Application of Refactoring and Design Pattern in Land Information System Development

CHEN Luyao  YUE Jianwei  JIANG Weiguo  LI Jing  CHEN Yunhao

Abstract  The unceasing change problem of land information systems can be resolved through the refactoring and design pattern. To promote the implementation of design pattern and refactoring methods in developing land information systems reusing software design, applying refactoring methods to the abstract factory and decorator design patterns in land information system development is discussed.

Keywords  design pattern; refactoring; land information system

Introduction

In recent years, land information systems have been applied on a larger scale and become more complex. In addition, the business of land management has been modified due to changes of related factors that have led to a shift in attention from systematic functions to non-functional ones such as modifiability, reliability, and reusability.

Software reusability is a crucial way to develop a land information system which can meet business requests in a relatively short time. Software reusability is a software component design course or a software system assembling process structured in advance and is for reusing[1]. Software reusing covers not only codes, modules, classes (objects) and software products, but also methods, standards, experiences and other knowledge[2]. Software design pattern and refactoring criteria are good examples of knowledge reuse.

Design pattern is a description of the classes which are used in certain scenarios to solve regular design problems, and objects which are used to intercommunicate with each other[3]. Design pattern, solving design problems in certain scenarios and recording the process can be regarded as a paradigm to similar problems in the future, which can reduce the cost of time and money. Different design patterns are made for different changes in software structure, which makes software accommodate certain changes and enable reuse. Therefore, design pattern can be used to address the problem of reusing a land information system when it needs to be changed.

Refactoring refers to the modification of software structure on the premise of fixing the observable behavior by using a series of refactoring criteria to
enhance software intelligibility and reduce the modification cost\[4\]. Requirements and their comprehension are changing every day with changes to the external environment, but refactoring criteria can be the guide when we deal with the changes.

Some domestic scholars use design pattern in geographic information system software development, but rare reports exist on the introduction or application of design pattern and refactoring criteria in land information systems. In this study, enhancing the reusability of land information systems by using design pattern and refactoring criteria is discussed, based on the application of such criteria in system software development.

1 Application of refactoring and design pattern in land information systems

1.1 Extract superclass and abstract factory pattern

In the construction of a land information system, many function controls are needed and involved. For example, print control must have the functions of print random maps and standard maps in a system. Land using a map printing model can be attained by searching for a map control model which has been developed previously. This model can print random maps and standard maps of land use, and the structure is shown in Fig.1.

![Fig.1 Sketch of previous map print class structure](image1)

It can be seen from Fig.1 that both classes can fulfill the function, but they deal with one kind of task in one way, some of which are duplicated or similar to each other. This feature coincides with the extract superclass, which can thus be used to reconstruct the classes. First, create a blank abstract superclass; second, respectively move up the common elements of the two child classes to superclass by using the pull up field, pull up method, and pull up constructor body; then compile and test after moving up once. The class structure after refactoring is shown in Fig.2.

Because the Print_MsgTxt() function of the child class deals with different tasks compared with that of the parent class, the Print_MsgTxt() function of the parent class is defined as a virtual function and redefined in the child class. According to design pattern, the abstract factory pattern can be formed after refactoring: define an abstract class clsPrintMap which contains virtual function Print_MsgTxt(), and define clsPrintFreeTK and clsPrintNormalTK as its child class. The Print_MsgTxt() is fulfilled in the child class, and corresponding child classes are initiated to accomplish the print function when the system calls the print model.

![Fig.2 Sketch of map print class structure after the superclass process](image2)
ate_GLW(), Print_Cross-Lines(), and Print_MsgTxt() are extracted to new classes respectively, and then are compiled and tested after extracting once. During the extracting process, function names are not sufficient to describe their purposes and not uniform, so their names are changed by renaming method refactoring task after extracting. The class structure after extracting and renaming is shown in Fig.3.

After refactoring, the functions of each class became clear and architectures have been improved. However, the class needs further refactoring if the cadastral map print function needs to be added and designed to a general print function in land information systems. The cadastral function also includes the random map print function and standard map print function. Accordingly, it can be accomplished by adding a child class to the clsPrintMapBase class, but the new two classes hold duplicate functions with the former classes. Based on regular rules, there is a multi-inheritance if the parent class is extracted here. To do so is against to the rule that "less inheritance and more aggregation are applied when developing"[3]. The task of refactoring should thus be suspended and map analyses used instead to find better ways to solve this problem.

Fig.3  Sketch of map print class structure after extracting

Analysis suggests that whatever the maps change (e.g., random map print does not need to print an adjacency diagram and standard map print does not need to print geographic coordinates), these maps do not need to print graphic elements, i.e., the key element of a map print is the map itself, while the adjacency diagram, frame, geographic coordinate and scale are all decorators and not the leading actor. To print different maps is to add various decorators to graphic elements. Hence, a decorator model should be adopted to refactoring, which enables the developer to add new map print types more easily and task performance without adding new child classes.

Based on the analysis, the refactoring task runs as follows: first, the extract class method is applied to extract decorator parts from each class (except graphic elements) to create a new class and remove the former two classes. Second, interfaces according to graph element print class are added to the new classes. Finally, the superclass (clsPrintMapDecorator) of the decorator classes is extracted to become the clsPrintMapBase child class together with the graphic element class (clsPrintMap). The Class clsPrintMapDecorator’s object takes responsibility for connecting the decorator object and clsPrintMap object. The instance of these classes can fulfill their functions after they get requests, and then transmit client requests to graph element objects to accomplish the print request.

The pattern structure is shown in Fig.4.

The decorator pattern resolves the problem of adding more responsibilities to an object dynamically. When developers deal with print function, they have options on print map decorator elements that various print types can be created conveniently to meet client requests.
1.3 Application

As the process of refactoring, map print control has been improved step by step under the refactoring criteria. Its transformation/development to decorator pattern means that the former map print control limited to two types of land information system maps has been improved to one that can process various types of maps. Reusability of the control has been enhanced, which makes the control a general print control in land information systems.

The method, which was for design pattern and under the guidance of refactoring, is also acceptable for improving other system controls. For example, the display control of map elements in GIS needs various attribute information for different map elements; other related map element information (e.g., certification delivered parcel-land of cadastral map, clients pay their attention not only on information of certification delivered parcel-land of cadastral map, but also on the information of its obligee) also needs to be shown to clients. To handle the task which has only one operation but several processing modes, a strategic pattern should be adopted to improve the original pattern with refactoring criteria. The crux of the implementation is the selection of the design pattern and how to process refactoring correctly with refactoring criteria to fulfill that pattern.

This method was adopted in the development of the Liuzhou Land Information System. It can reduce the difficulty of development and increase speed and quality significantly. Also, the feasibility of the method is verified. Since information systems in many fields have already been developed and accumulated a considerable number of documents and function models, this method offers the option of taking full advantage of previous system experiences and reduce duplication of labor.

2 Conclusions

Design pattern takes advantage of valuable experiences of object-oriented design, and its application in software design can enhance system openness, compatibility, stability and extension, which can make system development and maintenance easier. It is not hard to understand the pattern itself, but pattern selection and pattern application always hamper the development process[4]. However, refactoring is the guide for developers to select and apply the appropriate patterns. Refactoring can reduce the difficulties and alleviate pressure of designing[5]. Refactoring can help developers take full advantage of former experiences and reduce duplication of labor. Refactoring and design pattern complement each other: design pattern shows the way for refactoring and refactoring is the purpose of design pattern. This study discusses the application cases of design pattern and refactoring in land information systems.

References

[1] McClure C(1997) Software reuse techniques adding reuse to the systems development process[M]. Boston: Pearson Education
[2] Qu Honggang, Pan Mao, Wang Yong, et al.(2003) The research of design patterns on the development of GIS software[J]. Computer Engineering and Applications, 39(25): 1-4 (in Chinese)
[3] Gamma E, Helm R, Johnson R, et al.(1995) Design patterns: elements of reusable object-oriented software[M]. Boston: Addison-Wesley
[4] Zou Juan, Tian Yumin(2004) Selection and realization for software design patterns[J]. Computer Engineering, 30(10): 79-81 (in Chinese)
[5] Fowler M(1999) Refactoring: improving the design of existing code[M]. Boston: Addison-Wesley