Requirement engineering problems impacting the quality of software in Sub-Saharan Africa

Andrew Quansah, Asiamah Emmanuel, Bright Kyeremanteng, Esther Ntow Kesse
Department of Electrical and Computer Engineering, School of Engineering, University of Energy and Natural Resources, Sunyani, Ghana

Article Info

Article history:

Received Jul 12, 2022
Revised Nov 8, 2022
Accepted Nov 23, 2022

Keywords:

Formal inspections
Requirement engineering
Software development life cycle
Software engineering
Software quality

ABSTRACT

Poor software quality has led to tremendous financial losses, necessitating the goal of this study. This study aimed to find out the major cause of poor quality of software and propose solutions to mitigate the problem. Histogram analysis was conducted using data from software development firms' online applications used to track all defects and issues for each project, which are logged under a unique project ID. The requirement engineering stage was found to produce the most problems that directly or indirectly impact software quality. The capability maturity model integration, prototyping, ISO 9001, Walkthroughs, and Formal Inspections were proposed as solutions that could be used to mitigate the software quality problems that arise from the requirement engineering stage in the software development life cycle.

Corresponding Author:

Asiamah Emmanuel
Department of Electrical and Computer Engineering, School of Engineering
University of Energy and Natural Resources
BS-0061-2164, Sunyani, Ghana, West Africa
Email: info@uenr.edu.gh

1. INTRODUCTION

The world is governed by the use of software. Software affects nearly every aspect of our lives. From our normal day-to-day tasks to really complex tasks, almost everyone uses the software. Software is now omnipresent in almost every society [1], [2]. A lot goes into the engineering and development of software products that are now in use almost everywhere. A simple check against the ISO/IEC 5055:2021 which is an international organization for standards (ISO) standard for assessing software product quality based on four business-critical elements: Security, reliability, performance efficiency, and maintainability reveals this fact [3]. Poor software quality in the United States alone in 2020 was determined to have costed around $2.08 Trillion. This was primarily because of cybersecurity failures, operational failures, unsuccessful IT Projects, and Legacy Systems. The largest contributor to this cost was determined to be failures in operational failures, followed by unsuccessful development projects [4].

Due to the astronomical cost that can be incurred by the development of poor-quality software, this study aims to identify which stage in the software development life cycle contributes most to poor software quality and also propose solutions to mitigate it. Recent literature on software quality has focused on other metrics other than investigating the major stage in the software development life cycle (SDLC) that contributes to most of the problems that result in poor quality. Dlamini *et al.* [5] in order to monitor software quality throughout the later stages of the development process, they analyzed the current software quality models. They proposed a system architecture to evaluate the quality with embedded external systems, which rather adds to the complexity of an already complex process. A quality prediction model was created by Mohapatra *et al.*

[6] to determine whether or not software is prone to errors. They sought to identify the problematic components so that they might be fixed during further testing. Although this method might predict some faults, it may miss other faults due to the level of accuracy, which may have great consequences. This also throws a very complex solution to the problem and may handicap software development firms that do not have the technical expertise to create predictive models. O'Regan discussed the importance of process quality in the software engineering process. He stated adhering to best practices was crucial for the production of high-quality software products. This highlights the importance of paying attention to each stage of the process, especially the stages likely too problematic [7]. In their study, Martinez-Fernandez *et al.* [8] sought to determine if the inclusion of quality models in software analytics tools produced information regarding product or development process quality that was clear, accurate, valuable, and pertinent. For software firms without these analytical tools due to economic constraints, there is a need to find other straightforward solutions to curb the challenge of poor software quality. From the standpoint of software quality assurance, Chen *et al.* [9] investigated the mechanism that leads to software failure, assessed the degree to which the failure mechanism has an impact on software quality, suggested a management strategy to raise software quality, and created a quality management model. They didn't highlight which stage contributes most to software failure.

2. METHOD

2.1. The profile of the organization

The organization that was studied for this research is a Ghanaian-based firm that deals in the design and development of software for use in various sectors of the Ghanaian economy. The classic waterfall model is utilized in the software development life cycle. Among other programming languages, MySQL and Java are the languages that are mostly used. Requirements are gathered by the software developers themselves at the outset of any software project. The team leader gives a briefing to the developers, who then hold meetings with them to collect requirements. They're also in charge of preparing the software project's software requirements specifications (SRS) document. Frequently, the software product built is mostly subs standard, and this is realized towards the conclusion of the development life cycle, posing numerous challenges before going live.

2.2. Principal cause of software quality problems

To fix the underlying source of software quality problems it had to be first identified. The Histogram is among the seven quality control techniques that may be employed to swiftly pinpoint the underlying source of the poor-quality software products produced by the company. As a result, the Histogram was utilized to identify the primary reason for the low software quality in the organization in question. The procedure followed in the histogram analysis is as follows: Table 1 shows a series of challenges that have been experienced in projects based on information from company members. The organization employs the use of an online application to track all defects and problems for each project, which are logged under a unique project ID. All modification inquiries, rework inquiries, error repair inquiries, and support demands from customers are included in the problems. A sample of two human resource management projects was selected, and the series of problems documented for the projects were examined. Following the analysis, related issues were put together under a single problem. As shown in Table 2, each problem has been awarded a score depending on the frequency with which it occurred. The goal of this study is to improve the software's quality, which will lead to increased customer satisfaction. As a result, the technique for grading each problem is dependent on the number of times the issue or complaint has been logged on the web tool. Reduced complaints and issues will aid in improving software quality. The results of each cause were then tabulated and the histogram is then presented in Figure 1.

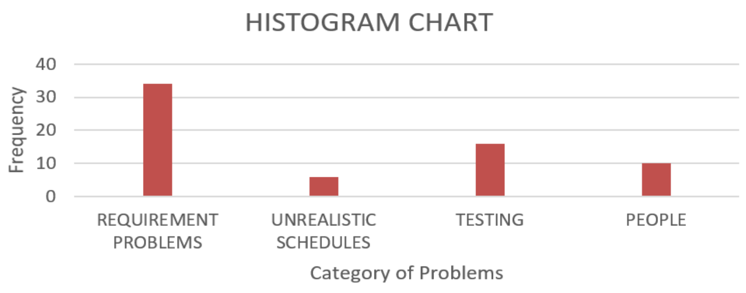

Figure 1. Histogram demonstrating the principal cause of software quality problem (Step 4)

Table 1. Breakdown of the bugs entered (the first step)

S/N	Issues	Frequency
1	Customers received the requested service very late due to a lack of developers.	6
2	Bug fixing was delegated to inexperienced personnel making the process consume time.	4
3	A few requirements were not found in the SRS document	7
4	Requirements that are lacking in detailed information.	4
5	Developers' inability to comprehend some requirements	6
6	Client-developer misunderstandings.	3
7	Changes in requirementsafter coding had already begun.	8
8	Reusing modules without doing a thorough analysis.	1
9	Reuse of the same components (module) several times	3
10	Incomplete testing because of schedule pressure.	4
11	Incomplete correction of bugs.	10
12	Absence of personnel responsible for testing.	3
13	Timelines Estimates were off because developers weren't consulted.	2
14	Clients were unsatisfied with requirements in the late stages of SDLC	2
15	Developers were unfamiliar with the customer's specific business.	3

Table 2. Score per cause

Category	Requirements gathering problems	Poor schedules	Inadequate testing	Inadequate staffing
Total	34	6	16	10

3. RESULTS AND DISCUSSION

For an ultra-quality data presentation, the data obtained from the bugs entered was subjected to regression analysis using the data analysis tool in Microsoft excel. The analysis was based on the responses given by the senior developers and junior developers as shown in Table 3. In the Software development firm. The results from the regression analysis shown in Table 4. shows an r-value of 0.5698 and 0.6564 for both senior developers and junior developers respectively. This, therefore, confirms that the principal cause of software quality problems is not discriminant towards just a particular group of employees. Both the senior developers and junior developers in the software firm share similar grievances.

As seen by the histogram chart in Figure 1, the primary reason for low-quality software is requirements difficulties. The figure shows that 51.5% of difficulties are caused by requirements. Software quality may be considerably enhanced by paying close attention to the requirement engineering process. Furthermore, as previously said, requirements are acquired by engineers who are unfamiliar with the different types of requirements-gathering techniques. There is also no standard structure for gathering requirements. Requirements represent the foundation of every project and, as such, must be given the highest care, as fixing faults at those latter phases in the life cycle entails a higher cost to the organization. If the list of requirements is almost flawless from the start of the software development life cycle, it will surely increase quality while saving money, resources, and time. This increases the likelihood of the software project's success.

Table 3. Entry of bugs by developers

	Senior developers	Junior developers	Total
Issue 1	4	2	6
Issue 2	3	1	4
Issue 3	4	3	7
Issue 4	2	2	4
Issue 5	1	5	6
Issue 6	1	2	3
Issue 7	4	4	8
Issue 8	0	1	1
Issue 9	1	2	3
Issue 10	3	1	4
Issue 11	3	7	10
Issue 12	1	2	3
Issue 13	0	2	2
Issue 14	1	1	2
Issue 15	0	3	3

Table 4. Regression analysis presentation

	Multiple R	R Square	Adjusted square	Standard error	Observa-tions	Intercept	Coefficients	P-value
Senior developpers	0.754871749	0.569831357	0.536741462	1.702549518	15		1.254201681	0.00114168
Junior developers	0.810211388	0.656442493	0.630014993	1.521528523	15		1.352348993	0.00024996

3.1. SOME SOLUTIONS PUT FORWARD

This segment discusses remedies and approaches that the firm and others may utilize to address the quality issue created by the present shortfalls in the requirement engineering process. These approaches each have their various advantages and disadvantages. It is important to compare them to see which one has least disadvantages and also the best advantages.

3.1.2. Capability maturity model integration

According to the software engineering institute (SEI), capability maturity model integration (CMMI) assists in the incorporation of multiple organizational roles, and also in creating process improvement goals, directing quality procedures, and giving a reference point for assessing current processes. CMMI recognizes 25 process areas during the development process. Every process area has its own set of "specific goals" and "specific practices" that aid in achieving those goals [10].

3.1.3. ISO 9001 standard

ISO 9001 is a set of rules that cover essential steps in the software development process. It ensures the effectiveness of processes, checks for errors in outputs, conducts regular reviews of specific processes, and promotes continuous development. The ISO 9001 standard involves managing the processes of a company, so that it may fulfill client expectations, offer consistent service, and continuously improve quality. The ISO 9001 standard uses document control as a control and verification tool [11].

3.1.4. Formal inspection

A Formal inspection is a technique that can aid in considerable software quality improvements. An inspection is a thorough technical examination that identifies problems as close to their source as possible. This procedure has the potential to significantly improve software quality [12]. Inspections conducted on Motorola's Iridium project, for example, found 80% of the faults present, whereas less formal examinations found just 60% [13]. Formal inspections help to ensure that defects are removed as quickly as possible. Formal inspection is typically followed by firms deemed to be "best in class" globally, according to research [14].

3.1.5. Walkthroughs

A walkthrough is a meeting that is unstructured in which requirements documents are examined and only after that they are passed on to the development team.

3.1.6. Prototyping

Building a prototype which is an early kind of the intended software that can be used for testing and gathering responses from the software system's customers and stakeholders is what is termed as prototyping. Prototyping is a method that allows for revisions till the program is complete and the client's expectations are met. We use prototyping for eliciting requirements since stakeholders may play with the software straight away and outline its strengths and faults [15].

3.2. RECOMMENDED SOLUTION: REQUIREMENT INSPECTION

When using the CMMI model, a company should consider each level as a target [16]. Furthermore, when it comes to the requirement engineering phase of software development, CMMI does not specify a specific path to the next level [17]. Rather than enhancing software quality, it tends to focus on management difficulties. In terms of its cost, it is quite expensive to contact CMMI experts to get CMMI level certified, this also being another significant disadvantage [18].

Obtaining ISO 9001 certification is an expensive procedure, particularly for small businesses [19]. Furthermore, the certification is strongly reliant on documentation and procedures, necessitating further hiring and training. Furthermore, research has revealed that the ISO standard registration process takes a long time [20]. In contrast to inspections, walkthroughs are different from inspections in that the author takes lead and chairs meetings in cases where no other specialized review responsibilities are usually specified. Walkthroughs are casual since they often do not follow a well-defined method, and also do not establish entry and exit criteria, involve no organizational reporting, and provide no system of measurement. Therefore, walkthroughs cannot be the operative approach for the firm studied in the paper [21].

Prototypes must be developed quickly so that they can be used early in the elicitation process. Due to a shortage of human resources, especially in the team of developers, they will be unable to produce prototypes quickly for the company discussed in this paper. The cost of developing a prototype for every project embarked on may proscribe the company under consideration [22].

The one inspection that should never be skipped is the requirement inspection [23]. Inspection avoids by saving a middling of nine labor hours in downstream rework for each significant problem discovered [24]. Requirements are gathered and recorded as precise software requirements during the requirements-gathering process. In this proposed solution, the SRS is the document to be examined. R1 inspection is what it's called. A requirements inspection verifies that specifications are written well, that is, every requirement in the SRS is consistent, precise, clear-cut, appreciable, and testable [25]. The knowledge gained from the inspection allows the remainder of the work to be completed more efficiently.

Furthermore, the company will not need to allocate additional resources to the inspection; in its place, an excellent plan will be established that can be followed to do the inspection. As a result, the inspection may be an effective tool for improving quality. As a result, the advised remedy is to establish a requirement inspection at the firm under consideration.

4. CONCLUSION

Several challenges arise in any software development project that has a direct or indirect impact on software quality. Software quality should not be bargained for because it shows how the software's needs and characteristics have been met, as well as whether or not customer satisfaction has been attained. Software is governed mostly by requirements. Users, developers, customers, and any other stakeholders participate in gathering requirements as part of a collaborative decision-making process. The proposed recommendation "requirements inspection" has been demonstrated for the software firm in Ghana, where a lot of flaws in the requirement engineering phase, including requirements gathering and management, the proposed recommendation "requirements inspection" can help. Many firms have used inspection as a technique to find faults and improve software quality all over the world, thus it could be a successful option. The organization will now have the ability to come up with concise full, and testable needs by conducting a formal inspection of requirements. This will not only save the organization money on maintenance and rework, but it will also enhance quality a lot and develop a quality-based way of life. Additionally, the benefits of proper requirement engineering will be seen throughout the SDLC.

REFERENCES

[1] C. R. Dexeus, "The deepening effects of the digital revolution," in *The Future of Tourism*, Cham: Springer International Publishing, 2019, pp. 43–69.

[2] F. J. Furrer, "Software everywhere," in *Future-Proof Software-Systems*, Wiesbaden: Springer Fachmedien Wiesbaden, 2019, pp. 3–10.

[3] ISO/IEC 5055 "Information technology-software measurement-software quality measurement-automated source code quality measures." ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission), 2021, [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:5055:ed-1:v1:en

[4] H. Krasner, "The cost of poor softwarequality in the USA," 2020. [Online]. Available: https://www.disputesoft.com/wp-content/uploads/2021/01/CPSQ-2020-Software-Report.pdf.

[5] G. Dlamini *et al.*, "Metrics for software process quality assessment in the late phases of SDLC," in *Intelligent Computing. SAI 2022. Lecture Notes in Networks and Systems*, Cham: Springer, 2022, pp. 639–655.

[6] A. Mohapatra, S. Pattnaik, B. K. Pattanayak, S. Patnaik, and S. R. Laha, "Software quality prediction using machine learning," in *Advances in Data Science and Management . Lecture Notes on Data Engineering and Communications Technologies*, Singapore: Springer, 2022, pp. 137–146.

[7] G. O'Regan, "Software quality assurance," in *Concise Guide to Software Engineering. Undergraduate Topics in Computer Science*, Cham: Springer, 2022, pp. 239–246.

[8] S. Martinez-Fernandez *et al.*, "Continuously assessing and improving software quality with software analytics tools: a case study," *IEEE Access*, vol. 7, pp. 68219–68239, 2019, doi: 10.1109/ACCESS.2019.2917403.

[9] Y. Chen, J. Chen, Y. Gao, D. Chen, and Y. Tang, "Research on software failure analysis and quality management model," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Jul. 2018, pp. 94–99, doi: 10.1109/QRS-C.2018.00030.

[10] M. Hoggerl and B. Sehorz, "An introudction to CMMI and its assessment procedure," 2006, [Online]. Available: https://www.softwareresearch.net/fileadmin/src/docs/teaching/WS05/SaI/Paper_Hoeggerl_Sehorz.pdf.

[11] J. J. Tarí, J. F. Molina-Azorín, and I. Heras, "Benefits of the ISO 9001 and ISO 14001 standards: A literature review," *Journal of Industrial Engineering and Management*, vol. 5, no. 2, Dec. 2012, doi: 10.3926/jiem.488.

[12] A. Ahad, "Software inspections and their role in software quality assurance," *American Journal of Software Engineering and Applications*, vol. 6, no. 4, p. 105, 2017, doi: 10.11648/j.ajsea.20170604.11.

[13] N. Brown, "High-leverage best practices: what hot companies are doing to stay ahead," *Cutter IT Journal*, vol. 12, no. 9, pp. 4–9, 1999.

[14] N. Brown, "Industrial-strength management strategies," *IEEE Software*, vol. 13, no. 4, pp. 94–103, Jul. 1996, doi: 10.1109/52.526836.

[15] R. M. Kimmond, "Survey into the acceptance of prototyping in software development," in *Proceedings Sixth IEEE International Workshop on Rapid System Prototyping. Shortening the Path from Specification to Prototype*, pp. 147–152, doi: 10.1109/IWRSP.1995.518584.

[16] Y. Sun and X. (Frank) Liu, "Business-oriented software process improvement based on CMMI using QFD," *Information and Software Technology*, vol. 52, no. 1, pp. 79–91, Jan. 2010, doi: 10.1016/j.infsof.2009.08.003.

[17] I. Keshta, M. Niazi, and M. Alshayeb, "Towards implementation of requirements management specific practices (SP1. 3 and SP1. 4) for Saudi Arabian small and medium sized software development organizations," *IEEE Access*, vol. 5, pp. 24162–24183, 2017, doi: 10.1109/ACCESS.2017.2764490.

[18] J. Iqbal, R. B. Ahmad, M. H. N. M. Nasir, M. Niazi, S. Shamshirband, and M. A. Noor, "Software SMEs' unofficial readiness for CMMI®-based software process improvement," *Software Quality Journal*, vol. 24, no. 4, pp. 997–1023, Dec. 2016, doi: 10.1007/s11219-015-9277-3.
[19] T. Lazibat, M. Damić, and I. Markotić, "Determinants, barriers and outcomes OF ISO 9001 implementation in SMEs," *Poslovna izvrsnost - Business excellence*, vol. 16, no. 1, pp. 93–104, Jun. 2022, doi: 10.22598/pi-be/2022.16.1.93.
[20] A. E. Wilcock and K. A. Boys, "Improving quality management: ISO 9001 benefits for agrifood firms," *Journal of Agribusiness in Developing and Emerging Economies*, vol. 7, no. 1, pp. 2–20, May 2017, doi: 10.1108/JADEE-12-2014-0046.
[21] M. Ciolkowski, O. Laitenberger, D. Rombach, F. Shull, and D. Perry, "Software inspections, reviews & walkthroughs," in *Proceedings of the 24th international conference on Software engineering - ICSE '02*, 2002, p. 641, doi: 10.1145/581339.581422.
[22] B. Camburn *et al.*, "Design prototyping methods: state of the art in strategies, techniques, and guidelines," *Design Science*, vol. 3, p. e13, Aug. 2017, doi: 10.1017/dsj.2017.10.
[23] T. G. Kirner and J. C. Abib, "Inspection of software requirements specification documents," in *Proceedings of the 15th annual international conference on Computer documentation - SIGDOC '97*, 1997, pp. 161–171, doi: 10.1145/263367.263389.
[24] T. Gilb and D. Graham, *Software inspection*. Addison-Wesley, Reading, 1993.
[25] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in software quality: concepts and definitions of software quality," 1977. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA049014.pdf.

BIOGRAPHIES OF AUTHORS

Andrew Quansah is a lecturer at the University of Energy and Natural Resources. He holds an MSc. Microelectronics and Wireless Intelligent System from Coventry University in the UK. He received his BSc. In Electrical and Electronics Engineering from the Kwame Nkrumah University of Science and Technology. His research interest includes wireless communication systems, artificial intelligence, software engineering, computer architecture, and microprocessors. He can be contacted at email: Andrew.quansah@uenr.edu.gh.

Asiamah Emmanuel received a B.Sc. degree in computer engineering from the University of Energy and Natural Resources, Ghana, and currently works as a teaching assistant in the Department of Electrical and Computer Engineering. His research interests include software engineering, artificial intelligence, and biomedical engineering. He can be contacted at email: easiamah81@gmail.com.

Bright Kwasi Kyeremateng received a B.Sc. degree in Electrical Engineering from the University of Energy and Natural Resources, Ghana, and currently works as a teaching assistant in the Department of Electrical and Computer Engineering. His research interests include software engineering, artificial intelligence, biomedical engineering. He can be contacted at email: brightkyeremantengkwasi@gmail.com.

Esther Ntow Kesse received her BS.c in Electrical and Electronic Engineering from the University of Energy and Natural Resources (2021). Her current research interests include software management, microgrids, renewable energy, modeling and control of power converters, and distributed generation. She can be contacted at email: estherntowkesse@gmail.com.