TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.5, May 2014, pp. 4018 ~ 4023
DOI: http://dx.doi.org/10.11591/telkomnika.v12i5.4664

Evaluations of Design Patterns for a Role-Playing Game

Jung-Hua Lo*, Chi-Shain Yeh
Department of Applied Informatics, Fo Guang University,
No.160, Linwei Rd., Jiaosi, Yilan County 26247, Taiwan (R.O.C.)/(886 3)-9871000
*Corresponding author, e-mail: jhlo@mail.fgu.edu.tw

Abstract

With the rapid of the deployment of computer systems, people in the modern society are increasingly dependent on the hardware and software systems. When the demand for computer systems increases, the possibility of crises from computer failure will also increase. Therefore, in order to achieve a desired level of quality, the experienced software team must have a better design method to establish the system. That is, it is very important to choose a better approach that can be applicable to develop the complex software system. The experienced designers will record the design problems which they felt hard to solve or they took much time to solve. When they meet the similar problems later, they would refer to the record. Design patterns are produced from the record. This Paper aims at explaining how to apply the design patterns in the game framework through a simple example by taking the Role Playing Game as example and adding design patterns; it also tries to make a judgment whether the software quality has been improved by using the object-oriented software quality model to test the software quality of the Role Playing Game without the design patterns and that with the deign patterns.

Keywords: *design pattern, role-playing game (RPG), quality model for object-oriented design (QMOOD), software quality measurementr*

1. Introduction

As the software industry gradually becomes mature, software quality is regarded as the life of a software enterprise. Software quality assessment is an important issue for most systems. It can be viewed as a powerful standard about quantifying to what extent a system or software possesses desirable characteristics. It also give us a confidence about the correctness of a given software system through qualitative or quantitative means or a mix of both. As a result, software quality measurement is an essential ingredient in customer satisfaction. A number of software quality assessment models have been developed to evaluate the quality of a software system. A common approach for assessing Software quality is by choosing a desirable characteristic and then using a set of measurable attributes the existence of which in a piece of software or system tend to be correlated and associated with this characteristic. Software failure is similar to hardware failure since both can be described by probability distributions. However, software faults are harder to visualize, detect, and correct, comparing with the hardware's physical faults. Since software is embedded in everything and permeates our daily life, the correct performance of software systems becomes an important issue of many critical systems. Therefore, in order to achieve a desired level of quality, the experienced software team must have a better design method to establish the system. That is, it is very important to choose a better approach that can be applicable to develop the complex software system. The experienced designers will record the design problems which they felt hard to solve or they took much time to solve. When they meet the similar problems later, they would refer to the record. Design patterns are produced from the record. Design patterns originate from Architecture [1] and the research achievements of E. Gamma and other three people are most famous and widely used. E. Gamma and other three people mentioned in their works that it was difficult to design object-oriented program and more difficult to make it reusable; engineers all knew that using design patterns could improve the software quality and using uniform form, writing pattern and language to describe the past design experience as design patterns could benefit designers' communication and understanding, thus people could reduce the cost for designing and maintaining the software [3-5]. This paper takes the Role-Playing Game, RPG as the research example to design two game versions. One is the initial version without the design

Received October 24, 2013; Revised December 25, 2013; Accepted January 15, 2014

patterns, the other one is the improved version which improves the imperfect program design by referring to the object-oriented designing principles. Our improvement strategy, based on the design patterns, improves the imperfect design to produce the contrast version, and finally uses the Quality Model for Object-Oriented Design, QMOOD) [6] to measure the comparing results of the two versions; it uses the practical data to prove that using design patterns can improve the software quality.

The organization of this paper is as follows. Section 2 presents an overview of several classical design patterns that can be applied to this RPG software development. Furthermore, we also describe in detail how systems of the above methods are utilized to validate the proposed approach. The quantitative valuation of wo versions of these RPG software packages are discussed in Section 3. Conclusions are presented in Section 5.

2. Methodology

2.1. Class Diagram for A Role-Playing Game

Role-playing Game is one kind of game which includes mountains of game contents, stories and playing principles. In this kind of game, the player must play a role in the setting story. The role is produces according to the game story and relevant numerical values like life value, physical strength, intelligence and response rate. There are four assemblies in this research example including player assembly, the enemy assembly, the map assembly and the fighting assembly. All the assembly category frames of the game are shown in Figure 1.

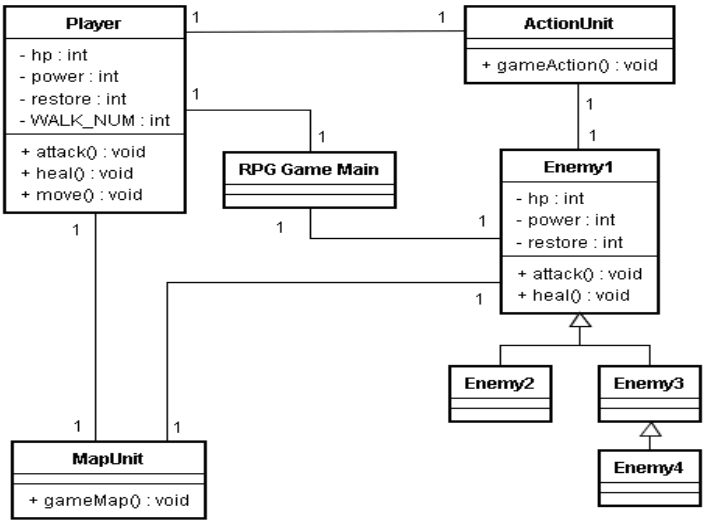

Figure 1. Effects of Selecting Different Switching under Dynamic Condition

During the game process, the main program controls the changing-over of the frames, the working timing of the assemblies and the starting and over of the game; the interactions and the behaviors of the roles in each assembly are controlled by each assembly. The main program is almost an intermediary which provides the working platform for the assemblies. In the map frame, the frame on the scenery will change from the overlooking map frame to the horizontal fighting frame, at the same time, the life values of the player and the enemy will be listed; when the player or the enemy acts, there will appear words explaining what they are doing; until one role's life value is 0, the game is over.

2.2. Template Method Design Pattern

In the Role-Playing game, sometimes because of the need of the plot, there are many roles that have most common attributes and methods and one or two differences, so there are many repeating program codes. If we need to modify one common attribute or method, we need

to modify all the attributes and the methods corresponding to this program code. For example, we can produce a new category called *GameObject*, the player and the enemy have the same attributes so they can inherit the attributes and the methods of *GameObject*; as to the role, they just need to define the unique attributes and methods, as shown in Figure 2.

After using this template method pattern, each time we add a new role, we don't need to program the same program as the new role, which inherits the *GameObject*, only need its unique attributes and methods. This method not only increases the reuse of the program code, but also benefits the expansion of the system as we only need to do some small correction. In addition, when we modify the common attributes and the methods, we only need to modify the *GameObject* instead of modifying all the roles, such improving the system's maintaining attribute.

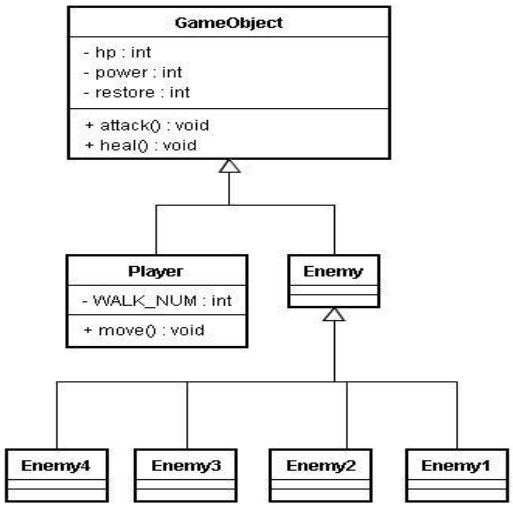

Figure 2. *GameObject* Template Method Pattern

2.3. Strategy Design Pattern

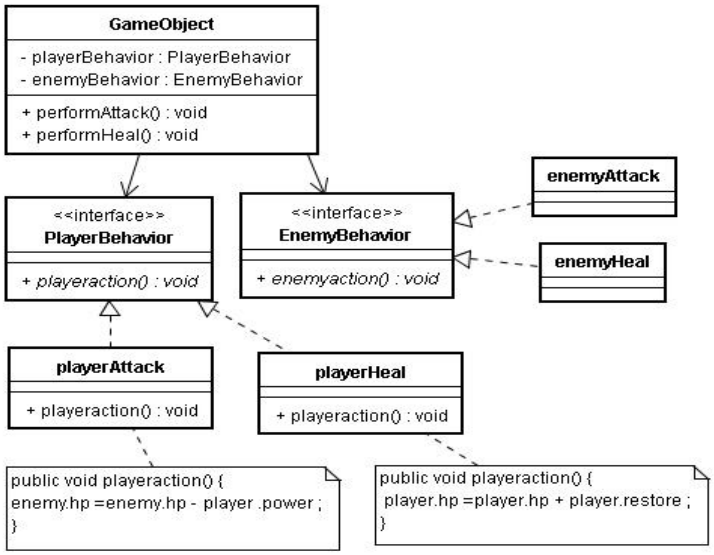

Figure 3. Strategy Design Pattern

In the Role-playing game, each role has many unique actions. For example, the attack from the player can reduce the life value of the enemy by multiplying according to its strength

while attacks from different enemies are different. So we need to design the program code for each role's actions. However, after using the template method pattern, though we have resolved the problem of the repeated program codes, when we meet many different actions, we need to produce many inherit reference categories, so there will be complex category association and difficulty in maintaining. Considering this, we put forward the solving problem for Strategy Pattern, i.e. classifying the categories. For example, classifying the attacks from players and the enemies, defining and naming them, and give them ports for external combination, like the attack from the player or the enemy, and what action they will do. As shown in Figure 3, the player role inherits *GameObject*, chooses the *PlayerBehavior*, and design that carrying out the playerAttack in the play actions is its own action method. After using the strategy patterns, the newly added action methods which modify the role will not affect the role's program code and will not produce repeated program code. In addition, new roles will be produced by changing or combining the action methods of the roles.

2.4. Factory Method Design Pattern

In the Role-playing game, there are interaction among roles, sometimes in the map frame, and sometimes in the fighting frame, all these depend on the story plot setting. The system deciding the appearing of the roles is controlled by the main game program codes. Once the method for building a role changes, it is necessary recheck and modify the main program. However, if we modify it very often, the mistake opportunity will improve. So we put forward Factory Method Pattern to solve this problem as shown in Figure 4. For example, if we need to build enemies in the fighting frame, we can use *EnemyFactory* to create enemy roles. As to which enemy roles we will create, we can refer to *EnemyFactory* inheriting category according to the reality. Then the main program will not need to be modified often as it is affected by the increased role categories and the responsibility burden of the main program will be reduced.

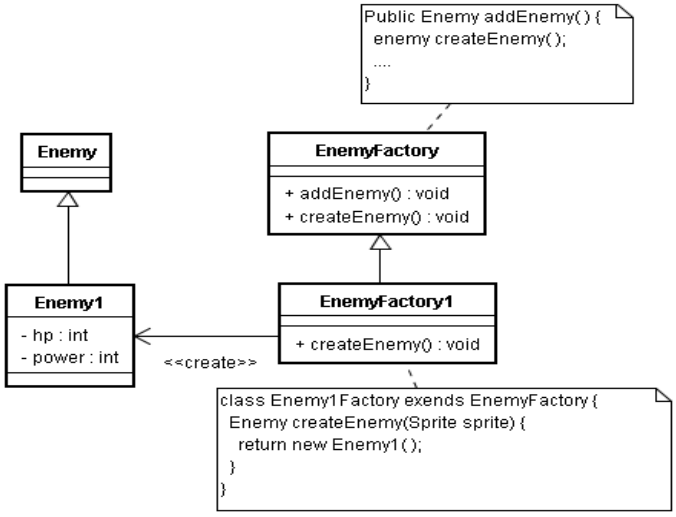

Figure 4. Factory Method Pattern

2.5. Quality Model for Object Oriented Design

Reducing of the production expenses is one of the software engineering aims. So, many method and methodology such as object-oriented, component-oriented, aspect-oriented, and service-oriented are preferred for developing software engineering [7]. The key measurement point of the software quality with traditional structural design is the program code. Some measurement of the quality has something to do with the length of the program code and the program grammar, and is not suitable for software designed according to the object-oriented software designing ideas [8]. For example, if we choose the attribute associated with portability, we need to compute the number of target-dependent statements in a program. More precisely, these measurable attributes are the "hows" that need to be enforced to enable the "whats" in

the software quality definition. The fundamental objective of this measurement is to address some of the well known human biases that can adversely affect the delivery and perception of a software development project. Chidamber and Kemerer [9] put forward C&K measurement which mainly measures the complexity and the inheriting relation of the object. Bansiya and other people referred to Dromey [10, 11]'s theory and put forward QMOOD (Quality Model for Object-Oriented Design) object-oriented design quality model. They combined the object-oriented quality measurement and the design characters, and then referred to the software's quality attributes. Our research put forward a stratum model whose design characters are relating to the measurement by using Dromey's theory and the reference definitions of the 6 quality attributes of ISO9126. Finally we get the software attributes quality by pulsing several design characters. Details are shown in Table 1.

Table 1. Expressions for Calculating the Software Quality Attributes Measurement

Quality Attribute	Index Computation Equation
Reusability	-0.25*Coupling + 0.25*Cohesion + 0.5*Messaging + 0.5*Design Size
Flexibility	0.25*Encapsulation - 0.25*Coupling + 0.5*Composition + 0.5*Polymorphism
Functionality	0.12*Cohesion + 0.22*Polymorphism + 0.22Messaging + 0.22*Design Size + 0.22*Hierarchies
Extendibility	0.5*Abstraction – 0.5 *Coupling + 0.5*Inheritance + 0.5*Polymorphism
Effectiveness	0.2*Abstraction + 0.2*Encapsulation + 0.2*Composition + 0.2Inheritance + 0.2*Polymorphism

3. Comparison and Results

In the practical process, we use two game software versions, one version is the initial version (RPG1.0) which does not use the design patterns, and the other version (RPG2.0) is the improved one which has improved the program design according to the object-oriented design principles. The comparing result of the measurement of the model designed according to QMOOD object-oriented quality model shown in Table 2. From the quality of the Role-playing game with design patterns and that without the design patterns, as shown in Figure 5, after using the design patterns, many quality attributes have been improved, especially the reusability and the extendibility. In the reusability we use the strategy patterns and the temple patterns. The strategy patterns adding many changeable assemblies in the system improve the reusability of the system assemblies. The template patterns make the assembly be able to inherit the father-category assembly and to be reused through the method of replacing the father-category assembly. In addition, as to improving the extendibility, as the strategy patterns define the whole group's method group, provide other users with suitable ports for development, if the system needs new methods, provided it matches the definition, it can add a category of special method through the port; so the convenience of extendibility is improved greatly. Through complete pattern, when the newly added assemblies have the common functions, we only need to add the unique function of the assembly.

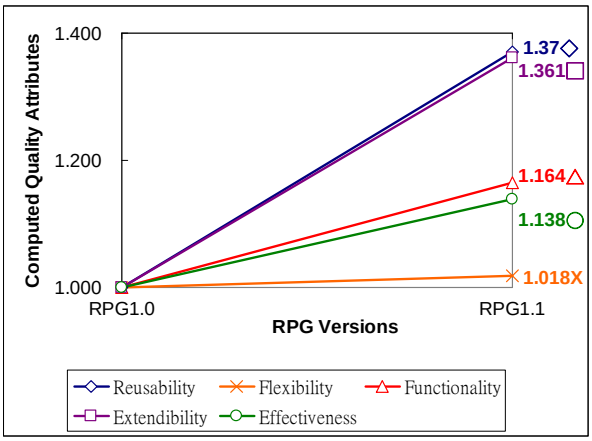

Figure 5. Comparison of the Design Pattern Quality Attributes

Table 2. Comparison of the Formal Measurement Values of Two RPG Software Versions

Metric	Design Property	RPG1.0	RPG2.0
DSC	Design Size	1	1.667
NOH	Hierarchies	1	0.937
ANA	Abstraction	1	1.654
DAM	Encapsulation	1	1
DCC	Coupling	1	1
CAM	Cohesion	1	1
MOA	Composition	1	0.968
MFA	Inheritance	1	1.001
NOP	Polymorphism	1	1.067
CIS	Messaging	1	1.073
NOM	Complexity	1	1.023

4. Conclusion

From the result of this research we can know that applying design pattern in the Role-playing game can efficiently improve the software quality, especially the reusability and the extendibility. The functional ability, the effectiveness and the elasticity are not greatly improved in this research. Maybe this is because the function demand of the system is not so great, and the design pattern does not change them very much. If we enlarge the system, or modify the design pattern more often, the expression in the quality will be more obvious. However, if we use the improper design or overuse the design pattern, which causes the increase of the coupling rate and difficulty of reading, the quality of the software may be decreased.

Acknowledgements

This research was supported by the National Science Council, Taiwan, R.O.C., under NSC 100-2221-E-431-002-.

References

[1] Alexander C, Ishikawa S, Silverstein M. A Pattern Language: Towns, Buildings, Construction. Oxford University Press. New York. 1977.
[2] Gamma E, Helm R Johnson, R Vlissides J. Design Patterns: Elements of Reusable Software. Addison-Wesley. 1994**.**
[3] Freeman, Eric, Elisabeth Freeman, Kathy Sierra, Bert Bates. Head First Design Patterns. O'Reilly Media. 2004.
[4] Larman, Craig. Applying UML and Patterns. Prentice Hall. 2005.
[5] Bernd Bruegge, Allen H Dutoit. Object-Oriented Software Engineering: Using UML, Patterns, and Java. Prentice Hall. 2010.
[6] Agdish Bansiya, Carl G Davis. A Hierarchical Model for Object-Oriented Design Quality assessment, *IEEE Transactions on Software Engineering*. 2002; 28: 4–17.
[7] Mehdi Yaghoubi, Mahmood Yaghoubi, Manoocher Babanezhad. Different Proposed Models to Mapping MDA to RUP, *TELKOMNIKA Indonesian Journal of Computer Engineering*. 2013; 13(3): 301-306.
[8] Sommerville, Ian. Software Engineering.7 ed. Pearson Education. 2008.
[9] Chidamber S, Kemerer CF. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*.1994; 20(6): 476–493.
[10] Dromey RG. A Model for Software Product Quality. *IEEE Transactions on Software Engineering.* 1995; 21(2): 146–162.
[11] Ning Jingfeng, Hu Ming. Study on Software Quality Improvement based on Rayleigh Model and PDCA Model. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(8): 4609-4615.