Application of Design Patterns in Process of Large-Scale Software Evolving

Wei WANG, Hai ZHAO, Hui LI, Peng LI, Dong YAO, Zheng LIU, Bo LI, Shuang YU, Hong LIU, Kunzhan YANG

Information Science and Engineering, Northeastern University, Shenyang, China.
Email: wangweiwin1@163.com, zhhai@neuera.com

Received September 15th, 2009; revised September 29th, 2009; accepted October 13th, 2009.

ABSTRACT

To search for the Design Patterns’ influence on the software, the paper abstracts the feature models of 9 kinds of classic exiting design patterns among the 23 kinds and describes the features with algorithm language. Meanwhile, searching for the specific structure features in the network, the paper designs 9 matching algorithms of the 9 kinds design patterns mentioned above to research on the structure of the design patterns in the software network. At last, the paper analyzes the evolving trends of the software scale and the application frequency of the 9 kinds of design patterns as the software evolves, and search for the rules how these design patterns are applied into 4 kinds of typical software.

Keywords: Design Pattern, Feature Model, Software Network, Evolving Trends

1. Introduction

With the increasing of system scale and complexity, the reliability and maintainability are highly required. Design Patterns describe common problems that frequently occur in the process of the object oriented software development and give the resolutions of these problems. Implication Design Patterns into software development can enhance the open, compatibility, stability and extensibility of software, which makes the development and maintenance much easier [1].

Does Design Patterns improve the quality of software efficiently? Can Design Patterns be widely used? How these patterns are composed reasonably? What is the reasonable scope of the ratio of the times patterns used and software scale? Facing these questions, it becomes an urgent issue to quantify and measure these patterns when they are used in software in the process of software designing.

If software continues to evolve, it needs to be reorganized [2–4]. This is called refactoring and the frames occur during this time. A better understanding of Design Patterns will reduce the time that is spent on refactoring. Looking into how design patterns are implied into some software organized fairly well and evolved continually can direct the design of software system positively. Therefore, it is significant to find out the evolving trends that design patterns are implied into software designing [5,6].

With the help of open-source software Doxygen, the object oriented software is abstract into XML. Then with the help of XmlParse which is developed in my lab, collect the nodes and edges from XML and abstract the software into software network.

The topology of software system can be represented by topology of network [7–9]. In the network, nodes represent the component of software and the edges represent the relation between nodes. Complex networks theory is applied into software system which mainly refers to open source software, reverse engineering that get the class graph and network model of the source code is taken to get and analyze the organization structure [10,11]. The abstraction process is shown as Figure 1.

2. Abstracting Process

According to the definition of the design patterns, the paper abstracts the structure features and expresses in mathematical language, which is used for designing and realizing the matching algorithm [12–15].

In the software network, nodes present the abstract data; edges present the relation between the nodes. Nodes can be classified into class, struct and interface; edges can be classified into inherit, usage, static, template and
Application of Design Patterns in Process of Large-Scale Software Evolving

2.1 Flyweight

*Flyweight* supports a number of fine-grained objects with sharing method. The frame of *Flyweight* is as shown in Figure 2.

Abstract the main features of *Flyweight*. Known from the frame of the *Flyweight*, classes in *Flyweight* can be mainly classified into *Flyweight* and *FlyweightFactory*. *FlyweightFactory* is an abstract class and the template of *FlyweightFactory* is defined in *Flyweight*. That is there is one-to-many relation between *Flyweight* and *FlyweightFactory*. *Flyweight* at least has two subclasses. The software network of the *Flyweight* is as shown in Figure 3.

All of the nodes in *Flyweight* are class. There is an edge with double value, template and usage, between node *FlyweightFactory* and node *Flyweight*, and the edge is from node *FlyweightFactory* to node *Flyweight*. The inherit in-degree of node *Flyweight* is more than 1.

2.2 The Other Eight Design Patterns

The abstracting processes of the other eight design patterns are similar to *Flyweight*. The software network of the nine design patterns are shown in Figure 5. The abstracting standards of the nine design patterns are shown in Table 1.
3. The Application and Analysis of Design Patterns in Software Evolving

The paper makes research on four kinds of open-source software: text-processing software abiword, image-processing software blender, web browser software firefox, and language-development software eclipse. There are more than one version can be used in these four widely used software, for this reason these software are taken as examples.

3.1 How Software Scale Changes in Software Evolving

Since there is linear relationship between number of nodes and number of edges, the software scale can be represented by the number of nodes. These results can be received: during the software evolving, number of nodes in abiword changes smoothly, that in blender and eclipse increases a little, and that in firefox increase first and decrease at last. Through checking software files, we find that the cores of abiword, blender and eclipse hardly change, while the core of firefox changes from the version of 3.0 to the version of 3.0.7.

3.2 The Application of Design Patterns in Software Evolving

The evolving trends of the implication of design patterns in abiword, blender, firefox and eclipse are shown as Figure 6, Figure 7, Figure 8, and Figure 9. The abscissa is the design patterns being used and the ordinate is times.
As Figure 6 shows, six patterns are used in *abiword*: Singleton, Bridge, Decorator, Memento, Mediator and Chain of Responsibility. The times of these six patterns being used goes up first, and then goes down, at last goes smoothly. Meanwhile, the average using times of each pattern in all versions are far away different. The average using times of Singleton, Bridge, Decorator and Chain of Responsibility are no more than ten, that of Mediator is slightly more than ten, while being different from the other patterns, the average using times of Memento is up to 30.

As Figure 7 shows, seven patterns are used in *blender*: Singleton, Bridge, Decorator, Flyweight, Memento, Mediator and Chain of Responsibility. And except Flyweight, the using times of the other six patterns are no more than 5. The using times of Flyweight goes up along with the software involving, and in the latest versions it goes up so quick that it goes far away from the usual linear growth mode. The other patterns are merely unchanged.

As Figure 8 shows, seven patterns are used in *firefox*: Singleton, Bridge, Decorator, Flyweight, Memento, Mediator and Chain of Responsibility. The using times of Bridge and Decorator are merely unchanged, and those of Singleton, Memento, Mediator and Chain of Responsibility show fluctuations that increase first, and then decrease, at last increase, but the amplitudes is very small. The using times of Flyweight increases to a large extent in the latest two versions.

As Figure 9 shows, seven patterns are used in *eclipse*: Singleton, Bridge, Decorator, Composite, Flyweight, Composite, and Mediator. And except Composite, the using times of the other six patterns are no more than 5. The using times of Flyweight goes up along with the software involving, and in the latest versions it goes up so quick that it goes far away from the usual linear growth mode. The other patterns are merely unchanged.
Application of Design Patterns in Process of Large-Scale Software Evolving

3.3 The Analysis on How Design Patterns are Used in Software Evolving

Known from Table 2 which shows the average using times of each design pattern in the chosen software, the average using times of Memento comes up to 32.71 which takes up of 49.7 percent of the sum of the using times of the six patterns used in abiword. Memento is used to catch the state of an object and store it outside of the object which is up for restoring the object in the future. Text-processing software must remember the state at any moment that can help the users restore the state when it is necessary. For these reasons, Memento is used far more frequently than the other patterns in abiword.

Known from Table 2, the average using times of Flyweight comes up to 24 which takes up of 54.5 percent of the sum of the using times of the seven patterns used in blender. Since the subclasses of class Flyweight are divided into shared data field and unshared data field, Flyweight deals with common graphs and exceptional graphs very well. A large amount of common graphs and exceptional graphs are provided for users in image-processing software. For these reasons, Flyweight is used...
Table 2. The average using times of each design pattern in the software

<table>
<thead>
<tr>
<th>Design Patterns</th>
<th>abiword</th>
<th>blender</th>
<th>firefox</th>
<th>eclipse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singleton</td>
<td>8.75</td>
<td>4.2</td>
<td>54.1</td>
<td>242.8</td>
</tr>
<tr>
<td>Bridge</td>
<td>5.875</td>
<td>2.8</td>
<td>12</td>
<td>14.91</td>
</tr>
<tr>
<td>Decorator</td>
<td>0.8125</td>
<td>1</td>
<td>2.7</td>
<td>25.64</td>
</tr>
<tr>
<td>Composite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.364</td>
</tr>
<tr>
<td>Flyweight</td>
<td>0</td>
<td>24</td>
<td>30.3</td>
<td>3.09</td>
</tr>
<tr>
<td>Observer</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Memento</td>
<td>32.71</td>
<td>3</td>
<td>130.4</td>
<td>0</td>
</tr>
<tr>
<td>Mediator</td>
<td>11.5</td>
<td>5</td>
<td>73.7</td>
<td>56.82</td>
</tr>
<tr>
<td>Chain of Responsibility</td>
<td>6.125</td>
<td>4</td>
<td>54.7</td>
<td>22.73</td>
</tr>
</tbody>
</table>

far more frequently than the other patterns in blender.

Known from Table 2: the using times of Singleton comes up to 54.1 which takes up of 15.1 percent of the sum of the average using times of the seven patterns used in firefox; the using times of Memento comes up to 130.4 which takes up of 36.4 percent of the sum of the average using times of the seven patterns used in firefox; the using times of Mediator comes up to 73.7 which takes up of 20.6 percent of the sum of the average using times of the seven patterns used in firefox. The using times of Chain of Responsibility comes up to 54.7 which takes up of 15.3 percent of the sum of the average using times of the seven patterns used in firefox. Mediator can deal with the communication among the objects implicitly. Chain of Responsibility can make these requests a line and pass these requests along the line till the final processing. Memento can store the reply data being received for client processing. At the same time, web browser takes C/S model, so users will send and receive large amount of data request and reply continuously. For these reasons, Memento, Mediator, Chain of Responsibility and Chain of Responsibility are used far more frequently than the other patterns in eclipse.

Known from Table 2, the average using times of Singleton comes up to 242.8 which takes up of 66.3 percent of the sum of the using times of the seven patterns used in eclipse. The developers have to call the system functions through the interfaces provided by eclipse when they use eclipse and these system interfaces permit being called but not changed, while Singleton can prevent the change made by developers when they use these interfaces. For these reasons, Singleton is used far more frequently than the other patterns in eclipse.

Definition 1: the using times of some design patterns takes up more than 50 percent of the sum of the using times of all the patterns in software, then this pattern is key pattern.

According to the Definition 1, the key pattern of software is similar to the key in database. Since the key pattern decides the main function of software, it can be some kind of symbol of the software. Based on the Definition 1, the key pattern of abiword is Memento; the key pattern of blender is Flyweight; the key patterns of firefox are Memento and Mediator; the key pattern of eclipse is Singleton.

As Figure 6, Figure 7, Figure 8, and Figure 9 show, the using times of Flyweight appears abnormal changes in the last versions of firefox and blender. Through referring to the white books of firefox and blender, we find that the cores of firefox and blender changed too where these abnormal changes happens [16].

4. Conclusions

According what mentioned above, the application and rule of software in the process of evolving is abstracted as follows:

The application of design patterns is changed along with the change of software core and the using times of key pattern of software increases first, then decreases, at last swing around a certain number.

REFERENCES


