ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 4, April 2018

A Survey on the issues of Refactoring

[1] Aparna K S, [2] Sai Deepthi.K, [3] Sharan Kumar.R, [4] Syeda Shameemunnisa, [5] Sree Geethika.M
[1] Assistant Professor, Dept. of CSE, RYMEC, Ballari
[2][3][4][5] B.Tech 4thSemester, Dept. of CSE, RYMEC, Ballari

Abstract: - **In this paper, a tremendous attitude of research towards refactoring the software is presented. This paper presents the areas where refactoring can be applied. The Scope of refactoring, how to refactor, when to refactor, impact of refactoring an a quality of software, different methods used, basic principles in refactoring software artifacts and the related work which needs to improve are presented. This paves the way for improvement, giving rise to many more research fields.**

Keywords: - **Refactoring, software re-engineering, formal methods.**

I. INTRODUCTION

A major part in software life cycle[1], is the software maintenance and majority of time and budget is invested in software evolution, which inturn depend on software maintenance. This software maintenance will be done using variety of techniques. Refactoring is one of the technique to improve the internal quality of the software without changing the external behavior. Refactoring also plays a vital role in software re-engineering, whose main objective is to restructure the legacy software. The most fascinating part is to estimate the parts of legacy software, which have to undergo changes, considering the parameter the software re-engineers are facing and the impact of these on evolving softwares. refactoring has wide scope in transforming the existing design code into a form, understandable by the reverse engineering tools and techniques. Refactoring[2] also fits in model-driven re-engineering process to implement platform migration by code generation from basic-modeling methods. Refactoring also happens to be one of the milestones in eXtreme-programming which is known to be arethe main proponent in ajile softwares are rapid application software development .Refactoring plays a quality improving role in software evolution, which plays a major role in software industries.

The remainder of the paper is structured as follows section II explains the stagesof refactoring and catagories of refactoring such as class method and attribute refactoring. Section III identifies the various issues involved like effects of refactoring on quality. Section IV discusses the impact of refactoring on software process, advantages and disadvantages

Basic principles used in Refactoring:-

1) The Dependency Inversion Principle: The states that depend an abstraction (interface) not an implementation.

2) The Interface Segregation Principle:This states that multiple small interfaces are better than one ' fat ' interface (big).

3) Acyclic Dependencies Principle: Dependencies between packages, classes or any resources should not form a cycle leading to a deadlock.

4) Common Closure Principle: States that any change made at one point should be reflected at all points, which -dependent.

5) Common Reuse Principle:If any subcomponent of the main component is used, then all other sub-components belonging to the main should be used.

II SOFTWARE REFACTORINGPROCESS

Commonly, the software refactoring process includes the following steps and activities.

1. Apply unit test to the program.

2. Identify which part of the code needs the refactoring using code smells.

3. Select a refactoring technique to remove the identified code smell.

4. Apply the selected refactoring technique.

5. Apply regression testing to the refactored code.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 4, April 2018

6. Assess the effect of the refactoring using software quality characters or the process.

7. Maintain consistency between the program and the other artifacts.

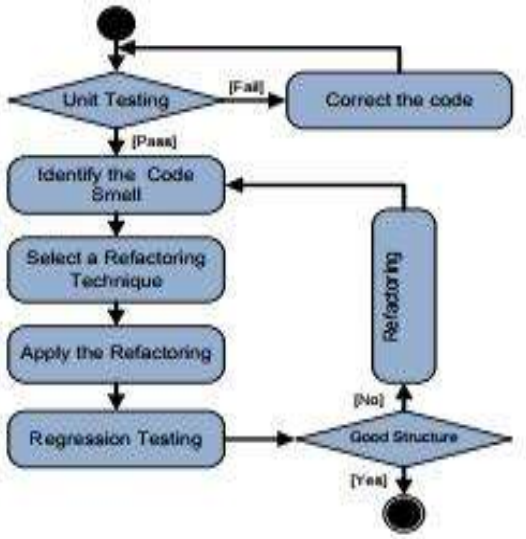

Figure1. The Steps in Refactoring Process

Refactoring Activities

1. Identification where the software has to be refactored,
2. Determination of which refactoring shall be applied.
3. Behavioralpresentation.
4. Apply refactoring.
5. Implement on refactoring on software.
6. Check the consistency between the manufactured program code and other software artifacts.

CATEGORIES OF REFACTORINGS

Class Refactoring:

- add (sub)class to hierarchy
- rename class
- remove class
- extract class

Method Refactoring:

- add method to class
- rename method
- remove method
- push method down
- push method up
- add parameter to method
- move method
- extract code in new method
- replace parameter with method

Attribute Refactoring:

- add variable to class
- rename variable
- remove variable
- push variable down
- pull variable up
- create accessors
- abstract variable

III. ISSUES IN SOFTWARE REFACTORING

Identifying the areas of applying Refactoring:
The major decision is to identify the level of abstraction, where the refactoring has to be applied. The dilemma is whether the refactoring should be applied to the program itself or to more abstract software design models or requirement documents.The widespread and popular approach to detect refactoring areas is to find the bad smell codes.As per the Kent Back, "bad smells are the structures that needs to be refactored "[7].Balazinske uses a clone analysis tool to identify duplicated code , suggesting candidates for refactoring [15].The surveys conclude that extracting duplicate code and inserting an intermediate subclass to factor out the common code [17] [18] [19] [20] .

Accessing the effects of refactoring on quality:
The quality attributes of a software such as robustness, extensibility,performance, allows us to improve the quality by application of refactoring at relevant points.Performance of a software can be improved by the concept of refactoring. Coupling metrics can be used to determine the effect of refactoring on maintainability of the program. There are soft goal graphs, depicts design decisions. These association o refactoring with a possible effect on soft goals, claims maintainability enhancements through primitive and non-primitive refactoring.

Causes and solutions for the bad smells in the code:

1. Duplicated code: In this case, we find the same code structure at several points The solution is to perform extract method and invoke the code from both places, using pull-up method.

2. Long method: whose statements operate on different parameters/ variables. The solution is to extract method and improve responsibilities distribution.

3. Large class: If a class has too many responsibilities it often has many instance variables. In such a case,

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 4, April 2018

perform extract (sub) class and improve responsibilities distribution.

4. Long-parameter list: If a method has a very long parameter list then replace the parameters as an object.

5. Divergent-change: This occurs when class is commonly changed by different aspects.This class should be identified and use extract method to put them all together.

6. Shotgun surgery: A Small modification in a program leads to lot of little change to a lot of different classes. The solution is to use either move method or move field to segregate all changes in a single class.

7. Feature envy: Any method making heavy use of data from another class. This can be solved using move/extract method to put in single class.

8. Lazy class: If a class is not doing good job, then use collapse hierarchy concept and all useless components should be subjected to inline –class.

IV. SOFTWARE PROCESS SUPPORT BY REFACTORING

The following Implications are made in Refactoring-

- Refactoring doesn't change the system.
- Refactoring doesn't mean rewriting from scratch.
- Refactoring is not just any restricting method to improve the code.

The goals of refactoring are to remove code duplication and improve comprehension and maintenance, reducing coupling. Refactoring uses ' the rule of three ' - defined by fowler is ' The first time you do something, you just do it. The second time you do something similar, you look at the duplication but you still do the duplication anyway. The third time you do something similar - you refactor. Refactoring supports the following research areas. In the process of Software Re-engineering, Refactoring fits very well in the process of software re-engineering,where it modifies the software to implement a new solution. Refactoring helps to identify parts of the legacysoftware to be converted and the process of conversion. Refactoring also supports model-driven re-engineering process,(which facilitates platform migration by code generation from abstract model).In this case, refactoring transform the design of existing code into form, understandable by MDA tools.Refactoring programs written in object-oriented language is more difficult to restructure because of tight data flow and control flow. The object-oriented
principles face difficulty due to inheritance mechanism, dynamic binding, overriding, polymorphism. Refactoring software artifacts has made the following assumptions

• Refactoring at design level can be done for class diagrams and activity diagrams.

• Design-patterns provides program description at high level of abstraction. Refactoring introduce new design-pattern instances into the software.

• Object oriented database schemes are ideal points for refactoring.

• Software architecture are refactored directly on the graphical representation of system architecture.

The Code refactoring has the following

Advantages:

- Makes code more readable
- Removes redundant, unused code and comments
- Improves performance
- Creates reusable code and is easier to maintain

Disadvantages:

- Database migration
- Published interfaces

V. CONCLUSION

Refactoring is a well-defined process that improves the quality of systems and allows developers to repair code that is hard to maintain. By careful application of refactoring the system's behavior will be same. Use of automated refactoring tools makes the developer perform necessary refactoring since tools are much quicker and reduce the bugs, but still there are many research issues to be tackled in using the refactoring tools. In each of these categories there are important open issues that remain to be solved. In general, a small approach was made to find the formalisms, processes, methods and tools that address refactoring in a more consistent, generic, scalable and flexible way. By the end of the paper we conclude that there is a lot of scope in restructuring and refactoring areas.

REFERENCES

[1]. MesfinAbebe and Cheol-Jung Yoo. Trends, Opportunities and Challenges of Software Refactoring: A Systematic Literature Review.

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

[3].http://sourcemaking.com /refactoring /defining-refactoring.

[4]. H. Liu, Z. Ma, W. Shao and Z. Niu, "Schedule of Bad Smell Detection and Resolution: A New Way to Save Effort", IEEE Transactions on Software Engineering, (2012).

[5]. [1] D. M. Coleman, D. Ash, B. Lowther, and P. W. Oman, "Using metrics to evaluate software system maintainability," IEEE Computer, vol. 27, no. 8, pp. 44–49, August 1994.

[6]. T. Guimaraes, "Managing application program maintenance expenditure," Comm. ACM, vol. 26, no. 10, pp. 739–746, 1983.

[7]. B. P. Lientz and E. B. Swanson, Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations, Addison-Wesley,1980

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

[9]. M. Balazinska, E. Merlo, M. Dagenais, B. Lag¨ueandKostas Kontogiannis, "Advanced clone-analysis to support objectoriented system refactoring," in Proc. Working Conf. Reverse Engineering. 2000, pp. 98-107, IEEE Computer Society.

[10]. S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," in Proc. Int'l Conf.Software Maintenance, 1999, pp. 109–

118, IEEE Computer Society.

[11]. T. Tourw´e and T. Mens, "Identifying refactoring opportunities using logic meta programming," in Proc. Int'l Conf. Software Maintenance and Re-engineering. 2003, pp. 91–100, IEEE Computer Society.

[12]. E. van Emden and L. Moonen, "Java quality assurance by detecting code smells," in Proc. Working Conf. Reverse Engineering 2002, pp. 97-108, IEEE Computer Society.

[13]. T. Dudziak and J. Wloka, "Tool-supported discovery and refactoring Computer Science, Technical University of Berlin, February 2002.

[14]. F. Simon, F. Steinbr¨uckner, and C. Lewerentz, "Metrics based refactoring," in Proc. European Conf. Software Maintenance and Reengineering. 2001, pp. 30–38, IEEE Computer Society.

[15]. L. Tahvildari and K. Kontogiannis, "A methodology for developing transformations using the maintainability soft-goal graph," in Proc. Working Conf. Reverse Engineering. 2002, pp. 77–86, IEEE Computer Society.