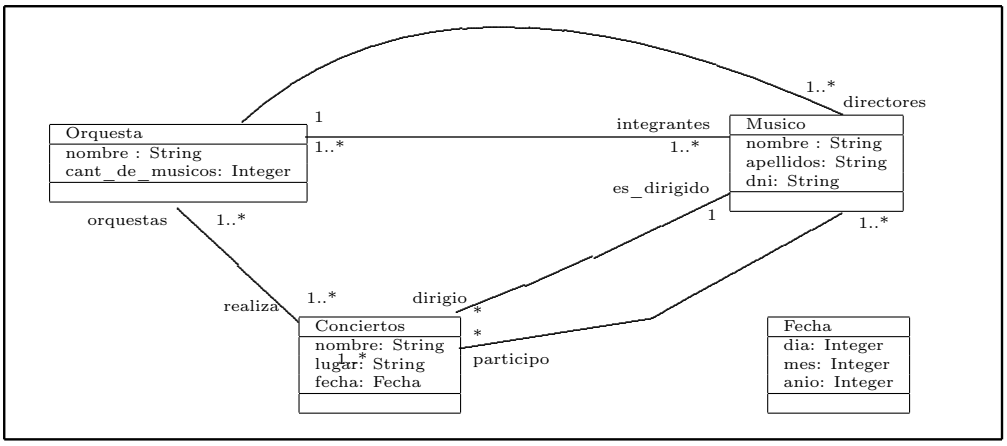


Figure D.22: 5^{th} Object, 3^{rd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.22:

```
context Orquesta inv:
        self.participa -> forAll (c | self.directores -> includes (c.es_dirigido))
```

Marque con una cruz la única opción que considera que satisface cada requerimento de modificación de la expresión anterior:

Requerimiento 1: El director de todo concierto realizado por una orquesta, es uno de los músicos que integran la orquesta.

context Orquesta inv:

- ☐ inv: self.realiza -> forAll (c | self.directores -> includes (c.es_dirigido))
- ☐ inv: self.realiza -> forAll (c | self.integrantes -> includes (c.es_dirigido))
- ☐ inv: self.realiza -> forAll (c | self.integrantes -> excludes (c.es_dirigido))

Requerimiento 2: Los conciertos en los cuales participaron los integrantes de una orquesta son conciertos que realizó dicha orquesta.

context Orquesta inv:

- ☐ inv: self.integrantes -> forAll (i | i.participo-> includes (self.realiza))
- ☐ inv: self.integrantes -> forAll (i | self.realiza-> includesAll (i.participo))
- ☐ inv: self.integrantes -> forAll (i | self.realiza-> includes (i.participo))

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

6^{th} Object, 3^{rd} Family of Experiment.

Part II

Por favor, indique el horario en el cual inicia el ejercicio:

hora..... minutos ..... segundos .....

.

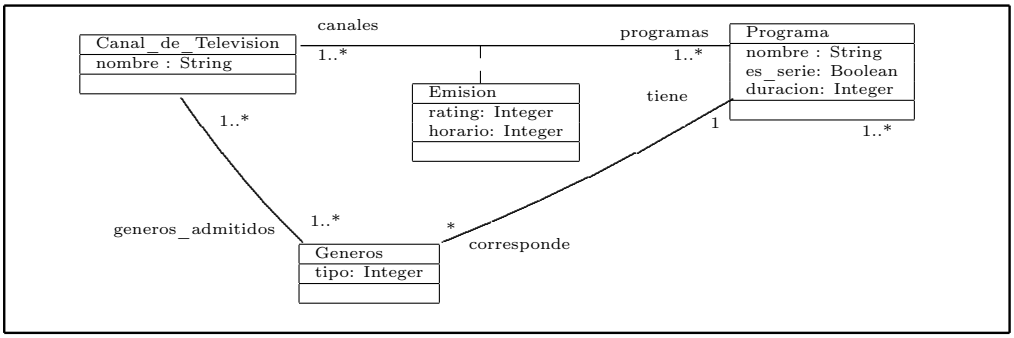


Figure D.23: 6^{th} Object, 3^{rd} Family of Experiments

Considere la siguiente expresión OCL válida para el diagrama UML de la Figura D.23:

```
context Canal_de_Television
      inv: self.programas->forAll(p | self.generos_admitidos->includes(p.corresponde) )
```

Marque con una cruz la única opción que considera que satisface cada requerimento de modificación de la expresión anterior:

Requerimiento 1: Todos los programas correspondientes a los generos admitidos de un canal de televisión, son programas de ese canal.

context Canal_de_Television inv:

- ☐ self.generos_admitidos->forAll(p | self.programas->excludes(p.tiene))
- ☐ self.generos_admitidos->forAll(p | self.programas->includesAll(p.tiene))
- ☐ self.generos_admitidos->forAll(p | p.tiene->includesAll(self.programas))

Requerimiento 2: Para todo genero admitido de un canal de televisión existe algun programa del canal con dicho genero.

context Canal_de_Television inv:

- ☐ self.generos_admitidos->exists (g | self.programas.corresponde->includes (g))
- ☐ self.generos_admitidos->exists (g | self.programas.corresponde->excludes (g))
- ☐ self.generos_admitidos->exists (g | self.programas.corresponde->includes (self))

Por favor, indique el horario en el cual finaliza el ejercicio:

hora..... minutos ..... segundos .....

.

D.5 A Sample of the Debriefing Questionnaire

This questionnaire was answered for the subjects of the first conducted experiment.

The information you provide in this questionnaire may be very valuable for us. Please answer each question as honestly as you can. Anything you write down will be treated confidentially. Thank you.

Personal details and experience

(1) First Name: ·····
(2) Last Name: ·····
(3) Age: ····· (4) Gender: ·····
(5) Years at the university: ·····
(6) Are you studying in another university or institute?: ·····
(7) Please tick the signatures you have approved?:
System Analysis ·····
Software Development ·····
Software Engineering ·····
System Project Administration ·····
Programming Languages ·····
(8) How many years have you experienced in programing? ·····
(9) Which programming languages have you used? ·····
(10) How many years have you experienced in modelling object-oriented software?
·····
(11) How many years have you experienced in modelling class diagrams?
·····
(12) How many years have you experienced with OCL? ·····
(13) Have you ever used a formal language?: ·····
(14) If you answer 'yes' to the previous question, please indicate which formal language you used. ·····
Any additional comments?
·····
·····
Thanks once again!