Transformation of Dependency and Association in UML Design Class

Ngamsantivong Thavatchai, Ratanavilisagul Chiabwoot

Abstract*: This paper presents two new conceptual relationships between classes of software development known as dependency and association. The design between the two relationships could be interchangeable because it always takes place in real-life situations — for instance, the relationship from friends to husband-wife and vice versa. However, in terms of coding, the most important factor is system performance. That means the designer could write the code as dependency or association to provide the same result. To improve the efficiency of the program, the researcher writes the code in the C++ language to execute four types of variables named messages, strings, calculation, and sorting. The four types of the variable used to test the performance of aggregation, composition, dependency, and functional programming, the timestamp was used to measure the execution time before and after for each case 50 times. The F-test statistic was used to compare the mean difference of each type of variable. The researcher found that for the Message variable. The functional programming is the fastest, followed by aggregation, composition, and dependency, the average C.P.U. time are 13566.60, 17891.70, 18532.66 and 19336.76, at 0.0 level of significance. For the String variable found that functional programming is the fastest followed by dependency, composition, and aggregation, the average C.P.U. time are 23785.88, 27449.76, 28478.24 and 28788.18, at 0.0 level of significance. For calculation found that functional programming is the fastest, followed by aggregation, composition, and dependency, the average C.P.U. time are 26982.68, 29311.86, 29377.50 and 29397.30, at 0.0 level of significance. For sorting found that functional programming is the fastest, followed by aggregation, composition, and dependency, the average C.P.U. time are 17925.20, 18408.36, 21641.68 and 22861.14, at 0.0 level of significance.

***Index Terms*: Association, Dependency, Composition, Aggregation, Relationship Between Class**

I. INTRODUCTION

In the real-world relationship between dependency and association, are interchangeable. The software developer could write the code in both dependency and association styles and come up with the same results, except for the fact that the execution time is still in question. Which method of code-writing will be the fastest? Therefore, intelligent software should know how to react accordingly when real-world situations change. It should provide the best practices for software performance. Software developers follow design concepts like a blueprint and rarely consider the best way to write the code. Therefore, designers should be aware of when to use dependency and association.

Revised Manuscript Received on July 20, 2019.

Ngamsantivong Thavatchai, Faculty of Applied Science, King Mongkut's University of Technology North Bangkok, Thailand

Ratanavilisagul Chiabwoot, Faculty of Applied Science, King Mongkut's University of Technology North Bangkok Department, Thailand.

II. RELATED LITERATURE

A. UML

Unified Modeling Language (UML) is the language of identifying, specifying, and documenting system and software artifacts. Therefore, the class diagram and relationships are the main focus of this study.

B. Dependency

Webster's Dictionary defines dependency as something dependent on something else. In UML, dependence represents a relationship between a client and a supplier. In which a change in the specifications of the supplier may affect the client. It represents a line of dashes (----), and the arrow provides the direction. For instance, if A is dependent on B (A--- > B) and then if B changes, A has to change too.
In UML, Class A is dependent on Class B, as represented in Fig. I.

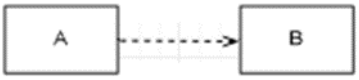

Fig. 1: Dependency relationship

C. Association

Associations represent structural relationships between objects of different classes or information that must preserve for some duration of time. It is not only characterized by procedural dependency relationships. It represents a line (——), and the arrow indicates the direction.

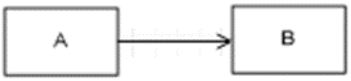

Fig. 2: Association relationship

One object has a permanent association with another object. There are two categories of associations; aggregation and composition. An aggregation is a stronger form of a relationship, in which a relationship exists between itself and its parts. The aggregate has an aggregation association with its constituent parts. A hollow diamond was attached to the end of an association path on the side of the aggregate (the whole ◇—) to indicate aggregation. A composition is a form of aggregation with strong ownership and coincident lifetimes on the part of the aggregate. When the whole is removed, the part is also removed. The part may be removed (by the whole) before the whole is removed. A solid filled diamond (◆—) was attached to the end of an association path (on the "whole side") to indicate composition.

D. Interchangeable relationships in real-world situations

As previously mentioned, in real-world situations, connections can be interchangeable. For instance, A and B, have a friendly relationship. After a while, A and B develop a husband-wife relationship, as shown in Fig 3.

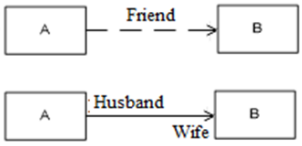

Fig. 3: Interchangeable relationship

E. Composition and Aggregation in Coding

In Object-Oriented Analysis and Design, an association relationship consists of composition and aggregation. The composition is a strong type of relationship. For instance, A composed of B, which means that A could not exist if B did not exist. In coding language, it is as follows:

```
Class A {
        A ( ) ;
        B _ref = new B ( );
          …
}
```

In terms of aggregation, this relationship is not strong. Therefore Class A could exist even though Class B does not exist. The code should be: -

```
Class A {
        A ( ) ;
        B _ref ;
}
```

F. Analysis of Variance

The ANOVA or Analysis of Variance is a statistical tool used to identify the mean and the differences between three or more sample groups. The ANOVA computed by using the formula below.

Mean Square Between / Mean Square Within or

$$F = \frac{Between}{Within} = \frac{MSG}{MSE}$$

III. CASE STUDY

This research created four situations and the execution of each relationship; the first sent the message, the second showed the string, the third calculation, and the last was sorting. An example of the output of execution illustrated in Fig 4.

A. Dependency class diagram

The researcher created three classes; Basic, Dependency, and Code. The basic class is the main class and includes strings, messages, sorting, and calculation. The Dependency

Fig. 4: Execution output

class demonstrated the relationship between Basic and Dependency. The Coding class was used to perform the execution of the program.

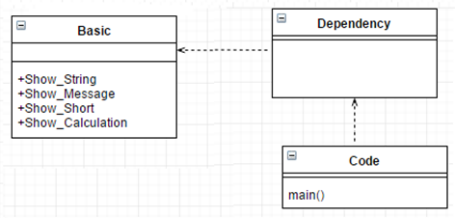

Fig. 5: Dependency class diagram

B. Aggregation class diagram

The result is similar to that found in Fig. 5. There were three classes created for the purpose of this research, known as Basic, Aggregation, and Code. The function and Code classes executed the same tasks as the participants in Fig. 5. The only change was the Aggregation class, which showed the relationship between the Aggregation class and functional programming.

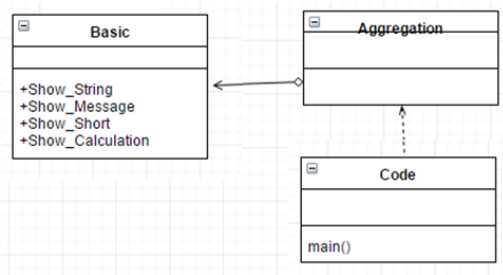

Fig. 6: Aggregation class diagram

C. Composition class diagram

In the previous example, the researcher replaced the Aggregation class with the Composition class. It changed the relationships in the Composition class while the rest remained the same.

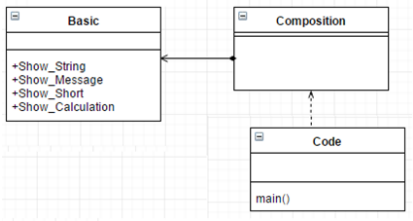

Fig. 7: Composition class diagram

D. Normal class diagram (Functional Program)

In the last case, the researcher moved all of the functionality in terms of dependency, aggregation, and composition into the main function and created none objected-oriented programming.

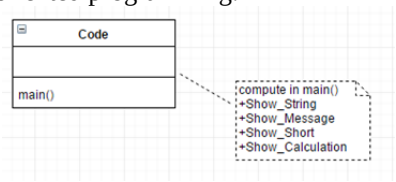

Fig. 8: None objected oriented programming

IV. RESEARCH FINDINGS

From the executable program, the timestamps recorded the execution before and after of each variable type in all cases.

A. Message execution time

The computation of the messages indicated a relationship between the Aggregation, Composition, Dependency, and functional show in Table 1.

Table 1: Message execution time

Descriptives

		N	Mean	Std. Deviation	Std. Error	Minimum	Maximum
SMessage	Aggregation	50	17891.70	1117.779	158.078	15001	20012
	Composition	50	18532.66	1386.841	196.129	14737	21953
	Dependency	50	19336.76	1211.394	171.317	18990	27244
	Functional	50	13566.60	1254.096	177.356	13176	22052
	Total	200	17331.93	2557.916	180.872	13176	27244

The results revealed that Aggregation was the fastest class (17891.70), followed by Composition (18532.66) and Dependency (19.336.76). The functional program did not include because it was not relevant in terms of computation. In real-life situations, developers could not write all of the code, only in the main function.

B. String execution time

Concerning string execution times, the results demonstrated that Dependency was the fastest (27449.76), followed by Composition (28478.24) and Aggregation (28788.19). The dependency relationship was the fastest, perhaps because the other two relationships took time to create the object.

Table 2: String execution time

Descriptives

		N	Mean	Std. Deviation	Std. Error	Minimum	Maximum
SString	Aggregation	50	28788.18	5411.671	765.326	21644	37170
	Composition	50	28478.24	2056.578	290.844	24743	35534
	Dependency	50	27449.76	3082.564	435.940	21197	29983
	Functional	50	23785.88	1925.830	272.354	21309	33841
	Total	200	27125.52	3935.561	278.286	21197	37170

C. Calculation execution time

Aggregation (29311.86) was the fastest class in terms of calculation, followed by Composition (29377.50), and finally, Dependency (29397.30). The results are shown below, as follows.

Table 3: Calculation execution time

Descriptives

		N	Mean	Std. Deviation	Std. Error	Minimum	Maximum
SCal	Aggregation	50	29311.86	973.713	137.704	26239	30137
	Composition	50	29377.50	1177.776	166.563	24617	30715
	Dependency	50	29397.30	976.654	138.120	26224	30275
	Functional	50	26982.68	71.335	10.088	26926	27300
	Total	200	28767.34	1370.818	96.931	24617	30715

D. Sort execution time

In terms of sorting, the aggregation (18408.36) relationship was the fastest way to execute, followed by composition (21641.68) and dependency (22861.14). The results show in Table 4.

Table 4: Sorting execution time

Descriptives

		N	Mean	Std. Deviation	Std. Error	Minimum	Maximum
SSort	Aggregation	50	18408.36	1276.999	180.595	18096	26567
	Composition	50	21641.68	2497.211	353.159	16905	25089
	Dependency	50	22861.14	1524.002	215.526	20514	30800
	Functional	50	17925.20	1836.955	259.785	16317	26183
	Total	200	20209.10	2783.520	196.825	16317	30800

V. CONCLUSION

An Analysis of Variance was employed to compare the average calculation times. The results are shown below, as follows:

Table 5: ANOVA of Aggregation, Composition, and Dependency

ANOVA

		df	Mean Square	F	Sig.
SMessage	Between Groups	3	332535805.4	214.091	.000
	Within Groups	196	1553247.902		
	Total	199			
SString	Between Groups	3	264210276.4	22.617	.000
	Within Groups	196	11681680.51		
	Total	199			
SCal	Between Groups	3	70844302.88	86.023	.000
	Within Groups	196	823553.938		
	Total	199			
SSort	Between Groups	3	292407756.3	86.232	.000
	Within Groups	196	3390944.318		
	Total	199			

Table 5 indicates a significant difference regarding the average execution times of messages toward aggregation, composition, and dependency. The more effective way to design the system to execute messages is the use of aggregate, composition, and dependency.

There were significant differences between the average string execution time of concerning aggregation, composition, and dependency. Therefore, the designer should use dependency, composition, and aggregation.

There are significant differences in the average execution time of calculation concerning aggregation, composition, and dependency. The best way to design the system to execute calculation was also to utilize aggregation, composition, and dependency.

There were also significant differences in terms of the average execution time of sorting toward aggregation, composition, and dependency. The best way to design the system to execute calculation was to employ aggregation, composition, and dependency.

Therefore, functional programming was the fastest but not relevant for this study. The designers and developers should not just consider real-life situations but also the fastest way to develop computer systems.

ACKNOWLEDGMENT

This work was supported financially by King Mongkut's University of Technology, North Bangkok, Grant Number: KMUTNB-61-GOV-A-08.

The researchers are thankful to the anonymous referees for their valuable suggestions.

REFERENCES

1. Craig Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development," 3rd Edition Pearson Education, 2005.
2. Hans-Erik Eriksson Others, "UML 2 Toolkit," OMG Press Advisory Board xix 2003.
3. Scott W. Ambler,"The Elements of UML(TM) 2.0," Style Cambridege University Press 2005.
4. Scott "W. Amber, "The Object Primer 3rd Edition: Agile Model Driven Development with UML 2," CambCambridge University Press, 2004.
5. Association vs. Dependency vs. Aggregation vs. Composition https://nirajrules.wordpress.com/2011/07/15/association-vs-dependency-vs-aggregation-vs-composition/
6. DependencyAndAssociation https://martinfowler.com/bliki/DependencyAndAssociation.html
7. Difference between association and dependency? https://stackoverflow.com/questions/1230889/difference-between-association-and-dependency
8. Relationship types https://www.ibm.com/support/knowledgecenter/SS5JSH_9.1.1/com.ibm.xtools.modeler.doc/topics/rreltyp.html
9. What is the difference between dependency and association in UML https://www.quora.com/Whats-the-difference-between-Dependency-Association-Aggregation-and-Composition-In-Class-Diagrams-in-UML

AUTHORS PROFILE

Assoc. Prof. Ngamsantivong Thavatchai received a BA. in Teaching Mathematics from Chulalongkorn University, Thailand and a Master's Degree in Educational Technology from Technological University of the Philippines. During newly graduated was sent trained on computerized information systems in many countries such as West-Germany, Italy, Japan, and Australia. The author of many textbooks in Thai such as UML, OOP, SPSS. At present research on IoT, Fuzzy logic, and software engineering.

Dr. Rantanavilisagul Chaibwoot received a Ph.D. in Electrical Engineering from King mongkut's institute of technology ladkrabang. I research in field Artificial Intelligence, Natural language processing, Machine learning, Searching Technique, Web Programming, Computer Network, Multimedia, Animation, and the author of "Optimization and Searching Algorithm".