Optimisation of class diagram modelling in UML

RAKOTONIRINA Andriamitantsoa Soloarinala, ROBINSON Matio, RANDIMBINDRAINIBE Falimanana

Student, Cognitive sciences and applications, University of Antananarivo in STII, Madagascar
Doctor, Cognitive sciences and applications, University of Antananarivo in ESPA, Madagascar
Professor, Cognitive sciences and applications, University of Antananarivo in ESPA, Madagascar

ABSTRACT

The development of software must first go through a design phase. Currently, the most widely used notation for designing an application is the Unified Modelling Language (UML), which is based on the use of objects to synchronize with object-oriented programming.
Designing with UML is carried out using diagrams, and UML provides several diagrams, the most crucial of which is the class diagram. Coupling at the class diagram level should not be disregarded in terms of software quality. This article presents a technique for modelling the class diagram to achieve lower coupling. Furthermore, quality software conforms to the criterion of low coupling between the classes in the class diagram.

Keyword: *UML, diagram, coupling, class, modelling.*

1. Introduction

Modelling is a system design technique that uses a set of predefined concepts [5]. The aim of modelling is to design models. It is necessary to design a good model because modelling has a major impact on the quality of the software. In other words, the software to be produced must meet the needs of the users in order to satisfy them. What's more, the quality of the software has a direct impact on the cost [1],[2],[3],[4].
Using the UML notation requires the creation of several diagrams. The class diagram shows the elements required to build the application and is therefore one of the most important diagrams. In other words, the quality of the class diagram defines the quality of the application.
Coupling between classes in the class diagram is one of the quality criteria that cannot be ignored, and low coupling is a software quality criterion.
So, in our article, we will explain the quality criteria for software, including coupling, followed by the modelling of the class diagram and ending with a discussion.

2. Some definition

A few definitions of software quality:
Quality is defined in terms of the end user and the software delivered. Quality software meets user's needs, is reliable and easy to maintain [11].
The quality characteristics of a software product relate to security, adaptability, performance, functional capacity, reliability and ease of use [12].
In addition to these characteristics, software quality is defined by cost and deadline constraints (as low as possible and in line with customer expectations) [13].
For developers, quality means proposing software solutions that meet user requirements while keeping development times and costs under control. In this case, quality focuses on the efficiency of the development process [14].

Quality is expressed in terms of controlled production time, insignificance of design defects, absence of complaints, etc. Quality characteristics relate to customer satisfaction and the development process [12].

Software quality is a multi-faceted concept, meaning that everyone has their own ideas about it. Software quality changes from one individual to another, in other words everyone has their own criteria for quality software, whether they are a computer scientist or an ordinary user

3. Quality criteria

There are different criteria for analysing software quality. To explain the concept of software quality, we will look at the ISO/IEC 9226 standard, which sets out all the criteria for quality software. According to this standard, the 6 main characteristics of quality software are: reliability, maintainability, portability, ease of use, performance and scalability. Of all these characteristics of quality software, reliability and maintainability are the most important. A lack of reliability can lead to operational errors, such as bugs, with potentially catastrophic consequences. As for maintainability, poorly maintainable software still requires investment in its development and maintenance.
Let's explain reliability and maintainability for a better understanding.

Reliability
Reliability is the ability of software to continuously provide the expected service.
It is the probability of performing an operation without failure over a fixed period of time and for a given context. Reliability is subjective: it depends on the user and the context of use. It provides a measure of the degree of confidence and measures the consequences of a fault

Maintainability
Maintainability is:
Ease of analysis: the effort required to diagnose deficiencies or causes of failure or to identify the parts to be modified;
Ease of modification: effort required to modify, remedy faults or change the environment;
Stability: the risk of unexpected effects of modifications;
Ease of testing: the effort required to validate the modified software.
In addition, IEEE [7] defines maintainability as the ease with which software can be maintained, improved, adapted, or corrected to meet specified requirements.
The objectives of maintenance are as follows:
Manage a modification process to avoid partial corrections being made outside the iteration process;
Build customer loyalty.

Another criterion of quality software that should not be overlooked is coupling.
Coupling
Coupling has been defined as one of the most fundamental qualitative attributes for measuring software performance during the design or implementation phase [19],[20].
Coupling is defined as the degree of interdependence between the different modules that make up software. In other words, coupling has been defined as the measure of the strength of the association established by a connection between one module and another [6]. It can also be defined as the degree of interdependence between modules [8].
In addition, coupling, which is another quality attribute, measures the degree to which one element is linked to another [10]. Good software design should minimize coupling interactions [21],[22],[23].
A software application consists of several modules. At the design stage, the principle is to ensure that all modules are virtually independent. Coupling indicates the degree of independence between the modules. If all the modules are interconnected, there is a high degree of coupling between the modules, which increases the complexity of the application [9]. In this case, changing one module means changing the others, and maintenance also becomes difficult in terms of understanding the code. It is therefore advisable to link only a few classes to keep a low coupling.
Low coupling at class diagram is a criterion of quality software.

There are several types of coupling, such as data coupling, stamp coupling, control coupling, external coupling and common coupling [8].

Of these different types of coupling, data coupling, which is considered to be the lowest, is the most desired, while common coupling, which is considered to be the strongest, is the least desired.
To obtain low coupling at the class diagram, it is necessary to model the class diagram well. Low coupling between the classes that make up the class diagram will lead to low coupling at the level of the class diagram itself.

4. Related work

Parul G. and Pradeep K. B. [15] have proposed the metrics MRC (Message Received Coupling), MPC (Message Passed Coupling) and DC (Degree of Coupling) for detecting a set of design problems. The authors have developed the MCG (Method Calling Graph) to calculate the values of the proposed metrics. These metrics help developers decide whether the design should be modified or left in its original form.
Daljeet S. and Sunit K. K. [16] have proposed a technique for measuring coupling in the UML sequence diagram using the sliced program. In their work, the SDG (Sequence Dependency Graph) is generated from the states and scenarios of the sequence diagram. The SDG is then dynamically sliced, taking different aspects into consideration. These slices can be used to measure coupling. This method is a direct measurement of the coupling of an object-oriented system from the sequence diagram.
Vipin S. and Santosh K. [17] measured the degree of coupling in the UML class diagram. They used the MRC (Message Received Coupling), MPC (Message Passed Coupling) and DC (Degree of Coupling) metrics described in [15].

MRC (Message Received Coupling)
The MRC measures the complexity of messages received by classes, because the MRC is the number of messages received again by a class from other classes.

MPC (Message Passed Coupling)
MPC is defined as the number of messages passed between objects in classes; this is the low-level coupling achieved by communication between components.

DC (Degree of Coupling)
The DC is calculated at class level. It can be defined as the ratio between the number of incoming declarations to a class and the number of outgoing declarations from a class. In other words, the degree of coupling [18] is computed as the ratio between the number of messages received and the number of messages passed.
This measure makes it possible to find and correct faults in the design of object-oriented software more efficiently.
The degree of coupling is given by

$$DC = \frac{MRC}{MPC}$$

Using Method Calling Graph (MCG) is required to calculate the value of these metrics. In this case, the relationships between classes in terms of method are presented in the form of a graph from which the values of these metrics are computed.

5. Proposed solution

Coupling concerns the relationship between classes, i.e. the class diagram. The class diagram is made up of a set of several classes linked together.
When we talk about coupling, we are no longer talking about the class itself, but about all the classes. The degree of coupling therefore depends on the degree of linkage between the classes.
The method calling graph traverses all the possible paths for methods in one class to be called by another class.
Figure 1 shows a method calling graph between classes. Classes are represented by A, B and C.

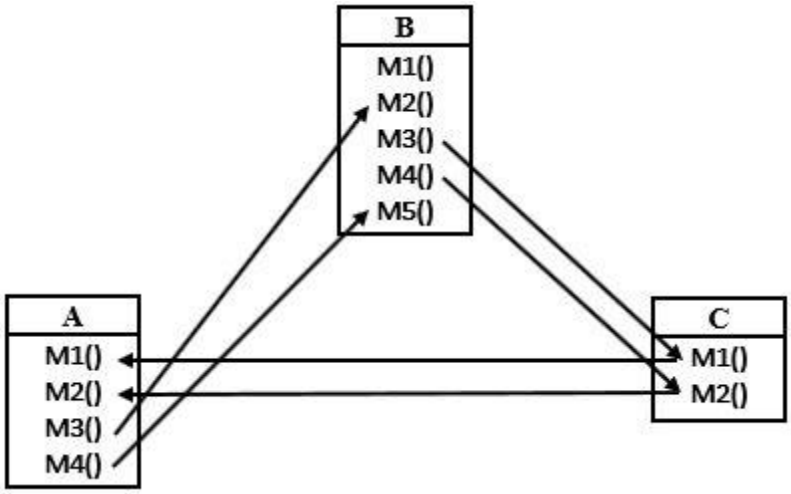

Fig -1: Method Calling Graph (MCG)

To compute the degree of coupling, this method call graph will be transformed into a message call graph whose graph nodes are of the form $A::M_i$ where Mi is a method of class A. If $A::M_n$ is called by $B::M_m$ and $B::M_m$ is also called by $C::M_k$ where {n, m, k} = 1, 2, 3. Then the message call graph is as follows:

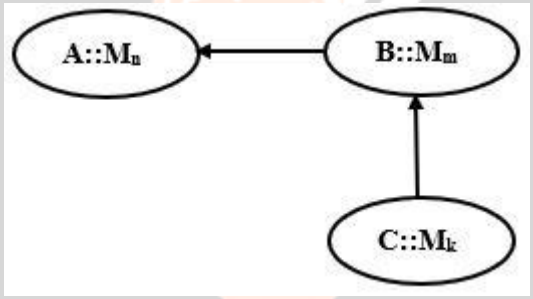

Fig -2: Message calling graph formalism

Normally, each class represented in the method calling graph has its own message calling graph, which is useful for calculating the degree of coupling. The coupling in the class diagram is therefore calculated for each class in terms of the degree of coupling.

Our work focuses on optimising the DC metric resulting from the MRC and MPC metrics of Parul G. and Pradeep K. B. on coupling computation.

The DC value should be between 0.5 and 2. If this is not the case, the designer must decide which method should be moved without affecting the inheritance property.
A DC value between 0.5 and 2, i.e. DC ∈ [0.5, 2], represents low coupling, which is desirable.
In addition, during the design phase, we must strive to achieve a low level of coupling. Low coupling between classes/modules minimises interactions and dependencies between classes/modules in the system.

Example

Let's take three classes whose Method Calling Graph is as follows:

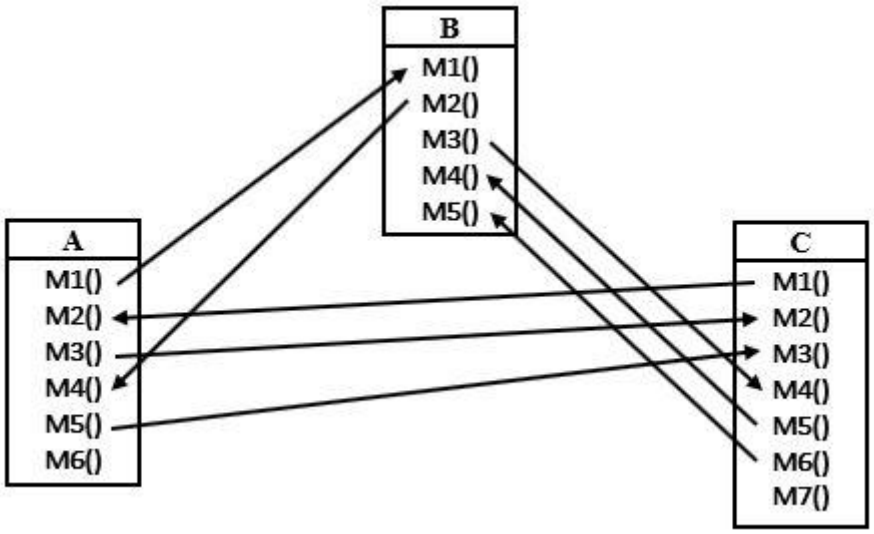

Fig -3: Example of Method Calling Graph

Let's calculate the value of the DC metric from the MRC and MPC metrics for all the classes shown in Figure -3.

Class A

Class A has 6 methods, so the message calling graph for this class is shown below:

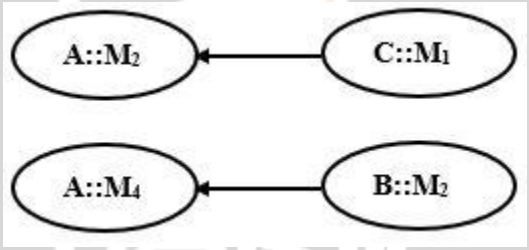

Fig -4: Message calling graph for class A

Thus, the MPC for class A is 3 because 3 consignment declarations belong to class A. The MRC for this class is 2. The MRC for this class is 2. Consequently, the degree of coupling is computed as:

DC = MRC/MPC = 2/3 = 0.66

Class B

Class B has 5 methods. The message calling graph for this class is as follows:

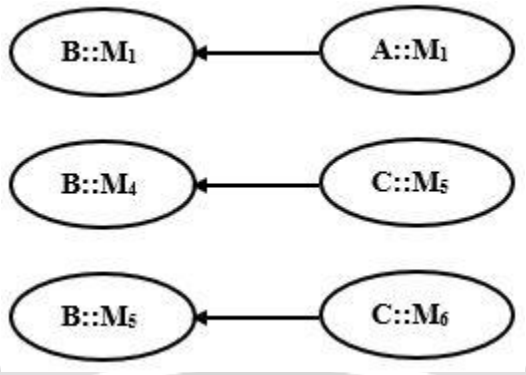

Fig -5: Message calling graph for class B

Thus, the MPC for class B is 2 because 2 consignment declarations belong to class B. The MRC for this class is 3. The MRC for this class is 3. Therefore, the degree of coupling is calculated as:

DC = MRC/MPC = 3/2 = 1.5

Class C

Class C has 7 methods. The message calling graph for this class is shown below:

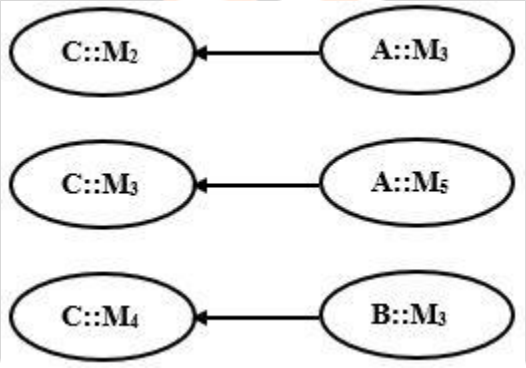

Fig -6: Message calling graph for class C

Thus, the MPC for class C is 3 because 3 consignment declarations belong to class C. The MRC for this class is 3. The MRC for this class is 3. Consequently, the degree of coupling is calculated as:

DC = MRC/MPC = 3/3 = 1

Table 1 shows the different coupling metrics for each class

Table -1: Class Level Metrics

Class	Object-Oriented Metrics		
	MPC	MRC	DC
A	3	2	0.66
B	2	3	1.5
C	3	3	1

The values of the DC metric are between 0.5 and 2, representing low coupling between classes.
Therefore, when modelling a class diagram, in order for the DC value to be between 0.5 and 2, the MPC and MRC values must be non-zero. In addition, one of the following conditions must be satisfied:

- **MRC = MPC – 1**
- **MRC = MPC**
- **MRC = MPC + 1**
- **MRC = MPC * 2**

In other words, the fulfilment of one of these conditions, combined with non-zero values of MPC and MRC, leads to a value of DC between 0.5 and 2, i.e. DC ∈ [0.5, 2], which justifies perfect coupling between classes, or more precisely, low coupling.

6. Discussion

Low coupling between the classes that make up the class diagram results in flexibility at the application. In other words, the application is made up of several modules, and in this case, changing one module does not cause the others to change. Only the module that is directly linked to the changed module is changed.
In application development, maintenance accounts for 80% of programme costs. In other words, a software development company spends a large proportion of its total resources on correcting, extending and restructuring its existing programmes. The principle of low coupling makes maintenance easier.
What's more, maintenance is a non-negligible part of software development because software has to evolve, which is why it is necessary to facilitate maintenance.

7. CONCLUSIONS

When designing software or an application, the designer must aim to ensure that the software to be designed will be quality software. There are several criteria for software quality, but the most important are reliability and maintainability. Reliability means that the software meets the user's expectations in terms of functionality. Maintainability is the ease with which the software can be maintained so that it can evolve in the future.
Another criterion that should not be overlooked is the coupling between classes in the UML class diagram. The coupling shows the relationship that exists between the different classes. Quality software is also characterised by the criterion of low coupling.
To obtain the low coupling criterion at the level of the class diagram, one of the above conditions must be met during modelling.
In this case, according to Parul G. and Pradeep K. B.'s metric, the DC value is between 0.5 and 2, which indicates low coupling, i.e. high-quality software. Moreover, it is necessary to integrate this functionality in the most used Software Engineering Workshops (SEW) in order to know in advance the quality of the design, which will reflect the quality of the software to be produced.

8. REFERENCES

[1]. B.P. Lientz, E.B Swanson et G.E. Tompkins (1978). Characteristics of Application Software. Maintenance Applications, volume 21(6), 466-477.
[2]. Martin J, McClure (1983). Software Maintenance: The Problem and its Solutions. Englewood Cliffs NJ: Prentice Hall.
[3]. Glenn E.K. et Stephen T.P. (1998). A cookbook for using the model-view-controller using interface paradigm in Smaltalk-80. Journal of Object-Oriented Programming, Volume 1(3), 26-49.
[4]. Lan Sommerville (2004). Software engineering (7^{e} éd.). British: edition British Library.
[5]. Raphaël M. (2003). Towards meta-modelling patterns. Fundamental Computing Laboratory of Lille MRU NCSR 8022, 1-5.
[6]. W.P. Stevens, G.J. Myers and L.L. Constantine. Structured Design. In IBM Systems Journal, Vol. 13, No. 2, pages 115-139, May 1974.
[7]. IEEE std 1219: Standard for Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA.
[8]. Yourdon, E. et Constantine, L. (1979). Structured Design. Englewood Cliffs: Prentice Hall.
[9]. Harrison, R. Counsell, S. et Nithi. R. (1998). Coupling metrics for object-oriented design. Dans METRICS '98: Proceedings of the 5th International Symposium on Software Metrics. IEEE Computer Society, Washington,

DC, USA, p. 150.
[10]. Larman C., UML and design patterns, Campus Presse 2003.
[11]. Beli D., Morrey I. et Pugli. J, Software Engineering: A Programming Approach, Prentice Hall Publisher, 1992.
[12]. Rigby P., Software design and quality, Afnor Editions, 1992. NORRIS M.
[13]. Arthur L.J., Improving Software Quality: An Insider's Guide to TQM, Wiley Series in Software Engineering Practice Publisher,1992.
[14]. Babey F., Software quality management, Afnor Editions, 1995.
[15]. P.Gandhi and P.K. Bhatia (2011). Optimization of Object-Oriented Design using Coupling Metrics. International Journal of Computer Applications, Volume 27 (10), 41-44
[16]. Daljeet S. et Sunint K. K. (2012). Measuring Cohesion and Coupling of Sequence Diagram Using Program Slicing. Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 84-88.
[17]. Vipin Saxena, Santosh Kumar (2012). Impact of coupling and Cohesion in Object-Oriented Technology. Journal of Software Engineering and Applications, 671-676
[18]. P. Gandhi and P. K. Bhatia. Optimization of Object- Oriented Design Using Coupling Metrics. International Journal of Computer Applications, Vol. 27, No. 10, 2011, pp. 41-44.
[19]. L.C. Briand, J.W. Daly, and J.K. Wust. A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE
[20]. Transactions on Software Engineering, vol. 25, no. 1, pp. 91–121, Jan 1999.
[21]. Huan Li. A Novel Coupling Metric for Object-Oriented Software Systems. IEEE International Symposium on Knowledge Acquisition and Modelling Workshop, pp. 609-612, 2008
[22]. W. Li and S. Henry. Object oriented metrics that predict maintainability. Journal of Systems and Software, pp. 111–122, Nov 1993.
[23]. S.L. Pfleeger and J.M. Atlee. Software Engineering: Theory and Practice, 3rd ed. Pearson Prentice Hall, 2006.