2011 International Conference on Modeling, Simulation and Control
IPCSIT vol.10 (2011) © (2011) IACSIT Press, Singapore

A Survey on the Maintenance of Software Structure in Thai Software Industries

Panita Meananeatra [1], Songsakdi Rongviriyapanish [1] and Taweesup Apiwattanapong[2]

[1] Computer Science Department, Thammasat University, Pathumthani, Thailand

[2] Software Engineering Laboratory, National Electronics and Computer Technology Center, Pathumthani, Thailand

Abstract. "One who knows the enemy and knows himself will not be in danger in a hundred battles", Sun Tzu, who is a Chinese scholar, said. The same saying is true in the software maintenance area. If we want to resolve maintenance problems, we should understand their root causes. However, existing research shows that most development teams poorly maintain software because they lack time, maintenance knowledge and support tools. Unfortunately, no research surveys the maintenance practice of the software structure in Thailand's software industries for knowing the current status. Therefore, the purpose of this survey is to find out the current status of that practice in Thai software industries. Our survey focuses on four topics: awareness, techniques, problems in that practice and refactoring practices. We use three data collection methods: material survey, web survey and telephone interview. The material survey results show the background of software maintenance. The web survey and telephone interview collected the opinions of interviewees at the organization level and the personal level on those four topics. Moreover, we conclude that Thai software industries are aware of the importance of the maintenance practice, and they would rather use a design pattern technique than a refactoring technique. In addition, they want a tool that helps suggest refactoring for improving the maintainability of the software structure.

Keywords: Software development, software maintenance, software structure

1. Introduction

In software industries, software development phases are the most important key to match the needs and qualities that customers desire. Sometimes products are being developed continuously; therefore, products should be easily maintainable before the beginning of later phases. For example, at the end of the design phase, the designer should maintain the structure in such a way that it can be changed easily. Similarly, at the end of the implementation phase should maintain the code in such a way that it can be read, understood, and changed easily. Moreover, most software needs to be change (e.g., requirement change, design change, technology change), so developers should modify the software. Modification affects the systematic design and code. The software becomes more complicated, so it is not easily understood and enhanced. The well-maintained software will be easily understood and developed. Software maintenance decreases the rework cost in development.

The definition of software maintenance by IEEE is as follows [3]: "The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment". According to Lientz and Swanson, there are four types of maintenance: corrective, adaptive, perfective, and preventive [5, 6]. Adaptive changes are made the software environment changes, and perfective changes are made to accommodate new user requirements. Corrective changes are made to fix errors, and preventive changes are made to prevent problems in the future. Corrective, adaptive and perfective changes have the long-term effects of increasing the system's complexity [7]. Unlike others, preventive maintenance makes programs easier to understand and hence facilitaties future maintenance

works [7]. Examples of preventive maintenance include restructuring and optimizing code as well as updating documentation. Two well-known restructure techniques are Design Pattern and Refactoring.

A Design pattern is a consensus on the most efficient solution to solve a given problem [9]. The use of a pattern guarantees good analysis and design practices. Refactoring is a technique for improving software structure without changing its behaviour[2]. A bad smell is defined as a program characteristic that indicates the need to use the design pattern [9] and to refactor a program[2]. Most bad quality programs have bad smells. Bad smell removal increases qualities such as understandability, maintainability and modifiability. However, existing research shows that most development teams poorly maintain software due to lack of time, maintenance knowledge and tools.

Thailand has many software houses, companies and organizations. Unfortunately, no research survey on the maintenance practice of software structure in Thailand's industry has been conducted. Therefore, the purpose of this survey is to investigate the current status on that practice in Thai software industries". Our survey focuses on four topics: 1) attitude about awareness, 2) techniques (e.g. design pattern and refactoring), 3) problems in that practice and 4) refactoring practice (how to apply, problems when they use). This paper is organized as follows. Section 1 introduces important background for implementing this survey. Section 2 explains the steps of this survey. Section 3 describes the results of three data collections: material survey, web page survey and telephone interview. Finally, Section 4 discusses conclusions and future work.

2. Methodology

This section explains the six steps of this survey: 1) establish the goal of the survey, 2) determine samples, 3) choose data collection methodologies, 4) crate questionnaires, 5) conduct surveys, and 6) analyze the data. In the first step, the goal of survey is "to investigate the current status on the maintenance practice of software structure in Thailand's industries". In the second step, the target group is software development teams in software houses or organizations that develop software (five companies). Data were collected through three methods: material survey, web survey, and telephone interview. Material survey has questions about basic background of this survey. After that, questionnaires of the web survey and telephone interview are created by using questions. Web survey has 15 questions that are organized into three parts: the general information of interviewees, the opinions of interviewees at the organization level, and opinions of interviewees at the personal level. We use Google Doc to create the web survey as shown in Fig 1. Telephone interview focuses on refactoring process and the problems that occurred when developers apply refactorings. In the survey step, we planned and gathered data. The plan has four parts; 1) material survey, we searched background of software maintenance, 2) web survey, we used data from previous part for creating questionnaire, 3) the questionnaire sent to interviewees by email, waited response, and summarized response, 4) telephone interview, we selected interviewees for interviewing from response. The last step, analyze the data, was shown in the next section. This survey lasted in six weeks.

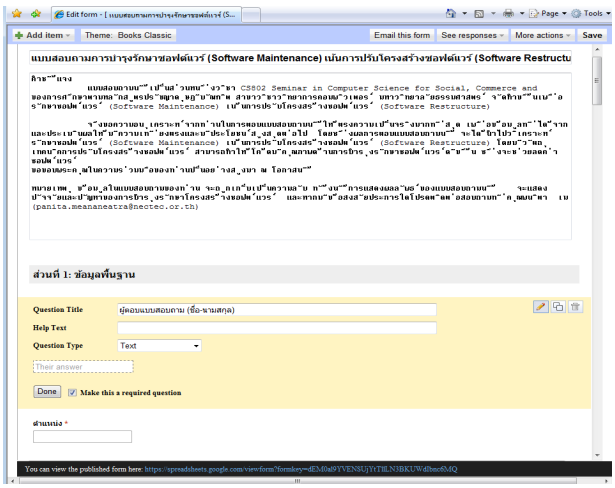

Fig. 1 (a): design form of questionnaire on Google Doc

Fig. 1 (b): questionnaire web page on Google Doc

3. Results and Discussion

This section discusses the results of three surveys; material survey, web survey, and telephone interview.

3.1. Material survey

Software development includes many phases: design, implementation, testing, deployment, and maintenance. The product should be easily maintainable before deriver in earlier phases. For example, the design phase should plan the structure in a way that structure can be easily changed. Similarly, the implementation phase should create code that can be easily read, understood, and changed. Most software projects, maintenance phase consumes a large part of the overall lifecycle costs and is required many changes. Therefore, software maintenance is necessary and helps to increase business opportunities [4]. The four major problems that can slow down the maintenance process are unstructured code, maintenance programmers having insufficient knowledge of the system, documentation being absent, out of date or insufficient, and unawareness to the importance of software maintenance [1]. The definition of software maintenance by IEEE is as follows [3]: "The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment". According to Lientz and Swanson, there are four types of maintenance: corrective, adaptive, perfective, and preventive [5, 6]. Corrective maintenance deals with fixing bugs in the code. Adaptive maintenance deals with adapting the software to new environments. Perfective maintenance deals with updating the software according to changes in user requirement. Finally, preventive maintenance deals with updating documentation and making the software more maintainable, such as updating documentation, adding comments, and improving the modular structure of the system [6]. The survey showed that around 75% of the maintenance effort was on the first two types, and error correction consumed about 21%. Many studies suggest similar results. These studies show that the new user requirements are the core problem for software evolution and maintenance.

Corrective, adaptive and perfective changes have the long-term effects of increasing the system's complexity [7]. A large program is continuously changed, thus, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it. This work is known as preventive change. The change is usually initiated from within the maintenance organization with the intention of making programs easier to understand and hence facilitaties future maintenance work [7]. Example of preventive change includes restructuring and optimizing code as well as updating documentation.

The term "restructuring" means reorganizing the procedural logic of a computer program so that it conforms to the rules of structured programming [8]. Two well-known restructure techniques are Design Pattern and Refactoring. A design pattern is a consensus on the most efficient solution to solve a given problem [9]. The use of a pattern guarantees good analysis and design practices. Refactoring is a technique for improving software structure without changing its behaviour[2]. A bad smell is defined as a program characteristic that indicates the need to use the design pattern [9] and to refactor a program[2]. Most bad quality programs have bad smells. Bad smell removal increases qualities such as understandability, maintainability and modifiability. However, existing research shows that most development teams poorly maintain software due to lack of time, maintenance knowledge and tools.

3.2. Web page survey

The questionnaire is divided into three parts: the general information of interviewees, the opinions of interviewees at organization level and at personal level. We gathered data by using the spreadsheet that is generated from Google Doc. A total of 32 respondents in 17 companies completed the survey. These participants have different roles in software development. These roles are senior manager, project manager, business analyst, software architect, software analyst, developer, tester, quality person, and marketing coordinator. Some participants may play more than one role in their projects. There are many programming languages used in the development, such as, Java, C++, PHP, MS C#, MS VB, Python, C, Perl, PL/SQL, Dephi and shell script. The result from responses showed that the number of respondents who used object-oriented languages: Java, C++ and MS C# is more than the one who use traditional languages. From our objective, we discuss three topics: awareness, techniques, problems in this section.

a. Awareness

In the study, 29 of 32 respondents felt that directors of their organizations support to the maintenance of software structure as seen from related activities: quality audit, training on the topic of software structure improvement and announced policy. It implies that most organizations have awareness on the maintenance of software structure. However, three respondents express their opinion that the directors of their organizations currently do not support the maintenance of software structure because of the lack of knowledge in maintenance, staffs and time. Two of them stated that the directors will support to improve maintenance of software structure in the future, but the other could not provide the answer.

In the opinions of interviewees at the organizations level and at personal level, they improved software structure in five phases: designing, implementation, testing, maintenance, and deployment. Moreover, they are awareness of the maintenance of software structure themself. The personal level, most interviewees improved software structure in the implementation phase more than the other phases because it helps them to easily understand, modify and reuse code.

b. Techniques

Two well-known techniques for improvement in software structure are design pattern and refactoring. At the organization level, 25 respondents used design pattern, and 20 respondents used refactoring. Some participants used both techniques in their projects. At the personal level, 16 respondents used design pattern and 11 respondents used refactoring. Therefore, design pattern are used more than refactoring at both the organization level and the personal level.

c. Problems

The problems in the improvement of software structure include six issues: time limit, developer skill, no standard of software structure improvement, rapid technology changes, diverse platforms and environments and complicated software design.

3.3. Telephone survey

In term of the respondents, refactoring skill, there are seven persons out of 32 who have one to ten-year experience. The result of the telephone interview came from four interviewees. Our other objective, we discuss the details of refactoring activities in four topics: motivation for refactoring, methods for assessing code that is refactored, the problem in refactoring activities and the developer suggestion.

For the motivation to refactor, all four interviewees refactor code when requirement changes occur, where as only two interviewees do when a bad smell occurs after coding.

The assessment of effectiveness of refactoring applied has four methods; 1) requirement matching with software functions or compile and test pass, 2) the assessment of understandability with an owner developer and a reviewer, 3) changes in execute time and 4) software metric (e.g. line of code) comparison between before applied refactorings and after them.

Three problems in refactoring activities are: 1) Developer Skill in refactoring, such as less experience and do not know how to select refactoring, framework and design pattern, 2) developer Skill in Tool such as do not know which tool or plug in support apply refactoring and design pattern and do not know how to apply program element using tool, and 3) tool such as cannot analysis and summarize of result after apply refactoring.

The suggestion of developers, they want tool that can present reasons of change, show requirement change impact which classes, illustrate refactor impact classes, and help to decide whether to refactor.

4. Conclusion and Future Work

This survey aims to investigate “current status on the maintenance practice of software structure in Thai software industry” and focuses on four topics in software structure: awareness, techniques, problems in that practice and refactoring practice. This survey consists of six steps: 1) establish the goal of the survey, 2) determine samples, 3) choose data collection methodologies, 4) crate questionnaires, 5) conduct surveys, and 6) analyze the data. The target group is software development teams in software houses or organizations that

develop software (five companies). Data were collected through three methods: material survey, web survey, and telephone interview.

The results of three methods reveal many findings. The first result, material survey, shows the answers to many questions such as "What is software maintenance and why software maintenance is important?", "What are techniques in software maintenance?", and "What problems are in software maintenance?" Next, the result of web survey is divided into three parts. 32 persons from 17 companies participated. 1) In general Information, programming languages used are Java, PHP, C++, MS C#, MS VB, and other. 2) At the organization level, 29 respondents felt that directors of their organizations support to the maintenance of software structure. However, three respondents express their opinion that the directors of their organizations currently do not support the maintenance of software because of the lack of maintenance knowledge, staffs, executive support and time. 3) At the personal level, developers improve software structure during five development phases and are aware of the importance of improving the software structure. Developers prefer design pattern than refactoring. The problems of software structure improvement are the lack of resource, the limitation of organization standard, limited developer skill, and complicated software design. Only seven persons out of 32 have experience in refactoring. For the result of telephone interview of four interviewees, all interviewees refactor code when requirement changes occur, where as only two do so when a bad smell occurs after coding. Participants use four methods for assessing the effectiveness of the refactoring applied. The problems in refactoring activity can be grouped into three categories: developer skill in refactoring, developer skill in tool, and the lack of tools.

The discussion focuses on the same four topics as the survey. This survey shows that Thai software industries are aware of the importance of the maintenance practice. Thai developers prefer the design pattern technique than the refactoring technique. Also, more developers refactor code when requirement changes occur than when bad smells are found. In addition, developers want tool that can present reasons of change, show classes that are affected by requirement changes and help decide whether to refactor.

5. References

[1] Y. Kataoka, T.Imai, H. Andou and T. Fukaya. Quantitative Evaluation of Maintainability Enhancement by Refactoring. in Proceedings of the International Conference on Software Maintenance (ICSM'02), 2002.

[2] M. Fowler. Refactoring: Improving the Design of Existing Programs. Addison-Wesley, 1999.

[3] IEEE Std. 1219: Standard for Software Maintenance. Los Alamitos CA., USA. IEEE Computer Society Press, 1993.

[4] K.H. Bennett and V.T. Rajlich. Software Maintenance and Evolution: a Roadmap. In ICSE 2000: Proceedings of the Conference on The Future of Software Engineering, pp. 73-87, 2000.

[5] B. P. Lientz and E. Swanson, Software Maintenance Management. Addison Wesley, Reading, MA, 1980.

[6] H. V. Vliet, Software Engineering: Principles and practices, 2nd Edition. John Wiley & Sons, West Sussex, England, 2000.

[7] A. A. Takang and P. A. Grubb , Software Maintenance Concepts and Practic. Thompson Computer Press London, UK,1996.

[8] E. Yourdon. Re-3, Part 1: Re-engineering, Restructuring, and reverse Engineering. American Programmer, vol. 2, No. 4, Apr. 1989, pp 3-10.

[9] A.L. Baroni, Y. -G. Gueheneuc and H. Albin-Amiot. Design patterns formalization. Rapport de recherche, Department d'informatique, Ecole des Mines de Nantes, number 01/01/INFP, 2003.