Rating of software trustworthiness via scoring of system testing results

Muhammad Dhiauddin Mohamed Suffian*,
Fairul Rizal Fahrurazi, Loo Fook Ann,
Nur Farahin Aman and Norzamzarini Bajuri

Business Solutions and Services,
MIMOS Technology Solutions Sdn. Bhd.,
Kuala Lumpur, Malaysia
Email: dhiauddin.suffian@mimos.my
Email: fairul.fahrurazi@mimos.my
Email: fa.loo@mimos.my
Email: farahin.aman@mimos.my
Email: norzam@mimos.my
*Corresponding author

Abstract: It is important to have a form of software trustworthiness rating serving as early health indicator on the capability of the software before it is released to end-users and customers. Although various software certification schemes are available in the market, there is limited information on how far results from system testing are exploited and used to rate the trustworthiness of the software under test. Thus, the proposed software trustworthiness rating strategy uses the scoring upon the completion of system testing execution. The strategy covers rating of software that has undergone system testing completely or partially. It is based on multiple levels calculation toward coming out with final rating: test strategies imposed, completeness of system test execution, test iterations, test case priority and test case result for each iteration. As a result, the multilevel scores calculation successfully derives meaningful trustworthiness rating for the software under test, whether it is a complete rating or partial rating.

Keywords: software trustworthiness; rating; system test; software health indicator; software testing.

Reference to this paper should be made as follows: Suffian, M.D.M., Fahrurazi, F.R., Ann, L.F., Aman, N.F. and Bajuri, N. (2018) 'Rating of software trustworthiness via scoring of system testing results', *Int. J. Digital Enterprise Technology*, Vol. 1, Nos. 1/2, pp.121–134.

Biographical notes: Muhammad Dhiauddin Mohamed Suffian currently serves as the Lead Business Analyst under Development team in the department. He has various exposures holding multiple roles in software development, software testing, e-learning and academia. He has also published and presented articles and papers in various local and international conferences/journals.

Fairul Rizal Fahrurazi currently serves as the Head of Development team overseeing business analysis, software development and software architecture. Previously, he was responsible for test research and test continuous improvement in the department. He has presented paper and served as keynote speaker in various international conferences.

Copyright © 2018 Inderscience Enterprises Ltd.

Loo Fook Ann is currently responsible for the development and implementation of automated test in the cloud platform. He has served as the Test Lead for network-related product and actively looking for continuous improvement of test automation initiatives.

Nur Farahin Aman had served as Business Analyst and Technical Lead for the department. She was responsible for managing the development of corporate website and supporting the business analysis activities for the department. Her previous key role was overseeing and managing the operation of ALM platform for the department.

Norzamzarini Bajuri has served various roles within the department focusing on system testing. He was the Test Engineer for various internal projects and currently assumes the Test Lead role for few external testing projects, as part of the Independent Verification & Validation (IVV) team adhering to ISO17025 standard.

This paper is a revised and expanded version of a paper entitled 'Software capability rating using system testing score' presented at 2016 International Conference on Open Systems (ICOS 2016), Langkawi, Malaysia, 10–12 October 2016.

1 Introduction

Software should work correctly and smoothly in its specified operating environment. Therefore, it is important to ensure that the requirements for the software operation are matched and compatible with specifications of the hardware that will host the software. Apart from the ensuring the proper functioning of software in the environment, other future events to take place should also be considered such as installation of new software modules, updating and/or customising software modules.

Software testing activities are carried out to achieve the expectations mentioned earlier, particularly system testing. Although unit testing and integration testing are conducted before execution of system testing, the former could not qualify the software being developed is fit for use. System testing assumes this responsibility by assessing the quality of the software under test both from technical and non-technical perspectives: developers and users.

System testing helps to reduce the risk of failure when software operates in its intended environment through the execution of functional testing and non-functional testing. Functional testing evaluates the features offered by the software while non-functional testing exercises quality aspects of the software in delivering the features comprising areas of but not limited to usability, performance, security and compatibility. This is aligned with the software trusworthiness by capturing functional and quality requirements (Nami and Suryn, 2012c).

Typically, pre-conditions for execution of system testing involve the readiness of system test cases, test environment and completion of software built, which has passed both unit and integration tests. The planned test cases are associated with requirements stated in software requirement specification (SRS) categorised into functional test cases and non-functional test cases. Non-functional test cases can be decomposed further into

further groupings, such as performance test cases, security test cases and usability test cases. All these cases are executed during system testing phase.

System testing phase is considered complete when all functional and non-functional test cases have been executed with PASS results, which could represent the trusworthiness and the goodness of the software in delivering the features as expected by the customer and users. Customers and users may perceive that the software that has passed system testing is stable and free of defects.

However, this may not be the case for every situation. In some conditions, not all test cases shall have PASS results and software may be released with few known caveats, subject to fulfilment of minimum of test exit criteria. This also means the software does not demonstrate a full trusted capable features since some features did not pass the test. On the other hand, there is also situation where software under test managed to pass all test cases but it still failed to completely work in the real operating environment. Thus, it could be deduced that the results of system testing does not guarantee that software can works correctly as expected when it is delivered and operated in actual environment (Nami and Suryn, 2012b).

This scenario serves as motivation to explore how the results of test cases for both functional and non-functional testing could be exploited to form an early health indicator once execution of system testing is completed. Few understandings and concerns have led to this work. Firstly, the way development team build the software may affect its health and trusworthiness. Ignorance of the importance unit testing before releasing to testing team can be one of the reasons. Secondly, the test cases designed for respective feature may be able demonstrate how good and stable a particular feature works. Thus, it gives an opportunity to explore how the result of each test case can be translated into health indicator for software either when the testing is in progress or once the system testing is completed.

There have been many attempts to evalue the quality level of software once it is released. This was done via rating or certification form. Perception on user experience was used to establish the rating for mobile application (Pradana and Ferdiana, 2014). The areas measured are novelty, efficiency, perspicuity, stimulation and dependability. Simplified model was introduced to rate the software health but specifically for real-time systems, which is done during the run-time condition (Dubey et al., 2011). As opposed to assessment via rating, software quality level is also assessed using certification mechanism. Generally, a particular software is certified during a quality gate called release readiness, which enables the software to be certified as holding a send, hold or partial release status (Port and Wilf, 2013). Criteria used for certifying the software are completeness of requirement implementation and documentations, availability of workaround, safety and how far the opne issues have been addressed. In automative industry, certification is given when the software achieves personnel qualification, tools qualification, process qualification, product certification, certificate of analysis as well as functional safety certification (Areias et al., 2014).

Since there is limited information on the work for evaluating software trusworthiness based on test results, the work presented in this paper establishes a mechanism of exploiting and manipulating the result of test cases executed during system testing. This shall demonstrate how good the software is from technical's point of view as well as user's point of view, quantitatively. Scores are given to each test case per test iteration for each test type or test strategy imposed. Cumulative scores from all test strategies are

regulated towards getting final rating of software trusworthiness, thus establish a kind of early indicator of the software health upon completion of system testing before releasing to the customers and end-users.

2 Prior works

Software quality is related to software health and trustworthiness. The ability of the software to handle faults, platform and privacy well can determine the trust (Xia and Pan, 2010). Similarly, usage functionalities, usage context and documentation can develop the trust of users in using the software (Nami and Suryn, 2013). Open source community observes the open source software as trusted software when it meets requirements, compliance, reliability and interoperability criteria (Bianco et al., 2011). Software is also trusted if user can use and operate it in effective, efficient and satisfactory manner (Zhao et al., 2010). Comparatively, the work presented in this paper emphasises on software trustworthiness rating based on the test strategies conducted to software under test, which covers the most popular ones: functional test, performance test, security test, usability test and compatibility test.

Nami and Suryn (2012a) proposed the use of finite state machine and requirements to assess the software trustworthiness. This comprises of scenario structure, statechart structure, semantic and a behaviouristic model. Furthermore, Bao et al. (2010) incorporated rules-based trustworthiness assessment across the life cycle processes and component level, which depends on environment and requirements of the software. From the context of open source trustworthiness, openness factor rating was applied (Bein and Jeffery, 2010). In contrary, Immonen and Palviainen (2007) determine the open source trustworthiness according component, architecture and system level comprises technical and non-technical assessment

In relation to health, Aspire Systems (2011) recommended the use of scientific indicators based on parameters from environment, requirement, design, coding and testing. It was called technical health index (THI). On the other hand, metrics of monotonicity, acceleration, sensitivity and substitutivity was adopted by Tao and Chen (2009) to model the software health and trustworthiness. Tao and Chen (2010) improvise the metrics-based model expanding them to critical and non-critical attributes. Having the health indicator for software could be translated into the confidence level of the customers and end-users in using the software in actual environment. This is relatively similar to the confidence level for purchasing new car based on the car safety rating adopted in many countries (ASEAN NCAP, 2016; EURO NCAP, 2016).

Despite the insufficiency of software testing to demonstrate software trustworthiness (Nami and Suryn, 2012b), this paper is aimed establish a mechanism that exercise the values behind the test cases executed during system testing, which cover both functional and non-functional requirements under the actual or similar operating environment once release to users.

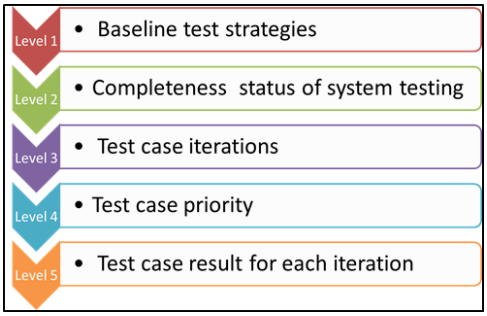
3 Methodology

The proposed methodology of using system testing results for software trustworthiness presented in this paper covers both complete rating and real-time rating. Complete rating

means software under test has completed the execution of system testing while real-time rating deals with rating while execution of system testing is still in progress. This means stakeholders of the test project can have the options of obtaining the trustworthiness rating on demand or wait until system testing ends completely.

The proposed methodology relies on the following levels of sequence: agreed test strategies, completeness status of system testing, test iterations, test cases priority and test case result. This is summarised in Figure 1.

Figure 1 Software trustworthiness methodology using system testing results (see online version for colours)



The details on how the methodology works towards producing the trustworthiness rating are explained below:

1 Identify the agreed or baseline test strategies together with the weightage assigned per test strategy before the start of system testing.

2 Identify the number of test cases involved per test strategy together with the priority assigned for each test case before the start of system testing.

3 Obtain the planned number of test iterations.

4 Once execution of system testing starts, obtain the information of number of test iterations involved per test strategy and completeness status of test execution.

5 Based on the completeness status and test iterations involved, calculate and regulate the score for each test case per test strategy using the preferred calculation method.

6 Perform the calculation for either total score or partial score.

7 Regulate the score in (6) against the assigned test strategy weightage.

8 Calculate either the final complete score or partial score.

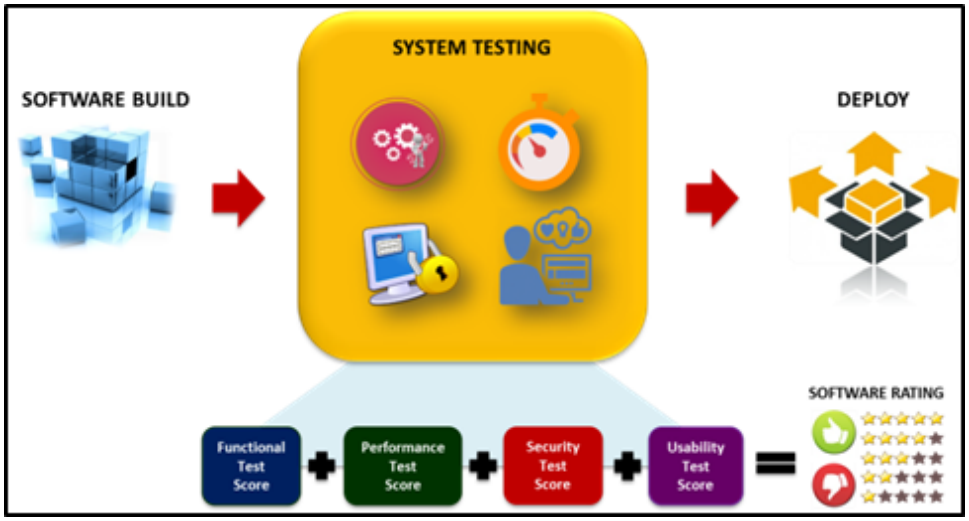
9 Assign trustworthiness rating to the software based on pre-defined rating table.

Generally, after software has been completely built, it is sent to independent testing team for system testing. During system testing, core test strategies executed for the software are functional test, performance test, security test and usability test. A weightage is assigned to every test strategy based on the importance of particular test strategy.

For every test strategy, scores are calculated based on number of test cases and test iterations either complete or incomplete. The total score for each strategy will be regulated against the strategy weightage assigned earlier. Then, the sum of all scores is

calculated to get the complete score or real-time score. The complete or real-time score is mapped with pre-defined software rating table to get the final or real-time software rating. The rating obtained serves as early health indicator of the software before deployment. The representation of how this methodology works as part of the software development process is shown in Figure 2:

Figure 2 Implementation of software trustworthiness rating as part of software development process (see online version for colours)



For a complete rating, it has to wait for all agreed testing strategies to be completed while for partial (real-time rating), it can be obtained at any point of time during execution regardless of any complete/incomplete test strategies.

At first stage, the minimum and maximum test iterations must be set, in which the minimum iterations are two (2) and four (4) for maximum iterations. Then, a fix base point is chosen and weightage per iteration for a test case is established. Total weightage must be equal to 100 across the iteration. This rule is applied to other test cases regardless of test strategies (except usability test since it may have own way of giving score). The calculation method is based on the following points system:

$$\text{Points} = \text{Weightage}(W) \times \text{Base point}(B)$$

Base point (B) remains the same for each iteration. The weightage (W) value shall differentiate the point given to each iteration. Total full points for each test case after taking into account the number of iterations it has to go through should be the same with other test cases. The weightage is further defined based on priority across all iterations. Two options can be considered for this:

- Option 1: 1st iteration carries highest weightage as compared to subsequent iterations. This is due to the importance of passing the test case for the first time it is executed to demonstrate the stability of the feature being tested. Thus, the weightage for 1st iteration should be a fix value for all test cases.

- Option 2: 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

Thus, simple points system is established by taking into. account the minimum and maximum iterations of 2, 3 and 4 as mentioned in option 2 earlier: two-points system, three-points system and four-points system. Example of the structure for applying option 1 into the points system is presented in Table 1. As for option 2, the points system shall look like in Table 2.

Table 1 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

Note: *[ ] = Iteration.

Table 2 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

Calculation result is defined as score. This score is then regulated against the priority or importance assigned to each test case. Total score for the test suite is calculated by dividing it over expected ideal score and multiply by 100.

Total score for each strategy is then regulated against the weightage of each test strategy as agreed before system testing starts. For example, a complete system testing may have a weightage of 50% functional test, 20% security test, 15% performance test and 15% 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.

Final score or real-time score is mapped against the software rating table to determine the rating assigned to the software test. Rating representation is subjected to any preference such as five-stars or good software.

4 Result and discussion

This section describes how the methodology works by using sample test results from hypothetical projects: project a, project b and project C. The assumption is project A has completed system testing, project B has test execution in progress with one of the test strategies has completed testing while project C has none of the test strategies completed.

Table 3 Test cases result from complete functional testing

Test case	*Iteration 1*	*Iteration 2*	*Iteration 3*
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

Project A executed all major test strategies, namely functional testing, performance testing, security testing and usability testing. The result of test cases from a complete functional testing is presented in Table 3. By applying the points system using option 1 and 10 as base point, the result of calculation should look like in Table 4. Thus, the functional test score for project A is as below:

$$\begin{aligned}&\text{Functional Test Score}\\&= (\text{Total Score/Ideal Score}) \times 100\%\\&= (3,760 / 4,300) \times 100\%\\&= 87.4\%\end{aligned}$$

However, when option 2 is applied into the points system, the result is as in Table 5.

The functional test score using option 2 is as below:

$$\begin{aligned}&\text{Functional Test Score}\\&= (\text{Total Score/Ideal Score}) \times 100\%\\&= (1,880 / 2,000) \times 100\%\\&= 89.7\%\end{aligned}$$

Table 4 Calculation result of functional testing score applying option 1

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						*3,760*	*4,300*

Earlier, it was set that the weightage assigned for test strategies in project A is distributed as follows: functional testing (50%), performance testing (20%), security testing (15%) and usability testing (15%). Hence, the final complete score for project A considering other test strategies have completed testing, can be illustrated as in Table 6.

Project B has different situation. One of the test strategies have completed and the others are either in progress or not started at all. Functional testing is completed with the score 89.7% while performance testing is currently in progress. Security testing and usability testing have yet to start. So, the calculation for execution of performance test cases should be done as shown in Table 7. It is called partial score calculation. Thus, the partial score is calculated as below:

$$\begin{aligned}&\text{Partial Performance Test Score}\\ &= (\text{Total Score/Ideal Score}) \times 100\%\\ &= (120/1,200) \times 100\%\\ &= 10\%\end{aligned}$$

Table 5 Calculation result of functional testing score applying option 2

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						*3,860*	*4,300*

Table 6 Final complete score for project A

Test strategy	*Total score*	*Weightage*	*Regulated score*
Functional	89.7	50% (0.5)	44.85
Performance	95.5	20% (0.2)	19.1
Security	90.3	15% (0.15)	13.55
Usability	80.9	15% (0.15)	12.14
Complete score			*89.64*

Table 7 Calculation result of in-progress performance testing score for project B

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*	*1,200*

Using the same weightage distribution as project A, the real-time score for project B can be depicted as shown in Table 8.

Table 8 Real-time score for project B

Test strategy	*Test exec. status*	*Total score*	*Weightage*	*Regulated score*
Functional	Complete	89.7	50% (0.5)	44.85
Performance	In-progress	10.0	20% (0.2)	2.00
Security	Waiting	-	15% (0.15)	-
Usability	Waiting	-	15% (0.15)	-
Real-time score				*46.85*

With regard to project C, only execution of functional testing has started out of four test strategies imposed. By using the similar calculation as performance testing in project B, the result is shown in Table 9. The partial score for functional testing of project C using the same test strategies weightage as previous two projects is calculated as below:

Partial Performance Test Score

$= (\text{Total Score/Ideal Score}) \times 100\%$

$= (680 / 2{,}100) \times 100\%$

$= 32.4\%$

Table 9 Calculation result of in-progress functional testing score for project C

TC	*IT1*	*IT2*	*IT3*	*TC priority*	*Total points*	*Actual score*	*Ideal score*
TC1	10	30	-	3	40	120	300
TC2	10	30	-	3	40	120	300
TC3	10	30	-	1	40	40	100
TC4	10	30	-	2	40	80	200
TC5	10	30	-	3	40	120	300
TC6	10	30	-	1	40	40	100
TC7	0	0	-	1	0	0	100
TC8	10	30	-	3	40	120	300
TC9	10	-	-	2	10	20	200
TC10	10	-	-	2	10	20	200
Total score						*680*	*2,100*

Table 10 outlines the real-time score calculation for project C.

Table 10 Real-time score for project C

Test strategy	*Test exec. status*	*Total score*	*Weightage*	*Regulated score*
Functional	In-progress	32.4	50% (0.5)	16.20
Performance	Waiting	-	20% (0.2)	-
Security	Waiting	-	15% (0.15)	-
Usability	Waiting	-	15% (0.15)	-
Real-time score				*16.20*

It is now the time to assign the suitable rating to the software developed under projects a, project b and project C. As a summary, the score for each project is as below:

1 Project A = 89.40%.

2 Project B = 46.85%.

3 Project C = 16.20%.

The scores as above need to be mapped to a kind of trustworthiness or health rating table. This rating table is developed based on the needs and practicality of organisation adopting the method. The percentage can be translated into STAR rating table (5-star, 4-star, 3-star, 2-star, 1-star), Grade-based rating table (grade A, B, C, D, E), verdict-based (excellent, good, fair, poor, very poor) or other similar rating definition. Sample of rating table using STAR approach is presented in Table 11.

Table 11 Sample rating table using STAR definition

Score	*Rating*	*Description*
80%–100%	5-STAR	To be defined based on context
60%–79%	4-STAR	To be defined based on context
40%–59%	3-STAR	To be defined based on context
20%–39%	2-STAR	To be defined based on context
0%–19%	1-STAR	To be defined based on context

By mapping the scores to Table 11, it is deduced that software in project A is rated as 5-STAR, software in project B is rated as 3-STAR while software in project C is rated as 2-STAR.

Based on the proposed method and its application to the sample test projects, there are few areas that could be discussed further:

- The proposed method assumes that the maximum cycles of testing are four and minimum is three. Thus, the pre-defined score mechanism is based on these minimum and maximum number of iterations.
- Minimum iterations are set to 2 since there might be a situation where test cases will not have to be executed again once it has two consecutive PASS results.
- As for maximum iteration of four, it only takes place when test cases fail at 3rd iteration. Therefore, another iteration is added to make sure the bugs are fixed and test case is PASS during last iteration
- The pre-defined base point is just for illustration purpose only. They might be value of base point for more realistic calculation.
- The more iteration involved for particular test case, the more unstable the corresponding feature is, which translates into inconsistency of the feature to deliver its intended capabilities or services. Therefore, maximum iteration is set to 4.
- As the test strategies differ for various nature and type of software under test, more sets of calculation can be added or expanded based on the imposed strategies. The more strategies imposed to the software, the more set of scores can be calculated.

- Average percentage is used to get the final score in order to have a simpler way to get the final rating of the software under test when mapping to health definition table.
- The proposed method able to accommodate rating for software under test that either has completed testing or execution is still in-progress.

5 Conclusions

The research work is able to establish a systematic way to rate software under test and provide early health indicator, which translates into software trustworthiness rating. The proposed method presented in this paper managed to measure the health of software quantitatively by using the scores assigned to test cases result executed during system testing. Regardless of the mechanism introduced in this paper, it widens the opportunities for the researcher to explore further into the values and meanings of test cases for system testing beyond ensuring that these cases obtained PASS result. As for the practical implementation, it shall help the management or stakeholders to understand the health of the software under test by doing comparative analysis between real-time ratings against complete rating before deciding on the release of the software. They might also decide on suspending the test should the real-time rating does not give good indication. From other perspectives, this area can also be seen as software confidence or software fitness. It has also proven that results from system test cases can be exploited to produce software trustworthiness rating.

For future improvement, the work can be expanded by having specific calculation method according to different test strategies imposed to the software. It can also involve adjusting the calculation and scoring mechanism according to different type of software as well as domain. Different rating table might need to be revisited as well. Simple example is by having a specific set of scoring and rating for mobile application in banking domain. Other than that, it is also beneficial to explore how consistency aspect in test cases result can be incorporated into the calculation. With such anticipated expansion works, it shall set a starting point on measuring the stability and capability of the features for any software produced and undergoing system testing phase.

References

Areias, C., Cunha, J.C., Iacono, D. and Rossi, F. (2014) 'Towards certification of automotive software', *IEEE Proceedings of 2014 International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp.491–496.

ASEAN NCAP (2016) [online] http://www.aseancap.org/our-test/the-ratings-explained/ (accessed 5 September 2016).

Aspire Systems (2011) *Continuous Software Quality through Technical Health Index*, Aspire System's whitepaper, pp.1–5 [online] http://www.aspiresys.com/WhitePapers/Whitepaper_Continuous_Software_Quality_through_Technical_Health_Index.pdf (accessed 22 September 2015).

Bao, T., Liu, S. and Han, L. (2010) '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.

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

Bianco, V.D., Lavazza, L., Morasca, S. and Taibi, D. (2011) 'A survey on open source software trustworthiness', *IEEE Software*, Vol. 28, No. 5, pp.67–75.

Dubey, A., Karsai, G. and Mahadevan, N. (2011) 'Model-Based software health management for real-time systems', *IEEE Proceedings of 2011 Aerospace Conference*, pp.1–18.

EURO NCAP (2016) [online] http://www.euroncap.com/en/about-euro-ncap/how-to-read-the-stars/ (accessed 5 September 2016).

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

Nami, M. and Suryn, W (2012a) 'Case study: using requirements and finite state machine for evaluating software trustworthiness', *Proceedings of 38th Annual Conference on IEEE Industrial Electronics Society (IECON 2012)*, pp.3095–3100.

Nami, M. and Suryn, W. (2012b) 'Software testing is necessary but not sufficient for software trustworthiness', *Proceedings of 2012 International Conference on Trustworthy Computing and Services (ICTCS 2012)*, pp.34–44.

Nami, M. and Suryn, W. (2012c) 'Software trustworthiness: past, present and future', *Proceedings of 2012 International Conference on Trustworthy Computing and Services (ICTCS 2012)*, pp.1–12.

Nami, M. and Suryn, W. (2013) 'Software trustworthiness: past, present and future', *Trustworthy Computing and Services*, Vol. 320, pp.1–12, Springer, Berlin, Heidelberg.

Port, D. and Wilf, J. (2013) 'The value of certifying software release readiness an exploratory study of certification for a critical system at JPL', *IEEE Proceedings of 2013 ACM International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp.373–382.

Pradana, D.S. and Ferdiana, R. (2014) 'Mobile applications rating assessments based on users experience perception', *IEEE Proceedings of 2014 Makassar International Conference on Electrical Engineering and Informatics (MICEEI)*, pp.175–179.

Tao, H. and Chen, Y. (2009) 'A metric model for trustworthiness of software's', *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.

Tao, H. and Chen, Y. (2010) 'A new metric model for trustworthiness of softwares', *IEEE Proceedings of 2010 International Conference on Information Science and Applications (ICISA)*, pp.1–8.

Xia, Z. and Pan, W. (2010) 'Research on the trustworthiness of software', *IEEE Proceedings of 2010 2nd International Conference on Information Science and Engineering (ICISE)*, pp.1–4.

Zhao, X., Liu, Y., Shi, Y. and Zhang, L. (2010) '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.