A Comprehensive Survey on Refactoring Paraphrase Tools and its Detection Methods

M. Sangeetha
Head & Asst. Professor, Department of Computer Science,
Shakthikailash Womens College, Dharmapuri, India

Dr. V. Sangeetha
Asst. Professor, Department of Computer Science,
PRUCAS, Dharmapuri, India

Abstract- With refactoring we preserve obtain a bad design, chaos even, and rework it hooked on well-designed code. In our anticipated approach, we can handle the time factor and detect resolution using different strategy. Most probably trying to optimize the work load get better source code for easy preservations. Our idea incorporated to proposes an optimal refactoring plan that augments detection and finding of bad smells. Create and construct a framework for automatically generating revealing algorithms for the formalized bad smells. The problem makes it very clear vision, where the smell being refactored might have dependency in increasing or resolving some other kind of smell which in turn results in increased effort and time. Before that to spotlight which kind of source code can be handled to restructure. This paper provides to formulate the vital to obtain the observers of refactoring to clutch exposed more survey on software refactoring rapidly.

Keywords – Refactoring, Code detection, Bad smells, Refactoring tools, Automated Tools.

I. Introduction

Developers should regularly refactor the code as per the Standard code lines. Refactoring leads to constant improvement in software quality while providing reusable, modular and service oriented components. It is a disciplined and controlled technique for improving the software code by changing the internal structure of code without affecting the functionalities[9]. In software engineering methodology, Refactoring progress is better to design the Software and crafts software easier to understand. It helps which is used to find out the bugs. "Restructuring of internal structure of object oriented software to improve the quality while the software's external behavior remains unchanged". According to Fowler design problems appear as "bad smells" at code or design problems appear as "bad smells" at code or design level and the process of removing them is called refactoring where the software structure is improved without any modification in the behavior. Distinct refactoring may perhaps pilot into numerous other typical refactoring schemes. In the circumstance of test driven development, code is refactored to eliminate the duplication and articulate objective as soon as it is written and gets ahead of the tests. When refactoring is used to modify the software, it improves readability, maintainability, and extensibility of the code. The Ultimate aim of our refactoring is to formulate code easier to retain and reusable in the future.

II. Refactoring Process Process Facts And Its Activities

The nomenclatures discriminate tools within numerous dimensions. The evaluation of the proposed technique consists of three parts. For the remainder of this survey it is sufficient to examine tools only in terms of their degree of automation following:

a) Manual: Refactoring tools automate behavior preserving program transformations. If the actual act of transformation is left up to the user, the tool cannot be considered a refactoring tool.

b) Semi-Automated: Semi-automated approaches are in-between the extremes of manual and full-automated refactoring. It is up to the user to identify the parts of the software to be refactored and to choose the refactoring. The tool may assist in the decision process by proposing refactoring opportunities, but the decision maker is the user. It is up to him to trigger the transformation.

c) Fully-Automated:Fully-automated refactoring approaches eliminate the user as the middle-man between the detection of a refactoring opportunity and the execution of the refactoring. When software systems evolve, a design can easily deteriorate. Fully-automated refactoring tools aim to help a programmer to reverse any such deterioration and to discover new and more suitable designs.

The refactoring process consists of an add up to dissimilar types of activities, each which can be automated to a firm coverage and The models do in a certain degree resemble the refactoring activities presented in this section. This is a good sign, as this means that there is a correspondence of how refactoring tools work and how programmers think about refactoring their code.

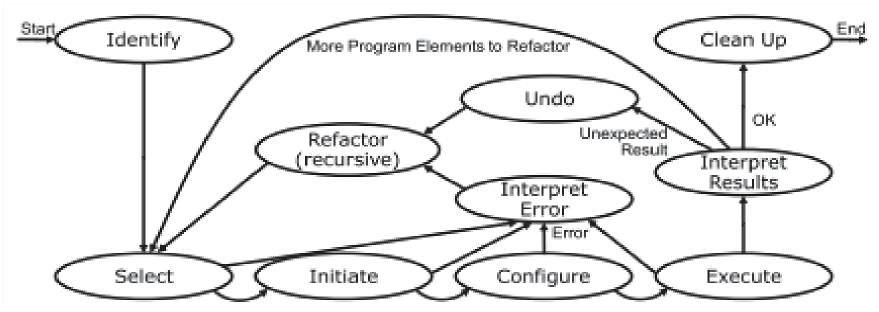

Figure 2.0: A model of how programmers use conventional refactoring tools.

1. Identify where the code should be refactored.
2. Analyze the source code and discover out where the Code should have to Refactored.
3. Determine when the Code should be refactor to identified the locate places
4. Mention why the Code Should be Refactor
5. Assess the type of Bad Smells and other software artifacts
6. Which Refactoring technique is more suitable to refactor the particular type of bad Smells?
7. Effect the refactoring of the Code.

III. Relate Works In The Litarature Reviews

Detection tool be able to exploit in detecting a specialized scheme of smell. A refactoring classification is used where the system is comprehensively refactored at one endeavor. Higher probability of the existing refactored code to penetrate new smells. Martin Fowler had been introduced codes have bad smells are rigid to retain and rigid to modify[1]. Already many more bad smell-level Schemes were introduced to detected and resolved using few automated tools like JDeodarant(Feature Envy).

A.Lozano et al.,[2] appraised the impact of bad smells, which means the degree to which dissimilar bad smells manipulate software maintainability. With the impact of the technique, might be possible to appraise the eminence of code according to the visual depiction. M. Fokaefs et al., [3] proposed an Eclipse plug-in type to identify and resolve feature envy bad smells. I). Identification of defects to detecting, II). Corrected the above said identified defects.

Occasionally developers may fail to invoke refactoring tools and smells detection tools which might be result in delay of refactoring and results in higher cost of refactoring. Almas Hamid, Muhammed Ilyas, Muhammad Hummanyun and Asad Nawaz[6] converse with the intention of refactoring is a technique to compose a program's source code more legible, proficient and maintainable.

Pietrzak and Walter[8] investigated the intersmell relationships to facilitate the detection of bad smells. They argued that detected or rejected bad smells might imply the existence or absence of other bad smells.

Francesca atArcelli Fontana[4] discuss in his article to facilitate recent research is active in detection tools to help the developer to detect the bad smells when the code becomes difficult to manage and understand.

Naouel moha[5], DETEX and DÉCOR methods are useful for identification of antipatterns, code smells with the help of existing literature concepts of code smells.

IV. Feasibility Analysis Of Software Refactoring Techniques And Tools

A. Techniques

As per the analysis wise[10], extensive assortment of techniques and formalisms have been projected and used to covenant with restructuring and refactoring.

- ✓ *Assertions* (invariants, pre-, and postconditions) can be used to express properties that should hold before and after the refactoring has been applied.
- ✓ *Graph transformations* provide an underlying theory of refactoring [8]. Each refactoring corresponds to a graph production rule, and each refactoring application corresponds to a graph transformation. The theory of graph transformation can be used to reason about applying refactorings in parallel, using theoretical concepts such as confluence and critical pair analysis.
- ✓ *Program slicing* can be used to guarantee that a restructuring preserves some selected behaviour of interest. It has been used to deal with program restructurings such as function extraction.
- ✓ *Software metrics* can be used before a refactoring, to measure the quality of a software system, and to identify places that need refactoring. It can also be applied after the refactoring, to measure improvements of the software quality.
- ✓ *Formal concept analysis* can be used to restructure object-oriented class hierarchies in a behaviour preserving way.
- ✓ *Program refinement* can be used to formally express program refinements in a behaviour preserving way.

B. Tools

Numerous companies preserve the availability of defect repositories where defects in projects beneath development are physically identified, corrected and documented. However, this valuable acquaintance is not used to excavation regularities about defect manifestations.

I. Eclipse: Eclipse is one of the typical Integrated Development Environment Tool, which is supposed to a workspace and an extensible plug-in for customizing the tool environment. Where written in independent platform using Java language and developed application. It may also helpful to create the application in C, C++, PHP and COBOL and so on. Especially it can be used to detect the bad smells in the code.

II. JDeodarant: Jdeodarant is not only detecting bad smells and helps to resolve them by applying appropriate refactoring. It provides a variety of methods and techniques in order to identify the bad smells and suggest the appropriate refactoring techniques that resolve them. It is used to detect four types of bad smells, namely “Feature Envy”, “State Checking”, “Long Method”, and “God class”.

III. PMD: PMD provides quick fix for bad smells. It helps to analyze the code, Highlight the problem and fix the problems automatically. It can be detect 5 types of bad smells. Where PMD detect the following: “Large Class”, “Long Method”, Duplicate Code”, “Dead Code” and “Large Parameter List”.

IV. IDEA: IDEA is a Commercial development environment created by IntelliJ. JetBrains creates the IntelliJ, Idea is a Integrated Development Environment in java. Idea has the refactoring features like Rename Method for Package, Class, field, Method Parameter and Local Variable, Extract Method, Introduce Variable.

V. Smalltalk Refactoring Browser: Smalltalk is very neat and clean language which has more automatic processing than other languages such as C++. Where the Smalltalk refactoring browser that used for implementing most of the standard classes, methods and fields refactoring for the Smalltalk Language[7].

While Refactoring can be applied to any programming language, the preponderance of refactoring contemporary tools has been developed for the platform independent language.

V. Scope Of The Refactoring Operations And Its Identifications

Without refactoring, the devise of the program will crumble. As natives transform the code-changes to realize short-term goals or amends made without a full conception of the design of the code-the code loses its structure. It becomes harder to perceive the design by reading the code. Refactoring is rather like tidying up the code and regular refactoring helps code retain its shape.

Refactoring, the practice of varying the structure of code without changing the way a program behaves, is a potentially constructive technique for building and maintaining code. For exemplar, using the Extract Method refactoring, a programmer may eliminate duplicated code by putting it into a novel method and calling that novel method instead. Semi automated refactoring tools in mainly commercial programming environments mitigate the programmer from having to formulate tedious and error-prone changes by hand. For example, a tool can encapsulate a field using getter and setter methods, and the tool will automatically replace all direct references to the meadow with references to the new methods. In this way, a refactoring tool offers the prospective for tremendous increases in refactoring productivity[9].

We engender feasible elucidation based upon the refactoring operations primitives and deemed into the following objectives:

1. Maximizing code quality by minimizing the number of detected defects using detection rules generated
2. Minimizing the effort needed to apply refactoring operations.

Reducing the amount of code won't make the system run any faster, because the effect on trail of the programs hardly ever is momentous.

VI. CONCLUSION

Sometimes Developers may have an inexperienced way to identify the software refactoring tools and methods. The consequence is to create the code further capable, scalable, maintainable or reusable, in fact without varying any functions of the program itself. In this paper we conversed about the various types of refactoring tools and methods are able to recognize by the employ of specific refactoring tools.

REFERENCES

[1] M. Fowler, K.Beck, J.Brant, W.Opdyke and D.Roberts, "Refactoring: Improving the design of Existing Code", Addison-Wesley, (1999).

[2] A.Lozano, M. Wermelinger, and B.Nuseibeh, "Assessing the impact of bad smells Using Historical Information", Proc. Ninth Int'l Workshop Principles of Software Evolution in Conjunction with the Sixth ESEC/FSE Joint Meeting, pp.:31-34, 2007.

[3] A. Fokaefs, N.Tsantalis, and A.Chatzigeorgious, "Jdeodorant Identification and Removal of Feature Envy Bad Smells", Proc. 12th European Conf. Software Maintenance and Reengg., pp. 329-331, Apr. 2008.

[4] Francesca Arcelli Fontana, Pietro Braione, Marco Zanoni, "Automatic Detection of Bad Smells in Code", In Journal of Object Technology, Vol.11, no.2, pp. 1-38, 2012.

[5] Naouel Moha, Yann-Gae "I Gue "he" neuc, Laurence Duchien, Anne-Franc, Osie Le Meur, "DÉCOR: A Method for the Specification and Detection of Code and Design Smells", IEEE Transactions on Software Engineering, Vol.36, No.1, January/Febrauary 2010.

[6] Almas Hamid, Muhammad Ilyas, Muhammad Hummayun and Asad Nawaz, "A Comparative Study on Code Smell Detection Tools", International Journal of Advanced Science and Technology, vol.60, 2013.

[7] Pisyush Chandi, "A Survey of Code Optimization using Refactoring", International Journal of Computer Science and Engineering(IJCSE), Vol.5, No.14, 2013.

[8] B. Pietrzak and B.Walter, "Code Smell Detection with Inter-Smell Relations", Proc. Seventh Int'l Conf. Extreme Programming and Agile Processes in Software Eng., pp. 75-84, June 2007.

[9] Dr. V. Sangeetha, M. Sangeetha, "Fascinating Perspective of Code Refactoring", International Journal of Advanced Research in Computer Science and Software Engineering(IJARCSSE), Volume 6, Issue 1, ISSN: 2277 128X, pp. 164-168, January 2016.

[10] T. Mens, S. Demeyer, and D. Janssens. "Formalising behaviour preserving program transformations. In Graph Transformation", vol.2505, Springer-Verlag, pp. 286-301, 2002.