Software testing for reliability and quality improvement

Mouad Bajjouk
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP058518@mail.apu.edu.my

Chandra Reka Ramachandiran
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
chandra.reka@apu.edu.my

Muhammad Ehsan Rana
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
muhd_ehsanrana@apu.edu.my

Sivananthan Chelliah
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
sivananthan@apu.edu.my

***Abstract*—Software systems are hard to complete as customers became hard to satisfy and their requirements are more complicated. In this paper, quality factors that affect software are discussed, with some well-known standards that improve and assist while planning for quality assurance. The testing strategies and types that contribute to improving quality are also highlighted. A development methodology was mentioned as it gives more support to the testing phase.**

***Keywords*—Software reliability, software quality, testing methodologies.**

I. INTRODUCTION

Nowadays, software systems became more complex and sophisticated to develop. Customers' requirements are also getting more complex to fully understand and implement. The competition in software development is getting high, as every development company is doing its best to attract more customers and do more profit. Therefore, software products must be reliable and high quality to meet and fulfill customers' expectations and requirements. Software reliability can be divided into many sub-attributes that must be addressed to improve overall software quality.

As companies are using different development methodologies, testing is one of the most important phases in any methodology, it can detect errors and bugs if available in the software. However, testing is not as easy as it seems, because many types of testing are existing, and every testing type can be performed in a particular situation. Therefore, this crucial phase must have more attention to improve reliability.

II. SOFTWARE RELIABILITY

[1] highlighted that software reliability can be defined as the probability of failure-free of a software in a specified environment for a given time.

According to [2], software reliability has developed many models from the early years until today to assess software reliability and improve the overall quality, including the testing while developing the software.

Furthermore, software reliability prediction models are being used before testing the software using a collection of data such as project complexity, used programming languages, architecture, etc. And after the software is being tested, a reliability model which is a mathematical formula is used to assess the software reliability by using the collected failure rates and fault data [3].

[4] stated that continuous testing is one of the most important steps in software development, as it can give to the quality assurance team an overall review of the quality of the developed software, which can be used to improve software reliability and meet customers' expectations.

Additionally, [5] noted that software testability is a considered important in software project development; many international standards like ISTQB, IEEE, ISO, IEC, and MIL-STD defined software testability as the degree to which a software or software component can be tested and validated according to the tests criteria imposed by the development company.

Testing has improved software reliability and code implementation (programming skills) two times more [6].

III. SOFTWARE RELIABILITY ATTRIBUTES

[7] defined in his statement that software reliability can have many attributes (non-functional requirements) that can produce better software quality if well considered. [7] has listed these attributes as the following order: availability, efficiency, installability, integrity, interoperability, modifiability, performance, portability, reliability, reusability, robustness, safety, scalability, security, usability, and verifiability. [7] also mentioned that these attributes must be combined in a good way as some of them can have a bad impact on the system if combined.

According to [8], availability can be defined as the time or period that a system or application can be accessed by the users; high availability can be 99.99%, the other left percentage can be reserved for the maintenance activities.

For software efficiency, it can be defined as the quantity or rate of the number of inputs needed to achieve a higher quantity of output under any workload [9]; [9] also highlighted that scalability which is increasing the capacity of a software by adding more functionalities and services.

[10] defined software installability as the ability for the software to be installed easily in a specific environment. [10] explained that software installability can be categorized from ‘easy to install’ to ‘hard to install’, or sometimes ‘failed to install’ due to implementation errors; the two categories (easy and hard) have been assessed using the installation time factor.

[11] asserted that integrity is one of the important aspects of a software, as integrity means the protection of the source code (logic) and the data of the software; and every modification on the internal implementation of the software can change its intended behavior; integrity can be a subpart of the security attribute, as strong security in software systems mean the ability to prevent any incoming malicious attacks, no matter what type of the attack is.

For the interoperability attribute, it can be defined as the ability of the software or hardware to work and interact smoothly with other systems [12].

As technology is evolving, customers’ requirements are also constantly changing, which forces developers to frequently modify the software to adapt to these changes; Modifiability is the ability of the software to be upgraded and improved to meet users’ requirements [13].

[14] claim that performance is also one of the most important aspects of a software. Performance is the ability of a software to quickly respond and interact with the user. [14] also stated that the throughput can be a part of the performance attribute.

As stated before, computer environments are frequently changing due to the rapid evolvement of the technology; therefore, portability is important for software systems to migrate from old to new computing environments [15].

[16] defined robustness in the testing approach as the capability of a software to be stable under unexpected or stressful environments, and function correctly while getting invalid inputs.

As software systems can be used in critical situations like controllers, safety attribute is the potential of a software system or controller to work properly during critical operations and environments [17].

IV. SOFTWARE QUALITY

Software quality as defined by the IEEE organization in their IEEE 730-2014 standard, is the ability of the developed software to meet all user’s requirements as well as the company’s stakeholders’ requirements [18].

Unlike [7], [18] states that software quality attributes are containing reliability and all other previously mentioned attributes by [7]. These quality attributes can be assured via the software quality assurance (SQA). The SQA consists of many activities to ensure that the produced software is adequate in terms of software services and is fulfilling all the described requirements by the client [18].

In [18]’s statement, software quality can be achieved by different approaches, during all the software development lifecycle (SDLC) phases. A software system that has several errors can be low quality, these errors can be caused by wrong requirements, lack of communication between developers and clients, deviating from client’s requirements, poor software design, implementation errors, non-conformance of documentation and code, inefficient testing processes, faulty user interfaces and procedures, and finally erroneous documentation [18].

V. QUALITY STANDARDS

Achieving software quality can be done by following the well known international standards such as ISO/IEC and IEEE [18]. The ISO/IEC 9000 standard family can be followed by the software manufacturer organization to achieve an optimal software quality management system, as this family of standards provides guidelines to reach a good level of quality[18].

For the quality assurance, [18] stated that IEEE Std. 730-2014 can be followed. For the lifecycle processes, ISO/IEC/IEEE 12207:2008 is the standard that helps to improve the SDLC processes. Finally, for the software verification and validation, IEEE Std. 1012-2012 can provide guidelines to better perform this task.

VI. SOFTWARE TESTING

Software testing, which can be defined as the evaluation of the produced software, is a very important step for ensuring a good software quality [18]. To add more clarity to this statement, [19] highlighted that software testing can be divided into two major categories, which are static testing and dynamic testing. The first type which is static is performed to evaluate the requirement document, software design, and source code through inspections, walkthrough, and reviews. On the other hand, dynamic testing is the examination of the produced software by executing it using different inputs; this type of tests let developers to observe the performance and behavior of the system [19].

A good software quality cannot be done using one type of dynamic testing. Hence, many types of dynamic testing are available like unit testing, integration testing, system testing, and acceptance testing [19].

These tests are not done without planning as [18] states, the software producer determine test strategies, then it plans and designs tests in order to perform them upon the software system. Two more important testing types are black box testing and white box (or glass box testing) [18]. The black box test consists of testing the system without knowing its code, it mainly serves for detecting bugs and malfunctioning functions. However, white box testing is a test where the internal code can be examined to find bugs and errors. Generally, black box testing can do more than the glass box testing, as it checks for correctness, availability, reliability, stress, security, usability, maintainability, flexibility, testability, portability, reusability, and interoperability. However, glass box testing can only serve for correctness, maintainability, and reusability [18].

All the mentioned tests including either automated or manual testing can serve for improving the overall reliability and quality of the produced software [20]. [20] added that development methodologies also play a great role in improving software quality, as an adequate methodology can give more efficiency to all lifecycle phases which includes the testing phase. [20] also refers to the scrum agile methodology as a good way of improving software quality as requirements are clear and unambiguous. Hence, performed tests can assess these requirements to check if it is conforming to the user’s needs.

TABLE I. LITERATURE REVIEW MATRIX

Author/ Date	Theoretical/ Conceptual Framework	Research Question(s)/ Hypotheses	Methodology	Analysis & Results	Conclusions	Implications for Future research	Implications For practice
Nikolay Pavlov, Georgi Spasov, Asen Rahnev, Nikolay Kyurkchiev (2018)	Using Gompertz model to measure software reliability.	Can Geompertz be the best model?	Using numerical data in the old and new model using the Gompertz model to calculate reliability.	The paper compares the generalized cut function and the Gompertz–Makeham software reliability model. The Gompertz model was found the best.	Gompertz function is the best model for data types that includes hours, number of failures, and cumulative failures.	N/A	Using the same data type and the new reliability model can give better results in terms of accuracy.
Yoshinobu Tamura, Shigeru Yamada (2016)	Choosing the best reliability model using deep learning.	What is the best reliability model that uses deep learning?	Using numerical data to perform a comparison of reliability models.	Selecting a reliability model using optimal release time is considered better than selecting using software cost.	The paper proposed a selection method for optimal software reliability model, compared different reliability evaluation models using deep learning.	N/A	Software reliability can be assessed better than before.
Kwang Yoon Song, In Hong Chang, Hoang Pham (2019)	Proposing a new reliability model by adding fault detection rate function on the non-homogeneous Poisson process.	What are the best models to measure reliability?	Studying several software reliability models using the non-homogeneous Poisson process.	The selected models and the proposed model were assessed based on 5 datasets, 2 datasets from a telecommunication system, and 3 datasets from an on-line communication system.	The proposed model has more efficiency in reliability prediction as it considers the uncertainty of operating environments.	The proposed models can have more validation with new datasets. New parameters estimation using Bayesian and big data can be conducted, considering the multi release point.	The proposed model gives more accuracy in reliability prediction in comparison with other models.
Roberto Pietrantuono, Antonia Bertolino, Guglielmo De Angelis, Breno Miranda, Stefano Russo (2019)	Introducing DevOpRET to continually perform reliability tests in the DevOps methodology.	How can developers estimate reliability?	DevOpRET is being applied to a case study using Discourse, which is a platform dedicated for discussions. Tests are performed on this platform to get reliability results.	Two profiles were used in the reliability estimation: uniform estimated profile and proportional estimated profile, the profiles were using a various number of tests on each DevOps cycle. Increasing the number of tests has significantly improved the accuracy of the prediction.	Using real-world applications and more usage data collection helped the DevOpRET to easily converge to the exact reliability prediction.	Using more advanced testing algorithms and machine learning can further improve the DevOps reliability assessment. DevOpsRET can also be more studied using actual application deployment, to assess the impact on other quality factors.	The DevOpsRET approach gives more accuracy to reliability assessment using more test cases.
Vahid Garousi, Michael Felderer, Feyza Nur Kılıçaslan (2019)	Summarizing the discipline of software testing to improve software testability.	How testing can improve reliability?	Compiling 208 papers to collect all pieces of information on software testing.	Many kinds of research addressed the improvement of software testability and extracted the factors that improve testability.	The conducted research is an index of the body of knowledge of testing, which helps to efficiently test software projects.	Testing trends are available in this research, which can be more investigated in future work.	Industrial collaborators found this paper useful when performing tests.

Otávio Augusto Lazzarini Lemos, Fábio Fagundes Silveira, Fabiano Cutigi Ferrari, Alessandro Garcia (2018)	Giving more knowledge in software testing to improve code reliability.	What are the steps that should be followed to improve the quality of code?	Conducting experiments with students and teachers to evaluate the knowledge of software testing among students, and teaching practices among teachers.	Learning principles and techniques of testing improved more than two times code implementation written by developers.	Having the test knowledge produced a reliable code by developers without even writing additional lines of code.	Conducting more researches with more students and professional developers, with analyzing how test training could impact the programming skills.	Teaching students (new developers) testing skills, tools, and techniques can impact positively code reliability.
Fernando Pinciroli (2016)	Improving compatible software quality attributes can give better results on software quality.	What are the compatible quality factors?	Using metrics to calculate solutions with different quality attributes.	Quality attributes like usability, security, and performance can have a negative impact on the system. Selecting different solutions and calculating their values using the provided metrics can give the best solution.	System identity which is the best combination of quality attributes can be achieved using the metrics. The expected value (level of quality) can be changing depending on the situation and the used quality attributes.	N/A	Choosing the adequate quality attributes can give the best software quality.
Mina Nabi, Maria Toeroe, Ferhat Khendek	Providing solutions for availability in cloud computing.	How can cloud providers improve availability?	Using 21 out of 100 relevant papers and conferences, with additional information from some of the main cloud providers.	Many cloud providers do not protect cloud services from application failure.	Cloud availability solutions are not the same, as every cloud provider has his own availability definition.	Future work can address cloud mechanisms (elasticity and inherent characteristic) in cloud upgrade.	N/A
Amro Al-Said Ahmad, Peter Andras (2019)	Using scalability metrics to measure cloud services scalability.	How scalability can be measured in cloud platforms?	Using the same/different services in the same/different cloud platforms to measures scalability.	Using two scenarios hosted in EC2 provided more improvement in cloud services scalability.	Different services were used in AWS and AZURE cloud platforms and assessed using metrics that address volume scaling as well as quality scaling, which gives more accurate scalability results.	Future work can address other cloud services and other cloud platforms.	Getting more accurate scalability results.
Serghei Mangul, Thiago Mosqueiro, Richard J. Abdill, Dat Duong, Keith Mitchell, Varuni Sarwal, Brian Hill, Jaqueline Brito, Russell JaredLittman, Benjamin Statz, Angela Ka-Mei Lam, Gargi Dayama, Laura Grieneisen, Lana S. Martin,	Providing solutions for increasing the stability and installability of software.	How to improve quality factors?	An empirical study of 36702 software resources varying from 2005 to 2017.	49% of 98 software products failed the easy installation, whereas 27.6% of these software failed to install due to some technical problems in their implementation.	Standard approaches must be applied to increase the installability and stability of the software.	N/A	Improving quality factors for software systems.

Jonathan Flint, Eleazar Eskin, Ran Blekhman (2019)							
Mohsen Ahmadvand, Alexander Pretschner, Florian Kelbert (2019)	Defending and protecting systems from integrity attacks.	What are the areas that must be addressed to protect systems?	Applying a taxonomy on literature papers, and evaluating it using 49 papers.	Coming up with correlated elements for protecting the integrity of the software systems.	Many techniques for protecting or mitigating integrity attacks like MATE attacks are available. Each one can have advantages and disadvantages.	Future research can address the resilience of these techniques. Performance can be benchmarked using more datasets on these various techniques.	Building more secure systems.
Gozde Basak Ozturk (2020)	Summarizing interoperability attribute trends and gaps.	N/A	Selecting and evaluating 2091 research papers from the Scopus database.	The 2091 papers were filtered by the English language to have 2052 article, then filtered by type, using only journal articles which let only 886 articles. The remained articles were reduced to 447 articles by including only interoperability related articles.	Problems remain when implementing BIM such as lack of communication and lack of data.	Future work can get more papers from other databases with different languages.	N/A
Milu Mary Philip, Nishank Singhal, Raagashree Ravi, Vijayakumar B. (2020)	Using architectural style in software development.	Will the proposed model improve software quality?	Using a document processing software as a case study.	Using the selected design, with the Java language and Ubuntu system, helped the selected software in the case study to have more modifiability with low coupling while increasing the number of parallel connections.	The used model in this paper will help in building flexible software that will easily meet users' requirements.	N/A	Software systems can be built more efficiently.
Huong Ha, Hongyu Zhang (2019)	Using a feedforward neural network in modeling configurable systems, with their performance prediction.	Can this neural model give more performance in prediction?	Using datasets to perform the proposed model experiments.	Conducting the experiment on different systems showed that the approach gives more accuracy in prediction without using a lot of input data.	The proposed deepperf method has outperformed various prediction models, using fewer data and different types of configurations.	The provided model design can be more improved by conducting researches on neural network universal property.	Accuracy in software performance prediction can be more accurate if using the deepperf model.
Hamza Ghandorh, Abdulfattah Noorwali, Ali Bou Nassif, Luiz Fernando Capretz, Roy Eagleson	Measuring software portability.	N/A	Using the systematic literature review using 49 research papers. The selected papers were chosen based on their quality. All data were collected from these 49 papers.	Most of the papers focused on measuring software portability in the development approach area.	Different software portability metrics were listed, using the collected papers that only focus on portability.	Future work can be conducted to find if neural networks can be used in software portability measurements.	N/A
Casidhe Hutchison, Milda Zizyte, Patrick E. Lanigan, David Guttendorf, Michael	Presenting robustness testing system for autonomous systems.	How to test robustness?	Using real systems and applying the selected testing approach.	Performing tests showed that the number of bugs increases when injecting single instances. Two of the dangerous bugs	Robustness tests can be useful, but at the same time can activate critical bugs that can	N/A	The given scalable approach helps to find serious bugs,

Wagner, Claire Le Goues, Philip Koopman (2018)				are floating point and NaN bugs.	damage the systems, like robots that have these types of software.		errors, and faults before happening at the runtime.
Jin Zhang, Jingyue Li (2020)	Performing testing and verification for a neural network in safety-critical cyber-physical systems.	N/A	Using six research libraries and filtering 950 articles.	After analyzing the selected papers, test and validation on neural network systems is gaining more focus by researchers.	The usage of neural networks in safety-critical systems is trending. Hence, test and validation for these systems is trending.	Future researches can be conducted to assess the neural network model, and compare it with other models, to get the best robustness model.	N/A
Nirali Honest (2019)	Emphasizing how testing can improve software quality using various types of tests and methods, throughout the software development lifecycle.	What are the good testing strategies for quality improvement?	Using existing literature to clarify levels of tests.	Software testing and its levels can improve the overall quality of the produced software.	To have good software quality among other competitors and sustain in the market, testing must be well performed.	N/A	Using tests to improve software quality.
Mohammad Daud Haiderzai, Mohammad Ismail Khattab (2019)	The impact of software testing on quality.	What are the quality factors for a quality software system? What are the suitable development methodologies in software testing?	Using a systematic literature review to find related literature to the research questions.	Different factors such as security, interoperability, portability, testability, usability, and correctness are considered important for improving quality. Additionally, testing methodologies such as functional, security testing, performance testing can also improve the quality of the software.	Software testing can improve software quality if well conducted, without deviating from customer's requirements.	N/A	Using different testing strategies can cover the functional and non-functional sides of software systems.

VII. CONCLUSION

Software quality and reliability got more attention from developers after the technological evolution. Customers' requirements were more sophisticated and more complicated. Therefore, many international organizations introduced standards for quality management systems and quality assurance, to maintain software quality and satisfy customers.

Software testing plays an important role in managing the quality and reliability and a software, as many testing strategies are available to cover and assess all software functionalities. Different development methodologies are available for developing projects. Yet, managing the time and cost of a project can still be challenging.

As a result, this affects testing strategies, because this phase can be costly in terms of time and budget. Future work can focus on how choosing methodologies can give more efficiency when performing different tests, to add more improvement upon software quality.

REFERENCES

[1] N. Pavlov, G. Spasov, A. Rahnev, and N. Kyurkchiev, "A new class of Gompertz–type software reliability models," Int. Electron. J. Pure Appl. Math., vol. 12, no. 1, pp. 43–57, 2018, doi: 10.12732/iejpam.v12i1.4.

[2] Y. Tamura and S. Yamada, "Software Reliability Model Selection Based on Deep Learning with Application to the Optimal Release Problem," J. Ind. Eng. Manag. Sci., vol. 2016, no. 1, pp. 43–58, 2016, doi: 10.13052/jiems2446-1822.2016.003.

[3] K. Y. Song, I. H. Chang, and H. Pham, "NHPP software reliability model with inflection factor of the fault detection rate considering the uncertainty of software operating environments and predictive analysis," *Symmetry (Basel).*, vol. 11, no. 4, 2019, doi: 10.3390/sym11040521.

[4] R. Pietrantuono, A. Bertolino, G. De Angelis, B. Miranda, and S. Russo, "Towards continuous software reliability testing in DevOPs," in *Proceedings - 2019 IEEE/ACM 14th International Workshop on Automation of Software Test, AST 2019*, 2019, pp. 21–27, doi: 10.1109/AST.2019.00009.

[5] V. Garousi, M. Felderer, and F. N. Kılıçaslan, "A survey on software testability," *Inf. Softw. Technol.*, vol. 108, pp. 35–64, 2019, doi: 10.1016/j.infsof.2018.12.003.

[6] O. A. Lazzarini Lemos, F. Fagundes Silveira, F. Cutigi Ferrari, and A. Garcia, "The impact of Software Testing education on code reliability: An empirical assessment," *J. Syst. Softw.*, vol. 137, no. Issre 2015, pp.

497–511, 2018, doi: 10.1016/j.jss.2017.02.042.

[7] F. Pinciroli, "Improving Software Applications Quality by Considering the Contribution Relationship among Quality Attributes," *Procedia Comput. Sci.*, vol. 83, no. Antifragile, pp. 970–975, 2016, doi: 10.1016/j.procs.2016.04.194.

[8] M. Nabi, M. Toeroe, and F. Khendek, "Availability in the cloud: State of the art," *J. Netw. Comput. Appl.*, vol. 60, pp. 54–67, 2016, doi: 10.1016/j.jnca.2015.11.014.

[9] A. Al-Said Ahmad and P. Andras, "Scalability analysis comparisons of cloud-based software services," *J. Cloud Comput.*, vol. 8, no. 1, 2019, doi: 10.1186/s13677-019-0134-y.

[10] S. Mangul *et al.*, "Challenges and recommendations to improve the installability and archival stability of omics computational tools," *PLoS Biol.*, vol. 17, no. 6, pp. 1–16, 2019, doi: 10.1371/journal.pbio.3000333.

[11] M. Ahmadvand, A. Pretschner, and F. Kelbert, "A Taxonomy of Software Integrity Protection Techniques," *Adv. Comput.*, vol. 112, pp. 413–486, 2019, doi: 10.1016/bs.adcom.2017.12.007.

[12] G. B. Ozturk, "Interoperability in building information modeling for AECO/FM industry," *Autom. Constr.*, vol. 113, no. January, p. 103122, 2020, doi: 10.1016/j.autcon.2020.103122.

[13] M. Philip, N. Singhal, R. Ravi, and V. B., "A Quantitative Approach to Analyze Modifiability in Software Architectural Design of Agile Application Systems," *Inf. Technol. Control*, vol. 49, no. 2, pp. 249–259, 2020, doi: 10.5755/j01.itc.49.2.22893.

[14] H. Ha and H. Zhang, "DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network," *Proc. - Int. Conf. Softw. Eng.*, vol. 2019-May, pp. 1095–1106, 2019, doi: 10.1109/ICSE.2019.00113.

[15] H. Ghandorh, A. Noorwali, A. B. Nassif, L. F. Capretz, and R. Eagleson, "A Systematic Literature Review for Software Portability Measurement," pp. 152–157, 2020, doi: 10.1145/3384544.3384569.

[16] C. Hutchison *et al.*, "Robustness testing of autonomy software," *Proc. - Int. Conf. Softw. Eng.*, pp. 276–285, 2018, doi: 10.1145/3183519.3183534.

[17] J. Zhang and J. Li, "Testing and verification of neural-network-based safety-critical control software: A systematic literature review," *Inf. Softw. Technol.*, vol. 123, no. April 2019, 2020, doi: 10.1016/j.infsof.2020.106296.

[18] D. Galin, *Software Quality Assurance: Concepts and Practice*. Wiley-IEEE Press, 2018.

[19] N. Honest, "Role of Testing in Software Development Life Cycle," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 5, pp. 886–889, May 2019, doi: 10.26438/ijcse/v7i5.886889.

[20] M. D. Haiderzai and M. I. Khattab, "How software testing impact the quality of software systems ?," *Int. J. Eng. Comput. Sci.*, vol. 1, no. 2, pp. 5–9, 2019.