How Good is My Software? A Simple Approach for Software Rating based on System Testing Results: A Case Study via Test-in-the Cloud Platform

Muhammad Dhiauddin Mohamed Suffian
Business Solution & Services
MIMOS Berhad
Kuala Lumpur, Malaysia
dhiauddin.suffian@mimos.my

Loo Fook Ann
Business Solution & Services
MIMOS Berhad
Kuala Lumpur, Malaysia
fa.loo@mimos.my

Farah Farhana Mohd Nazri
Business Solution & Services
MIMOS Berhad
Kuala Lumpur, Malaysia
farhana.nazri@mimos.my

Fairul Rizal Fahrurazi
Business Solution & Services
MIMOS Berhad
Kuala Lumpur, Malaysia
fairul.fahrurazi@mimos.my

Soo Shi Tzuaan
Business Solution & Services
MIMOS Berhad
Kuala Lumpur, Malaysia
soo.st@mimos.my

Mohamed Redzuan Abdullah
Business Solution & Services
MIMOS Berhad
Kuala Lumpur, Malaysia
redzuan.abd@mimos.my

***Abstract*—Knowing how good your software is prior to release could indicate whether the software can really work in the actual environment. Executing the system test allows for this to take place. By applying simple analytics approach to the system test cases results of PASS or FAIL for each test strategy imposed, points can be assigned per test case for every test iteration. Then, scores can be calculated. This shall be done to every test tool used per test strategy. The average of accumulated scores from all test strategies is mapped to the pre-defined rating table to establish software product rating. The proposed approach can be used for complete system testing or ongoing system testing, which serves as early indicator for the software's expected behavior in the actual environment.**

Keywords—good software, software product rating, cloud, test, system test cases, analytics approach

I. INTRODUCTION

It is imperative to gauge the quality of software before releasing it to users. Software is said to be "good" once it could serve its purpose in the specified situation [1]. Various mechanisms have been proposed to evaluate the quality of software produced such as by using quality model [2] and rating based on system metrics [3]. The assessment of the quality of software may cover the whole life cycle by taking into account the audit activities at micro level from requirements to system testing [4]. ISO/IEC 9126 standard has also been succeeded by ISO/IEC 25010 standard to accommodate more comprehensive characteristics and sub-characteristics of software quality: Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability Security, Maintainability and Portability (5).

The most significant avenue to conclude the quality of the software upon release is after the completion of system testing. System testing emphasizes on functional and non-functional aspects of software under test in the environment that is similar or closer to actual operating environment [6]. By executing system test cases that exercise functionalities, usability, performance, security and other quality aspects, it could indicate the level of trustworthiness exhibited by the software under test [7].

In practice, system testing can only start once the entry criteria have been fulfilled: Complete or partially testable code is available, requirements are defined and approved, availability of sufficient and desired test data, test cases are developed and ready as well as test environment has been set-up and all other necessary resources such as tools and devices are available [8]. Test cases designed for system testing are derived from the software requirement specification (SRS), comprising set of functional test cases and set of non-functional test cases. All these are aimed to make sure overall aspects of the software under test conform to every requirement in the SRS. By having PASS result for all test cases at the end of system testing regardless of how many test iterations imposed, it serves as indicator that system testing has already completed and the software is fit for use. However, this ideal scenario will not take place in actual situation since software that passes the system testing can be released with known issues, caveats and uncertainties. This demonstrates that there is no guarantee software can operate correctly despite the rigor and thorough system testing conducted [8]. So, can all these system test cases give any value to the goodness or health the software delivered to users or at least instill certain level of confidence to the customers?

This concern serves as the premise to explore how far the result of system test cases could be exploited and analyzed in determining the quality of the software produced. Since each test case bring the value in a form of PASS or FAIL, it could be transformed into a quantitative measurement that could indicate how the software would behave in the actual environment.

II. RELATED WORKS

Software quality can be associated with software health and trustworthiness by being able to handle faults, platform and privacy in determining the trust [9]. Users' trust in using the software can also be gathered via the software usage functionalities, usage context and documentation [10]. Within the open source community, as long as the software able to fulfill its requirements, compliance, reliability and interoperability criteria, the particular open source software can be rated as trusted software [11]. This was supported by the application of Openness Factor Rating [12]. Furthermore, users will also trust any software when they can use it in effective, efficient and satisfactory manner [13], which is aligned to the aspects evaluated in this paper namely functional, performance, security and usability. However, [14] proposed the trustworthiness assessment based component, architecture and system level comprising technical and non-technical assessment.

978-1-7281-3500-7/19/$31.00 ©2019 IEEE

There was also a proposal for measuring software trustworthiness by using finite state machine and requirements in a form of scenario structure, state chart structure, semantic and a behaviouristic model [7]. In assessing the trustworthiness assessment across the lifecycle and components, a set of rules was established that rely on software environment and requirements [15].

An index was given to indicate health status of software based on following parameters: environment, requirement, design, coding and testing, which was referred as Technical Health Indicator (THI)[4]. On the different view, [16] used Monotonicity, Acceleration, Sensitivity and Substitutivity metrics to model the software health and trustworthiness, which was later expanded further into critical and non-critical attributes [17]. Relatively, the use of specific rating has been used for quite some time by automotive sector. This was done via the use of car safety rating to indicate how safe a particular car prior to the purchase decision [18][19]. However, every aspect assessed to come out with complete vehicle rating has its own formula toward reaching the final scoring.

The rationale for understanding the works on rating and how they are calculated is mainly to measure how far software is ready for release [20][21]. Having the rating based on certain type of test could support in determining how good or bad the software is at a particular point of time, especially once the test is completed. From the rating obtained from test activities, it may complement the selection of release readiness metric [20] and support the release readiness index [21]. In the long run, it may contribute to the calculation of quality index [22]. Thus, this paper tries to emulate those rating provided in the prior works towards establishing a quantitative approach for software product rating based on the results obtained from test cases of system testing. The results of test cases execution either PASS or FAIL or not yet executed are transformed into quantitative values towards producing the score and ratings.

This paper is organized into following sections: Section I on the Introduction of the paper, Section II on the related works on rating, Section III on the approach introduced for calculating the score and rating, Section IV on the rating implementation via test-in-the-cloud platform and finally Section V on the conclusion and recommendation.

III. Simple Analytics Approach for Software Product Rating

The proposed approach introduced in this paper deals with complete software rating and real-time software rating. Complete software rating refers to rating given once the system testing fully completed, in which all test cases for functional and non-functional successfully obtained PASS result. As for real-time software rating, it refers to rating given while software still undergoes system test execution, in which some test cases might have completed and some are not.

A. The Approach

The approach relies on the following level of steps as depicted in Figure 1:

- Baseline test strategies – specify the test types or strategies imposed to the software under test with prioritization weightage for each
- Number of testing tools per test strategy – Specify the number of test tools used per test strategy, for example the use of Jmeter and YSlow for conducting performance test
- Completeness of system test execution – Specify the status of system test completion either totally completed or partially completed, which is real-time status while system test is in progress
- Test case iterations – Finalize the number of test iterations or cycles conducted per test strategy during system test
- Test case priority – Assign the priority to each test case per test strategy
- Test case result for each iteration – Identify the result of each test case by every iteration for each test strategy, either PASS or FAIL

Based on the values or figures obtained from the steps above, by using preferred calculation method, the final complete score or partial score is calculated. This score is then mapped to pre-defined rating table to get the final software product rating.

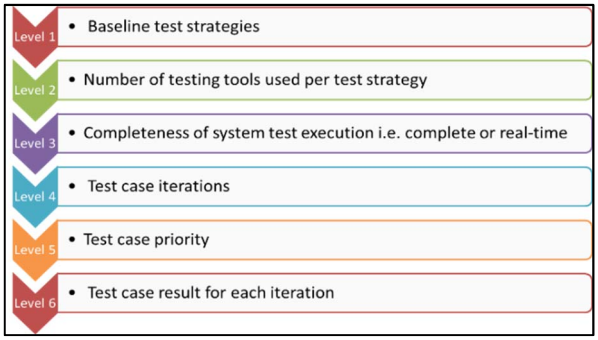

Fig. 1. Simple analytics approach for software product rating

Test strategies in the context of this paper refers to types of test conducted, which include the following tests but not limited to functional testing and common non-functional testing: performance testing, security testing and usability testing. Other potential non-functional testing that could be considered could be compatibility testing and portability testing.

For complete software rating, all test cases executed for every test strategy imposed to the software under test must be completed with the final iteration result is PASS. The minimum and maximum test iterations must be finalized, in which the proposed minimum iteration is two (2) and maximum is four (4) since in real practice, independent testing team that executes system testing does not have the luxury of time to test the software due to schedule crunch. Most of the time, the team can test up to three iterations. Although in some occasions test iterations may go beyond four iterations, but the increase in the number of iterations may trigger questions by the management on why particular test cases need to go beyond the common iteration limit.Then, for each test case per test iteration, a fixed base value is assigned with the total weightage should be equal to 100. This is applicable to functional, performance and security testing as usability testing might use different approach to calculate the score. Thus, the calculation uses the following formula:

$$Points = Weightage\ (W) \times Base\ point\ (B) \quad (1)$$

Across all iteration, the base point (B) remains the same while for the weightage, it determines and differentiates the point given for each iteration. The calculation of total points is the

same for all test cases subject to fulfilling the minimum and maximum iterations suggested earlier. Assignment of weightage across iteration may use either of these two options: For Option 1, it is assumed that 1st iteration carries highest weightage as compared to subsequent iterations due to the importance of passing the test case for the first time it is executed. This is to demonstrate the stability of the feature being tested. Thus, the weightage for 1st iteration should be a fix value for all test cases. As for Option 2, it assumes last iteration (2nd, 3rd or 4th iteration) carries the highest weightage as compared to preceding iterations. This is due to the need of passing the test case during final iteration as part of testing exit criteria. Thus, the weightage for last iteration should be a fix value for all test cases. Any of these two options shall establish a point system. The differences between the point system applying Option 1 and Option 2 are illustrated below in Table I and Table II:

TABLE I. POINT SYSTEM APPLYING OPTION 1

Type of Point System	Example
2-points System	6B [I] + 4B [II] = 10B
3-points System	6B [I] + 3B [II] + 1B [III] = 10B
4-points System	6B [I] + 2B [II] + 1B [III] + 1B [IV] = 10B

TABLE II. POINT SYSTEM APPLYING OPTION 2

Type of Point System	Example
2-points System	4B [I] + 6B [II] = 10B
3-points System	1B [I] + 3B [II] + 6B [III] = 10B
4-points System	1B [I] + 1B [II] + 2B [III] + 6B [IV] = 10B

The result of the calculation is defined as score. The score is regulated against the assigned priority for each test case. Total score for the test suite is calculated by dividing it over expected ideal score and multiply by 100. For each test strategy, total score is then regulated against the weightage of each test strategy that has been agreed prior to the start of system testing. Example of weightage per test strategy for a complete system testing is 50% for functional test, 20% for security test, 15% for performance test and 15% for usability test (total of 100%). The result is considered as final score. Same approach is applied for partial (real-time) rating, in which the score is based on number of iterations or test strategies completed at particular point of time. In this case, the result is considered as real-time score. The final score either complete score or real-time score is mapped against the rating table, which has been pre-defined earlier. The way rating table is based on the organizational context. Example of software rating table can be depicted as below in Table III:

TABLE III. EXAMPLE OF SOFTWARE RATING TABLE

Score	Rating
80% - 100%	5 Star
60% - 79%	4 Star
40% - 59%	3 Star
20% - 39%	2 Star
0% - 19%	1 Star

B. Examples of Score Calculation

Three hypothetical scenarios are created: Project I, Project II and Project III. Project I has fully completed system testing, Project II is undergoing system test execution with one of the test strategies completed while Project III is still undergoing system test execution with none of the test strategies completed. Project I has fully completed all test strategies comprise of functional, performance, security and usability testing. By using only functional testing result having 20 test cases with 3 iterations completed and Option 1 as the weightage option, following formula is used:

- Base point = 10
- Points calculation = 6B + 3B + 1B (Iteration is 3)

The points obtained for overall test suite of functional test cases are shown below in Figure 2:

TC	IT1	IT2	IT3
TC1	Passed	Passed	Passed
TC2	Passed	Passed	Passed
TC3	Passed	Passed	Passed
TC4	Passed	Passed	Passed
TC5	Passed	Passed	Passed
TC6	Passed	Passed	Passed
TC7	Failed	Failed	Passed
TC8	Passed	Passed	Passed
TC9	Passed	Passed	Passed
TC10	Passed	Passed	Passed
TC11	Passed	Passed	Passed
TC12	Passed	Passed	Passed
TC13	Passed	Passed	Passed
TC14	Failed	Passed	Passed
TC15	Passed	Passed	Passed
TC16	Passed	Failed	Passed
TC17	Passed	Passed	Passed
TC18	Failed	Passed	Passed
TC19	Passed	Failed	Passed
TC20	Passed	Passed	Passed

TC	IT1	IT2	IT3	TC PRIORITY	TOTAL POINTS	ACTUAL SCORE	IDEAL SCORE
TC1	60	30	10	3	100	300	300
TC2	60	30	10	3	100	300	300
TC3	60	30	10	1	100	100	100
TC4	60	30	10	2	100	200	200
TC5	60	30	10	3	100	300	300
TC6	60	30	10	1	100	100	100
TC7	0	0	10	1	10	10	100
TC8	60	30	10	3	100	300	300
TC9	60	30	10	2	100	200	200
TC10	60	30	10	2	100	200	200
TC11	60	30	10	1	100	100	100
TC12	60	30	10	1	100	100	100
TC13	60	30	10	3	100	300	300
TC14	0	30	10	3	40	120	300
TC15	60	30	10	3	100	300	300
TC16	60	0	10	3	70	210	300
TC17	60	30	10	1	100	100	100
TC18	0	30	10	2	40	80	200
TC19	60	0	10	2	70	140	200
TC20	60	30	10	3	100	300	300
TOTAL SCORE						3760	4300

Fig. 2. Calculation result of functional testing score applying Option 1

Thus, functional test score is calculated as below:

Functional Test Score
= (Total Score/Ideal Score) x 100%
= (3760/4300) x 100%
= **87.4%**

However, if the calculation is replaced with Option 2 weightage, following formula is applied:

- Base point = 10
- Points calculation = 1B + 3B + 6B (since iteration is 3)

The points obtained for overall test suite of functional test cases are shown below in Figure 3:

TC	IT1	IT2	IT3
TC1	Passed	Passed	Passed
TC2	Passed	Passed	Passed
TC3	Passed	Passed	Passed
TC4	Passed	Passed	Passed
TC5	Passed	Passed	Passed
TC6	Passed	Passed	Passed
TC7	Failed	Failed	Passed
TC8	Passed	Passed	Passed
TC9	Passed	Passed	Passed
TC10	Passed	Passed	Passed
TC11	Passed	Passed	Passed
TC12	Passed	Passed	Passed
TC13	Passed	Passed	Passed
TC14	Failed	Passed	Passed
TC15	Passed	Passed	Passed
TC16	Passed	Failed	Passed
TC17	Passed	Passed	Passed
TC18	Failed	Passed	Passed
TC19	Passed	Failed	Passed
TC20	Passed	Passed	Passed

TC	IT1	IT2	IT3	TC PRIORITY	TOTAL POINTS	ACTUAL SCORE	IDEAL SCORE
TC1	10	30	60	3	100	300	300
TC2	10	30	60	3	100	300	300
TC3	10	30	60	1	100	100	100
TC4	10	30	60	2	100	200	200
TC5	10	30	60	3	100	300	300
TC6	10	30	60	1	100	100	100
TC7	0	0	60	1	60	60	100
TC8	10	30	60	3	100	300	300
TC9	10	30	60	2	100	200	200
TC10	10	30	60	2	100	200	200
TC11	10	30	60	1	100	100	100
TC12	10	30	60	1	100	100	100
TC13	10	30	60	3	100	300	300
TC14	0	30	60	3	90	270	300
TC15	10	30	60	3	100	300	300
TC16	10	0	60	3	70	210	300
TC17	10	30	60	1	100	100	100
TC18	0	30	60	2	90	180	200
TC19	10	0	60	2	70	140	200
TC20	10	30	60	3	100	100	300
TOTAL SCORE						3860	4300

Fig. 3. Calculation result of functional testing score applying Option 2

Thus, functional test score is calculated as below:

Functional Test Score
= (Total Score/Ideal Score) x 100%
= (1880/2000) x 100%
= **89.7%**

On the other hand, Project II involved one completed test strategy while the remaining test strategies are still in progress. In this situation, functional test has completed and

performance testing is still ongoing. This is where real-time score is calculated. Assuming the completed functional testing result has the same score as Figure 2, the result of the ongoing performance test execution is shown below in Figure 4:

TC	IT1	IT2	IT3	TC PRIORITY	TOTAL POINTS	ACTUAL SCORE	IDEAL SCORE
TC1	10	-	-	3	*10*	*30*	300
TC2	10	-	-	3	*10*	*30*	300
TC3	10	-	-	1	*10*	*10*	100
TC4	10	-	-	2	*10*	*20*	200
TC5	10	-	-	3	*10*	*30*	300
TOTAL SCORE						120	1200

Fig. 4. Ongoing performance test result

Thus, the real-time score is calculated as below:

Performance Test Score (Real-Time)
= (Total Score/Ideal Score) x 100%
= (120/1200) x 100%
= **10%**

As for Project III, since the situation involved none of completed test strategies, the calculation of score shall use the same calculation for real-time score.

The score is calculated for every test suite per test tool in every test strategy imposed. Thus, final score for each test strategy is the average of the total scores obtained from the all tools used for that particular test strategy. For example, if two tools are used for functional test, then the final score for functional test is the average of score for Tool 1 and score for Tool 2.

C. *Application in Hypothetical Scenarios*

In actual application, after software is completely built, it is sent to system testing for thorough assessment. This is where the software product rating is applied. If the stakeholders or project team requires rating while system testing is in progress, real-time score will be calculated and real-time rating will be given, On the other hand, complete score is calculated and complete rating is provided when project team or stakeholders requires rating after system testing completed. Figure 5 depicts the application of software product testing at system testing phase before software is deployed and delivered to users.

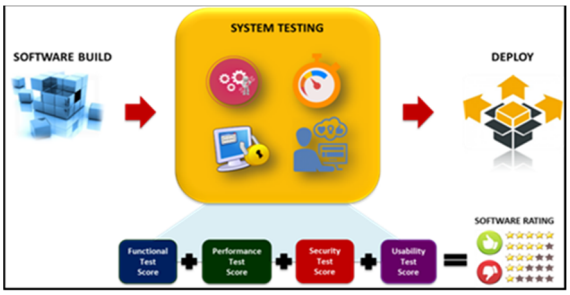

Fig. 5. Application of software product rating at system testing

Assumes that the following were the weightage assigned for each test strategy across all three projects, which in total makes up to 100%:

- Functional Test Weightage = 50% or 0.5
- Performance Test Weightage = 20% or 0.2
- Security Test Score Weightage = 15% or 0.15
- Usability Test Weightage = 15% or 0.15

Table IV were the scores obtained from each test strategy and complete score for Project I, Project II and Project III:

TABLE IV. COMPLETE SCORE CALCULATION FOR PROJECT I

Test Strategy	Total Score	Weightage	Regulated Score	Score (Complete/ Real-Time)
Project I				
Functional	89.7	50% (0.5)	44.85%	**89.64% (Complete Score)**
Performance	95.5	20% (0.2)	19.1%	
Security	90.3	15% (0.15)	13.55%	
Usability	80.9	15% (0.15)	12.14%	
Project II				
Functional	89.7	50% (0.5)	44.85%	**46.85% (Real-Time Score)**
Performance	10.0	20% (0.2)	2.00%	
Security	-	15% (0.15)	-	
Usability	-	15% (0.15)	-	
Project III				
Functional	32.4	50% (0.5)	16.20%	**16.20% (Real-Time Score)**
Performance	-	20% (0.2)	-	
Security	-	15% (0.15)	-	
Usability	-	15% (0.15)	-	

In order to assign the software product rating, a rating definition table should be established, in which depends on how the organization will define it. Example of rating table that could be defined is shown below in Table V:

TABLE V. EXAMPLE OF SOFTWARE RATING TABLE

Score	Rating	Description
80% - 100%	5-STAR	Excellent
60% - 79%	4-STAR	Above Average
40% - 59%	3-STAR	Average
20% - 39%	2-STAR	Below Average
0% - 19%	1-STAR	Poor

By using rating table in Table V, the rating for each Project I, Project II and Project III respectively is shown below in Table VI:

TABLE VI. RATING FOR PROJECT I, PROJECT II AND PROJECT III

Project	Score	Type of Score	Rating
Project I	**89.64%**	Complete Score	5-STAR
Project II	**46.85%**	Real-time Score	3-STAR
Project III	**16.20%**	Real-time Score	1-STAR

The implementation of the approach using test-in-the-cloud platform discussed in subsequent section.

IV. IMPLEMENTATION OF SOFTWARE RATING VIA TEST-IN-THE-CLOUD PLATFORM

The test-in-the-cloud platform is the automated web testing in the cloud offering a platform for end-to-end test automation using cloud infrastructure. In this platform, several testing tools were integrated together to establish a centralized platform for system testing. These tools were grouped into common testing strategies as mentioned in this paper: functional, performance, security and usability testing. In alignment to ISO25010, the categorization of the tools is as follows in Table VII:

TABLE VII. TESTING TOOLS IN THE CLOUD PLATFORM AGAINST ISO25010 CHARACTERISTICS

Quality Aspect	Name of Test Tools
Functional Suitability - Functional Testing	
Web Application Function	Robot Framework RobotMTS

Quality Aspect	Name of Test Tools
Web Application Workflow	Robomachine
Web Application URL Link	Linkchecker
Web Application HTML Syntax	NU HTML Checker
Performance Efficiency - Performance Testing	
Backend Web Application Response Time	Jmeter
Frontend Web Application Performance Metrics (based on Yslow Rulesets)	Yslow
Frontend Web Application Performance Metrics (based on Pagespeed Rulesets)	Pagespeed
Frontend Web Application Performance Metrics (based on Sitespeed.io Rulesets)	Sitespeed.io
Backend Web Server Response Time	ApacheBench
Compatibility - Compatibility Testing	
None	None
Usability - Usability Testing	
Frontend Web Application UI Layout and Responsiveness	Galen Framework
Frontend Web Application Accessibility	AChecker
Reliability - Reliability Testing	
None	None
Security - Security Testing	
Web Application Security Scanning	OWASP ZAP,
Network Security Scanning	Nmap
Application source code library dependencies security scanning	OWASP dependency-check
Maintainability - Maintainability Testing	
None	None
Portability - Portability Testing	
None	None

The reason for more than one tool used per test strategy is because different test tools exercise different quality aspects per characteristic in ISO25010. All the tools above were used to test a real project referred as TXC. TXC comprises seven (7) subsystems that are related with dashboard, data entry, workflow management, project management and business profiling. It is a web application system that was developed using PHP, JavaScript and HTML5 with MYSQL as the database. The system itself is hosted on Google Cloud platform. After completing the full system test on TXC system, the score for each test strategy obtained as below in Table VIII:

TABLE VIII. SCORE FOR EACH TEST PER TEST TOOL

Test Strategy	Tool	Score	Complete Score
Functional Testing	Robot Framework	90%	84.4% (84%)
	RoboMachine	63%	
	RobotMTS	90%	
	NUHTMLChecker	94%	
	LinkChecker	85%	
Performance Testing	Jmeter	100%	83.2% (83%)
	YSlow	76%	
	PageSpeed	68%	
	Sitespeed.io	72%	
	ApacheBench	100%	
Security Testing	OWASP Dependency-Check	57%	73.3% (73%)
	OWASP Zap	63%	
	NMap	100%	
Usability Testing	AChecker	94%	94.0% (94%)

Based on Table VIII, the complete score for each test strategy is the average scores of all tools used. Thus, the complete score for functional testing is 84.4% or 84%, performance testing is 83.2% or 83%, security testing is 73.3% or 73% and usability testing is 94%. The visualization of each result is presented in Figure 6. In totality, the final complete score after completing the system test execution for TXC system is represented in Figure 10. Based on the result shown in Figure 7, the average final complete score for TXC system is 84% and this is equivalent to 5-STAR rating or verdict of Excellent. This means that the system will work correctly (functional suitability), fast (performance efficiency), secured (security) and user-friendly (usability) as per specification when it is deployed in actual environment.

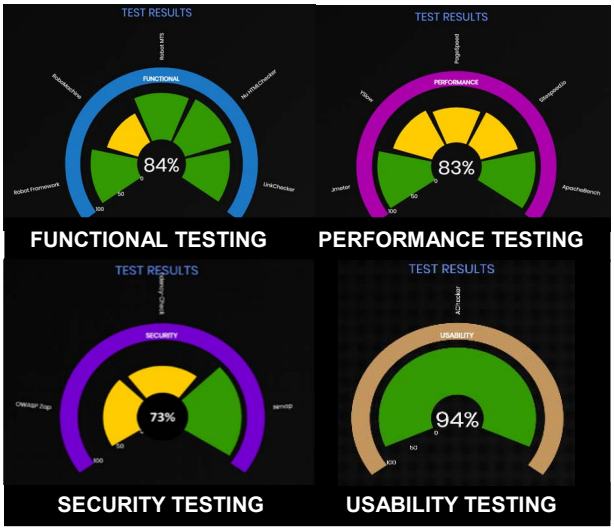

Fig. 6. Complete score of each test strategy

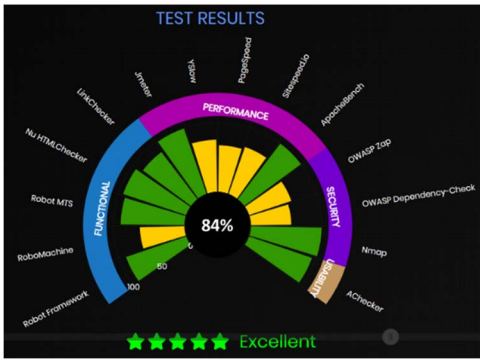

Fig. 7. Final complete score and rating of TXC system

V. DISCUSSION

Based on the simple approach introduced in this paper, there are several key points to discuss and justify further. Firstly, the proposed approach sets 4 iterations as the maximum cycles of testing and 3 as the minimum cycle, which establishes the basis for the pre-defined score. Specifically, the total of 3 test iterations serve as the minimum iterations for the approach due to common industry practice to execute a test 3 times. It may come into a situation whereby test cases may pass for 2 consecutive iterations, but 3rd iteration is still executed to verify that the test cases really pass. As for maximum iterations, the 4th iteration will only take place when there are failed test cases at 3rd iteration. Thus, this 4th iteration is added as part of the approach to make sure the bugs are fixed and test case is PASS during last iteration, Besides, the pre-defined score used in this paper is just for illustration purpose. Future works might find other alternatives to come out with the score. It should be clearly understood that the more iterations required by particular test

case for execution to get PASS result, the more unstable or unreliable the feature exercised by the test case is. This means the particular feature is not good enough to deliver its intended capabilities or services as required by user. Different organization may adopt different test strategies depending on the nature and type of software under test. Thus, there might be more sets of calculation that can be added. The more strategies imposed to the software, the more set of scores can be derived and calculated. Average is used to get the final score in order to have a simpler way to get the final rating of the software under test.

The proposed approach allows for the stakeholders to get the health status of the software under test at any particular point of time, regardless the system testing is still in progress or already completed. Although testing team typically use one test tool per test strategy, it does not mean the use of more than one tool is wrong. Therefore, the proposed approach takes into account how the scores and ratings can be manipulated to get further insights of the quality of the software under test since each test tool assess different aspects of the software. As presented in Table VII, for functional suitability via functional testing, Robot Framework focuses on functions while Linkchecker emphasizes on URL Link.

VI. Conclusion and Recommendation

The paper has successfully established the fundamental work on establishing a simple analytics approach for software product rating by using results from system testing execution, either as complete rating or real-time rating. Having the software rating in place as part of the development process shall serve as early health indicator for the software, thus giving early confidence to the users on how the intended software will work later. Furthermore, it provides new avenue for researchers in software testing on the benefits given by system test cases beyond just PASS and FAIL results. Practically, management and respective stakeholders would be able to decide on the software release based on the rating status of the software, either to give approval for release or to suspend the release.

As part of future work, the approach could be revised by having different way of calculation for different test strategy instead of just using average score as presented in this paper, for example applying statistical technique to calculate the score. The approach may also be adjusted to suit different types of software and domain such. Organizations that are interested to adopt this approach may consider to establish different set of scoring and rating table, such as scoring and rating for enterprise application is different from scoring and rating for e-commerce application. All these shall serve as preliminary work towards having a better and reliable approach for measuring the capability and confidence of software under test.

References

[1] D. Grundy, "What makes a 'good' program?", Proceedings of Pharmaceutical Users Software Exchange 2010 (PhUSE2010), Paper AD07, pp. 1-5, October 2010.

[2] M. Kläs, K. Lochmann and L. Heinemann, "Evaluating a quality model for software product assessments – a case study", 4. Workshop zur Software-Qualitätsmodellierung und -bewertung (SQMB 2011), Karlsruhe, Germany, pp. 14-24, 2011.

[3] S. Apel, F. Hertrampf and S. Späthe, "Towards a metrics-based software quality rating for a microservice architecture - case study for a measurement and processing infrastructure", Innovations for Community Services (I4CS 2019), Communications in Computer and Information Science, vol 1041, pp. 205-220, Springer, Cham.

[4] Aspire Systems. "Continuous software quality through technical health index", Aspire System's whitepaper, pp. 1-5, 2011. (Retrieved from http://www.aspiresys.com/WhitePapers/Whitepaper_Continuous_Software_Quality_through_Technical_Health_Index.pdf on 25 July 2019)

[5] IEEE, "ISO/IEC 25010:2011 Systems and software engineering -- Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and software quality models" (Retrieved from https://www.iso.org/standard/35733.html on 20 July 2019)

[6] R. Black, E.V. Veenendaal and D. Graham, "Foundations of Software Testing ISTQB Certification", Cengage Learning, 2008.

[7] M. Nami and W. Suryn, "Software trustworthiness: past, present and future", Proceedings of 2012 International Conference on Trustworthy Computing and Services (ICTCS 2012), pp.1-12, 2012

[8] ThinkSys, "Entry and exit criteria in software testing" (Retrieved from https://www.thinksys.com/qa-testing/entry-exit-criteria/ on 23 July 2019).

[9] Z. Xia and W. Pan, "Research on the trustworthiness of software", IEEE Proceedings of 2010 2nd International Conference on Information Science and Engineering (ICISE2010), pp. 1-4, 2010.

[10] M. Nami and W. Suryn, " Software trustworthiness: past, present and future", Trustworthy Computing and Services, Springer, vol. 320, pp. 1-12, 2013.

[11] V.D. Bianco, L. Lavazza, S. Morasca and D. Taibi, "A survey on open source software trustworthiness", IEEE Software, vol. 28 (5), pp. 67-75, 2011.

[12] W. Bein and C. Jeffery, "Towards an openness rating system for open source software", IEEE Proceedings of 2010 43rd Hawaii International Conference on System Sciences (HICSS), pp. 1-8, 2010.

[13] X. Zhao, Y. Liu, Y. Shi, and L. Zhang, "An empirical study of the influence of software trustworthy attributes to software trustworthiness", IEEE Proceedings of 2010 2nd International Conference on Software Engineering and Data Mining (SEDM), pp. 603-606, 2010.

[14] A. Immonen, and M. Palviainen, "Trustworthiness evaluation and testing of open source components", IEEE Proceedings of 2007 Seventh International Conference on Quality Software (QSIC '07), pp. 316-321, 2007.

[15] T. Bao, S. Liu, and L. Han, "Research on an analysis method for software trustworthiness based on rules", IEEE Proceedings of 2010 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 43-47, 2010.

[16] H. Tao and Y. Chen, "A metric model for trustworthiness of softwares", Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '09), vol. 3, pp. 69-72, 2009.

[17] H. Tao and Y. Chen, "A new metric model for trustworthiness of softwares", IEEE Proceedings of 2010 International Conference on Information Science and Applications (ICISA), pp. 1-8, 2010.

[18] ASEAN NCAP {Retrieved from http://www.aseancap.org/our-test/the-ratings-explained/ accessed on 28 July 2019).

[19] EURO NCAP (Retrieved from http://www.euroncap.com/en/about-euro-ncap/how-to-read-the-stars/ accessed on 28 July 2019).

[20] P. R. Satapathy, "Evaluation of software release readiness metric [0,1] across the software development life cycle", 2008 (Retrieved from https://www.researchgate.net/publication/229001077_Evaluation_of_Software_Release_Readiness_Metric_0_1_across_the_software_development_life_cycle).

[21] N. Koprowski, M.F. Harun and H. Lichter, "Release readiness measurement: a comparison of best practices", IEEE Proceedings of the 2014 8th Malaysian Software Engineering Conference (MySEC), pp. 166-171, 2014.

[22] A. Atanasov and L. Qin, "Software testing process evaluation with quality index", Proceedings of 2011 3rd World Congress in Applied Computing,Computer Science, and Computer Engineering, pp. 185-192.