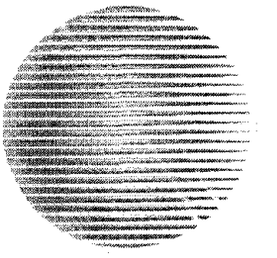
JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2005; **17**:1–22
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr.312

Research

Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation



Marcela Genero Bocco[1,*,†], Daniel L. Moody[2,3] and Mario Piattini[1]

[1]*Department of Computer Science, Universidad de Castilla-La Mancha, Ciudad Real, Spain*
[2]*Department of Computer Science, University of Iceland, Reykjavik, Iceland*
[3]*Department of Cybernetics, Czech Technical University (CVUT), Prague, Czech Republic*

SUMMARY

The complexity of software artifacts is widely believed to be an important determinant of maintenance effort. This paper conducts an experimental analysis of the impact of complexity on the maintenance of the Unified Modeling Language (UML) class diagrams. This represents an analysis of the effect of an internal quality attribute on an external quality attribute. A range of complexity metrics are proposed based on an ontological analysis of the UML language and previous research. The relative influence of these metrics on maintenance effort is then evaluated using a laboratory experiment. A within-subjects design was used, with subjects required to modify a range of UML class diagrams with different levels of complexity. Only two of the metrics emerged as significant determinants of maintenance effort: number of methods and number of associations. Together these explain around 28% of the variation in maintenance effort. While these findings are encouraging, further research is necessary to explore the ability of these metrics to predict maintenance effort. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: complexity; Unified Modeling Language (UML); class diagram; object-oriented (OO) modeling; experimental research; software quality

*Correspondence to: Marcela Genero Bocco, Paseo de la Universidad, Nº 4, 13071 Ciudad Real, Spain.
†E-mail: marcela.genero@uclm.es

Contract/grant sponsor: Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha; contract/grant number: PCC-03-003-1
Contract/grant sponsor: Dirección General de Investigación del Ministerio de Ciencia y Tecnologia; contract/grant number: TIC2003-07804-C05-03

Received 22 April 2004
Revised 16 February 2005
Accepted 17 February 2005

1. INTRODUCTION

1.1. Maintainability of information systems

The adaptability of a system to changes in its environment is considered to be one of a system's most important characteristics [1,2]. In practice, maintenance represents the major source of work for software organizations [3]. It is estimated that 61% of the professional life of programmers is devoted to maintenance [4]. Also, maintenance costs represent two thirds of the total lifetime costs of information systems in commercial mainframe environments [5]. For this reason, *maintainability* is an important issue in the design of information systems. However in practice, it is often given relatively little attention in the interests of completing development projects 'on time, on budget'. The result is that information systems often have substantially reduced operational lifetimes, and are sometimes obsolete before they are delivered [6,7].

Maintainability is generally accepted to be an important aspect of software quality, and is one of the six software quality characteristics defined in the international standard for software quality: ISO/IEC 9126 [8]. Maintainability is defined as 'the capability of the software to be modified' and consists of four subcharacteristics:

- *analyzability*, the capability of the software to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified;
- *changeability*, the capability of the software product to enable a specified modification to be implemented;
- *stability*, the capability of the software to minimize unexpected effects from modifications of the software; and
- *testability*, the capability of the software to enable modified software to be validated.

Maintainability is an external quality characteristic, as it relates to the external behavior of the product [9]. However, internal quality characteristics (characteristics of the product which are not visible to the user) are the key to satisfying external quality characteristics. Internal quality characteristics define the 'hows' by which external quality characteristics or 'whats' will be met [10]. Understanding the relationship between internal and external quality characteristics is essential for developing software which satisfies user needs [11].

1.2. The effect of complexity on maintenance effort

Software complexity has long been recognized as a determinant of maintenance effort [12–20]. There are several reasons why complexity affects maintenance effort.

- Complex systems are more difficult to understand, and understanding a system is a prerequisite for making changes to it [21,22]. Empirical studies show that humans have a strictly limited capacity for processing information [23–25]. Once the level of complexity exceeds this limit, a state of *information overload* ensues and comprehension degrades rapidly [26–29]. Both field and experimental studies show that information overload results in reduced performance on cognitive tasks [24,29]. This relates to the *analyzability* subcharacteristic in ISO/IEC 9126 [8].
- More complex systems will tend to have higher *entropy* because of the number of interacting components, which increases their volatility and susceptibility to change [30,31]. This increases

the number of changes required, thus contributing to maintenance effort. This relates to the *stability* subcharacteristic in ISO/IEC 9126 [8].

- As systems increase in complexity, the impact of changes becomes more complex and unpredictable, which increases the effort required to implement changes [32–34]. Changes to one part of the system are more likely to affect many other parts of the system, often referred to as the 'ripple effect' [2,32,35]. This relates to the *changeability* subcharacteristic in ISO/IEC 9126 [8].

1.3. Objectives of this research

Most previous research on maintainability [16,17,36–38] has focused on the maintainability of the final system (i.e., code-level maintenance). Our focus is different in the sense that we are interested in evaluating maintainability from the early stages of software development. For that reason, this paper investigates the relationship between complexity (an internal quality characteristic) and maintainability (an external quality characteristic). The specific focus of this research is on the Unified Modeling Language (UML) *static structure models*, commonly called *class diagrams* [39]. We evaluate the effect of a set of metrics for measuring complexity of UML class diagrams [40,41] on maintenance effort through a controlled experiment.

We performed a previous controlled experiment [42], pursuing a similar objective. In that experiment, as in this, the independent variable was UML class diagram complexity. In the previous experiment the dependent variables were three maintainability subcharacteristics (understandability, analyzability and modifiability) measured by means of subjective ratings on a seven-point scale. Even though the results obtained in the previous experiment (through correlation analysis) reflect that the metrics proposed were related to class diagram maintainability, we are aware that the dependent measures were subjective and therefore less reliable. Therefore, we decided to carry out another experiment measuring the dependent variable in a more objective way, which is the experiment presented in this paper.

The outline of the paper is as follows:

- Section 2 defines a set of candidate metrics for measuring complexity of UML class diagrams;
- Section 3 describes the experimental design used to evaluate the metrics;
- Section 4 presents the results of the experiment and evaluates their validity; and
- Section 5 summarizes the findings and discusses the main contributions of the research.

2. METRICS FOR MEASURING COMPLEXITY OF UML CLASS DIAGRAMS

This section defines a set of metrics for measuring complexity of UML class diagrams.

2.1. UML class diagrams

UML defines an industry standard approach to modeling object-oriented (OO) systems [39]. So far there has been very little research into the maintainability of UML models. However, UML is usually used in a *model-driven* development approach, so changes to systems should be driven by changes to

the underlying models. Class diagrams define the architecture of an OO system—the objects and their behavior—and represent the starting point for all changes to an OO system.

2.2. Metrics for UML class diagrams

A set of candidate metrics was developed for measuring the complexity of UML class diagrams. These have been published in [41,42], but are summarized here to provide context for understanding the experiment. The metrics were developed based on:

- an analysis of the ontological structure of UML class diagrams;
- a review of the OO metrics literature.

2.2.1. *Ontological analysis*

According to general systems theory [21,31], the complexity of a system is determined by the *number and variety of elements and the number and variety of relationships between them.* In developing measures of complexity for UML class diagrams, a natural starting point therefore is to define the types of elements and relationships such diagrams may contain: the *ontological structure* of the notation [2]. At the requirements level, a UML class diagram may consist of the following constructs [39].

- Classes, which consist of:
  - attributes (with their name, type and visibility scope);
  - methods (with their name, list of parameters, return type, visibility and scope).
- Relationships, which may be of four types:
  - association;
  - aggregation: this includes both composition relationships and part-whole relationships as defined in UML 1.4;
  - generalization;
  - dependency.

Potential measures of UML class diagram complexity include counts of each construct. This results in seven (simple) measures of complexity.

M1. Number of classes: the number of classes in a class diagram. This metric corresponds to OA1 in [43], DSC (Design Size in Classes) in [44] and C_s in [45]. This is the most common measure of class diagram complexity used in the literature.

M2. Number of methods: the total number of methods defined in a class diagram, including those defined at class and instance level, but not including inherited methods (this would lead to double counting). This is a modification of the WMC (Weighted Method per Class) metric [46], in which: (a) the weightings of all methods are 1; and (b) the value is totaled across all classes (as WMC is a class-level metric). Weightings are not applicable at the requirements stage because the code for methods is not available. This also corresponds to the NOM (Number of Methods) metric in [17,44] and the combination of NIM (Number of Instance Methods) and NCM (Number of Class Methods) in [47] (although all of these metrics are defined at the class level).

M3. Number of attributes: the total number of attributes defined in a class diagram, including those defined at class and instance level, but not including inherited attributes or attributes defined within methods. This is a generalization of the NOA (Number of Attributes) metric [45] to the class diagram level.
M4. Number of associations: the number of association relationships in a class diagram. This is a generalization of the NAS (Number of Associations) metric [48] to the class diagram level.
M5. Number of aggregations: the number of aggregation relationships in a class diagram.
M6. Number of dependencies: the number of dependency relationships in a class diagram.
M7. Number of generalizations: the number of generalization relationships in a class diagram (each parent–child pair in a generalization relationship).

Together these metrics enable measurement of 'the number and variety of elements and the number and variety of relationships between them' in a UML class diagram, as required by our definition of complexity.

2.2.2. *Review of OO metrics literature*

A thorough review of the OO metrics literature was conducted in order to identify additional metrics for measuring the complexity of UML class diagrams at the requirements stage. Most research on OO software measurement has focused on the measurement of code and detailed design artifacts [9,45,49,50]. However, there are now a significant number of proposals for metrics which can be applied at the analysis level [17,43–48,51–55]. Of these, most are defined at the class level, but some can be generalized to the class diagram level. A number of proposals evaluate the complexity of the UML notation itself (*meta-level* complexity), so are not applicable in this research [53–55]. Based on these previous proposals, we identified four additional metrics.

M8. Number of generalization hierarchies (NGenH): the number of distinct generalization hierarchies in a class diagram. This corresponds to the OA2 metric in [43] and the NOH metric in [44].
M9. Number of aggregation hierarchies (NAggH): the number of distinct aggregation hierarchies in a class diagram. This is a generalization of M8 to aggregation hierarchies.
M10. Maximum depth of inheritance (MaxDIT): the maximum DIT value for all classes in a class diagram. The DIT value for a class in a generalization hierarchy is defined as the length of the longest path from the class to the root of the hierarchy. This is a generalization of the DIT metric in [45] to the class diagram level.
M11. Maximum height of aggregation (MaxHAgg): the maximum HAgg value for all classes in a class diagram. The HAgg value for a class in an aggregation hierarchy is defined as the length of the longest path from the class to the leaves. This is a generalization of M10 to aggregation hierarchies.

All of the metrics (M1–M11) satisfy the ISO/IEC 9126 required properties for measures used for comparison, as they are objective, empirical and repeatable [8]. They are also simple, which Fenton and Neil [56] argue is a desirable property for software metrics in general. Another issue is that the proposed metrics could also allow designers and conceptual modelers to make better decisions when they are modeling class diagrams. In particular, they allow selection among several class diagram alternatives with equivalent semantic content. Moreover, they could be useful for predicting, for example, the

maintenance effort needed for modifying UML class diagrams. That potential needs to be investigated empirically and is the main objective of the current study reported here.

2.2.3. *Predicted impact of the metrics on maintenance effort*

The metrics M1–M7 (which represent counts of individual UML constructs) are expected to increase maintenance effort, as adding a construct of any type will increase overall complexity of the class diagram. However, some types of elements are likely to affect complexity more than others e.g., adding a new class is likely to increase complexity more than adding a new attribute. Thus, we would expect overall complexity of a class diagram to be a *weighted sum* of these measures.

- We predict that the number of classes (M1) will have the greatest effect on maintenance effort. The reason for this is that the number of classes corresponds to the number of distinct concepts or 'things' in the model. Based on theories of human cognition, increasing the number of concepts will increase the effort required to understand the model, which in turn will increase maintenance effort [23–25,57,58]. This is also consistent with software estimation models, in which the number of entities (which correspond to classes in an OO environment) is the most influential determinant of software size [59–61].
- We predict that the metrics corresponding to the number of relationships (M4–M7) will be the next most influential. According to *cognitive load theory*, cognitive load is increased by the level of interactivity among elements (the number of relationships among concepts) [62,63]. Increasing cognitive load reduces understanding and is therefore likely to increase maintenance effort.
- The metrics M2 and M3 provide measures of the internal complexity of classes. One would expect a class diagram with high internal complexity (many methods and attributes per class) to be more difficult to understand than a model with the same number of classes and relationships (external or system complexity) but lower internal complexity. However, this is expected to have a weaker impact than external complexity.

Because generalization hierarchies and aggregation hierarchies define subsystems within the class diagram, we predict that the metrics M8 and M9 will *reduce* complexity and therefore reduce maintenance effort. Given two models with the same number of classes, we would expect one with more generalization and/or aggregation hierarchies to be easier to understand, because hierarchical structures facilitate 'chunking' and thus reduce load on working memory [23,24,64].

Finally, we predict that M10 and M11 will increase maintenance effort. The rationale for this is that human classification hierarchies tend to be limited in depth, usually to around three levels [27]. Thus, very deep hierarchies may overtax powers of human information processing [65,66]. The literature has shown conflicting results for the effects of inheritance hierarchies on maintenance: for example, Basili *et al.* [67] found that the larger the DIT value, the greater the probability of fault detection, Daly *et al.* [68] found that systems with three levels of inheritance had lower maintenance time than those with no inheritance and Cartwright [69] found that three levels of inheritance reduced maintenance time. On the other hand, Unger and Prechelt [70] found that inheritance depth had no effect on maintenance effort, Harrison *et al.* [71] found that systems with no inheritance are easier to modify than systems with three or five levels, while Poels and Dedene [72] found that extensive use of inheritance leads to models that are more difficult to modify.

In summary, we predict that the metrics will influence maintenance effort in ascending order of influence:

- M1 (corresponding to the number of elements or concepts);
- M4, M5, M6, M7 (corresponding to the number of relationships or interactivity);
- M2, M3 (corresponding to the internal structure of elements);
- M10, M11 (corresponding to depth of hierarchies);
- M8, M9 (expected to reduce complexity and therefore maintenance effort).

We are conscious that these predictions must be empirically evaluated to extract final conclusions.

3. RESEARCH METHODOLOGY

3.1. Research objectives

The purpose in generating the initial set of metrics was to be as complete as possible—to try to measure all possible aspects of UML class diagram complexity. However, some of these metrics are likely to be more influential than others in determining maintenance effort. The purpose of this research is to empirically evaluate the relative influence of all of these metrics on maintenance effort and use this to build a prediction model. The validity of each of the predictions stated in Section 2.2.3 (related to the order of influence) is out of the scope of the paper and will be pending for further research.

3.2. Experimental design

The experimental design is summarized in Figure 1. A *within-subjects* design was used, in which each participant was given nine UML class diagrams of different levels of complexity. Participants were required to make a number of prescribed changes to each model and to record how long it took them to complete the task.

3.3. Participants

There were 30 participants in the experiment: 20 final-year Computer Science students and 10 Software Engineering academics at the University of Castilla-La Mancha. All participants had prior experience using UML to develop software. The student participants had all completed at least two software engineering courses in UML. All participants were also given an intensive 'refresher course' in UML class diagrams prior to the experiment.

3.4. Materials

Participants were given a guide summarizing UML notation and nine UML class diagrams representing different application domains. The experimental models had different levels of complexity, covering a broad range of metric values (see Table I). Each experimental model included a brief description of the underlying problem domain and an associated maintenance task. An example of one of the class diagrams used (CD5) and maintenance task is included in Appendix A (all experimental materials

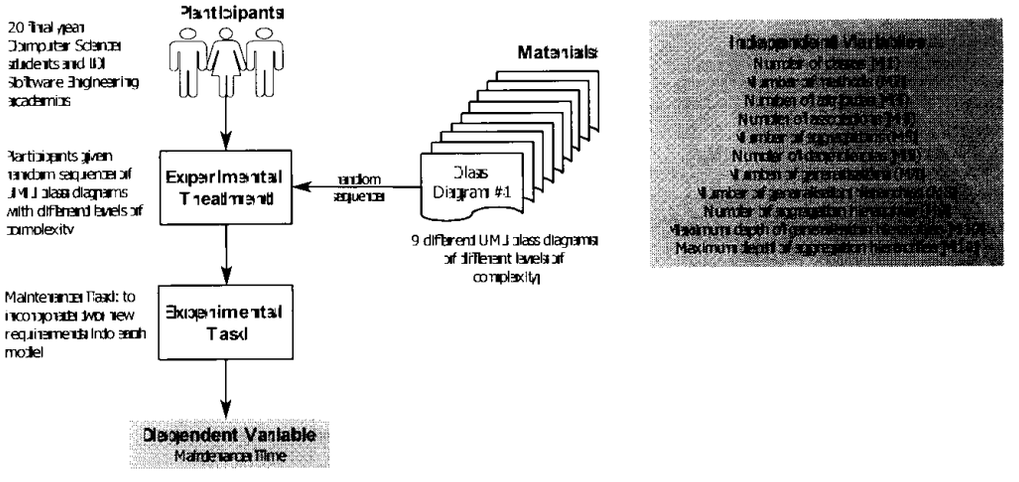


Figure 1. Summary of experimental design.

Table I. Complexity metrics for the experimental models.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
CD1	7	11	22	1	0	0	5	1	0	2	0
CD2	8	12	31	1	6	0	1	1	1	1	2
CD3	3	17	24	2	0	0	0	0	0	0	0
CD4	10	12	21	15	3	0	0	0	2	0	1
CD5	9	19	29	3	3	0	3	1	3	2	1
CD6	7	16	7	6	0	0	0	0	0	0	0
CD7	23	33	66	4	5	2	16	3	2	3	3
CD8	20	30	65	6	5	0	14	3	4	3	2
CD9	23	65	80	20	3	2	3	1	3	2	3
AVG	12.22	23.89	38.33	6.44	2.78	0.44	4.67	1.11	1.67	1.44	1.33
STD DEV	7.63	17.3	25.27	6.65	2.33	0.88	6.12	1.17	1.5	1.24	1.22

are available upon request to the corresponding author). The experimental models were given to participants in random order, to control for learning effects.

3.5. Experimental task

Each experimental model had an associated experimental task, consisting of two new requirements to be incorporated into the model. The activities that the subjects had to perform fit into the category

of *enhancive maintenance*, according to the taxonomy of software maintenance activities proposed by Chapin *et al.* [73], as they involved adding new functionality to the system. The tasks defined for each class diagram were all of equivalent complexity (in terms of number of diagram elements added, modified or deleted), so the only source of variation in effort to perform the tasks should be model complexity. An example of one of the experimental tasks is included in Appendix A. Each subject had to modify the class diagrams according to the specifications and to record the time taken for each task—they were asked to write down their start time (to the nearest second) prior to beginning the task, and then to write down their finish time at the end. Each subject was allowed an elapsed calendar time of one week to carry out the experimental task.

3.6. Independent variable

The independent variable was the complexity of the experimental models. This was measured by the metrics previously defined—these represent empirical indicators of the underlying theoretical construct. While there is only one *latent* independent variable (complexity), there are 11 *observed* independent variables (M1–M11). The *levels* of the independent variable are embodied in the different experimental class diagrams: each model represents a different combination of metric values (Table I).

3.7. Dependent variable

The dependent variable in this experiment was maintenance effort, which was defined as 'the effort required by a participant to make the prescribed changes to an experimental model'. This was measured by the time taken to perform the experimental task, expressed in seconds. This includes the time to comprehend the class diagram, to analyze what changes were required and to complete them. The dependent variable thus incorporates the subcharacteristics of *analyzability* and *changeability* defined in ISO/IEC 9126 [8]. Time taken is an objective measure of the effort required to perform the experimental task, and is one of the most commonly used measures of effort in performing cognitive tasks [27].

3.8. Summary of theoretical model

Figure 2 summarizes the theoretical model tested by this experiment.

- Circles indicate latent variables (theoretical constructs) while rectangles indicate observed or measured variables (empirical indicators).
- The solid lines indicate *causal relationships* hypothesized between latent variables. The direction of the arrows indicates direction of causation.
- The dotted lines indicate *measurement relationships* between latent variables and their empirical indicators. The direction of the arrows indicates whether they are reflective or formative. The $\ominus$ symbol indicates an inverse indicator (where the measure is inversely related to the underlying construct).

As shown in Figure 2, we hypothesize a causal relationship between complexity of UML class diagrams and maintenance effort. The metrics defined in Section 2 are all *formative* measures of

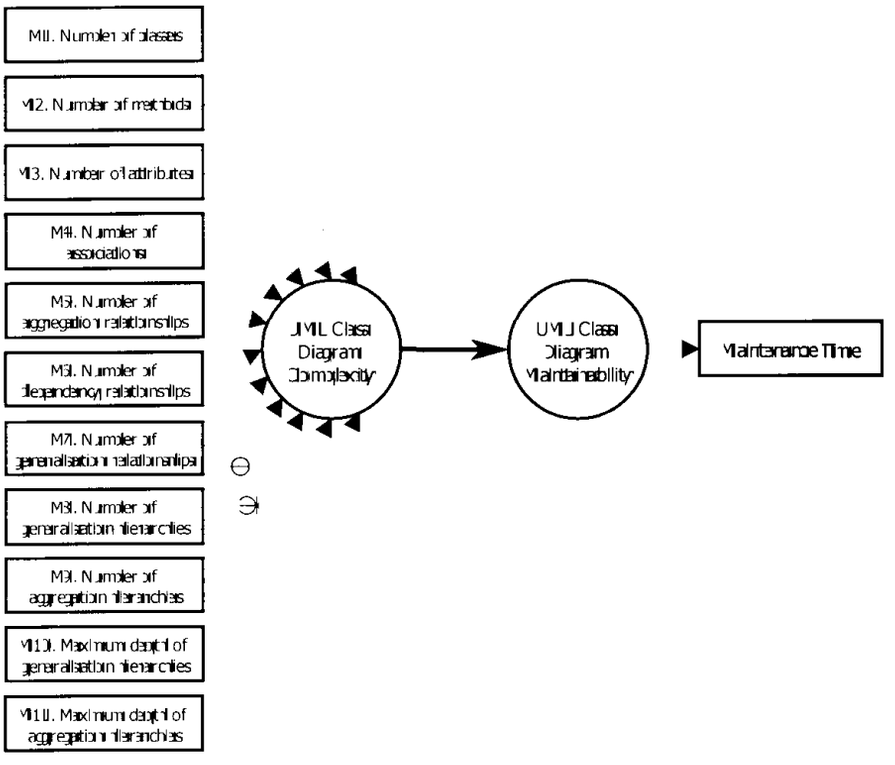


Figure 2. The *a priori* theoretical model.

complexity, while maintenance time is a reflective measure of effort. The purpose of this research is to analyze the effects of different aspects or dimensions of complexity (represented by the different metrics) on maintenance effort.

3.9. Hypotheses

We want to investigate whether the complexity metrics proposed can be used as predictors of maintenance effort, so we formulate the following null and alternative hypotheses.

- H_0: There will be no relationship between the complexity metrics (M1–M11) and maintenance time.
- H_1: There will be positive relationships between (M1–M7, M10, M11) and maintenance time and negative relationships between (M8, M9) and maintenance time.

For simplicity, we express these as single hypothesis, but in reality they incorporate 11 distinct null and alternative hypotheses as shown in Figure 2.

4. RESULTS

This section begins with the data collection and continues with the analysis the empirical data and the interpretation of the analysis results.

4.1. Data collection

Data were collected by calculating the value of the metrics for each experimental data model and entering start and finish times for each experimental task. Metrics were calculated automatically using a tool, while maintenance time was calculated from start and finish times using a spreadsheet. The experiment resulted in a total of 270 data points (30 participants $\times$ 9 experimental tasks) for the dependent variable, which formed the basis for the data analysis.

4.2. Statistical significance versus practical meaningfulness

In interpreting empirical results, it is important to distinguish between statistical significance and practical meaningfulness of results [74,75]. *Statistical significance* measures whether the observed relationship between variables is likely to be a 'real' relationship or if it is likely to be the result of random variation. Statistical significance is measured by alpha (α), which represents the probability that the result could have occurred by chance.

Effect size has been suggested by a number of researchers as an index of practical meaningfulness [74,76–78]. In studies involving relationships between variables, effect size is most commonly measured by the standardized regression coefficient (β). In terms of deciding whether an effect is 'meaningful' or not, the most frequently used guidelines are those proposed by Cohen [74]:

- $|\beta| < 0.1$, not meaningful;
- $|\beta| \geq 0.1$, small;
- $|\beta| \geq 0.3$, medium;
- $|\beta| \geq 0.5$, large.

4.3. Construct validity

The metrics defined in Section 2.2 are proposed as measures of a single underlying construct: UML class diagram complexity. Construct validity is one of the most important constructs in measurement and refers to the level of confidence that empirical indicators reflect the construct they purport to measure [75]. Factor analysis is the preferred technique among researchers for evaluating construct validity. However in this case, the sample size was too small, so *inter-item correlation analysis* was used instead. Out of 55 inter-item correlations between all metrics, 49 were significant at the 0.01 level, two more were significant at the 0.05 level and four were non-significant. Convergent validity (measured as the average correlation of each metric with the other metrics) for nine of the 11 metrics

was large (> 0.5) and medium for the remaining two ($r > 0.3$). This provides strong evidence that they are all measuring the same underlying construct: each of the metrics measures a different dimension of the construct.

The construct validity of the metrics used for the independent variable is guaranteed by the DISTANCE framework [79] which was used for their theoretical validation [40].

4.4. Regression analysis

Multivariate regression analysis is a technique for analyzing causal relationships between continuous variables. All of the independent and dependent variables in this experiment are continuous variables so regression analysis is an appropriate technique for analyzing the relationships between them. The result of regression analysis is a prediction model relating the independent and dependent variables.

Stepwise regression analysis was used to evaluate the relative influence of the metrics in determining maintenance effort. This is a statistical technique used in exploratory research to discover the relationship between variables when a range of different causal factors have been hypothesized [75]. This is an appropriate technique to use here as the research is exploratory. Stepwise regression analysis may be carried out in the following two ways [80].

- Top down: start by including the most highly correlated variable in the regression equation and then adding the variable with the highest partial correlation until there is no significant change in r^2.
- Bottom up: start by including all variables in the regression equation and then removing the least influential variable in each stage.

In this paper, we use top down analysis, but both methods lead to the same results.

4.4.1. Regression analysis stage 1

Bivariate correlation analysis was first carried out between all observed independent variables (the complexity metrics) and the dependent variable (maintenance time). All correlations were found to be statistically significant except for number of dependencies (M6). The number of methods (M2) metric emerged as the most influential variable, with a correlation coefficient of 0.499. Regression analysis was therefore carried out using M2 as the sole independent variable and maintenance time as the dependent variable. The results of the analysis were:

- $r^2 = 0.249$;
- $p = 0.000$;
- $\beta = 0.499$.

The r^2 statistic shows that M2 explains almost 25% of the variation in maintenance time. The standardized regression coefficient is 0.499, which represents a medium effect [74]. The regression equation which results is

$$\text{Maintenance Time} = 2.07 + (0.05 \times \text{Number of Methods}) \quad (1)$$

Table II. Separate effects of metrics on maintenance time.

Metric	Significance (p)	Effect size (β)
Number of methods (M2)	0.000	0.388
Number of associations (M4)	0.002	0.217

4.4.2. *Regression analysis stage 2*

In the second stage of the analysis, the metric with the highest partial correlation (while controlling for M2) was entered into the regression model. Partial correlation analysis revealed that number of associations (M4) was the only metric which had a significant partial correlation with Maintenance Time. Most of the remaining metrics were negatively correlated with the dependent variable after controlling for M2. M4 was therefore entered into the regression model. The results of the analysis were:

- $r^2 = 0.277$;
- $p = 0.000$;
- $\beta = 0.533$.

This shows that number of methods (M2) and number of associations (M4) together explain around 28% of the variation in maintenance effort, which is a significant increase in the predictive power of the model. The standardized regression coefficient is 0.533, which represents a large effect [74]. The regression equation which results is

$$\text{Maintenance Time} = 1.94 + (0.04 \times \text{Number of Methods}) + (0.09 \times \text{Number of Associations}) \quad (2)$$

The coefficients in the regression equation measure the 'raw' effect of each independent variable on the dependent variable. The regression coefficient for M2 is 0.04, which means that each additional method increases maintenance time by 2.4 seconds or 1%. The regression coefficient for M4 is 0.09, which means that each additional association increases maintenance time by 5.4 seconds or 2.27%.

The separate effects of each independent variable on Maintenance Time are shown in Table II. Both metrics were found to be significant predictors of maintenance time ($\alpha < 0.01$), with M2 having a medium effect and M4 having a small effect.

While each association has a larger incremental effect on maintenance time than each method (as measured by the regression coefficient), the much larger number of methods (on average, there were about six times as many methods as associations in the experimental models) means that they have a much greater effect on maintenance time overall (as measured by the effect size). While the regression coefficient for M2 is less than half the size of that for M4, the standardized regression coefficient (β) is almost twice the size.

4.4.3. *Regression analysis stage 3*

In the third stage of analysis, partial correlation analysis was carried out on the remaining metrics to see if there was a significant correlation with maintenance time after controlling for M2 and M4.

None of the remaining metrics had significant effects on maintenance time, nor were they even close (p values were all greater than 0.6). This provides strong evidence that the stepwise regression analysis is complete, and that the regression model defined in (2) is fully specified.

4.5. Validity analysis

In this section we analyze the internal, external and statistical validity of the experiment.

4.5.1. Internal validity

The following variables were controlled as part of the experiment.

- Participant characteristics: use of a within-subjects design minimizes threats to internal validity by equalizing participant characteristics between groups.
- Task complexity: the experimental tasks for all experimental models were equivalent in complexity.
- Instrumentation: the same measurement techniques were used for independent and dependent variables for all participants. The risk of measurement error was reduced by calculating metric values automatically.
- Training: all participants were given the same amount of prior training.
- Learning effects: experimental models were given to subjects in random order to minimize learning and sequence effects.

However, we identified two possible threats to the internal validity of the study.

- Control of environment: one threat to the validity of the experiment is that it was not conducted under controlled conditions. Participants were allowed to do the experimental tasks in their own time, which means that environmental factors (e.g., time of day, location) were not explicitly controlled. This was done because of the number of separate tasks involved: it would be quite onerous for subjects to complete all nine experimental tasks in a single laboratory session, and may have led to fatigue, which could have adversely affected the results. We argue that because of the within-subjects design, these effects would be randomly distributed across levels of the independent variable, and therefore do not represent a serious threat to the conclusions of the study.
- Measurement error: another threat to internal validity is the fact that subjects were responsible for recording the time it took to perform the experimental tasks. This increases risk of measurement error for the dependent variable, as subjects may have recorded the time inaccurately. However, again, the within-subjects design means that the error should be randomly distributed across levels of the independent variable, so does not represent a serious threat to the findings. If anything, this would reduce the significance of the regression results by introducing random error, so the results most likely underestimate the strength of the relationship between complexity and maintenance effort.

4.5.2. *External validity*

In this study, external validity is a much greater issue than internal validity. We identify three possible threats to the external validity of this study.

- Sample population: a clear threat to the generalizability of the findings was the choice of experimental subjects. In general, the population from which one selects subjects for the experiment should be representative of the population to which the researcher wishes to generalize results [81]. In this study, we used a sample population of students and academics, but wish to generalize the results to practice. Most previous experimental studies of conceptual modeling have used Computer Science or Information Systems students as proxies for practitioners, so this study has no greater external validity problems than most (e.g., [64,82–90]). According to the literature, causal relationships are more generalizable across populations than values of variables [75,81]. We can therefore be reasonably confident that the causal relationships found between the complexity metrics and maintenance effort will also hold in the target population, although the level of performance (i.e., time taken to perform the tasks) is likely to be significantly different.
- Experimental models: all of the class diagrams used in this experiment were created by the researchers specifically for the purpose of the experiment. As a result, they may not be representative of models found in practice in terms of distribution of metric values. In particular, the experimental models were much less complex than models that would be found in practice, where models frequently consist of 100 or more classes. This raises questions about the *scalability* of the results.
- Experimental task: a final weakness of this experiment was the artificiality of the experimental task. Participants were required to make two rather simple changes to each experimental model—in practice, maintenance changes would be much more complex. It is difficult to realistically simulate the task of maintaining UML models in a laboratory environment and this would be better tested in a real-world maintenance environment.

4.5.3. *Statistical validity*

There are a number of assumptions underlying the linear regression model that need to be satisfied for the results of regression analysis to be statistically valid. However, in many if not most reports where linear regression analysis is used, these assumptions are not even addressed [75].

- *Linearity*. Inspection of the scatterplots of both independent variables in the final regression model (M2 and M4) against Maintenance Time showed no evidence of curvilinearity.
- *Random error*. Correlation coefficients were calculated between independent variables and unstandardized residuals in each stage of the regression analysis and found to be 0.000.
- *Residual normality*. Shapiro–Wilk tests of normality were carried out on unstandardized residuals in each stage of the regression analysis, and they were found to be normally distributed.
- *Homoscedasticity*. Heteroscedasticity exists when the variance of residuals increases or decreases with values of the independent variable. Inspection of scatterplots showed that the variance of residuals was randomly distributed in all cases.

27. Eysenck M, Keane M. *Cognitive Psychology: A Student's Handbook*. Lawrence Erlbaum Associates: Mahwah NJ, 2000; 540.
28. Jacoby J, Speller D, Kohn C. Brand choice as a function of information load. *Journal of Applied Psychology* 1978; **63**:83–91.
29. Lipowski Z. Sensory and information inputs overload: Behavioural effects. *Comprehensive Psychiatry* 1975; **16**(3):105–124.
30. Heales J. Factors affecting information system volatility. *Proceedings 21st International Conference on Information Systems (ICIS'2000)*, Ang S, Kremar H, Orlikowski WJ, Weill P, DeGross JI (eds.). Association for Information Systems: Atlanta GA, 2000; 1–3.
31. Klir G, Elias D. *Architecture of Systems Problem Solving*. Plenum: New York NY, 2003; 349.
32. Arthur LJ. *Measuring Programmer Productivity and Software Quality*. Wiley: New York NY, 1985; 292.
33. Brooks FP. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley: Reading MA, 1975; 195.
34. Meyer B. *Object Oriented Software Construction*. Prentice-Hall: Nyack NY, 1997; 1296.
35. Alexander C. *Notes on the Synthesis of Form*. Harvard University Press: Boston MA, 1968; 224.
36. Oman P, Hagemeister J. Metrics for assessing a software system's maintainability. *Proceedings Conference on Software Maintenance*. IEEE Computer Society Press: Los Alamitos CA, 1992; 337–344.
37. Welker KD, Oman PW. Software maintainability metrics models in practice. *Crosstalk* 1995; **8**(11):19–23, 32.
38. Welker KD, Oman PW, Atkinson G. Development and application of an automated source code maintainability index. *Journal of Software Maintenance* 1997; **9**(3):127–159.
39. OMG. *Unified Modeling Language (UML) Specification, Version 1.5*. Object Management Group: Needham MA, 2001; 736.
40. Genero M. Defining and validating metrics for conceptual models. *Doctoral Dissertation*, University of Castilla-La Mancha, Ciudad Real, Spain, 2002; 437.
41. Genero M, Piattini M, Calero C. Early measures for UML class diagrams. *L'Objet* 2000; **6**(4):489–515.
42. Genero M, Olivas J, Piattini M, Romero F. Using metrics to predict OO information systems maintainability. *CAISE 2001 (Lecture Notes in Computer Science*, vol. 2068). Springer: Berlin, 2001; 388–401.
43. Marchesi M. OOA metrics for the Unified Modeling Language. *Proceedings 2nd Euromicro Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press: Los Alamitos CA, 1998; 67–73.
44. Bansiya J, Davis C. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering* 2002; **28**(1):4–17.
45. Henderson-Sellers B. *Object-Oriented Metrics—Measures of Complexity*. Prentice-Hall: Englewood Cliffs NJ, 1996; 252.
46. Chidamber S, Kemerer C. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 1994; **20**(6):476–493.
47. Lorenz M, Kidd J. *Object-Oriented Software Metrics: A Practical Guide*. Prentice-Hall: Englewood Cliffs NJ, 1994; 146.
48. Harrison R, Counsell S, Nithi R. Coupling metrics for object-oriented design. *Proceedings 5th International Symposium on Software Metrics*. IEEE Computer Society Press: Los Alamitos CA, 1998; 150–157.
49. Zuse H. *A Framework of Software Measurement*. Walter de Gruyter: Berlin, 1997; 775.
50. Etzkorn LH, Bansiya J, Davis C. Design and code complexity metrics for OO classes. *The Journal of Object-Oriented Programming* 1999; **12**(1):335–340.
51. Briand L, Daly J, Wüst J. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering* 1999; **25**(1):91–121.
52. Brito e Abreu F, Carapuça R. Object-oriented software engineering: Measuring and controlling the development process. *Proceedings 4th International Conference on Software Quality*. Amercian Society for Quality (ASQ): Milwaukee WI, 1994; 212–224.
53. Siau K, Cao Q. Unified Modeling Language—a complexity analysis. *Journal of Database Management* 2001; **12**(1):26–34.
54. Siau K, Erickson J, Lee L. Complexity of UML: Theoretical versus practical complexity. *Proceedings 12th Workshop on Information Technology and Systems (WITS'02)*. Association of Information Systems: Barcelona, 2002; 13–18.
55. Siau K, Tian Y. The complexity of Unified Modeling Language: A GOMS analysis. *Proceedings 14th International Conference on Information Systems (ICIS'01)*. Association for Information Systems (AIS): Atlanta GA, 2001; 443–448.
56. Fenton N, Neil M. Software metrics: A roadmap. *Future of Software Engineering*, Finkelstein A (ed.). ACM Press: New York NY, 2000; 359–370.
57. Moody D. Complexity effects on end user understanding of data models: An experimental comparison of large data model representation methods. *Proceedings of the 10th European Conference on Information Systems (ECIS'2002)*. University of Gdañsk: Gdañsk, Poland, 2002; 53–58.
58. Shanks CG, Moody DL, Nuredini J, Tobin D, Weber RA. Representing things and properties in conceptual modelling: An empirical evaluation. *Proceedings of the 11th European Conference on Information Systems (ECIS 2003)*. Seconda Università degli Studi di Napoli: Naples, 2003; 112–124.

Marcela Genero Bocco

De: "John Wiley & Sons, Ltd." <alistair.smith@sunrise-setting.co.uk>
Para: <marcela.genero@uclm.es>
CC: "SMRedNC2" <NedChapin@acm.org>; "SMR proofs" <smrproofs@wiley.co.uk>; <dm@hi.is>; <mario.piattini@uclm.es>
Enviado: martes, 29 de marzo de 2005 17:04
Adjuntar: smr312.pdf; Offprint order.doc
Asunto: Your proofs of SMR312

Dear Author

Your proofs of SMR312

Please find attached proofs of your article with instructions on how to return your corrections and an editable offprint order form.

Do you want to be alerted when your paper is published online? Simply register for Contents Alerts and automatically receive the table of contents of Journal of Software Maintenance and Evolution: Research and Practice. Just select e-mail alert on the journal home page http://www.interscience.wiley.com/jpages/1532-060X

If you have any queries please contact Sunrise Setting Ltd.

Thank you in advance for your co-operation.

Kind regards,

Tim Williams

SMR Project Manager

Sunrise Setting Ltd
12a Fore Street
St Marychurch
Torquay
Devon
TQ1 4NE
UK

Phone: +44 (0) 1803 322635
Fax: +44 (0) 1803 323565
E-mail: smr@sunrise-setting.co.uk