Traceability and Factorization in Class Diagrams: an Experimentation of their Correlation

Cam-Ngan Tran[1] and Michel Dao[2]

[1] France Télécom R&D DTL/TAL
2 avenue Pierre Marzin
22307 Lannion Cedex – France

[2] France Télécom R&D DTL/TAL
38-40, rue du général Leclerc
92794 Issy Moulineaux Cedex 9 – France

Abstract. In this article, we present a study of the correlation between factorization and the quality criterion of traceability. Our work is based on a set of new factorization metrics and a specific definition of traceability. The results of our experiment show a good correlation between the increase of the factorization of a UML class diagram and its traceability.

1 Introduction

A quality model defines a methodology and a set of measures in order to evaluate the quality of a software system (source code, analysis or conception models, etc.) [1, 2]. MACAO project[3] has specified its own quality model, based on existing ones [3, 4]. It uses, besides "classical" object-oriented metrics [5, 6], a body of specific metrics dedicated to the measurement of factorization[4] in UML class diagrams [7]. In our quality model, the results of the computation of the metrics on a given UML class diagram are combined in order to obtain an evaluation of several quality criteria such as encapsulation, consistency, testability, etc.

Assessing that a quality model actually measures what it is supposed to measure is an essential task in order to ensure confidence in its results. This article presents a study of a limited aspect of the validation of MACAO project quality model, namely the correlation between the degree of factorization of a UML class diagram and the evaluation of the traceability criterion.

This article is organized as follows:

In section 2 we briefly present quality model validation and explain the validation we have carried out, which is based on a set of factorization metrics presented in section

[3] MACAO is a common project of France Telecom R&D, SOFTEAM and LIRMM (CNRS and University Montpellier 2.) supported by the french national network on software technologies RNTL

[4] We define factorization as the amount of features (attributes and methods) that are shared through inheritance. The more features are shared, the better the factorization.

Tran C. and Dao M. (2004).
Traceability and Factorization in Class Diagrams: an Experimentation of their Correlation.
In *Proceedings of the 1st International Workshop on Software Audits and Metrics*, pages 38-48
DOI: 10.5220/0002673500380048

3 and on a specification of traceability specified in section 4. We then analyze the results of our study in section 5 before drawing some conclusions.

2 Quality model validation

Validation of quality models may be roughly divided into two types of activities [8, 9]:

- the validation that the metrics used in the model actually measure what they claim to measure. This activity is known as “internal validation”;
- the determination of correlations between an external quality criterion (such as traceability) and an internal measurement. This activity is known as “external validation”.

We have chosen to study the set of metrics specific to factorization [7] defined in MACAO project and we have made a limited internal validation of thoses metrics [10]. Our internal validation was focused on 11 properties extracted from litterature: *non-negativity*, *null value*, *simplicity*, *non-coarseness*, *non-uniqueness*, *minimum factorisation*, *maximum factorisation*, *monotonicity*, *non-equivalent of interaction*, *tracking property*, *importance of implementation* and one property specific to factorization: *redundancy*. This internal validation is studied and discussed in [10].

Regarding external validation, which is the subject of this article, we have carried out a study of the correlation between factorization measured by our set of metrics and the traceability criterion.

Our aim during this study was to (in)validate the following assumption:

> *The increase of factorization of a UML class diagram leads to an increase of the traceability criterion*

The problem regarding the evaluation of such a correlation is that one must measure traceability in a way that is not directly based on the factorization. We propose a definition and a means to compute traceability in section 4 that we believe to be a good indicator of tracebility’s estimation and to be independent (in its definition) of factorization.

3 Factorization measurement

Our set of factorization metrics is based on our previous work on class hierarchy restructuration [11]. We have designed and implemented several algorithms that reconstruct class hierarchies in order to maximize factorization. Those algorithms use a structure known as Galois sub-hierarchy which is a restriction of Galois lattice. The Galois sub-hierarchy provides an organization of classes in a hierarchy where features are maximally factorized (they are introduced by only one class and inherited by the others), thus improving generalization and the number of classes introduced is minimal.

The metrics we have defined measure the “distance” between the existing class hierarchy and the “ideally factorized” hierarchy that results from the application of one of our algorithms. Metrics are defined at different levels: feature (attribute and method),

generic feature (set of semantically close features such as all methods with the same name), class and hierarchy.

As our algorithms reorganize a class hierarchy so that factorization metrics are optimal, we used reorganized hierarchies as a benchmark in our study.

4 Traceability

Traceability is a key issue to guarantee consistency among software artifacts of the development cycle phases. Together with changeability, modifiability and stability, traceability is a quality criterion closely tied to the software maintenance. The improvement of those quality criteria will reduce maintenance cost. Their assessment relies, to a large extent, on software evolution and change impact analysis and is the subject of numerous researches.

Changeability, as "the capacity of the software product to enable a specified modification to be implemented", was studied by [12]. The author showed the existence of a correlation between coupling and changeability and the absence of a significant correlation between changeability and cohesion.

Modifiability, which is similar to changeability is defined by [13] as "the facility with which a system can be adapted to the changes in the functional specification, the environment or the expression of the requirements". This work is in connection with the architecture of the system. It argues that modifiability has to be taken into account in earlier phases of the systems development life cycle.

Stability is also a criterion on which maintainability depends. It is defined as "the capacity of the software product to avoid the unexpected effects of the modifications of the software". Studies on stability are mainly based on the comparison of various versions of the software [14]. These studies allow to determine the most unstable components.

4.1 Traceability concept

The traceability quality criterion deals with the software modifications which consist in updating and/or expanding a system during the implementation or maintenance phases. It is classified in two categories [15]:

- *internal traceability* shows the dependency between objects within a development phase. It is based on tangible links of the same abstraction level. These tangible links make the internal traceability easier to compute than the external traceability. In software design, the internal traceability can be divided into three main types [16]: object to object, association to association and use case to object;
- *external traceability* concerns relations between objects in different phases of development cycle. For example, it could be the relationship between specifications and testing. External traceability is ensured by cognitive links. These links, by which objects are connected to others objecs at different abstraction levels, are determined by an expert who is familiar with the system. Thus, the external traceability links must be manually created, based on the expertise and the intuition of the designers [17].

In this study, we consider traceability as the facility to find elements impacted by a modification. A number of research studies deals with code traceability [17–19] and there are various ongoing works on external design traceability. However, just few works still focus on internal design traceability.

Our work addresses internal traceability by examining links between elements within a class hierarchy (features, links and classes). In this way, we deal only with the modification of *class diagrams*.

The traceability issue regarding class diagrams is to discover which classes, or at the lower level, which features will be affected when a modification occurs. On the one hand, traceability does indicate the dependency level of an element in a class diagram. On the other hand, the dependency level of the element reveals a certain facility of tracing in the class diagram. Therefore, internal traceability is closely connected to complexity and comprehensibility and other quality criteria as stated above. For example, if a diagram is too complex or too difficult to understand and to analyze, modifications will be more difficult to trace. In other words, traceability provide designers with an indicator predicting the consequences of a modification within a class diagram.

4.2 Definitions

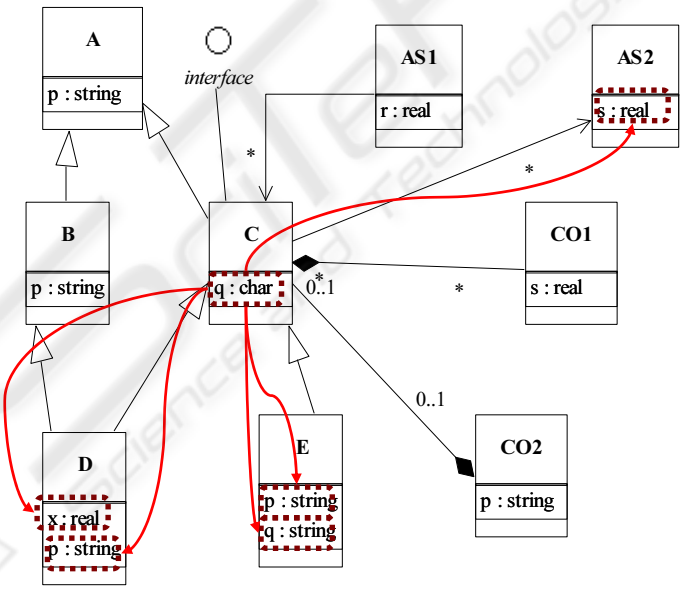

Fig. 1. Trace and impacted elements of a modification.

Before presenting the calculation method of the traceability concept, we present hereunder some terms and definitions used in this article.

Modified Element: is a class or a feature where a modification occurs. A *modification* could be :

1. an *internal modification* such as a modification of an element within a class (for example: feature redefinition, feature addition/deletion, etc.);
2. an *external modification* such as a modification of a relation between classes (for example : class redefinition/addition/deletion or links addition/deletion etc.).

Impacted Element: is a class or a feature that must be adapted after implementing a modification.

In figure 1, a modification of feature *q* in class *C* demands adaptations of features *x* and *p* in class *D*, *p* and *q* in class *E* and *s* in class *AS2*. So, features *x*, *p*, *q*, *s* are defined as impacted elements by a modification of feature *q*.

If *x/A* denotes *¨the feature x of the class A¨* and *SET_ELEM_IMP(X)* is defined as the set of all impacted elements of *X*, then *SET_ELEM_IMP(q/C)* will be {*x/D;p/D; p/E;q/E;s/AS2*}.

Dependency: is a link between a "*modified element*" and an "*impacted element*". We say that there exists a dependency between them.

4.3 Impact analysis

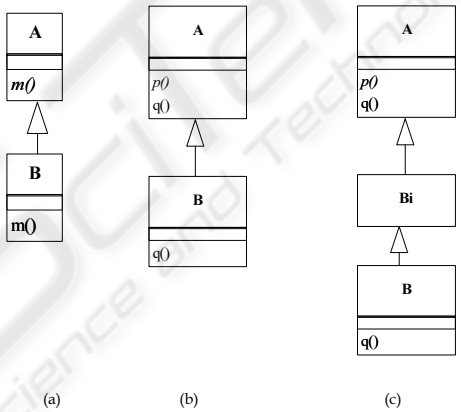

Fig. 2. Dependency in an inheritance relationship.

Firstly, our analysis is based on the dependency analysis during which we list down all the possible cases which are called *dependency rules*. 15 dependency rules were defined in our project.

Hereunder is an example of the dependency rule *"features of a child class depend potentially on all the features of its super class"* illustrated in figure 2. In *a)*, feature *m* of child class *B* depends on feature *m* of super class *A*. In *b)* and *c)* feature *q* of *B*

depends on features *p* and *q* of *A*. There exists no explicit dependency of a feature of *A* on any feature of *B*.

Secondly, we used modification scenarios. Three groups of modification scenarios were delimited:

- manipulation of features (addition, deletion, redefinition);
- manipulation of links (association link, inheritance link, etc.);
- manipulation of classes (addition, deletion, redefinition). The manipulation of features and links may be a sub-group of classes manipulation.

The two last groups of scenarios are based on the first one. According to the dependency rules, we found that:

- features of the same class are impacted;
- features of the children classes are impacted;
- features of the associated classes (direct and indirect) are impacted. (Based on the dependency rules defined for the association links, and taking into account the navigability, the composition, aggregation, etc.);
- features of the dependent classes are impacted.

For example, in figure 3, the features impacted by a suppression or an addition of the feature "*nom:string*" of class *Utilisateur* are:

- all features of class *Utilisateur*
- all features of class *Chauffeur*
- all features of class *TypeVehicule*
- all features of class *Vehicule*
- all features of class *Agence*

Altogether, 8 features are impacted by a suppression, an addition or a modification of feature *nom:string* of class *Utilisateur*. This impact covers all the possible scenarios for the feature.

4.4 Computation method

Two kinds of measures are used to quantify a quality criterion: direct or indirect measures. *Direct measures* are the measures of an attribute which does not depend on the measures of another attribute. *Indirect measures* are measures which must be calculated by other measures.

We chose to use direct measures in our project because of the following. Several existing methods use checklists or interviews to evaluate a quality criterion. Such methods are based on a cognitive system where information is extracted from the experiences and expertise parameters. These methods require a significant number of tests, samples and human resources. The context of our project did not enable us to carry out the external validation by such methods. Thus, we tried to find internal measures of the class diagrams to estimate objectively their traceability. Moreover, computation of these internal measures may be more easily automated.

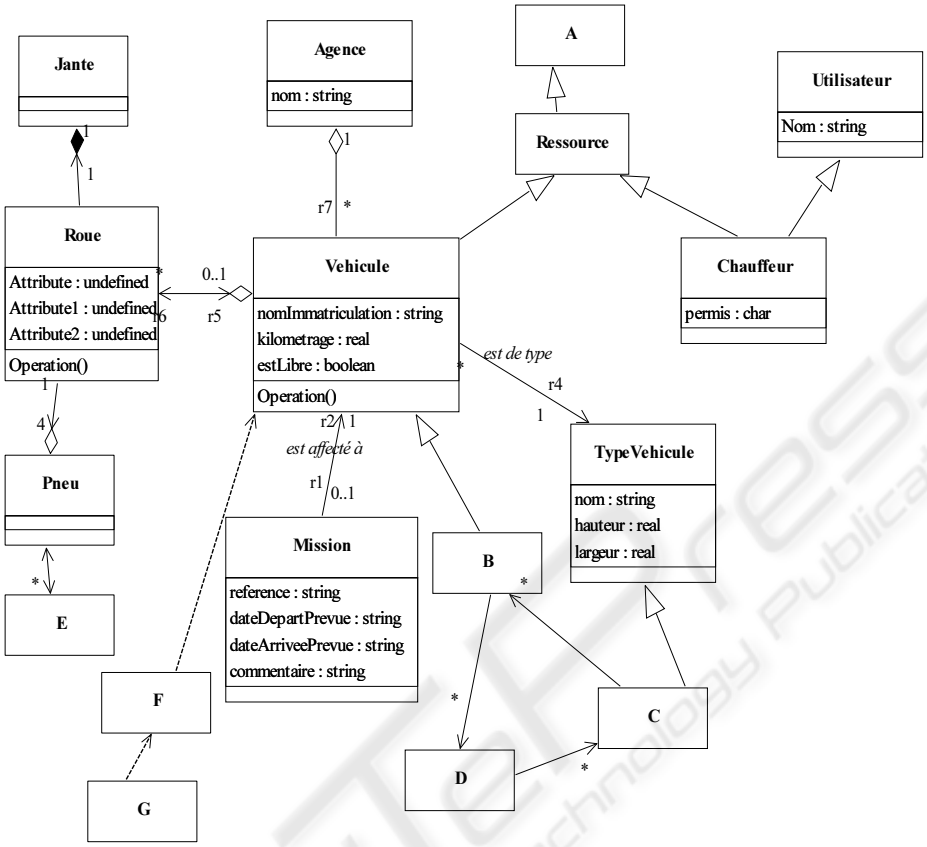

Fig. 3. Example of impact analysis in a class diagram.

The distance between the modified elements and the elements impacted was used to calculate the traceability. The distance from element *X* to *Y* is calculated by all the elements we have to pass from *X* to Y (figure 4). The relationship between the traceability and the distances is: *the longer the distance, the more difficult to trace and therefore the worse the traceability*.

If there exists more than one track between modified element and impacted element, the average of distances will be computed.

To calculate the degree of traceability for a given feature, we look for a set of impacted elements of this features. This set includes all the features in the impacted classes and all the empty impacted classes (an empty class is a class without feature). The distance between this feature and all impacted elements will be added together. The traceability is the inverse of the distance. If the distance is equal to *0*, the traceability is equal to *1*. This means that if the set of impacted elements is *null*, there is no effort to trace when a modification is implemented to this feature.

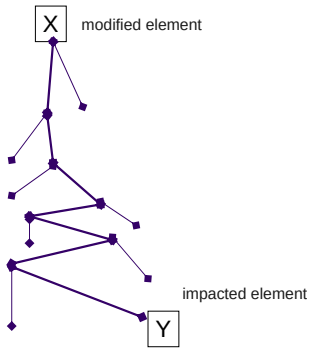

Fig. 4. Distance trace between modified element and impacted element.

The traceability of a class will be the sum of the traceabilities of all its features plus the reverse of the distance between this class with other empty impacted classes.

Suppose that *X* is an modified element and *SET_ELEM_IMP(X)* contains all the impacted elements of *X*. We can resume our computation of the traceability of *X* like following:

Find SET_ELEM_IMP(X)=$Y_1..Y_n$
Initiate the variable distance DIS=0
For each i
    Calculate DIS(X,Y_i)
    DIS=DIS+DIS(X,Y_i)
EndFor
If DIS==0 *then*
    TRA(X)=1
else
    TRA(X)=1/DIS
EndIf

5 Analysis of experiment results

The traceability measurement is implemented in the UML CASE tool Objecteering in J language. J is the Objecteering internal object-oriented language that allows to implement new behaviors on the UML meta-model. The module created allows to find the impacted elements and to compute the distance and the traceability accordingly.

5.1 Test models

To analyze the correlation between traceability and factorization, we used one of the algorithms of factorization of class hierarchies (see section 3). This algorithm transforms a hierarchy into an optimally factorized hierarchy which comprises new classes allowing to factorize features. As a result, the factorization metrics of the elements after

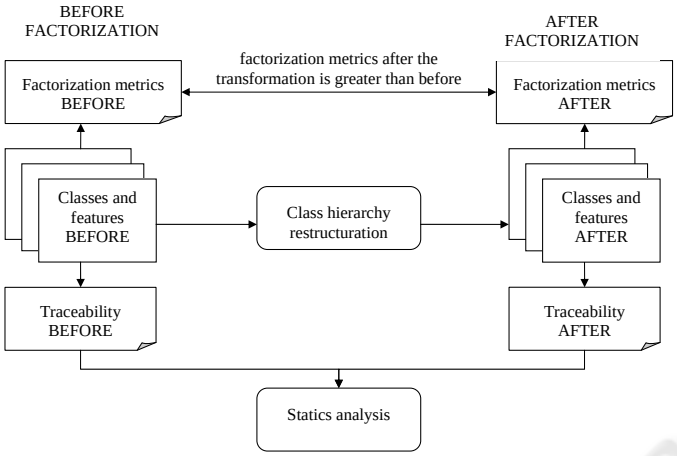

Fig. 5. Test protocol.

the transformation are necessarily better than before, so a computation of factorization metrics is not needed.

We use a method combining the replicated and synthetic methods [20]. The computation was realized on three projects with 100 classes and 264 properties. If a significant change between the traceability before and after the transformation is found, we will be able to identify a correlation between the traceability and the factorization of the elements in the class hierarchy.

5.2 Experiment results

The results of the tests summarize as follows:

For classes:

1. Only one class among 100 has a worse traceability after the restructuring and the reduction for this class is -0,0032 whereas the average increase is 0,1457. The overall results are:
   - 88% of classes have a better traceability after the transformation;
   - 11% of classes have the same traceability after the transformation;
   - 1% of classes have a worse traceability after the transformation.
2. the average of the traceability degree before the transformation is 0,1023 and the average of the traceability's degree after the transformation is 0,4051 (4 times more).

For features

1. Only 5 features out of 264 features have a worse traceability after the transformation and the average reduction for these five features is -0,0052 where as the average increase is 0,1077. The overall results are:
   - 95% of features have a better traceability after the transformation;
   - 2% of features have the same traceability after the transformation;

   - 3% of features have a worse traceability after the transformation.
2. the average of the traceability before the transformation is 0,0560 and the average of the traceability after the transformation is 0,1638 (3 times enhancement).

As a result, this experiment shows that, on our examples, it is easier to trace modifications within UML class hierarchies reorganized by one of our algorithms. Therefore, applying the algorithm of reorganization, allows to increase tracebility, thereby making maintenance easier.

6 Conclusion

In this article, we have presented a limited experiment of the validation of the correlation between the factorization of a UML class hierarchy and the traceability within this hierarchy. We have proposed a definition of the concept of traceability that relies on the fact that the majority of the impacts of a modification in a UML class hierarchy may be found using the different relations between the classes of the hierarchy (inheritance and associations). We have implemented an algorithm to compute this traceability and compared the results of this computation on class hiearchies before and after their restructuration by an algorithm that optimizes factorization. On those examples, we could observe a definite improvement of traceability once the restructuration of the class hierarchy has been carried out.

This work should be extended in two directions. Firstly, we should realize other experiments in order to confirm the positive results of this study. Ideally, we should be able to have enough numerical results in order to make a statistical analysis. Secondly, there exists many more quality criteria on which an external validation should be performed in order to validate our entire quality model.

References

1. IEEE1061: Software Quality Metrics Methodology. IEEE Standard. 1061-1992 edn. Institute of Electrical and Electronics Engineers, Inc., New York (1998)
2. ISO9126: Information Technology - Software Product Evaluation - Software Quality Characteristics and Metrics. International Organization for Standardization, Geneva, Switzerland (1998)
3. Bansiya, J., Davis, C.: A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering **28** (2002) 4–17
4. Dromey, R.G.: A Model For Software Product Quality. IEEE Transactions On Software Engineering **21** (1995) 146 –162
5. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented Design. IEE Transactions on software engineering **20** (1994) 476–493
6. Lorenz, M., Kidd, J.: Object-Oriented Software Metrics, a Practical Guide. Prentice Hall (1994)
7. Dao, M., Huchard, M., Leblanc, H., Libourel, T., Roume, C.: A New Approach to factorization – Introducing Metrics. In: Proc. Metrics 2002 – 8th International Software Metrics Symposium, Ottawa, Canada (2002)
8. Fenton, N.E.: Software Metrics: A Rigorous Approach. Chapman & Hall, London (1992)

9. Zuse, H.: Software Complexity: Measures and Methods. 1st edn. Walter de Gruyter, Berlin (1991)
10. Rasheed, T.M., Tran, C.N., Dao, M.: Validation théorique des métriques de factorisation (Internal Validation of Factorization Metrics). MACAO project report 5.2.1, France Télécom R&D (2003)
11. Dao, M., Huchard, M., Libourel, T., Roume, C.: Evaluating and Optimizing Factorization in Inheritance Hierarchies. In Black, A.P., Ernst, E., Grogono, P., Sakkinen, M., eds.: The Inheritance Workshop at ECOOP 2002, University of Jyvāskylā (2002) 38 – 43
12. Kabaili, H.: Changeability of Object-Oriented Software Systems: Architectural Properties and Quality Indicators. Doctoral thesis, Universit de Montréal, Montréal, Qubec, Canada (2002)
13. Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H.: Architecture-Level Modifiability Analysis (ALMA). Journal of Systems and Software **69** (2003) 129 – 147
14. Sahraoui, H.A., Grosser, D., Valtchev, P.: Predicting software stability using Case-Based Reasoning. In: Proceedings of the 17th IEEE International Conference on Automated Software Engienering (ASE '02). (2002)
15. Lanubile, F., Visaggio, G.: Decision-driven maintenance. Journal of Software Maintenance: Research and Practice **7** (1995) 91 – 115
16. Lindvall, M., Sandahl, K.: Practical Implications of Traceability. Software Practice and Experience **26** (1996) 1161 – 1180
17. D. Kung, J. Gao, P.H., Wen, F.: Change Impact Identification in Object Oriented Software Maintenance. Proceedings of the International Conference on Software Maintenance (1994)
18. G. Antoniol, G. Canfora, A.d.L.: Maintaining Traceability During Object-Oriented Software Evolution, a Case Study. Proceedings of the IEEE International Conference on Software Maintenance (1999)
19. P. Clarke, B. Malloy, P.G.: Using a Taxonomy Tool to Identify Changes in OO Software . Proceedings of the European Conference on Software Maintenance and Reengineering (2003)
20. Emam, K.E.: A methodology for validating software product metrics. Technical report, National Research Coucil, Canada (2000)