Defining and Validating Metrics for UML Statechart Diagrams

Marcela Genero, David Miranda and Mario Piattini
ALARCOS Research Group
Department of Computer Science
University of Castilla - La Mancha
Paseo de la Universidad, 4
13071 - Ciudad Real – SPAIN
dmiranda@proyectos.inf-cr.uclm.es, {marcela.genero, mario.piattini}@uclm.es

Abstract

Maintainability is an increasingly relevant quality aspect in the development of object oriented software systems (OOSS). It is generally accepted that OOSS maintainability is highly dependent on the decisions made early in the development life cycle. Conceptual modelling is an important task of this early development. So that the maintainability of conceptual models have a great influence on the maintainability of the OOSS which is finally implemented. For assessing the conceptual models maintainability it is useful to have quantitative and objective measurement instruments. Conceptual modelling focus on either static aspects or dynamics aspects of the OOSS. Using the Unified Modelling Language (UML) static aspects at conceptual level are mainly represented in structural diagrams such as class diagrams, whilst dynamic aspects are represented in behavioural diagrams such as statechart diagrams, activity diagrams, sequence diagrams and collaboration diagrams. There exists several works related to metrics for structural diagrams such as class diagrams. However, behavioural diagrams have been little studied This fact leaded us to define measures for UML statechart diagrams. The main goal of this paper is to show how we defined those measures in a methodological way, in order to guarantee their validity. We used the DISTANCE framework, based on measurement theory, to define and theoretically validate the measures. In order to gather empirical evidence that the proposed measures could be early maintainability indicators of UML statechart diagrams, we carried out a controlled experiment. The aim of the experiment was to investigate the relationship between the complexity of UML statechart diagrams and their understandability (one maintainability subcharacteristic).

Keywords. Object Oriented Software Systems maintainability**,** UML behavioural diagrams, UML statechart diagrams, complexity metrics, theoretical validation, empirical validation

1. Introduction

Nowadays, maintainability has become one of the most pressing challenge facing OOSS development organisations. It is generally accepted within the OOSS development field that the maintainability of OOSS is highly dependent on decisions made early in the development

life cycle. Conceptual modelling is an important task of this early development. Most of the approaches towards OOSS development (OMT, Catalysis, Rational Unified Process, OPEN, etc.), have considered conceptual modelling as a relevant task. Therefore, the maintainability of conceptual models has a significant impact on the maintainability of OOSS, which is ultimately implemented.

In the development of OOSS the conceptual models represent not only the structure of the OOSS but also their behaviour. Modelling OOSS with UML [23] the static aspects at conceptual level are mainly represented in structural diagrams such as class diagrams, whilst dynamic aspects are represented in behavioural diagrams such as statechart diagrams, activity diagrams, sequence diagrams and collaboration diagrams. For assessing quality aspects of such diagrams it is useful to have quantitative and objective measurement instruments, avoiding thus bias in the quality evaluation process.

There exists several works related to metrics for structural diagrams such as class diagrams [6] [12] [16] [18] [21] [22]. However, there is little reference to metrics for behavioural diagrams such as statechart diagrams in existing literature. One of the first approaches towards the definition for behavioural diagrams can be found in 0, where metrics are applied to statechart diagrams developed with Object Modelling Technique (OMT) [29]. Yacoub et al. [35] propose dynamic metrics for Real-Time Object Modeling (ROOM) [32]. In [27] complexity measures for event-driven object-oriented conceptual models are defined. These proposals of metrics have not gone beyond the definition step. In our knowledge, there is no published works related to their theoretical an empirical validation. Therefore, and as was pointed out in [7] [8] [9] [27] the definition of metrics for diagrams that capture dynamics aspects of OOSS it is an interesting topic for future investigation. This fact motivated us to define metrics for behavioural diagrams, starting with metrics for measuring one critical internal quality attribute such as the complexity of UML statechart diagrams.

Even though our goal is to centre on OOIS maintainability, it is an external quality characteristic that can only be evaluated once a product is finished or nearly finished. So that we centre our work on measuring an internal quality characteristic, the complexity of UML statechart diagrams, and ascertaining trough experimentation if there exist any correlation between those metrics and their understandability (a subcharacteristic of maintainability). We consider that evaluating (and if it is neccesary improving) the complexity of UML statechart

we can contribute to the understandability of UML statechart diagrams and therefore to the understandability of OOSS.

The definition of metrics must be done in a methodological way to assure the reliability of the proposed metrics. According some suggestions provided in [4] and [11] about "how to define valid measures" we have followed a process consisting of three main tasks: measure definition, theoretical validation and empirical validation.

For the definition and the theoretical validation of the measures we follow the DISTANCE framework [26][28], which assures the theoretical validity of the defined measures, i.e., that they measure the attribute they intend to measure. By means of the usage of the DISTANCE framework we can also assure that the proposed metrics are characterised by the ratio scale type, which as Zuse [36] pointed out it is a main concern when defining metrics for software artifacts.

But theoretical validation it is not enough, empirical validation is critical for the success of any measurement activity [1] [15] [20] [30], because by means of empirical validation we can demonstrate with real evidence that the measures we proposed serve the purpose they were defined for and that they could be fruitful in practice. In this paper we show an experiment we carried out in order to evaluate if there is empirical evidence that UML statechart diagram complexity metrics are related to UML statechart diagram understandability.

This paper is organised in the following way: Section 2 presents a proposal of measures for the complexity of UML statechart diagrams. The theoretical validation of the proposed metrics is carried out in section 3. A controlled experiment, for empirically validating the proposed metrics, is presented in section 4. Finally, the last section shows the conclusions and some lines which are still open for further investigation.

2. A proposal of measures for UML statechart diagrams

The complexity of a statechart diagram is determined by the different elements that compose it, such as states, transitions, activities, etc. It is not advisable to define a single measure for the complexity of UML statechart diagrams, since a single measure of complexity cannot capture all possible aspects or viewpoints of complexity [14] instead several measures are needed, each one focusing on a different statechart diagram elements.

Table 1 outlines the set of metrics we propose to measure the complexity of UML statechart diagrams:

Metric Name	Metric definition
NUMBER OF ENTRY ACTIONS (NEntryA)	The total number of entry actions, i.e. the actions performed each time a state is entered.
NUMBER OF EXIT ACTIONS (NExitA)	The total number of exit actions, i.e. the actions performed each time a state is left.
NUMBER OF ACTIVITIES (NA)	The total number of activities (do/activity) in the statechart diagram.
NUMBER OF STATES (NS)	The total number of simple states, considering also the simple states within the composites states.
NUMBER OF TRANSITIONS (NT)	The total number of transitions, considering common transitions (the source and the target states are different), the initial and final transitions, self-transitions (the source and the target state is the same), internal transitions (transitions inside a state that responds to an event but without leaving the state)

Table 1. Metrics for UML statechart diagram complexity.

A relevant property is that the proposed metrics are simple and ease to automate, which are, as Fenton and Neil [16] remark in a recent paper related to the future of software metrics, desirables properties for software metrics.

3. Theoretical validation of the proposed metrics

Firstly, in this section we will introduce the DISTANCE framework [26] [28] and later we will use this framework to theoretically validate the metrics proposed below.

3.1 DISTANCE framework

The DISTANCE framework provides constructive procedures for modelling software attributes and defining the corresponding measures. The different steps of the procedure are inserted into a process model for software measurement that (i) details the required inputs, underlying assumptions and expected results, for each task (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, for instance, GQM [2] [3]. In this section we summarise the procedures for attribute definition and measure construction for ease of reference.

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 characterise and other software entities that serve as reference points or norms for measurement. These distances are then measured by functions that are called "metrics" in mathematics. These are functions that satisfy the metric axioms, i.e. a set of axioms that are necessary and sufficient to define measures of distance [28].

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 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.

The distance-based measure construction process consists of five steps (see table 2). The process is triggered by a request to find or build a measurement instrument for a software attribute *attr* that characterises the software entities in a set P. There might for instance be a request that expresses the need for a measure of some complexity aspect of a statechart diagram.

Table 2 summarises 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 metric 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 2. 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 metric with additive segments [33]. According to the uniqueness theorem associated with the representation theorem for segmentally additive proximity structures the resulting measures are characterised by the ratio scale type. For further details on the measurement theoretical principles underlying the approach we refer to [26] [28].

Before proceeding to the distance-based definition of the metrics for the UML statechart diagrams in the next section, we wish to clarify why we emphasise the theoretical validation of the proposed measures. In our opinion, the validity of the measurement instruments used for the variables of an empirical study is a key factor in the overall study validity. On the one hand, the theoretical validity of the complexity measures is used to claim their construct validity of the empirical studies carried out using those measures On the other hand, knowledge of the scale type of the measures is useful for choosing the appropriate statistical techniques to analyse the data obtained in the studies carried out for the empirical validation of the measures.

3.2. A distance-based definition of complexity measures for statechart diagrams

In this section the distance-based definition of a particular measure NS (number of states) is elaborated in detail. We will follow each of the steps for measure construction proposed in the DISTANCE framework. In order to exemplify the process we will use the statechart diagrams shown in figure 1.

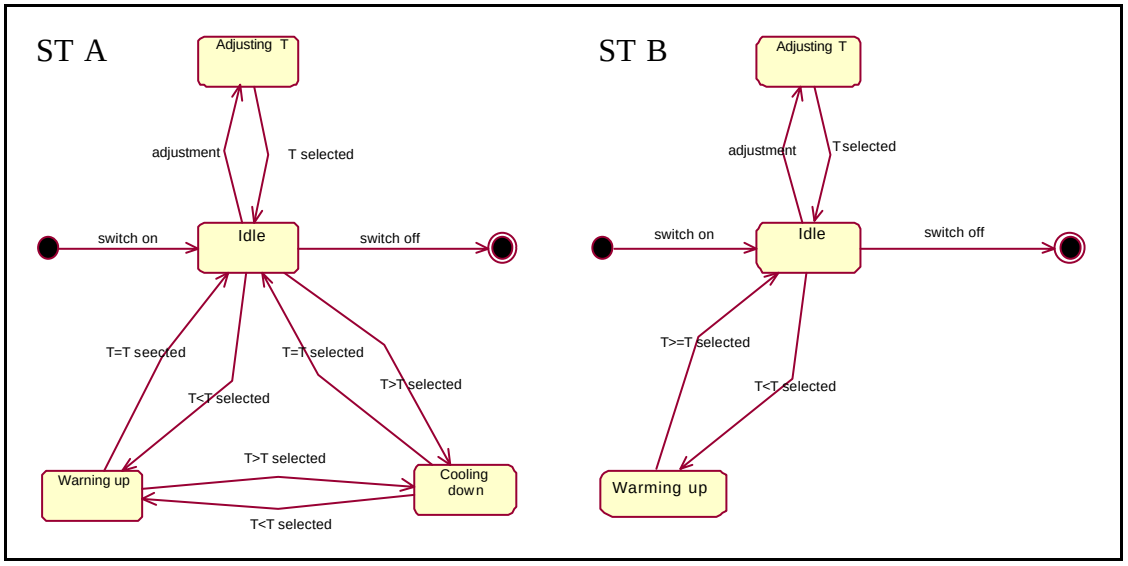

Figure 1. Two examples of SDs (SD A, SD B)

- **Step 1. Find a measurement abstraction.** In our case the set of software entities P is the Universe of SDs (USD) that is relevant for some application domains and p is a SD (i.e. $p \in USD$). The attribute of interest *attr* is the number of simple states, i.e. a particular aspect of SD complexity. Let US be the Universe of States relevant to one application domain. The set of states within an SD, called S(SD) is then a subset of US. All the sets of states within the SDs of USD are elements of the power set of US, denoted by $\wp(US)$. As a consequence we can equate the set of measurement abstractions M to $\wp(US)$ and define the abstraction function as:

  $abs_{NA}: USD \rightarrow \wp(US): SD \rightarrow S(SD)$

  This function simply maps a SD onto its set of states.

  In our example we have the set of states attributes of SD A and of SD B:

  $abs_{NS}(SD\ A) = S(SD\ A) = \{A, B, C\}$

  $abs_{NS}(SD\ B) = S(SD\ B) = \{ABD\}$

- **Step 2. Model distances between measurement abstractions.** The next step is to model distances between the elements of M. We need to find a set of elementary transformation types for the set of measurement abstractions $\wp(US)$ such that any set of simple states can be transformed into any other set of simple states by means of a finite sequence of elementary transformations. Finding such a set is quite easy in the case of a power set. Since the elements of $\wp(US)$ are sets of states, T_e must only contain two types of elementary transformations: one for adding a state to a set and one for removing a state from a set. Given two sets of states $s_1 \in \wp(US)$ and $s_2 \in \wp(US)$, s_1 can always be transformed into s_2 by removing first all states from s_1 that are not in s_2, and then adding all states to s_1 that are in s_2, but were not in the original s_1. In the 'worst case scenario', s_1 must be transformed into s_2 via an empty set of states. Formally, $T_e = \{t_{0\text{-}NS}, t_{1\text{-}NS}\}$, where $t_{0\text{-}NS}$ and $t_{1\text{-}NS}$ are defined as:

  $t_{0\text{-}NS}: \wp(US) \rightarrow \wp(US): s \rightarrow s \cup \{a\}$, with $a \in US$

  $t_{1\text{-}NS}: \wp(US) \rightarrow \wp(US): s \rightarrow s - \{a\}$, with $a \in US$

  In our example, the distance between $abs_{NS}(SD\ A)$ and $abs_{NS}(SD\ B)$ can be modelled by a sequence of elementary transformations that removes the state C from S(SD A) and that adds the simple state D to S(SD A). This sequence of two elementary transformations is sufficient to transform S(SD A) into S(SD B). Other sequences, in which for instance the

order of these two transformations is changed, do, of course, exist and can be used to model the distance in sets of states between SD A and SD B. It is, however, obvious that no sequence can contain fewer than two elementary transformations if it is going to be used as a model of this distance. All 'shortest' sequences of elementary transformations qualify as models of distance.

- **Step 3. Quantify distances between measurement abstractions.** In this step the distances in $\wp(US)$ that can be modelled by applying sequences of elementary transformations of the types contained in T_e, are quantified. A function δ_{NSS} that quantifies these distances is the metric (in the mathematical sense) that is defined by the symmetrical difference model, i.e. a particular instance of the contrast model of Tversky [33]. It has been proven in [26] that the symmetrical difference model can always be used to define a metric when the set of measurement abstractions is a power set.

  $\delta_{NST}: \wp(US) \times \wp(US) \rightarrow \Re: (s, s') \rightarrow |s - s'| + |s' - s|$

  This definition is equivalent to stating that the distance between two sets of states, 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 symmetrical difference model results in a value of 2 for the distance between the set of states of SD A and SD B. Formally,

  $\delta_{NS}(abs_{NS}(\text{SD A}), abs_{NS}(\text{SD B})) =$

  $|\{A, B, C\} - \{A, B, D\}| + |\{A, B, D\} - \{A, B, C\}| = |\{C\}| + |\{D\}| = 2$

- **Step 4. Find a reference abstraction.** In our example the obvious reference point for measurement is the empty set of attributes. It is desirable that a SD without states will have the lowest possible value for the NSS measurement. We therefore define the following function:

  $ref_{NS}: \text{USD} \rightarrow \wp(US): \text{SD} \rightarrow \varnothing$

- **Step 5. Define the software measurement.** In our example, the number of states of a statechart diagram $SD \in USD$ can be defined as the distance between its set of states S(SD) and the empty set of states $\varnothing$, as modelled by any shortest sequence of elementary transformations between S(SD)and $\varnothing$. Hence, the NS measurement can be defined as a

function that returns for any SD $\in$ USD the value of the metric δ_{NS} for the pair of sets S(SD) and $\varnothing$:

$$\forall SD \in USD: NS(SD) = \delta_{NS}(S(SD), \varnothing) = |S(SD) - \varnothing| + |\varnothing - S(SD)| = |S(SD)|$$

As a consequence, a measurement that returns the count of states in a SD qualifies as a number of states measurement. It should be pointed out here that, although this result seems trivial, other measurement theory approaches to software measurement definition cannot be used to guarantee the ratio scale type of the NS measurement. The number of states in a SD cannot for instance be described by means of a modified extensive structure, as advocated in the approach of Zuse [36], which is the best known way to arrive at ratio scales in software measurement.

Due to space constraints we cannot present the measurement construction process for the other measures. However complexity aspects of UML statechart diagrams can be modelled by means of a set abstraction and as a consequence all the measurements take the form of a simple count, so their process construction is analogous followed by the metric NS.

4. Empirical validation of the proposed metrics

In this section we describe an experiment we have carried out to empirically validate the proposed measures as early maintainability indicators. We have followed some suggestions provided in [5] [25] [34]on how to perform controlled experiments and have used (with only minor changes) the format proposed by Wohlin et al. [34] to describe it.

4.1. Definition

Using the GQM (Goal-Question-Metric) template [2] [3] for goal definition, the goal of the experiment is defined as follows:

Analyse	*UML statechart diagrams complexity metrics*
For the purpose of	*Evaluating*
With respect to	*The capability of being used as understandability indicators of the UML statechart diagrams*
From the point of view of	*OOSS designers*
In the context of	*Undergraduate students in the final year of*

Computer Science and teachers of the Area of Software Engineering at the Department of Computer Science in the University of Castilla-La Mancha

4.2. Planning

After the definition of the experiments (why the experiment is conducted), the planning took place. The planning is preparation for how the experiment is conducted and includes the following activities:

- **Context selection.** The context of the experiment is a group related to the area of Software Engineering at the university, and hence the experiment is run-off line (not in an industrial software development environment). The subjects were eight teachers and eleven students. Students are enrolled in the final-year of Computer Science at the Department of Computer Science in the University of Castilla-La Mancha in Spain. All of the teachers belong to the Software Engineering area.

  The experiment is specific since it focuses on UML statechart diagram complexity metrics. The ability to generalise from this specific context is further elaborated below when we discuss threats to the experiment. The experiment addresses a real problem, i.e., which indicators can be used to assess the understandability of UML statechart diagram? To this end it investigates the correlation between metrics and understandability.
- **Selection of subjects.** The subjects are chosen for convenience, i.e. the subjects are students and teachers that have experience in the design and development of OOSS.
- **Variables selection.** The independent variable is UML statechart diagram complexity. The dependent variable is UML statechart diagram understandability.
- **Instrumentation.** The objects used in the experiment were 20 UML statechart diagrams. The independent variable was measured by the metrics, presented in section 2. The dependent variable was measured by the time the subject spent answering the questionnaire attached to each diagram. We called this time "understandability time".
- **Hypothesis formulation.** An important aspect of experiments is to know and to state in a clear and formal way what we intend to evaluate in the experiment. This leads us to the formulation of a hypothesis (or several hypothesis):

- Null hypothesis, H_0: There is no significant correlation between the UML statechart diagrams complexity metrics and understandability time.
- Alternative hypothesis, H_1: There is a significant correlation between the UML statechart diagrams complexity metrics and the understandability time.

– **Experiment design**. We selected a within-subject design experiment, i.e. all the tests had to be solved by each of the subjects. The subjects were given the tests in different order.

4.3. Operation

It is in this phase where measurements are collected including the following activities:

▪ **Preparation.** At the time the experiment was done all of the students had taken two courses on Software Engineering, in which they learnt in depth how to build OOS using UML, and all of the selected teachers had gained enough experience in the design and development of OOS. Moreover, the subjects were given an intensive training session before the experiment took place. However, the subjects were not aware of what aspects we intended to study. Neither they were informed about the actual hypothesis stated.

We prepared the material we handed to the subjects, which consisted of a guide explaining the UML statechart notation and twenty diagrams. These diagrams were related to different universes of discourse, that were easy enough to be understood by each of the subjects. The complexity of each diagram is different, covering a broad range of metrics values (see table 2).

Diagram	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
NentryA	1	1	2	0	3	6	1	1	0	2	1	1	2	1	1	0	2	2	0	0
NexitA	1	0	0	0	2	6	0	0	1	1	2	1	1	1	0	0	0	0	1	0
NA	0	3	2	2	2	0	1	3	4	0	1	1	0	2	4	5	1	1	0	0
NS	3	4	4	4	4	6	5	5	5	4	6	3	2	3	9	9	5	12	2	5
NT	7	7	7	11	13	13	11	13	10	6	17	5	4	8	13	24	8	24	6	12

Table 3. Metrics values for each UML statechart diagram

Each diagram had a test enclosed which includes a questionnaire in order to evaluate if the subjects really understand the content of the UML statechart diagram. Each questionnaire contained exactly the same number of questions (four) and the questions were conceptually similar and in identical order. Each subject had to write at the beginning the time he started answering the questionnaire and at the end the time he finished. The

difference between the two is what we call the understandability time (expressed in seconds).

- **Execution.** The subjects were given all of the material described in the previous paragraph. We explained to them how to carry out the experiment. We allowed them one week to do the experiment, i.e., each subject had to carry out the test alone, and could use unlimited time to solve it. We collected all of the data with the understandability time calculated from the responses of the experiments.
- **Data Validation.** Once the data were collected, we controlled if the test were completed and if the questions have been answered correctly. All the tests were considered valid because all the questions were correctly answered.

4.4. Analysis and Interpretation

Our goal is to ascertain if any relationship exists between each of the proposed metrics (see section 2) and the understandability time. We have used the data collected in order to test the hypotheses formulated in section 3.2.

First, we applied the Kolmogorov-Smirnov test to ascertain if the distribution of the data collected was normal or not. As the data was not normal we decided to use a non parametric test like Spearman's coefficient, with a level of significance α=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).

Using Spearman's correlation coefficient, each of the metrics was correlated separately with maintenance time (see table 4).

	NA	**NS**	**NT**	**NEntryA**	**NExitA**
Undertandability time	0.483	0.500	0.6048	0.2145	0.285

Table 4. Spearman´s correlation coefficients between metrics and understandability time

Analysing the Spearman´s correlation coefficients shown in table 4, we can conclude that it only seems to exist correlation (rejecting hypothesis H_0) between the metrics NA, NS and NT and understandability time. We can deduce this from the fact that only those metrics have a correlation near or greater than 0.5, which is a common threshold to evaluate correlation

values. Another issue that can be pointed out seeing table 4, is that the Number of Transitions (NT) seems to have a greater correlation with the understandability time than the Number of States (NS). Nevertheless, more empirical studies have to be carried out to confirm these findings.

4.5. Validity evaluation

We will discuss the various issues that threaten the validity of the empirical study and the way we attempted to alleviate them:

Threats to Conclusion Validity. The conclusion validity defines the extend to which conclusions are statistically valid. The only issue that could affect the statistical validity of this study is the size of the sample data (380 values, 20 diagrams y 19 subjects), that perhaps are not enough for both parametric and non-parametric static test [4]. We are aware of this, so we will consider the results of the experiment as preliminary findings.

Threats to Construct Validity. The construct validity is the degree to which the independent and the dependent variables are accurately measured by the measurement instruments used in the experiment. The dependent variable we used is the understandability time. We propose objective measures for it: the understandability time, i.e., the time each subject spent answering the questions related to each diagram that it is considered the time they need to understand it. The construct validity of the metrics used for the independent variable is guaranteed by Poels and Dedene framework [26] [27] used for their theoretical validation.

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. The following issues have been dealt with:

- DIFFERENCES AMONG SUBJECTS. Using a within-subjects design, error variance due to differences among subjects is reduced. As [5] remarks in software engineering experiments when dealing with small samples, variations in participant skills are a major concern that is difficult to address fully by randomisation or blocking. In the experiment, teachers and students had the same degree of experience in modelling with UML.
- KNOWLEDGE OF THE UNIVERSE OF DISCOURSE AMONG STATECHART DIAGRAMS. The statechart diagrams were from different universes of discourse but they were general

enough to be easily understood by each of the subjects. In this way, knowledge of the domain does not affect internal validity.

- PRECISION IN THE TIME VALUES. Subjects assumed the responsibility of writing the time they began and finished answering each diagram.
- LEARNING EFFECTS. All the tests in each experiment were put in a different order, to avoid learning effects. Subjects were required and controlled to answer in the order in which the tests appeared. However, we could not control if they really followed the order because they did the tests alone.
- FATIGUE EFFECTS. On average the experiment lasted for less than one hour, so fatigue was not very relevant. Also, the different order of the tests helped to cancel out these effects.
- PERSISTENCE EFFECTS. In order to avoid persistence effects, the experiment was run with subjects who had never done a similar experiment.
- SUBJECT MOTIVATION. All the teachers and students who were involved in this experiment have participated voluntarily, in order to help us in our research.
- OTHER FACTORS. Plagiarism and influence between subjects really could not be controlled. However, before starting the experiment, it was explained to the participants, amongst other things, that they should answer the questionnaire themselves and that the results would not be valid if the questions were discussed whilst the experiment was being carried out.

Threats to External Validity. External validity is the degree to which the research results can be generalised to the population under study and to other research settings. The greater the external validity, the more the results of an empirical study can be generalised to actual software engineering practice. Two threats to validity have been identified which limit the ability to apply any such generalisation:

- MATERIALS AND TASKS USED. In the experiment we tried to use statechart diagrams and tasks which can be representative of real cases, but more empirical studies taking "real cases" from software companies must be done in the future.
- SUBJECTS. To solve the difficulty of obtaining professional subjects, we used teachers and students from advanced software engineering courses. We are aware that more experiments with practitioners and professionals must be carried out in order to be able to

generalise these results. However, in this case, the tasks to be performed do not require high levels of industrial experience, so, experiments with students could be appropriate [1].

4.6. Presentation and Package

As the diffusion of experimental data is important for the external replication [10] of the experiments we have put all of the material of this experiment onto the website http://alarcos.inf-cr.uclm.es.

5. Conclusions and future work

As Scheneidewind [31] remarks, measuring quality aspects, such as maintainability among others, is the key of developing high-quality OOSS. It is widely recognised that the maintainability assurance of OOSS must be guaranteed since the early phases of its life cycle. Behavioural diagrams, such as UML statechart diagrams are a key artifact of the early development, so focusing in their maintainability should contribute to the maintainability of the OOSS which is finally delivered.

With the hypothesis that the complexity of UML statechart diagrams may influence their maintainability, we have defined a set of metrics for the complexity of UML statechart diagrams. Trough a controlled experiment we have partially corroborated that hypothesis. Only the metrics NA, NS and NT seem to be correlated with one maintainability subcharacteristic, the understandability of UML statechart diagrams.

Nevertheless, despite the encouraging results obtained we consider them as preliminaries. We are aware that both internal and external replication of the experiment [10], would be necessary to assess if the presented metrics could be really used as early maintainability indicators. Also, data of "real projects" on UML statechart diagrams maintainability efforts would be useful, as well as time spent on maintenance tasks in order to predict data that can be highly fruitful to software designers and developers. However the scarcity of such data continues to be a great problem we must find other ways to tackle validating metrics. Brito e Abreu et al. [7] [8] [9] suggested the necessity of a public repository of measurement experiences, which we think would be a good step towards the success of all the work done

on software measurement. It will possible to do that when more "real data" on systems developed using UML is available, which is the challenge of most of the researchers in this area.

Once these metrics were empirically validated, our idea is to build understandability prediction models, based on the metrics values. For this end, we plan to use advanced techniques such as Fuzzy Classification and Regression Trees 0 and Fuzzy Prototypical Knowledge Discovery [24], which have been used for prediction purposes in others domains obtaining accurate predictions.

Finally, another research line of interest would be to look at the relationship between the complexity of the UML statechart diagrams and other maintainability sub-characteristics, other than understandability, such as modifiability and analysability [19].

Acknowledgements

This work is supported by a FPI grant from the Spanish Science and Technology Ministry. This research is part of the DOLMEN project supported by CICYT (TIC 2000-1673-C06-06).

References

[1] Basili V., Shull F. and Lanubille F. (1999). Building Knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4), 456-473

[2] Basili V. and Weiss D. (1984). A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, 10.

[3] Basili V. and Rombach H. (1988). The TAME project: towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6), 728-738.

[4] Briand L., El Emam K. and Morasca S. (1995). Theoretical and empirical validation of software product measures. *Technical report ISERN-95-03*, International Software Engineering Research Network.

[5] Briand L., Arisholm S., Counsell F., Houdek F. and Thévenod-Fosse P. (2000). Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions. *Empirical Software Engineering*.

[6] Brito e Abreu F. and Carapuça R. (1994). Object-Oriented Software Engineering: Measuring and controlling the development process. *4th Int Conference on Software Quality*.

[7] Brito e Abreu, F., Zuse H., Sahraoui H. and Melo W. (1999). Quantitative Approaches in Object-Oriented Software Engineering. *ECOOP'99 Workshops Reader*, LNCS 1743, A. Moreira and S. Demeyer (eds), Springer-Verlag, 326-337.

[8] Brito e Abreu F., Poels G., Sahraoui H. and Zuse H. (2000). Quantitative Approaches in Object-Oriented Software Engineering. *ECOOP´2000 Workshop Reader*, LNCS 1964, Springer-Verlag.

[9] Brito e Abreu F., Henderson-Sellers B., Piattini M., Poels G. and Sahraoui H. (2001). Quantitative Approaches in Object-Oriented Software Engineering. *ECOOP´01 Workshop Reader*, LNCS, Springer-Verlag. (to appear).

[10] Brooks A., Daly J., Miller J., Roper M. Wood M. (1996). Replication of experimental results in software engineering. *Technical report ISERN-96-10*, International Software Engineering Research Network.

[11] Calero C., Piattini M. and Genero M. (2001). Empirical validation of referential integrity metrics. *Information and Software Technology*, 43, 949-957.

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

Delgado M., Gómez Skarmeta A. and Jiménez L. (2001). A regression methodology to induce a fuzzy model. *International Journal of Intelligent Systems*, 16(2), 169-190.

[13] Derr K. (1995). *Applying OMT*. SIGS Books. Prentice Hall. New York.

[14] Fenton N. (1994). Software Measurement: A Necessary Scientific Basis. *IEEE Transactions on Software Engineering*. 20(3), 199-206.

[15] Fenton N. and Pfleeger S. (1997). *Software Metrics: A Rigorous Approach*. 2nd. edition. London, Chapman & Hall.

[16] Fenton N. and Neil M. (2000). *Software Metrics: a Roadmap. Future of Software Engineering*. Anthony Finkelstein Ed., ACM, 359-370.

[17] Genero M., Piattini M. and Calero C. (2000). Early Measures For UML class diagrams. *L´Objet*. 6(4), Hermes Science Publications, 489-515.

[18] Genero M. (2002). Defining and Validating Metrics for Conceptual Models. *Ph.D. thesis*, University of Castilla-La Mancha.

[19] ISO 9126 (1999). Software Product Evaluation-Quality Characteristics and Guidelines for their Use. *ISO/IEC Standard 9126*. Geneva.

[20] Kitchenham B., Pflegger S. and Fenton N. (1995). Towards a Framework for Software Measurement Validation. *IEEE Transactions of Software Engineering*, 21(12), 929-943.

[21] Lorenz M. and Kidd J. (1994). *Object-Oriented Software Metrics: A Practical Guide*. Prentice Hall, Englewood Cliffs, New Jersey.

[22] Marchesi M. (1998). OOA Metrics for the Unified Modeling Language. *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering*, 67-73.

[23] Object Management Group (1999). *UML Revision Task Force, OMG Unified Modeling Language Specification, v. 1.3*, document ad/99-06-08.

[24] Olivas J. and Romero F. (2000). FPKD. Fuzzy Prototypical Knowledge Discovery. Application to Forest Fire Prediction. *Proceedings of the SEKE'2000, Knowledge Systems Institute*, Chicago, Ill. USA, 47-54.

[25] Perry D., Porter A. and Votta L. (2000). *Empirical Studies of Software Engineering: A Roadmap. Future of Software Engineering*. Anthony Finkelstein Ed., ACM, 345-355.

[26] Poels G. and Dedene G. (1999). DISTANCE: A Framework for Software Measure Construction. *Research Report DTEW9937*, Dept. Applied Economics, Katholieke Universiteit Leuven, Belgium, 46 p. (submitted for publication).

[27] Poels G. and Dedene G. (2000a). Measures for Assessing Dynamic Complexity Aspects of Object-Oriented Conceptual Schemes. *19th International Conference on Conceptual Modelling (ER 2000)*. Salt Lake City, USA, 499-512.

[28] Poels G. and Dedene G. (2000b). Distance-Based software measurement: necessary and sufficient properties for software measures. *Information and Software Technology*, (42), 35-46.

[29] Rumbaugh J., Blaha M., Premerlani W., Eddy F. and Lorensen W. (1991). *Object-Oriented Modelling and Design* .Prentice Hall, USA.

[30] Schneidewind N. (1992). Methodology For Validating Software Metrics. *IEEE Transactions of Software Engineering*, 18 (5), 410-422.

[31] Schneidewind N. (2002). Body of Knowledge for Software Quality Measurement. *IEEE Computer*, 35(2), 77-83 .

[32] Selic B., Gullekson G., and Ward P. (1994). *Real-Time Object Oriented Modelling,*. John Wiley & Sons, Inc.

[33] Suppes P., Krantz M., Luce R. and Tversky A. (1989). *Foundations of Measurement Vol. 2*. Academic Press, New York.

[34] Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B. and Wesslén A. (2000). *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers.

[35] Yacoub S., Ammar H. and Robinson T. (1998). Dynamic Metrics for Object Oriented Designs. *Proceedings of the Sixth IEEE International Symposium on Software Metrics*.

[36] Zuse H. (1998). *A Framework of Software Measurement*. Berlin, Walter de Gruyter, (1998).