Optimisation of class 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 design phase is a crucial stage in the software application development. It involves representing models through diagrams, of which the most significant is the class diagram. There are numerous criteria for software quality, and one that should never be ignored is the cohesion at the class level. We suggest a technique for modelling classes to attain strong cohesion in this article. In addition, quality software meets the criterion of strong cohesion at class level in the class diagram.

Keyword: *diagram, cohesion, class, modelling.*

1. Introduction

Modelling is a system design technique that utilises a number of predefined concepts [5]. Its main objective is to design models. It is strongly recommended to design software efficiently, such that it satisfies the user's requirements and expectations, resulting in high-quality software. In addition, program quality has a direct impact on cost [1],[2],[3],[4].
The design, presented using UML notation, comprises of diagrams, the most important and only mandatory of which is the class diagram. This diagram shows the application interface in advance. In other words, the quality of the class diagram will represent the quality of the application. Cohesion is one of the non-negligible quality criteria in the classes of the class diagram, and strong cohesion is a criterion of software quality.
So, in our article, we'll explain the quality criteria for software, including cohesion, then class modelling and we'll finish 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 users' needs, is reliable and easy to maintain [14].
The quality characteristics of a software product relate to security, adaptability, performance, functional capacity, reliability and ease of use [15].
In addition to these characteristics, software quality is defined by cost and deadline constraints (as low as possible and in line with customer expectations) [16].
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 [17].
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 [15].

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 various 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 and performance/scalability. Of all these characteristics of quality software, reliability and maintainability are the most important. A lack of reliability can lead to operating errors such as bugs, with potentially catastrophic consequences. As far as maintainability is concerned, 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 cohesion.
Cohesion
Cohesion is defined as the manner and degree to which the tasks performed by a single software module are linked together [6]. The cohesion of a module is defined not only in terms of the number of interconnections, but also in their type. Therefore, cohesion is the link that exists between the different functions of a class.
Cohesion has been defined for procedural systems as the degree to which elements belonging to the same module are linked [8]. A coherent module has its elements closely linked and performing a reduced number of operations [11].
Cohesion measures the degree of connection between the parts of a software component which are involved in carrying out a well-defined task [13].
Additionally, the cohesion of a class is defined as a characteristic concerning the connectivity between the members of the class [9].
Strong cohesion at the level of the classes that make up the class diagram is a criterion of quality software.

There are several types of cohesion, such as accidental or random cohesion, logical cohesion, temporal cohesion, procedural cohesion, communicational cohesion, functional cohesion and informational cohesion [10],[12].
Of these different types of cohesion, functional cohesion, which is considered to be the strongest, is the most desired, whereas accidental cohesion, which is considered to be the weakest, is the least desired.
To achieve a high level of cohesion in the class diagram, it is necessary to model the class correctly. Strong cohesion of the classes that make up the class diagram will lead to strong cohesion at the level of the class diagram.

4. Related work

Bieman and Kang's [19] proposal in terms of cohesion is TCC (Tight Class Cohesion) and LCC (Loose Class Cohesion). They are closely related to that of Hitz and Montazeri, although there are some differences. They consider the criterion of sharing instance variables and method calls. An instance variable can be used directly or indirectly by a method. An instance variable is used directly by an Mi method if it appears in the body of this

method. It is indirectly used by a method Mi, if it is directly used by a method Mj which is itself directly or indirectly called by Mi. Two methods are directly linked or connected if they both directly or indirectly use at least one instance variable. TCC is defined as the percentage of the class's public method pairs that are directly connected. In addition to the direct connections between the methods of a class, LCC takes into account the indirect connections that may exist between them. LCC is defined as the percentage of pairs of public methods of a class that are directly or indirectly connected.

Bieman and Kang's approach can be summarised as follows:

Number of possible connections in a class of N methods:

NC= N*(N-l)/2

NCd: number of direct connections between methods and

TCC= NCd/NC

NCid: Number of indirect connections between methods and.

LCC = (NCd + NCid)/NC

5. Proposed solution

Cohesion concerns the class itself. The cohesion of a class is defined as a characteristic concerning the connectivity between the members of the class [9]. A class is comprised of attributes and methods which are linked together.

Mathematically, the relationships between the attributes and methods of a class are modelled in the form of a graph: The nodes correspond to the attributes and methods of the class.

Methods are represented by rectangles and attributes are represented by rounded rectangles. There may also be relationships between methods.

The following diagram shows the relationship between the attributes and methods of a class in the form of a graph.

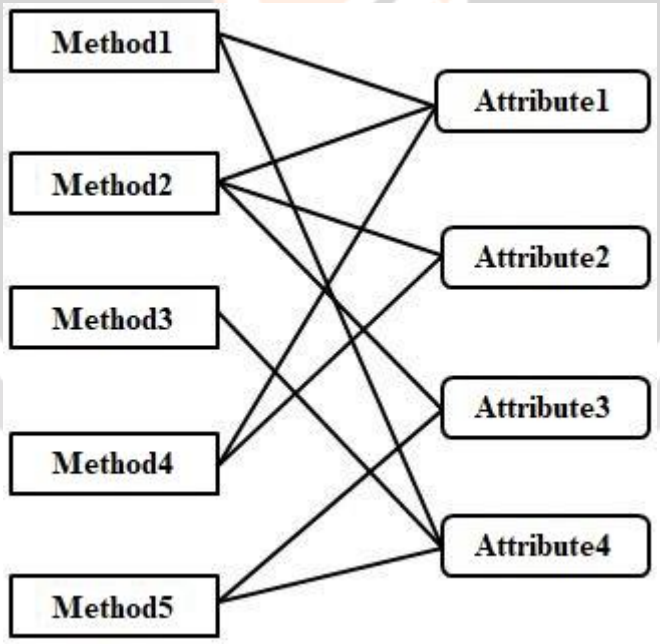

Fig-1: Relationship between class attributes and methods

Our work focuses on the optimisation of the result of the TCC and LCC metrics of Bieman and Kang on the cohesion calculation.

The values given by the TCC and LCC metrics are between 0 and 1.
The value between 0 and 0.4 corresponds to a class that is not cohesive and should be revised.
A value between 0.5 and 0.9 corresponds to a cohesive class.
The value 1 corresponds to a class with perfect cohesion, in other words, the value 1 means that we have high cohesion.
Additionally, when designing, we should strive to achieve high cohesion. A cohesive class/module focusses on a single task and has little interaction with the other classes/modules in the system.

Example

Let's take a class whose graph representing the relationship between attributes and methods is as follows:

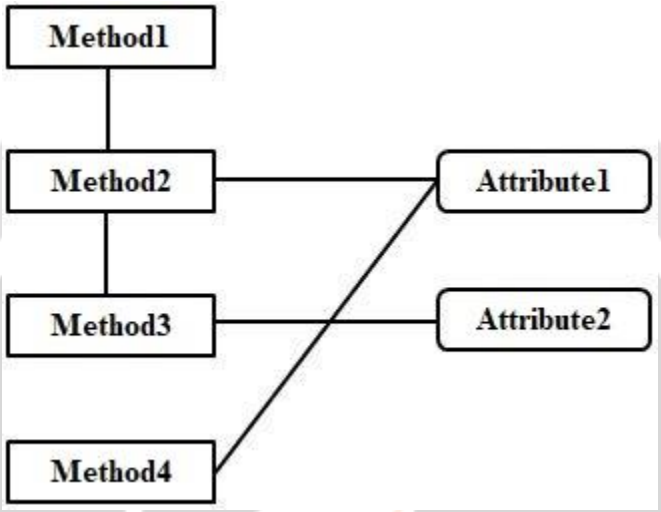

Fig- 2: Example of a graph

From this graph, we have the following data.
NC = 4(4-1)/2 = 6
NCd = (M1,M2), (M2,M3), (M2,M4)
NCd = 3
TCC = 3/6
NCid = (M1,M3), (M1,M4), (M3,M4)
NCid = 3
LCC = (3+3)/6 = 1

We can say that when all the methods are linked, the value of LCC is equal to 1, which is more desirable.
Therefore, during the modelling of a class, all the methods must be linked. In this case, the value of LCC is equal to 1, which justifies perfect cohesion in a class, or more precisely, strong cohesion.

6. Discussion

The strong cohesion of UML classes helps the designer to design modular applications, i.e. an application made up of several modules. In this case, the elements that are linked together are located in the same module, which makes maintenance easier. In other words, the elements involved in the same task are grouped together and can be easily found, which makes maintenance simpler. Software has to evolve, and software developed under such conditions may no longer adapt to future conditions, which is why it is necessary to facilitate maintenance. In addition, the modules that make up the application have their own functionality, which means that the application is easy to understand.

Strong cohesion also reduces the complexity of an application. In other words, complex code leads to errors, is difficult to test and is a challenge to maintain. The costs of an erroneous or simply poor-quality programme can be very high and can even prove critical for a company. So, designing software of reasonable complexity saves us money. It is in this sense that it is necessary to reduce the complexity of an application, which is possible and even feasible with strong cohesion.

7. CONCLUSIONS

Software quality is a multifaceted concept. In other words, everyone has their own criteria for quality software. The ISO/IEC 9226 standard proposes 6 main characteristics: reliability, maintainability, functional capability, portability, ease of use and performance/scalability. Reliability and maintainability are considered to be the most important.
Another criterion that should not be overlooked is cohesion within the classes of the UML notation class diagram. Cohesion shows the link that exists between the different functions of a class. Quality software is also characterised by the criterion of strong cohesion at class level. To obtain the criterion of strong cohesion at class level, during the modelling of the class, all the methods must be linked. In this case, the LCC value, according to the Bieman and Kang metric, is equal to 1, which indicates strong cohesion, i.e. high-quality software. Besides, it is necessary to incorporate this functionality into the Software Engineering Workshops (SEW) that are most widely used in terms of design 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 ingeneering (7e éd.). British: edition British Library.
[5]. Raphaël M. (2003). Towards meta-modelling patterns. Fundamental Computing Laboratory of Lille MRU NCSR 8022, 1-5.
[6]. The Institute of Electrical and Electronics Enginneers (1990). IEEE Standard Glossary Of Software Engineering Terminology.
[7]. IEEE std 1219: Standard for Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA.
[8]. 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.
[9]. Bieman, J. M. et Kang, B. (1995). Cohésion and retise in an object-oriented system. ACM Press. p. 259 262.
[10]. Yourdon, E. et Constantine, L. (1979). Structured Design. Englewood Cliffs: Prentice Hall.
[11]. Larman C., UML 2 and design patterns (3e ed.). Pearson Education. 2005.
[12]. Pressman R. S., Software Engineering: A Practitioner's Approach (3e ed.). New York: McGraw-Hill. 1992.
[13]. Larman C., UML and design patterns, Campus Presse 2003.
[14]. Beli D., Morrey I. et Pugli .J, Software Engineering: A Programming Approach, Prentice Hall Publisher,1992.
[15]. Rigby P., Software design and quality, Afnor Editions, 1992. NORRIS M.
[16]. Arthur L.J., Improving Software Quality: An Insider's Guide to TQM, Wiley Series in Software Engineering Practice Publisher,1992.
[17]. Babey F., Software quality management, Afnor Editions. 1995.