Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 4, April 2015, pg.88 – 91

RESEARCH ARTICLE

ADVANCE TECHNIQUE FOR MEASURING CODE QUALITY

Shweta N.Shindode
Rohini S.Wagh
Priyanka A.Narke
Priyanka V.Gite
sshindode@gmail.com

-ABSTRACT

Specification Oriented Programming (AOP) is a new programming paradigm that offers a novel modularization unit for the refactoring concerns. Functions spread across many modules and tangled with each other can be factored out into a single, separate unit, called a specification. Formal specifications can help with program code testing, code optimization, code refactoring, documentation, and, very importantly, debugging of code and repair of that code. However, they are very difficult to written manually, and automatically mining technicians suffer from 91–99% false +ve rates. To address this problem, we propose to augment a temporal-property miner by incorporating code quality metrics. Abstract A refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. Refactoring systematically organized into catalogues, in a similar way as design patterns. in that certain are applicable are provided by so-called code smells: Suspicious code parts that require improvement.

Index Terms—specification mining, machine learning, program understanding

--INTRODUCTION

In all engineering disciplines, humans cope with complex systems by using a "divide and conquer" approach: they divide a complex problem into many simpler sub-problems. Each sub-problem in-stance should have a low level of coupling with the other ones, so the team assigned to this sub problem can reason about it as a whole and can devise a solution for it. The composition of the sub-problem solutions produces the solution for the whole complex problem. In software engineering, many modularization mechanisms have been introduced. For example object oriented programming (OOP) provides the object model.

Incorrect and buggy behavior in deployed software costs up to $70 billion each year in the US [46], [53] Thus debugging ,testing, maintaining, optimizing, refactoring, and documenting software, while time-consuming, remain critically important. Such maintenance is reported to consume up to 90% of the total cost of software projects [49]. A key maintenance concern is incomplete documentation [15]: up to 60% of maintenance time is spent studying existing software (e.g., [47]).Human processes and especially tool support for finding and fixing errors in deployed software often require formal specifications of correct program behavior (e.g., [43]); it is difficult to repair according error without a

clear notion of what "correct" program behavior entails. Unfortunately, while low level program an notations are becoming more and more prevalent [14], comprehensive formal specifications remain rare.
This models usually a natural decomposition of the domain, because it tends to break systems down into units of data and behavior that correspond to real world entities. Whatever decomposition type is used, it becomes the main architecture of the system under development, and all future changes will refer to it. Software systems are so complex that they cannot be developed without dividing them into sub-modules.

--CONCEPT OF ASPECT MINING

The main aim of project is to and the Cross Cutting in the program as well as optimize the project space and time complexity, also provide suggestion for the new programmer (i.e. Refactoring) Aspect mining is a relatively new research domain whose aim is to study and to develop techniques and tools for automatic identification of crosscutting concerns in legacy systems. The goal is to refactor them to aspects.
A manual approach for crosscutting concerns discovery is difficult and error prone due to the size of the software system, its complexity and lack of documentation. A refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing [11. its observable behavior. Refactoring's are systematically organized into catalogs, in a similar way as design patterns. Hints that certain refactoring's are applicable are provided by called code. suspicious code parts that require improvement.

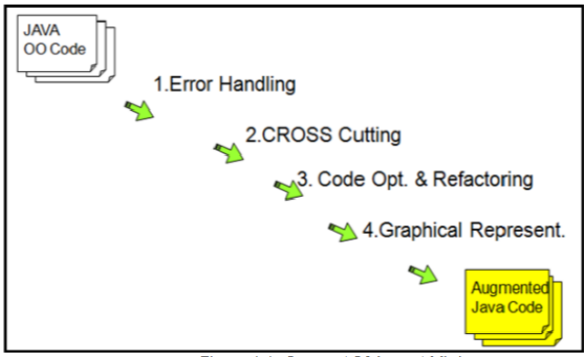
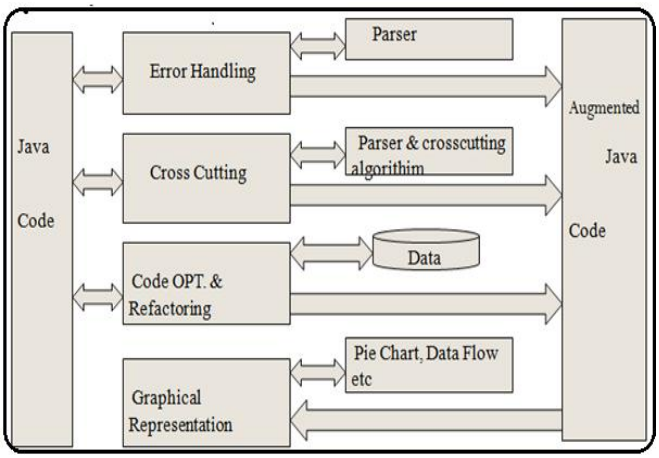


Figure 1.1: Concept Of Aspect Mining

--SYSTEM DESIGN:



The iteration involves the redesign and implementation of a task from the project control list, and the analysis of the current version of the system. The goal for the design and implementation of any iteration is to be simple, straightforward, and modular, supporting redesign at that stage or as a task added to the project control list. The level of design detail is not dictated by the interactive approach. In a light-weight iterative project the code may represent the major source of documentation of the system; however, in a critical iterative project a formal Software Design Document may be used. The analysis of an iteration is based upon user feedback, and the program analysis facilities available. It involves analysis of the structure, modularity, usability, reliability, efficiency, achievement of goals. The project control list is modified in light of the analysis results.

--METHODS

1. Error Finding (Parser)

The rest module of the project is Error Finding, for that purpose a parser is used .Parser is a recognizer which recognize the context free grammar, Error handing module hands the error into the source code if error are found out that error is solved. In the project we generate our own parser which generates tokens using that tokens we generate the parse tree and the leaf node indicate the solution of the tree.

2. Cross Cutting

Cross Cutting is the second module of the project in cross cutting we find out the cross cutting concerns .Cross Cutting concerns means finding out the unused code, dependencies, not used objects(sometimes we create objects that has allocated memory but using that objects we do not access anything, that is variable methods etc).Identifying those objects and remove from source code, so due to that time complexity of source code is reduced as well as space complexity is also reduced.

3. Code Optimization Refactoring

Refactoring means process of changing a internal structure of software system without changing its function or behavior. After that we optimize the code size, code time complexity and space complexity. This is done by removing unused code, repeated code and replacing code with better code. Code Smell In computer programming, code smell is any symptom in the source code of a program that possibly indicates a deeper problem. Often the deeper problem hinted by a code smell can be uncovered when the code is subjected to a short feedback cycle where it is refactored in small, controlled steps, and the resulting design is examined to see if there are any further code smells that indicate the need of more refactoring. From the point of view of a programmer charged with performing refactoring, code smells are heuristics to indicate when to refactor, and what specific refactoring techniques to use. Thus, a code smell is a driver for refactoring.

4. Visual Representation

Graphical representation can be represented using Pie Charts. The Project will consider to Pie-charts, one for the original code 44 before reduction and second one for the code which is reduced both the pie-charts are compared for the time complexity and space complexity which will show the difference in both.

---ALGORITHM

k-means Algorithm for Aspect Mining :Kam

In order to avoid the two main disadvantages of the traditional k-means approach, we propose a new k-means based clustering algorithm, kAM (k-means in Aspect Mining), that uses an heuristic for choosing the number of clusters and the initial centroids. This heuristic is particular to aspect mining and it will provide a good enough choice of the initial centroids. After selecting the initial centroids, kAM behaves like the classical k-means algorithm. The main idea of kAMs heuristic for choosing the initial centroids and the number k of clusters is the following:

(i) The initial number k of clusters is n (the number of methods from the system).

(ii) The method chosen as the _rst centroid is the most distant method from the set of all methods (the method that maximizes the sum of distances from all other methods).

(iii) For each remaining methods (that were not chosen as centroids), we compute the minimum distance (dmin) from the method and the already chosen centroids.

The next centroid is chosen as the method m that maximizes d min and this distance is greater than a positive given thresh-old (dist Min). If such a method does not exist it means that m is very close to its nearest centroid nc and should not be chosen as a new centroid (from the aspect mining point of view m and nc should belong to the same (crosscutting) concern). In this case, the number k of clusters will be decreased. (iv) The step (iii) will be repeatedly performed, until k centroids will be reached. We have to notice that step (iii) described above assures, from the aspect mining point of view, that near methods (with respect to the given threshold dist Min) will be merged in a single (cross-cutting) concern, instead of being 10 distributed in different (crosscutting)concern. We mention that at steps (ii) and (iii) the choice could be a non-deterministic one. In the current version of kAM algorithm, if such a non-deterministic case exists, the _rst selection is chosen. Improvements of kAM algorithm can tackle these kind of situations. kAM algorithm is presented in Algorithm 1. We mention that the algorithm stops when the clusters from two consecutive iterations remain unchanged, or the number of steps performed exceeds the maximum number of iterations allowed.

Input:

the setM = m1, . . . ,mn of methods to be clustered

the metric dE between methods in a multidimensional space

distMin ">" 0 the threshold for merging the clusters

noMaxIter the maximum number of iterations allowed

Output:
K = K1, . . . ,Kp the partition of methods in M
Algorithm kAM is:
k = n // the initial number of clusters
i1 argmaxi=1,n Pnj=1,J!=i dE(mi,mj) // the index i1 of the _rst centroid
is chosen nr =1 // the number of centroids already chosen
46
while nr < k do D = j | 1 j n, j /2 i1, ..., inr1, d = minl=1,nr1 dE(mj ,mil
) , d >
distMin
if D = ; then
k =k 1 // the number of clusters is decreased
else
nr = nr + 1 // another centroid is chosen
inr = argmaxj2D minl=1,nr1dE(mj ,mil)
endif
endwhile
for j = 1, k do
fj = mij // the centroids are initialized
endfor
while (K changes between two consecutive steps) and (there were not per-
formed noMaxIter iterations)
do
for j = 1, k do
Kj = mi | 8 1 r k, r 6= j, dE(mi, fj) dE(mi, fr)
fj = the mean of objects in Kj
// the j-th centroid is recalculated
endfor
endwhile

ACKNOWLEDGEMENTS

Aspect mining research is concerned with the development of concepts, principles, methods and tools supporting the identification of aspects in object-oriented software systems as well as the subsequent refactoring of such systems into aspect oriented systems. In this paper, we explored the state of the art in aspect mining research, and we identified a series of promising research directions. Research results can be used to support software development, continuously analyzing a system while it is built in order to identify (1) code smells requiring aspects; and(2) aspect smells requiring (aspect specific) refactoring's. Moreover, aspect mining can be used on completed systems, in order to over better support for future evolution of the system.

REFERENCES

[1] S. Breu and J. Krinke. Aspect mining using dynamic analysis. In Workshop on Software-Reengineering, Bad Honnef, 2003.
[2] A. van Deursen and T. Kuipers. Identifying objects using cluster and concept analysis. In Proc. Int. Conf. on Software Engineering (ICSE), pages 246255. ACM, 1999.
[3] S. G. Eick, J. L. Ste_en, and E. E. Sumner. See soft tool for visualizing line oriented software. IEEE Transactions on Software Engineering, 18(11):957 968, November 1992.
[4] J. Hannemann and G. Kiczales. Overcoming the prevalent decomposition of legacy code. In Proc. Workshop on Advanced Separation of Concerns. IEEE, 2001.
[5] D. Janzen and K. De Volder. Navigating and querying code without getting lost. In Proc. 2nd Int. Conf. on Aspect-Oriented Software Development (AOSD), pages 178187. ACM Press, March 2003.

