Effects on Software Quality with Clean Room Software Coding

Niveda Pareek
Department of Computer Science and Technology
Central University of Punjab
Bathinda , India

Satwinder Singh
Department of Computer Science and Technology
Central University of Punjab
Bathinda , India

Abstract- **Software engineering and its maintenance is the most emerging topic now-a-days. As people are becoming more tech-savvy so with improvements in technology and in turn improvement in the internal working of the code and interface is new demand as well as prominent area of research. Software maintenance epitomize cost factor from customer point of view therefore developer tends to showcase software as brand new entity rather than improvised one. Defect avoidance is the key area rather than defect removal here.**

Keywords— Clean room processing, Coupling and cohesion, Refactoring, Software Maintenance

I. INTRODUCTION

Cleanroom is derived from the method used to fabricate semiconductor. It combines many of the formal ways and software package quality improvement approaches. In the study [1] bulk fixing coding issues were analysed which gave insight into the virtual world of software engineering. This defines quality of a software system is generally outlined by its source code. Software evolves unendingly; it gets modified and enhanced as new necessities perpetually arise. If proper time is not dedicated on improving source code, it becomes filthy and its quality will decrease without any doubt. After that key importance was given to refactoring which improves the existing code in such a manner that external user will never get to know about what is happening inside the software which makes it more interesting to explore. As search goes deeper in journals then it was found that most of the cost is spent on software maintenance rather than software development which in turn inspired to know the factors and causes involved in this concept. It was analysed that all multi-national companies participate and spend in big projects to revise their internal coding section i.e. to apply refactoring operations using some tools and hire people in bulk for the same[2]. It is not secured that refactoring will always improve the code but still it is a part of present software development practices. This study deals with the same objective within the same context. Whenever it comes to improve the quality of code, some techniques are applied. For this data can be chosen in bulk and show results in every possible way.

Next step is to analyse how these refactoring can mould coupling and cohesion characteristics, rules can be set to optimize their usage for improvement. There are several findings which describe where and how these rules can be applied. A survey is also a good alternative to know about the results or findings of this topic. So with these numerous options, this concept can be easily defined and analysed .It should be done in order to find out the results according to the objective. Many findings are there which describe conditions in which if refactoring is applied well then it can improve specific dimensions of coupling and cohesion. Guidelines for applying the refactoring under these conditions are composed and validated on an open source software system regarding its quality. Individual workmanship, ordered development, individual unit testing, informal coverage testing, unknown dependability, informal style were replaced by peer reviewed engineering, progressive development, team correctness verification, applied math usage testing, measured dependability, disciplined engineering specification and style respectively.

II. LITERATURE REVIEW

Cleanroom software development is a software package development method that avoids package defects by exploiting formal strategies of development and a rigorous scrutiny process. If defect prevention is the point of concern rather than defect removal then cleanroom software engineering can be considered as high quality software development method along with reliability. Quality of software product can be improved by regularly refactoring it which is defined as "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour" [1]. In other words, refactoring can be applied to improve software in terms of its design, understand ability as well as it will make it easier to find errors which will in turn help to program faster[3] . Single refactoring can make small changes or sometimes decrease the quality but when applied in block, it significantly increases the quality. Unfortunately it is not an assurance that quality will always increase with refactoring technique. Major portion of total life cycle cost of a system is consumed by software, when cost is calculated as total systems cost along with programming resources consumed. After reviewing various studies[4][5][6] about software quality and maintenance, it was observed that out of total resources of the system and programming groups, a lot of them are used by maintenance and enhancement. As per consideration of management, maintenance and enhancement are somewhat more important than new application software development[7]. In maintenance and enhancement, technical problems tend to be more vital than those of management. Demand for intensification and extension by an user constitute the most ponderous management problem area[5]. Usually, management problems associated with maintenance grabs more attention. To collect more detailed management information in practical scenario, maintenance work should be categorized. Systems should be in such flexible state that is capable to handle various types and tasks of maintenance and enhancement. Refactoring restructures a program in such a manner that changes to the program can be done in much easier way although it do not change behaviour of a program. Complicated changes to a program need both refactoring and additions[8] such as extracting a reusable element, improving consistency among components, supporting the unvaried structure of an Object-Oriented Application framework. One among the explanations why programs are not refactored is that any type of changes to the program runs the danger of introducing defects into the program which might be a lot dangerous than what is expected. Research is still going on to apply refactoring without changing the program's behaviour. Best possible way to automate refactoring can be considered as to ensure that defects are never introduced in the program. All applied refactoring will improve design of a program is not always true. On the contrary, applying arbitrary refactoring can corrupt software design instead of improving it, preserving its behaviour as per the definition [9]. Outcome of a refactoring are meaningful modifications which make the program easier to refine or reconstruct.

To deal with these complexities, there is an urgent need for techniques to scale back software system complexity by slowly raising the internal software system quality. Restructuring is the research domain that addresses such issues. Restructuring[10] is defined as “the transformation from one representation to another at the same relative abstraction level, while preserving the subject system’s external behaviour (functionality and semantics)”. A restructuring modification can appear as code altered to enhance its design in the trivial sense of structured design. Restructuring usually does not involve modifications and updates due to new requirements although it creates new versions that propose changes to the subject system. Many aspects of the system can be improved as restructuring lead to better observations of underlying subject of the system.

Refactoring activities analyze where the computer code ought to be refactored , which refactoring(s) should be imposed to the known places, assures that applied refactoring maintains behavior of the system, application of refactoring, analyze effect of the refactoring on quality measures of computer code such as quality, understandability, comprehensibility, maintainability, method productivity, cost, effort, maintain consistency between restructured computer code and other software artifacts such as documentation, design documents, requirements specifications, tests, etc. These steps are applied on real time software, embedded software and safety critical software. Above mentioned categories are important open issues to be solved in future.

Refactoring can be addressed in more consistent, scalable, general and versatile method by identifying need for process and methods. In programming, the key plan is to spread instance variables and functions across the class in order to make the code more comprehensible, reusable and maintainable[11]. It somehow reach to the extent where it can be said that for small nesting levels, if statement is sometimes a little quicker than the equivalent virtual methodology. One might prefer an if-statement over a virtual method for conditions with nesting level 4 or less[10]. A validated reason behind low maintainability, low reuse, high complexity and erroneous behavior of the programs is design flaws introduced in initial stages of development or throughout system evolution. One of the taboo research objectives is to preserve the correct system design. However, modifications, detection and correction of design flaws may be complicated and resource-consuming task for huge systems subject to timely modifications. Quality of object systems can be improved by use of metrics for quality estimation and automated transformations[12].In general, both aspects have been treated independent of each other. Further these efforts can be used to observe the interaction of particular modification and metrics in an orderly manner to suggest the use of modifications that may be helpful in improving quality as calculated by metrics. The answer to the question, Can metrics help to bridge the gap between the improvement of object oriented design quality and its automation can be yes or no. It can be yes; the reason can be defined as automation of quality improvement can be done with the use of metrics. Metrics can help to automate a large part of whole process hierarchy of detecting flaws and rectifying them. The answer can be no as the results of some findings show that a prescription cannot be executed without validation of a programmer or designer. Although it cannot determine all aspects to permit such type of automation. It is clear that refactoring can be used to restructure software systems; it is not clear how to use them to enhance specific quality attributes that indicates a good design.

In the context of improving [13], it is assumed that coupling and cohesion characteristics may serve as indicators for the optimal distribution of responsibilities over the class hierarchies. Thus, coupling and cohesion is a less ambitious goal than refactoring which will improve the design .Cohesion corresponds to the degree to which elements of a class belong together and coupling is the strength of association established by a connection from one class to another [13]. Some conditions will improve specific coupling and cohesion dimensions by applying refactoring. It is identified that specific applications of Move Method, Replace Method with Method Object, Replace Data Value with Object and Extract Class are beneficial. However, guidelines for these methods can be inadequate. Specific application of Extract Method was harmful for cohesion so it is not true that every practice will give the expected result. Being a researcher it is to be taken into account that some findings and experiments will lead to the desired point and some will take it to some other aspect which in turn can be considered as new research area to work on. Many findings made above research area very vast and allow dealing with new errors so that findings can be defined and analyzed in order to improve knowledge and in future new framework can be proposed.

It is experienced that it is possible to achieve quality enhancements with restricted refactoring efforts by exploiting the results from coupling/cohesion impact analysis. Analysis and resolution of a limited set of refactoring opportunities are known to enhance the concerned quality attributes which can be considered as a restriction of this effort[14]. Certainly refactoring will make software go more slowly, but it also makes the software more flexible to performance tuning[12]. Now after refactoring, quality of code can be improved by maintenance of the software which is one of the most crucial and costly phase of the software lifecycle since so long. Out of total cost of developing the software, maintenance costs can be more than 40 per cent. Software lifecycle is very long that is why maintenance cost is high. Large projects may take several years to complete and take more time to be maintained. It is highlighted fact that software systems often survive much longer than the developers analyzed which can be a good sign from a developer's point of view. Software is easy to modify during maintenance as compared to hardware[2]. Software maintenance changes are more extensive and more frequent than maintenance performed on less mutable systems. The assumption that these changes may add defects to the system is increased by the fact that high personal turnover in software industry increases the probability of non-availability of original developer at the time of consultation and long process of development. To analyze reasons behind the choices made by the developers during design and implementation is the major problem in maintaining a huge system[14]. As the whole idea behind these studies is mainly to improve understandability and modifiability by proposing refactoring framework and make the code cheaper to modify as well as easier to understand. However, this tip can also be used for security, versatility, usability, performance and more. Functionalities are termed as goals while qualities and the factors affecting them are termed as soft-goals[2]. Precisely, they can be categorized into resources and tasks. The dependencies among goals, soft-goals, tasks and resources can be represented by a soft-goal interdependence graph. Functionality goals mainly focus on performance and code complexity of soft goals concerned with the set of software metrics. Within quality space, modifications on the state of a program measures refactoring. By further monitoring the process of developing a new software from scratch[10], it is coined that to balance productivity goal with refactoring goal, quality space should be measured along with progress. Different priority at different development phases can determine the region for quality space that ought to be adjusted dynamically during the software evolution i.e. development. Refactoring changes are invertible if no functionality change happens because these are non-functional.

Program refactoring is a method to improve the maintainability of a program. Although the concept itself is considered to be effective, there are few quantitative findings of its impact to the software maintainability. It is sometimes difficult to judge whether the refactoring in software should be applied or not without knowing the effect accurately. Effect of program refactoring on maintainability is measured by a quantitative evaluation based on coupling metrics to determine its effect. Degree of maintainability enhancement can be evaluated by comparison i.e. comparing the coupling before and after the refactoring. Then bad smell is introduced which is defined as a program characteristic which alludes to the necessity of program refactoring[6]. Duplicate code can be improved by unifying the duplicated parts, which is the main reason why duplicate code is considered as bad smell. A class that does not do anything specific is termed as lazy class which is also bad smell. Refactoring candidates' analyze the source code to detect bad-smell. Code cloning i.e. copy pasting the code is also a good alternative to achieve design goals but it carries the danger of code quality within time frame. However, deciding which clones to be eliminated is a cumbersome task. Modifying a clone needs effort, cost, and risk that such a change contains but all refactoring are not worth it because external refactoring is needed to serve the purpose[10]. Furthermore, cloning should not be refactored at all if it fulfills a useful design role. When a developer finds the same research work as the one under consideration and copy-paste the code then software clones come into existence. The step by step procedure of cloning is detection of clone, identification of clone refactoring cases and clone instances that are not refactored, extraction of features from clone instance and assessment of classifier's performance. Empirical proofs provide evidence that cloning is not always harmful and makes it a major engineering tool. Cloning is often used as a convenient design shortcut to reuse an existing solution by duplicating and then specializing code fragments within a software system. An easy way to reuse existing code is by cloning in which code is duplicated and then those parts are specialized within the system. However, sometimes bugs are introduced in the system due to cloning and create long-term software maintenance issues.

A question arises after learning about these findings that does refactoring improve software quality. System has to undergo modification, improvement and enhancements to handle evolving software requirements. To evaluate effectiveness of refactoring which is used to enhance software quality, open source system detect changes referred as refactoring. When the development team performs refactoring then analysis can be made on how metrics of open source system can be affected irrespective of the reasons that led to that decision. It sometimes leads to change of certain metrics to the worse.

In the agile community[14] it is accepted that refactoring contributes to confine the quality of source code and incorporates a positive impact on the maintainability and comprehensibility of a software system. Code that is frequently refactored is assumed to be correct, easier to understand and align to new needs. From economic point of view, refactoring is found to have beneficial impacts on maintenance activities and other software quality attributes thus it is highly motivated. It can be seen that this work contributes to a far better understanding on the consequences of refactoring on code quality as well as code development in industrial and agile development environment. It particularly deals with code maintenance. It suggests that refactoring increases rather than decreasing the code quality, productivity and improves quality factor. All findings rely on industry based evidences. All these factors are measured using common internal quality attributes and reduce code complexity along with coupling but increases cohesion.

III. RESEARCH GAP

Literature name and author	Work done by author	Research gap in Literature
Bulk fixing coding issues and its effects on software quality by Gabor Szoke, Gabor Antal, Csaba Nagy, Rudolf Ferenc,	They studied thousands of refactoring commits during refactoring period. Rather than fixing code smells indicated by metrics or automatic smell	It is not specified What should be refactored. How it can be done. Can it be automated, what can be the

and Tibor Gyimo	detectors, developers preferred to fix main coding issues.	shortcomings? One more point is, code quality will never improve if developers do not refactor the source code time to time. Whole quality of software product cannot be affected by a single refactoring. Moreover, sometimes it may have negative impacts.
Representing and Using Non functional Requirements: A Process-Oriented Approach By John Mylopoulos, Lawrence Chung, and Brian Nixon	This study offers a framework to embed non-functional requirements into software development process especially for information systems.	Implementation of prototype is still under process for this framework. It needs to be applied to other non-functional requirements too.
Rationale Support for Maintenance of Large Scale Systems By Janet E. Burge and David C. Brown	Software Engineering Using Rationale system abbreviated as SEURAT is developed. The SEURAT system will make modifications to the software; maintainer can make use of this software development tool.	In enhancement of the study, evaluation of the choices made and their impact provides support to the maintainer to view reasons for the same. Further, as per our findings this study is found not to be suitable for large scale systems.
Software refactoring guided by multiple soft-goals By Yijun Yu John Mylopoulos Eric Yu	A case study in this work has shown that transformation on the state of program measures refactoring in the quality space. Modifications to the state of the program without changing the state of the data are non-functional requirements.	The scope for improvement of the study is by balancing refactoring goal with productivity goal in order to measure the quality space along with progress. During software development quality space should be adjusted dynamically and

		as per the requirements.
Refactoring - Improving Coupling and Cohesion of Existing Code Bart Du Bois and Serge Demeyer and Jan Verelst	It is demonstrated that it is possible to achieve quality improvements with restricted refactoring efforts by exploiting the results by coupling/cohesion impact analysis. Concerned quality attribute can be improved by analysis and resolution of refactoring activities.	Guidelines for improvement of cohesion and coupling are insufficiently specific. Specific extract method was not found to be useful for cohesion.
Recommending Clones for Refactoring Using Design, Context, and History By Wei Wang and Michael W. Godfrey	Code can be reused by copying the existing code and improving those parts in the software system. However, cloning may introduce bugs and cause software maintenance issues.	Code management can be improved by better resource allocation.
A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team By Raimund Moser, Pekka Abrahamsson, Witold Pedrycz, Alberto Sillitti and Giancarlo Succi1	Refactoring increases the productivity and quality factors of code rather than decreasing it. Empirical evidences prove it. They are measured using internal quality attributes which reduces code complexity and coupling but increases cohesion.	The effects of defects are not elaborated and its generalization in large context is not possible.
Can Metrics Help to Bridge the Gap Between the Improvement of 00 Design Quality and Its Automation By Houari A. Sahraoui Robert Godin Thieny Miceli	To automate the process of quality improvement, metrics can be used. Metrics can also automate process of analyzing defects and rectifying them.	A designer or programmer is needed to validate the whole concept. So it can't be considered as automated. Moreover, the situations where these transformations should be applied are not specified.
	A quantitative method is proposed	Refactoring effects can also

A Quantitative Evaluation of Maintainability Enhancement by Refactoring By Yoshio Kataoka ,Takeo Imai ,Hiroki Andou and Tetsuji Fukaya	to analyze the effect of maintainability enhancement of the source code .Its implementation somehow made reasonable judgment regarding what programmer can or cannot do to enhance code maintainability.	be calculated based on metrics. Although refactoring effects calculated by coupling metrics are limited.
Refactoring – Does it improve software quality? By Konstantinos Stroggylos, Diomidis Spinellis	How metrics of open source projects were affected can be examined by refactoring performed by developers. It may lead to modifications in the metrics to the worse.	In real life system, enhancement to one matric does not affect various metrics.

IV. CONCLUSION

All the findings and conclusion made clean room software coding vaster for research. Existing research work suggests that in order to safeguard productivity and inevitability of development goals, refactoring must be small enough. Software engineering case studies are very rear in industries so it gives huge confidence on these findings. However, it should be kept in mind that findings of research work are valid only in specific areas of study. To achieve confidence in such studies, it is recommended to research in all contexts and make results more common.

To analyze refactoring, effect of change in various metrics can be used which enhances overall quality of the system. Empirical analysis can be repeated on more proprietary and open source systems. Findings showed that it is possible to implement design into existing code automatically.

REFERENCES

[1] G. Szoke, G. Antal, C. Nagy, R. Ferenc, and T. Gyimothy, “Bulk fixing coding issues and its effects on software quality: Is it worth refactoring?,” Proc. - 2014 14th IEEE Int. Work. Conf. Source Code Anal. Manip. SCAM 2014, September 2014, pp. 95–104.

[2] Y. Yu, J. Mylopoulos, E. Yu, J. C. Leite, L. L. Liu, and E. D’Hollander, “Software refactoring guided by multiple soft-goals,” in Proceedings of The First International Workshop on Refactoring: Achievements, Challenges, Effects (REFACE 2003), 2003, pp. 7–11.

[3] S. Demeyer, “Refactor conditionals into polymorphism: What’s the performance cost of introducing virtual calls ?,” IEEE Int. Conf. Softw. Maintenance, ICSM, vol. 2005, pp. 627–630, 2005.

[4] J. Mylopoulos, L. Chung, and B. Nixon, “Representing and using nonfunctional requirements: a process-oriented approach,” IEEE Trans. Softw. Eng., vol. 18, no. 6, pp. 483–497, 1992.

[5] T. Bakota, P. Hegedűs, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, “A Probabilistic Software Quality Model,” 27th IEEE International Conference on Software Maintenance (ICSM), September 2011, pp. 243–252.

[6] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, “A quantitative evaluation of maintainability enhancement by refactoring,” in Proceedings of International Conference on Software Maintenance, 2002, pp. 576–585.

[7] T. Bakota, P. Hegedus, G. Ladanyi, P. Kortvelyesi, R. Ferenc, and T. Gyimothy, “A cost model based on software maintainability,” in Software Maintenance (ICSM), 2012 28th IEEE International Conference on, 2012, pp. 316–325.

[8] L. Tahvildari and K. Kontogiannis, “A metric-based approach to enhance design quality through meta-pattern transformations,” in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2003, pp. 183–192.

[9] K. Stroggylos and D. Spinellis, “Refactoring--Does It Improve Software Quality?,” Fifth Int. Work. Softw. Qual. (WoSQ’07 ICSE Work. 2007), pp. 3–8.

[10] W. Wang and M. W. Godfrey, “Recommending clones for refactoring using design, context, and history,” Proc. - 30th Int. Conf. Softw. Maint. Evol. ICSME 2014, pp. 331–340.

[11] F. W. Opdyke, “Refactoring object-oriented frameworks,” 1992.

[12] H. A. Sahraoui, R. Godin, and T. Miceli, “Can metrics help to bridge the gap between the improvement of OO design quality and its automation? in Proceedings. of Int. Conf. Softw. Maintenance, pp. 154–162, 2000.

[13] B. Du Bois, S. Demeyer, and J. Verelst, “Refactoring - Improving coupling and cohesion of existing code,” Proc. - Work. Conf. Reverse Eng. WCRE, pp. 144–151, 2004.

[14] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti, and G. Succi, "A case study on the impact of refactoring on quality and productivity in an agile team", vol. 5082 LNCS. 2008, pp. 252–266.