CHAPTER-6
CONCLUSION AND FUTURE WORK

The software evolution is one of the challenging issues in today’s business environment. It is necessary for the organizations, which make use of Information and Communication Technologies (ICT) will have to align their business processes to compete with global business. The business organizations have perceived that a generous amount of existing software frameworks fail due to unstructured architectural design. The value as well as quality must be taken in to account, while choosing a software evolution approach.

Research suggests that refactoring is considered as best-practice for creating, reusing, maintaining the software system

Modern Object Oriented Programming languages provides more reusability, modularity, and understandably, maintainable with low costs, which allows the precious development resources to the business development. But, at the same time, there is a need of new requirements that can be introduced more efficiently with less problems. These new requirements should be fit cleanly without changing the internal structure of the original design with less maintenance effort. Also, Code smells are the indicator of the presence of anti-patterns in the code, the presence of which will lead to the bugs in the near future. Efficient refactoring process is required which aids to nondestructive changes to the structure of the source code and to improve code clarity and maintainability.

6.1 SUMMARY OF CONTRIBUTIONS

We have proposed an effective tool named GATAOOS to do the software archeology outspreading several of the open source procedures in vogue. ANTLR is unified in a multi-threading environment aimed at producing manifold instances. This stage then takes multiple files as input to provide operative outputs for improved analysis of OO systems. The tool will have twin phases designed to identify the smells. In the first phase, the tool tries to parse the given source code files and garners the required data including class names, declared variables and methods. In the subsequent stage, it employs this data with the view to parse all the code again in order to identify the code smells or design flaws existing in the system. The parsing and analysis approaches are
driven simultaneously to identify or spot the code smell occurrence from a source code. Finally, we have analyzed the proposed tool GATAOOS by considering 18 open source java projects with better complexity to detect the code smells. Each one is tested for presence of smells- Data Class, Primitive Obsession, Duplicate Code and Message Chains. It was observed that, the Data Class Code smell detected in most of projects (88.8%) followed by Message Chains (61.1%), Primitive Obsession (44.4%) and Duplicate Code (33.1%). We have calculated the density of smells detected by tool for better analysis, it was observed that proposed tool identified the Data Class code smell whose average density is 2.5 followed by Message Chains (1.52), Primitive Obsession (0.86) and Duplicate Code (0.36). Finally, each test case verified manually to assess the effectiveness of the tool.

Next, we have proposed a novel refactoring tool called GAFactor using Dynamic Approach. This GAFactor system detects a developer’s java code, reminds to the programmer that the automatic refactoring is available and if the programmer accepts then GAFactor complete the refactoring automatically. GAFactor automatically performs static analysis for analyzing the flow of data of the code that saves the programmer from doing error-prone work. The refactoring process model performs a series of small transformations, by preserving the behavior of the software. Even though each transformation perform little, but a series of transformations can produce a substantial restructuring using dynamic programming approach. The main purpose of using this approach is that, it provides the flexibility to application developer to take the decision based on each refactoring process step, which is small and less likely to go wrong. This process also keeps the system fully functional after each small refactoring, reducing the chances that a system can get seriously broken during the restructuring. The refactoring path is evaluated with various maintainability metrics and decision tree is constructed based on these metrics as Weight Method for Class [WMC], Lack of Cohesion in Method [LCOM] and Coupling between Object Classes [CBO] (Efferent Coupling and Afferent Coupling). The approach uses the principle of optimality. If all the refactoring paths, for a refactoring technique are optimal, the solution is optimal and those refactorings are applied on given source code. However the choice is given to the programmer to commit the refactored code.

Source code which encompasses code smells will be explored for places to be changed and refactoring techniques to be applied. Each position will be applied refactoring
techniques to create possible refactoring techniques usage paths or simply refactoring path. Each altered source code segment will be calculated for software maintainability metric. Selected source code will be a base source code to be applied refactoring techniques on the remained positions. All positions are already changed, maintainability metrics for all refactoring paths for a specific refactoring technique were calculated. From the experimental results we have obtained the cumulative metrics–(WMC, LCOM, AC, EC) for Composing Methods- (98, 5.101, 19, 11), Moving Features between the Objects(84, 5.126, 20, 14), Simplifying Conditional Expressions (36, 1.944, 9, 5), Organizing Data (35, 1.987, 8, 4), Simplifying Method Calls( 27, 1.971, 7, 7) and Generalization(78, 2.657, 8, 8). Now Programmer can take a decision to commit for the refactored code by comparing these metric values with threshold values selected.

It is observed that, the proposed GAFactor tool refactors the code effectively and from results, we can conclude that for each refactorings performed, cohesion (LCOM) is very high due to the sub optimal solutions. Moreover, the proposed method provides low WMC, due to less time complexity among member functions within the class in all the cases and minimum CBO.

Finally we have implemented the complete automated verification mechanism that certifies all the functional components of application and various process involved during the certification phase. This mechanism mainly saves the cost and time compared to the manual Quality Assurance approach. Moreover, this mechanism does not need any human intervention, will provide a 100% bug free certification. We have evaluated our approach by comparing software maintainability of source code before and after applying the refactoring techniques. All these metrics were used to evaluate the effectiveness of the proposed method. It is also observed that the proposed system provides better refactoring results.

The experimental results shows for the given test case before and after refactoring technique- composing methods - WMC decrease from 123 to 98, LCOM increased from 4.3 to 5.101, AC decreases from 22 to 19 and EC from 14 to 11. Similarly for other refactoring techniques there is increase in LCOM, decrease in CBO and WMC, which improves the quality of the system.
6.2 FUTURE WORK

The different applications may require dissimilar thresholds for the metric to be considered. Machine learning techniques are suitable for selecting the threshold. Using our approach for improving quality will produce better results. But the main issue is how to maintain the consistent accuracy due the overhead added by metrics measurements.

The proposed approach is implemented in Java programming language. So, the performance of the approach may be changed for other programming languages and platforms. The assumption in the approach is that there will be extensive test cases for the functionalities of the classes. If there are not enough test cases to reflect the functionalities, the dynamic analysis will not be fruitful in determining the client usage of subclass’s methods. However, ensuring the coverage of test cases is a different research topic.

The proposed approach is experimented on a limited set of projects. How the proposed approach would work for other projects is one of the future works. The accuracy of the proposed approach is determined by manually inspecting the source codes. Even though there are different standard processes for identifying the code smells from the source code and refactoring the code, there is more scope to improve the software quality of the system.