Journal of Engg. Research Vol.9 No. (4A) pp. 59-73 DOI:10.36909/jer.9481

Software quality improvement and validation using reengineering

Jaswinder Singh*,*, Kanwalvir Singh Dhindsa** and Jaiteg Singh*****

*Department of Computer Application, IK Gujral Punjab Technical University, Kapurthala, Punjab, India
** Department of Computer Science and Engineering, Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, Punjab, India
*** Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India
* Corresponding Author : jaswinder.singh@chitkara.edu.in

Submitted *:* 05/01/2020
Revised *:* 09/02/2021
Accepted *:* 22/02/2021

ABSTRACT

In software development life cycle, software maintenance is among the critical phases. It is a post-implementation activity that requires rigorous human efforts. For any software developer, maintaining software for a longer period is the primary objective. This objective can be accomplished if good quality software is developed. Maintainability is one of the vital characteristics of software maintenance. Maintainability enables developers to keep the system alive for a longer period of time at a limited cost. Software Maintainability can be enhanced using reengineering. The proposed research validates improvement in the quality of the reengineered software system. The quality of the software is analyzed using a coupling, cohesion, inheritance, and other essential design metrics. The observed improvement in the software design is 62.1%. The execution time of the software is also reduced by 6.5%. Reduction in the cost of maintenance is also another important outcome of this research. The observed reduction in the maintenance cost is 36.8%. Thus, the main objective of the proposed research is to analyze and validate the quality improvement in the reengineered software. Agile Scrum methodology has been used to perform software reengineering. Design Metrics are measured using the Chidamber and Kemerer Java metric (CKJM) version-9.0 tool. For reengineering implementation, Net Beans 7.3 has been used.

Keywords: Software quality; Software metric; Software maintainability; Software engineering; Reengineering.

INTRODUCTION

Maintenance activity is an integral part of software engineering. Irrespective of the Software type, maintenance is required to keep the software in tune. Maintenance may occur when there are changes in user requirements, identification of bugs and their removal, or adapting a changed/new environment. Performing various maintenance activities may reduce the quality of software in terms of degradation in software architecture (Baabad et al., 2020) as well as rapid software product aging (Tripathy & Naik , 2014). The consequences are code decay that is difficult in changing the software code and thus increases the cost of change.

The improvement in the quality of the software can be obtained with the process of software reengineering. Reengineering improves software quality and makes the system more Maintainable. The reengineering process includes reverse engineering, alteration, and forward engineering (Figure 1).

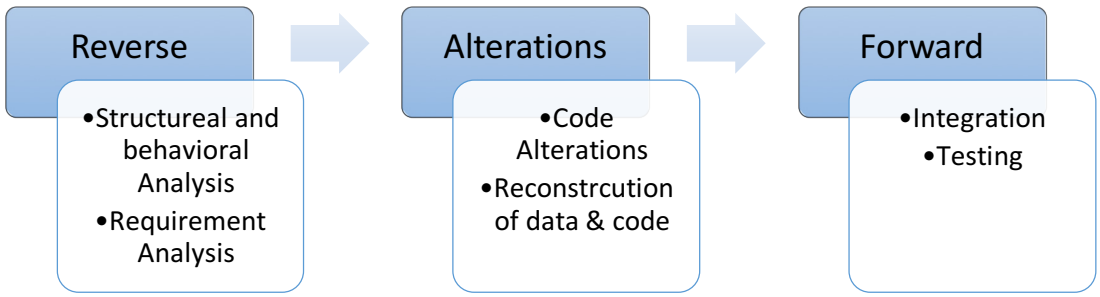

Figure 1. Reengineering Process

As shown in Figure 1, Reverse Engineering is concerned with the structural and behavioral analysis of an existing system. The design of the system is recovered through the existing documents. New requirements are identified and analyzed. An alteration includes code alterations as per the specifications and reconstruction of the system is performed by making changes in the data and code. Forward Engineering includes integration of various reconstructed modules and performing an adequate level of testing for an error-free system.

The quality of the software can be determined by considering various metrics. Chidamber and Kemerer gave important quality metrics in terms of CK metric suit (Chidamber et al., 1994) to determine the software quality.

The proposed research used a basic set of CK metrics to measure software design complexity. Software design before and after reengineering is compared using Ck metrics. The primary metric suite includes six metrics given in Table 1.

Table 1. Design Metrics (CK Metric Suit) considered validating the quality improvement

Sr. No.	Metric	Meaning
1	Coupling Between Object (CBO)	The measure of coupling (dependency) of a given class with the number of other classes and hence, shows the dependency of one class on other classes. It gives a total dependency count.
2	Depth of the Inheritance Tree (DIT)	Inheritance hierarchy, this metric measures the length of the longest path from a given class to the root class.
3	Number of Children (NOC)	This metric counts the number of immediate child classes of the given class.

4	Response for a Class (RFC)	For an object of a particular class, this metric counts the number of procedures (methods) that get activated in response to a message received by an object.
5	Lack of Cohesion of Methods (LCOM)	This metric counts cohesiveness in class design. A count of the number of method-pairs whose similarity is zero minus the count of method pairs whose similarity is not zero.
6	Weighted Methods per Class (WMC)	It is the summation of the number of methods defined in a class.

Metrics given in Table 1 have proven to be a useful indicator to analyze the quality of object-oriented software. In the proposed work, the software quality improvement is validated by comparing existing software with its reengineered counterpart. For the last two decades, software development using agile methodology gains lots of importance. Agility ensures adequate user involvement while developing software projects to make good quality software. Agile methodology is also helpful in improving the critical quality attribute that is the maintainability of software.

Software used in the proposed work is reengineered (Singh et al., 2019) with the help of agile methodology. The main objective is to validate the quality improvement of reengineered software in terms of internal design complexity, execution time, and cost of maintenance. Mean Time to Execute (MTTE) metric is another measure used to access the execution time improvement of the reengineered system. Proposed work also evaluates maintenance costs effect in software after reengineering. Cost Reduction is validated by estimating and comparing the efforts required to implements the requirements before and after reengineering. Requirements are estimated using the planning poker method. The next section discusses the literature review of software reengineering, CK metric usability, requirements estimation methods, and the Agile process.

LITERATURE REVIEW

The term Reengineering is in existence for the last four decades. Many researchers contributed to the field of reengineering. Grady et al. (1994) used a synthesis process with reengineering to enhance maintainability and reuse. In his work, the synthesis process with reengineering used domain engineering and application engineering. Cagnin et al. (2001) used a case study of the legacy project and converted it into object-oriented software. They also compared the average time spent on maintaining the legacy, segmented, and reengineered systems. Their work proved that after reengineering, the average time spent on maintenance is reduced. According to Khomh & Gueheneuc (2018), to keep a legacy software system continue to perform, Reengineering is the only solution. Muhammad et al. (2018) emphasize reengineering as an essential approach to enhance the quality of the software. Work proves that design and coding can be improved using software reengineering. Smiari & Bibi (2018) performed on the smart retail business platform application to easily make changes in the system and also to easily maintain the retail application. Reengineering is applied to easily adopt alternative features without causing structural changes in the main version. Majthoub et al. (2018) summarized various software reengineering models and approaches. The needs of various reengineering tools have been identified to increase the use of reengineering practices in organizations. Earlier work also shows the importance of the CK metric suit for analyzing the complexity of the software. Their work used the CK metric for

determining maintainability on OO software (used WMC and CBO). CK metric is used to (Binanto et al., 2018) measure the quality of various versions of the software by applying CKJM tools to the classes of each version. By measuring the values of the CK metric suit for various versions, the quality of software in terms of various design metrics is analyzed.

Malhotra & Jain (2019) used the CK metric to analyze internal software quality factors on object-oriented software for refactoring techniques analysis. Basili et al. (1996) validated the CK metric as an important quality indicator. Work validated that the larger the value of the CK metric suit, the more immense is the complexity of software and the probability of more fault-prone software. Shyam et al. (1998) analyzed the three financial software devices using the CK metric and find their impact on managerial design decisions.

The proposed research used the Planning poker method for Requirement size estimation. Planning poker is one of the important techniques used in Agile for size-based effort estimations. For size and effort estimations in agile software development projects, Fernandez-Diego et al. (2020) identified Planning Poker as most preferred estimation method (24.66% of total number of studies). According to Usman et al. (2014), planning poker is useful for estimating efforts in agile software development. Planning poker is also used for user story estimations (Haugen 2006, Molokken-ostvold et al., 2008 & Mahnic et al., 2012). According to Cohen (2006), planning poker is a highly used approach in Scrum. Francisco et al. (2011) proposed Agile MANTEMA for medium and large maintenance software. It is also identified (Gandomani et al., 2019) that consensus-based cost estimation using planning poker is more accurate as compared to performing estimations using absolute values.

Tarwani & Chug (2016) stated the importance of agile in maintenance and also comment that "Agile methods provide faster delivery of product in a short period and ensure a high level of software quality at the same time." Ming et al. (2004) compared the waterfall and agile in terms of quality assurance, verification, and validation attributes and found agile practices more suitable as compared to the waterfall process.

METHODOLOGY

For reengineering, the first need is to identify that when the software should require reengineering. Singh et al. (2019) proposed a framework for identifying reengineering requirements. The proposed agile cost estimation model performs cost estimations (Singh et al., 2019). Reengineering is performed on software classes using agile scrum methodology. A quality factor for object-oriented software is measured using the CKJM metric tool (Spinellis 2005). Six basic metric sets of CK metric suit is used which includes Number of Children (NOC), Lack of Cohesion of Methods (LCOM), Weighted Methods per Class (WMC) Depth of the Inheritance Tree (DIT), coupling between object classes (CBO) and Response for a Class (RFC). Meantime to execute Metric (MTTE) is used as a time metric. Samples of 35 executions are taken on the system with the configuration of 8 GB RAM, HDD 1TB i3-4th Gen Processor. Net beans 7.0 is used for JAVA 7. Complexity measures for the object-oriented software system measured using CK metric suit are given in Table 2. The CK metric values are determined using the CKJM tool. DIT values represent the inheritance level of the class. Values of WMC in the class represent the number of methods/functions inside the class. NOC value 0 means that there is no immediate child class of the given classes. Level of coupling, lack of cohesion, and various methods call in the class are determined using different values measured by the CKJM tool.

Table 2. CK metric Suit Complexity measure (Singh et al., 2017).

Sr. No.	Metric &Classes	DIT	WMC	NOC	RFC	CBO	LCOM
1	IDE	6	17	0	121	17	70
2	UserDetail	5	23	0	109	12	183
3	Login	6	12	0	78	9	60

By performing the reengineering process, classes are interpreted, redesigned by altering the methods, and integrated with the whole software using forward reengineering.

SOFTWARE QUALITY IMPROVEMENT ANALYSIS AND VALIDATION

As stated by Sneed (2008), Reengineering enhances software quality. The overall improvement in the software quality is proposed via analyzing improvement in three aspects of the software.

- Improvement in Design Complexity of the software
- Improvement in the execution time
- The proposed reduction in maintenance cost

The main reasons for the improvement in the CK metric are identified in Table 3.

Table 3. Reasons for Quality Improvement in CK Metric.

Sr. No.	Metric	Reengineering Impact
1	WMC	The reduction of the number of functions and operators in the classes to a significant level. In place of different function calls, the functionality of different functions is integrated in a better way. Encapsulation of data and methods are much better as compare to the earlier design. The reduction in Independent calls of different functions is also observed.
2	CBO	Object or functional dependencies to other classes are reduced in classes. In Login and user details classes, object creation of other classes is minimized.
3	RFC	The numbers of functions and object creations are less in reengineered classes. Because of these reasons, the number of instances and calling of methods is less.
4	LCOM	Sharing of Instances among classes is reduced using the reengineering process. Lesser value results in more cohesiveness in classes.
5	DIT & NOC	After reengineering, there are no significant changes for these metrics.

Table 4 shows the results of the change in the number of functions for IDE, UserDetail, and Login class. The total number of functions in Login class before reengineering was eight but after applying reengineering, functions were reduced to four. Similarly, in IDE and UserDetail classes, functions are reduced to four and five. respectively.

Table 4. Functional Reengineering.

Sr. No.	Attribute	Login Before Reengineering	Reengineered Login	IDE Before Reengineering	Reengineered IDE	UserDetail Before Reengineering	Reengineered User Detail
1	Total number of Methods	8	4	10	4	15	5

After the reengineering, quality for classes against attributes of the CK metric is analyzed.

Validation of Quality Improvement in Software Design

Software accuracy highly depends upon good software design. The proposed work used Ck metric suit to access the design complexity of the software. Design complexity is measured in terms of cohesion, coupling, number of methods, and level of inheritance values. After reengineering, software design complexity is reduced significantly. Software classes are reengineered, which results in reducing the overall complexity of the software. Quality improvements in software design are analyzed below.

Analysis of Quality improvement for Login Class

Quality improvement in terms of various design attributes is depicted in Figure 2. WMC value is reduced from 12 to 4. WMC represents the complexity of the individual class. Reduction in this metric means a reduction in the complexity and, thus, results in more maintainability. The leading cause of the reduction in WMC is the reduction in the number of functions. The number of functions has been reduced from 8 to 4, which is a 50% reduction in the number of functions. More CBO means more coupling in the class, which represents more dependency. CBO is reduced from 9 to 5. The main reason for this reduction is that the dependencies of various functions in the classes have been reduced.

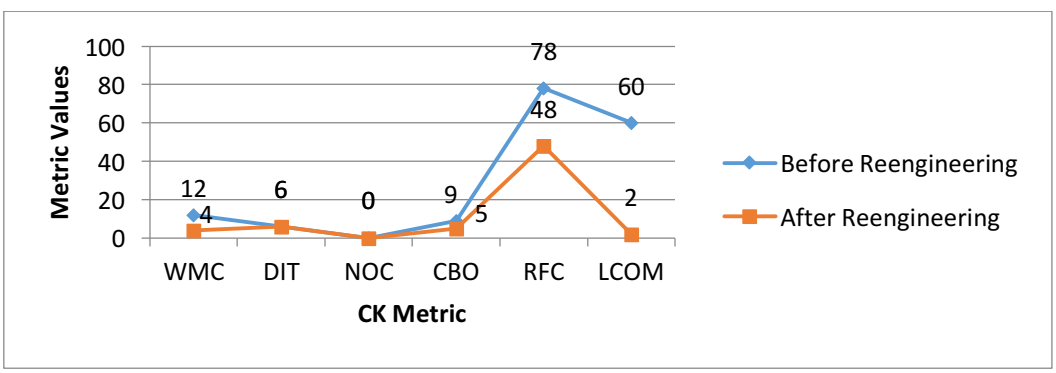

Figure 2. Analysis of Login Class before and after reengineering.

Larger the RFC more is the complexity of the functionality of a class. There is a drastic reduction in RFC from 78 to 48. The number of method calls in response to massage has been reduced from 4 to 1 in reengineered class. Lack of cohesion shows inappropriate design. LCOM is reduced from 60 to 2. Classes are designed such that there is the least dependency on other classes or functions for execution. There is no change in DIT and NOC as several descendants in the class before and after reengineering is the same. The overall quality has been improved to a greater extent.

Analysis of Quality Improvement for User Detail Class

As shown in Figure 3, in UserDetail class WMC is reduced from 23 to 6. The number of functions reduction is from 15 to 5, that is the functionality reduction is significant. A decrease in the number of functions results in less dependency of class to other classes; thus, CBO is reduced from 12 to 4.

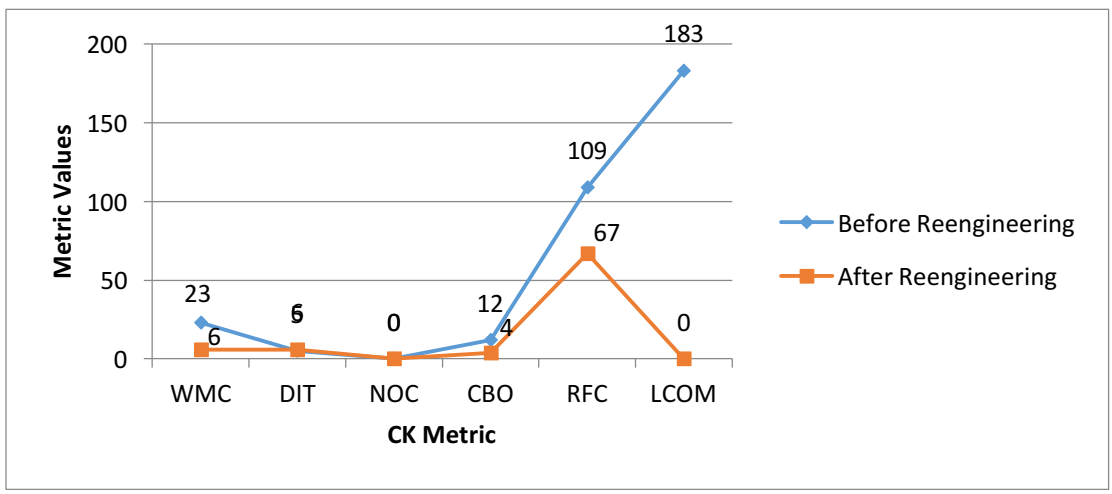

Figure 3. Metrics analysis of UserDetail Class.

There is a drastic reduction in RFC from 109 to 67. The number of method calls in response to massage has been reduced from 3 to 1 in reengineered class. The drastic reduction is noticed in LCOM as it is reduced from 183 to 0. After reengineering, 5 new functions have been designed aiming at maximal cohesiveness in these functions. There is no change in NOC and DIT as the number of descendants after and before reengineering is the same. With the reduction in the metric values, complexity in the class has been reduced.

Analysis of Quality Improvement for IDE Class

In IDE Class, WMC is reduced from 17 to 4 for this class. The reduction in the number of functions in the class is from 10 to 4. The dependency metric is reduced from 17 to 12. There is a reduction in RFC from 121 to 102. This is a reflection of the reduction in the number of method calls in response to massage. Calls have been reduced from 19 to 13 in coding. Because of only 4 functions in the class and designing more cohesive functions,

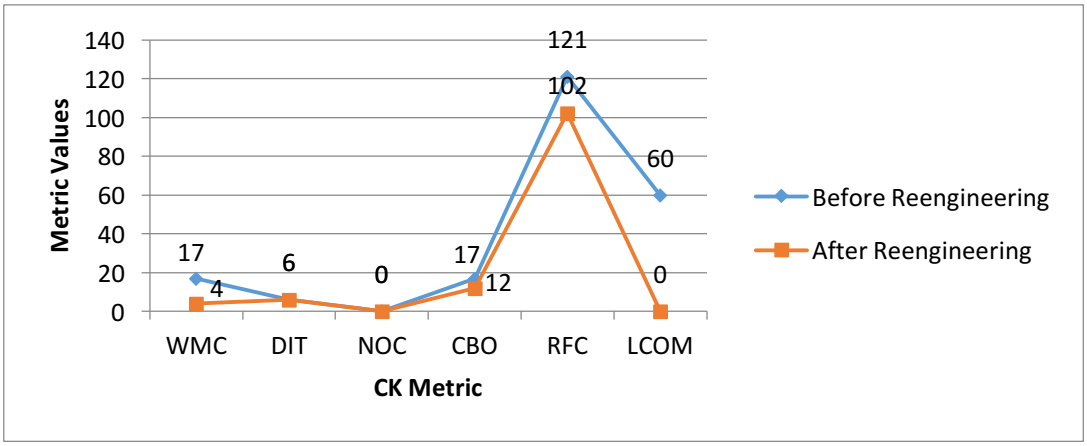

Figure 4. Metrics improvement analysis for IDE Module.

As visible in Figure 4, LCOM is reduced from 60 to 0. As the case with other classes, there is no change in NOC and DIT.

As validated by Basili et al. (1996), the improvement in CK metrics enhances the quality of the software. The proposed research uses reengineering to improve the CK metric values and hence improves software quality.

Validation of Quality Improvement in Execution Time

Another quality measure is to use the meantime to execute metrics (MTTE). In the proposed work, samples of 35 executions are collected by executing the software. Net beans 7.3 software is used to run Java software. System configuration includes i3 (4th Gen. Processor), 8GB RAM, 1 TB HDD, and JAVA7. Table 5 represents 35 samples of the execution time of classes in milliseconds. These results are measured before applying to reengineer the classes.

Table 5. Execution time of classes before applying Reengineering process.

Execution Time Samples	Login Class execution time Before Reengineering (ms)	IDE Class execution time Before Reengineering (ms)	UserDetail Class execution time Before Reengineering (ms)
1st Data Sample	113	145	60
5th Data Sample	122	143	25
10th Data Sample	117	144	21
15th Data Sample	114	143	19
20th Data Sample	119	143	18
25th Data Sample	134	143	16
30th Data Sample	153	159	15
35th Data Sample	114	150	16

As shown in Table 5, execution time in a millisecond is observed for login, IDE, and UserDetail Classes. Login is the first screen when the software is opened. IDE represents the environment carrying different menu-based options. The UserDetail is one option in the IDE. UserDetail takes very little time as compared to Login and IDE module as this module is available only after logging in to the software and is executed by clicking the UserDetail' option available on the IDE of the software. All the modules are executed one by one, and 35 samples are collected to analyze the execution time improvement. Execution time after applying the reengineering process to the software classes is again collected for comparison purposes. Table 6 shows the 35 samples collected after performing reengineering to the software.

Table 6. Execution time of classes after applying the reengineering process.

Execution Time Samples	Login class execution time after reengineering	IDE class execution time after reengineering	UserDetail class execution time after reengineering
1st Data Sample	106	139	38
5th Data Sample	120	129	37
10th Data Sample	101	125	29
15th Data Sample	102	130	29
20th Data Sample	103	130	26
25th Data Sample	103	156	28
30th Data Sample	138	134	26
35th Data Sample	117	130	23

The execution time for the three classes is collected in Table 6. This sample data is analyzed to validate the performance improvement in the software. Based on the collected data, the classes are compared in the subsection below.

Execution Validation for IDE Class

Figure 5 shows the executions of class before and after reengineering. The MTTE for the Old IDE module comes out to be 146.9 milliseconds, and for New IDE is 135.6 milliseconds. For old and reengineering IDE Modules, comparisons are shown in Figure 5. MTTE is the mean of observed time of execution in millisecond for thirty-five executions.

Figure 5. OLD IDE Modules Vs New IDE Module

Execution Validation for Login Class

For the login Module, MTTE, before applying the reengineering process, it comes out to be 123.2, and after reengineering, it is 106.4. Comparisons are depicted in Figure 6.

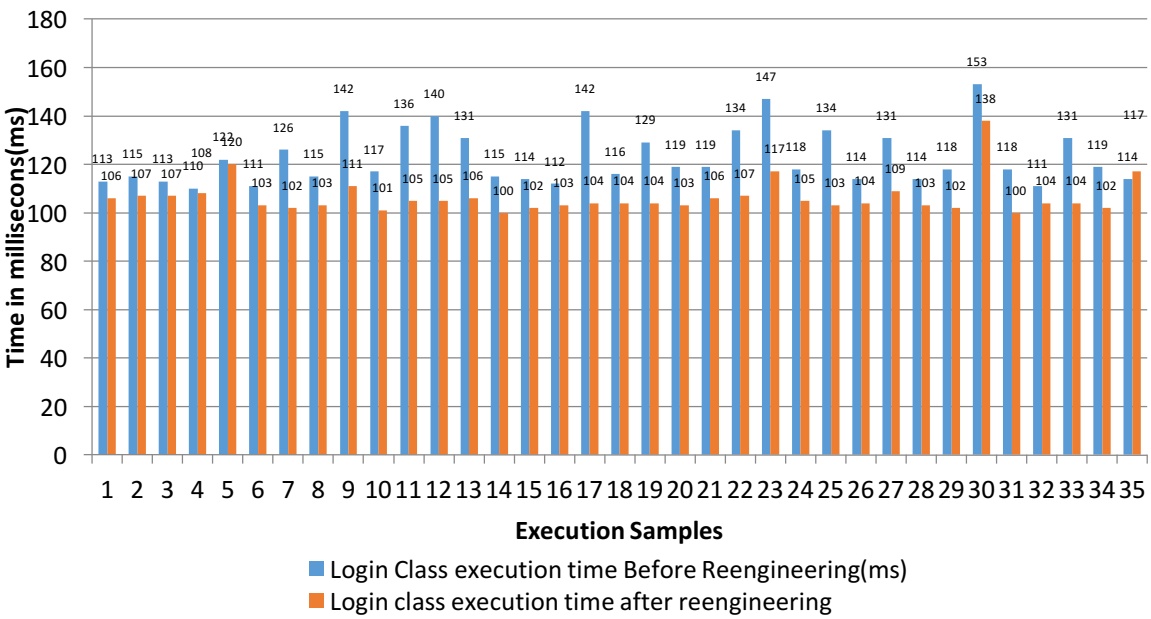

Figure 6. Old Login Vs. New login (Reengineered) Module.

Execution Validation for UserDetail Class

There is an exception in the case of UserDetail in terms of execution sample. The MTTE for the Old UserDetail module comes out to be 20.4 milliseconds, and for New, UserDetail is 29.6 milliseconds. The comparison is visible in Figure 7.

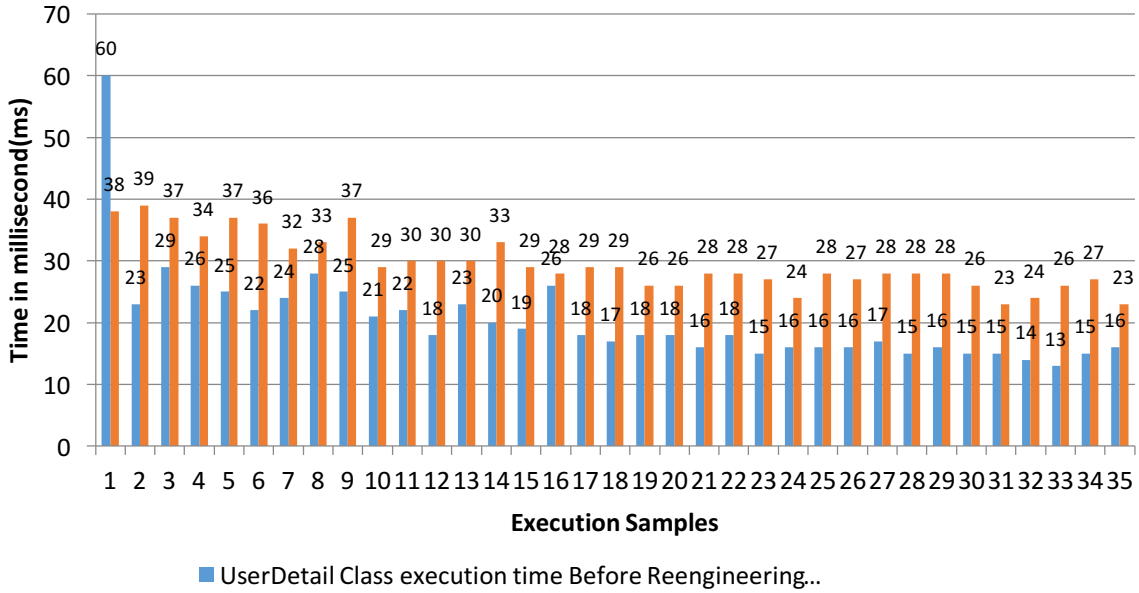

Figure 7. Old UserDetail Vs. New UserDetail (Reengineered) Module.

The MTTE for all three modules of the software is measured as 290.6 milliseconds for old software and 271.7 milliseconds for reengineered software. This shows the improvement in the MTTE for reengineered software. The comparison is visible in Figure 8.

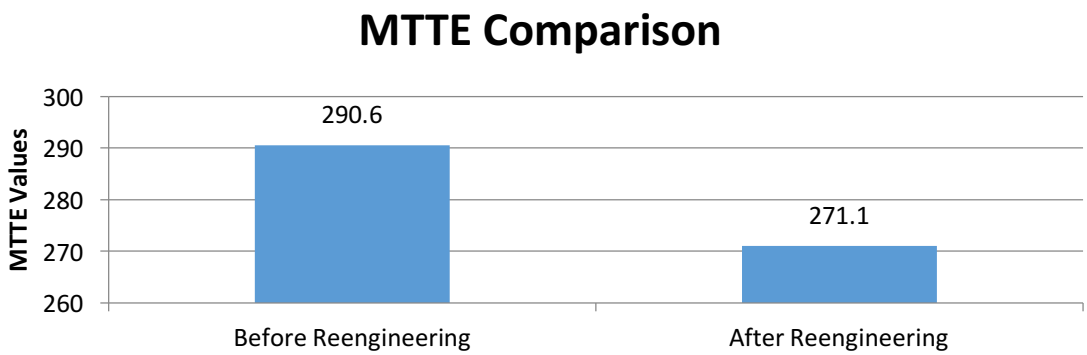

Figure 8. The MTTE for reengineered Software and Software before Reengineering.

Reduction in Maintenance Cost

For any software company, cutting the cost of maintenance is a major concern. Successful reengineering reduces the cost of maintenance. To validate this, consider the size allocation to classes. Effort estimation using planning poker for reengineering requirements are given in Table 7. Estimations for UserDetail, IDE, and Login class is 8, 5,

and 2, respectively. These estimations are performed using the Planning Poker estimation Technique (Singh et al., 2019).

Table 7. Reengineering Requirement Estimations using planning poker.

Sr. No.	Requirements	Estimated Story Points
1	Size estimation for UserDetail Class	8
2	Size estimation for IDE Class	5
3	Size estimation for LoginClass	2

For cost calculation, an updated agile effort estimation model (Rosa et al., 2017) is proposed. The basic estimation Model is given in equation (1).

$$Effort = 14.5xREQ0.5009 \quad (1)$$

Here Efforts are given in Person-Month (PM), and REQ represents Requirement size.

For the Java classes, size is measured using story points. So REQ in research work is proposed to represent the size of story points assigned to classes. The total number of story points assigned to classes is fifteen. So assigning REQ= 15 in equation (1), the effort measured is 56.29 PM. After performing reengineering, story points are reestimated for the classes, and the results are shown in Table 8. Software Classes are more maintainable after reengineering, and requirements size is also reduced. Estimated requirements sizes became 1, 2, and 3 for Login, IDE, and UserDetail classes, respectively.

Table 8. Requirement Estimations using planning poker after Reengineering.

Sr. No.	Requirements	Estimated Story Points
1	Size estimation for LoginClass	1
2	Size estimation for IDE Class	2
3	Size estimation for UserDetail Class	3

For cost calculation, the total story points assigned to classes are 6. So, assigning REQ= 6 in equation (1), the effort measured is 35.57 person-month. Thus, reengineering not only improves the maintainability, but also reduces the maintenance cost. Table 9 shows the maintenance cost reduction for the software.

Table 9. Maintenance Cost Reduction.

Sr. No.	cost of maintenance before Reengineering	cost of maintenance After Reengineering
1	56.29	35.57

Reduction in the cost of maintenance is significant as most of the work in the Industry is to perform software maintenance, and by performing reengineering, the reduction in the maintenance cost can be achieved.

COMPARISON

In the similar research (Sahoo et al., 2016), N-Process Model is proposed for reengineering Implementation. Reengineering using agile is achieved through quick planning and iteration. Reengineering is performed based on the Implementation Sequence Diagram (ISD) and the Implementation Class Diagram (ICD). Using ISD and ICD, the work is more focused on reverse engineering in place of overall software improvement. The reengineering process is depicted but the agility is not completely realized. Although the reverse engineering is supported with adequate diagrams, but research is lacking in the validations of overall quality improvement goals through reengineering.

CONCLUSION

Research validates the improvement in software quality due to the reengineering process. Various quality factors are considered, and the performance of the software is analyzed and validated for these factors. For measuring the internal design of the software, the CK Metric suite is used. Classes of the software that need to be reengineered are identified based on the value of the CK metric set. The CKJM tool is used to measure metric values. CK metric values are measured for classes before and after applying to reengineer. Classes are optimized to have better internal quality factors. It is observed that the quality of reengineered software is enhanced, and software complexity has been reduced, thus results in more maintainability. Table 10 summarizes the improvement in software quality in terms of internal design complexity, MTTE, and cost of maintenance. Complexities of all the classes have been reduced after reengineering.

Table 10. Quality Improvement Validation Summary.

Sr. No.		Metric	Before Reengineering	After reengineering	Percentage Improvement
1	1.1	Login Class Design Complexity	165	65	621.%
	1.2	IDE Class Design Complexity	221	124	
	1.3	UserDetail Class Design Complexity	332	83	
2		Total MTTE of Classes	290.6 ms	271.7 ms	6.5%
3		Cost of Maintenance	56.29 PM	35.57 PM	36.8%

This research work validates the quality improvement in terms of internal software design, execution time, and cost of maintenance for the reengineered system. It is important to observe that reengineering plays a vital role in keeping the software system alive for a longer period and thus cutting maintenance costs. The work can be extended by applying reengineering on more complex software of different domains and validating the outcome.

REFERENCES

Baabad, A. Zulzalil, H. B., Hassan, S., & Baharom, S. B. 2020. Software Architecture Degradation in Open Source Software: A Systematic Literature Review, in IEEE Access. 8:173681-173709. DOI: 10.1109/ACCESS.2020.3024671.

Tripathy, P. & Naik, K. 2014. Software Evolution and Maintenance: A Practitioner's Approach, Wiley Publication, pp-1-416.

Chidamber, S.R. & Kemerer, C.F. 1994. A metrics suite for object-oriented design. IEEE Transaction on Software Engineering. 20:476-493. DOI: 10.1109/32.295895,1994.

Singh, J., Singh, K., & Singh, J. 2019. Reengineering Framework to Enhance the Performance of Existing Software. International Journal of Advanced Computer Science and Applications (IJACSA), 10(5):536-543.

Grady, H. & Campbell, Jr. 1994. Reengineering to Increase Maintainability and Enable Reuse. In 4th NSWC Systems Reengineering Technology Workshop, 1994.

Cagnin, M., Penteado, R., Mtasiero, P. C., & Maldonado, J. C. 2001. Comparison of maintainability improvement by segmentation and reengineering-a case study, In 5th European Conference on Software Maintenance and Reengineering, Lisbon, Portugal, pp. 158-167.2001.doi: 10.1109/CSMR.2001.914980

Khomh, F. & Gueheneuc, Y. .2018. Design patterns impact on software quality: Where are the theories? In 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso. pp.15-25.

Muzammula, M. & Awaisb, M. 2018. An empirical approach for software reengineering process with relation to quality assurance mechanism, Advances in Distributed Computing and Artificial Intelligence Journal .7(3):31-45.

Smiari, P. & Bibi, S. 2018. A Smart City Application Modeling Framework: A Case Study on Re-engineering a Smart Retail Platform. In 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Prague, pp.111-118. DOI: 10.1109/SEAA.2018.00027.

Majthoub, M., Qutqut, M. H. & Odeh, **Y. 2018.** Software Re-engineering: An Overview. In 8th International Conference on Computer Science and Information Technology (CSIT), Amman, pp. 266-270. DOI: 10.1109/CSIT.2018.8486173.

Binanto, I., Warnars, H. L. H. S., Gaol, F. L., Abdurachman, E., & Soewito, **B. 2018.** Measuring the quality of various version an object-oriented software utilizing CK metrics. In International Conference on Information and Communications Technology, Yogyakarta, pp. 41-44. DOI: 10.1109/ICOIACT.2018.8350760.

Malhotra R. & Jain, J. 2019. Analysis of Refactoring Effect on Software Quality of Object-Oriented Systems. In: Bhattacharyya S., Hassanien A., Gupta D., Khanna A., Pan I. (eds): International Conference on Innovative Computing and Communications. Lecture Notes in Networks and Systems, 56. Springer, Singapore.

Basili, V. R., Briand, L. C., & Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators, In proc. of IEEE Transactions on Software Engineering. 22(10): 751-761.

Shyam, R., Chidamber, David P. Darcy, & Kemerer, C. F. 1998. Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis, IEEE Trans. Software Eng. 24:629-639.

Fernandez-Diego, M. Mendez, E. R. Gonzalez-Ladron-De-Guevara, F. Abrahao, S., & Insfran, E. 2020. An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review, In IEEE Access, 8(166768:166800). DOI: 10.1109/ACCESS.2020.3021664.

Usman, D., Mendes, E., Weidt, F., & Britto, R. 2014. Effort Estimation in Agile Software Development: A Systematic Literature Review. In 10th International Conference on Predictive Models in Software Engineering, pp.82-91.

Haugen, N.C. 2006. An empirical study of using planning poker for user story estimation. In AGILE 2006 Conference (AGILE'06), Minneapolis, pp. 23–34.

Molokkenostvold, K., Haugen, N.C., & Benestad H.C. 2008. Using planning poker for combining expert estimates in software projects, Journal of Systems and Software. 2106–2117.

Mahnic, V. & Hovelja, T. 2012. On using planning poker for estimating user stories. Journal of Systems and Software, LXXXV (9): 2086-2095. http://dx.doi.org/10.1016/j.jss.2012.04.005

Cohan, M., 2006 Agile Estimating and Planning, Pearson Education.

Francisco, J. P., Francisco, R., Garcıa, F., & Piattini, M. 2011. A software maintenance methodology for small organizations: Agile_MANTEMA, Journal of Software Maintenance and evolution, ©John Wiley & Sons, Ltd. 851-876.

Gandomani,T. J.Faraji, H. and Radnejad, M. 2019. Planning Poker in cost estimation in Agile methods: Averaging Vs. Consensus. In 5th Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, Iran, pp. 066-071. DOI: 10.1109/KBEI.2019.8734960.

Tarwani, S. & Chug, A. 2016. Agile Methodologies in Software Maintenance: A Systematic Review. Informatica, 40(4):415-214.

Ming, H., Verner, J., Zhu, L. & Babar, M. A. 2004. Software quality and agile methods, In Proc. of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004., Hong Kong.520-525. DOI: 10.1109/CMPSAC.2004.1342889

Singh, J., Singh, K. & Singh, J. 2019. Reengineering framework for open source software using decision tree approach, International Journal of electrical and computer engineering (IJECE). 9(3):2041-2048.

Singh, J., Singh, K., & Singh, J. 2019. Reengineering Cost Estimation using Scrum Agile Methodology. International Journal of Computer Information Systems and Industrial Management Applications. 11:208-218.

Spinellis, D. 2005. Tool writing: A forgotten art?" IEEE Software, 22(4):9–11, (DOI:10.1109/MS.2005.111.) Tool Available at http://www.spinellis.gr/sw/ckjm/doc/ver.html

Singh, J., Singh, K., & Singh, J. 2017. Identification of requirements of software reengineering for JAVA projects. In International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India. 931-934.

Sneed, H.M. 2008. 20 Years of Software-Reengineering: A Resume". In 10th Workshop on software reengineering (WSR'08). 115-124.

Rosa, W., Madachy, R., Clark B., & Boehm, B. 2017. Early Phase Cost Models for Agile Software Processes in the US DoD. In ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Toronto,.30- 37.

Sahoo, A., Kung, D., & Gupta, S. 2016. An Agile Methodology for Reengineering Object-Oriented Software. In 28th International Conference on Software Engineering & Knowledge Engineering, California, USA, pp. 638-648.