International Journal on Advanced Science Engineering Information Technology

Vol.11 (2021) No. 1
ISSN: 2088-5334

Test Case Prioritization Approach for Sequence of Events Using Complexity Factor

Emyreema Ja'afar[a,*], Sa'adah Hassan[a], Salmi Baharom[a], Johanna Ahmad[a]

[a]*Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia*
Corresponding author: [*]emy.jaafar@yahoo.com

***Abstract*—Test case prioritization (TCP) is a method to prioritize and schedule test cases. Some approaches have been introduced to minimize the time, cost, and effort for testing the software based on the test cases that are higher priority. Since the more complex the software program, the more intensive the test should be carried out. Thus, complexity is one of the factors that affect the effectiveness of the test case prioritization. However, the existing approaches for measuring complexity have some limitations. This is due to inaccuracy in finding the weightage value for complexity as the value is useful to determine the test case prioritization. Consequently, a complexity metric measurement is needed to determine the weightage value. Hence, this paper presents work on TCP using complexity factors to enhance the accuracy in prioritizing the test cases for event sequences. This work uses Branch Coverage Expectation (BCE) for complexity measurement, in which BCE has been proven its usefulness empirically in the previous research. The event-weightage value based on the complexity is then assigned and used to prioritize the test cases while the Average Percentage of Fault Detected (APFD) metric is used to evaluate the proposed approach. A tool has been developed to ease the process as well as to facilitate the evaluation purposes. The results show the need to combine the complexity factor with other factors to improve the proposed TCP's effectiveness.**

***Keywords*— Test case prioritization; software testing; complexity measurement; branch coverage expectation; sequence of events.**

Manuscript received 24 Oct. 2019; revised 6 Nov. 2020; accepted 17 Dec. 2020. Date of publication 28 Feb. 2021.

I. INTRODUCTION

While the technologies are evolving, the number of software users is expanding; consequently, it increases the demand for advancing software functionalities. As a result, the software becomes more complicated. Software testing is a critical phase in the software development cycle. Particularly for complex software. Organizations must carry out software testing before delivering the product to the users. The objective of software testing is to validate and verify the software product and discover as many errors as possible. However, the developers are not taken seriously due to significant resources in terms of cost, time, and effort; need to be consumed for testing the software. Therefore, test case prioritization (TCP) is one of the research areas aiming to reduce the effort, time, and cost of software testing.

Test case prioritization techniques schedule test cases so that the test cases that are higher in priority are executed earlier than the test cases that are a lesser priority. TCP prioritizes the test cases to increase a test suite's fault detection rate [1]. There has been an increasing amount of work on TCP that shows the researchers' interest in TCP [2]–[4]. However, further work is still needed to improve TCP's effectiveness so that it can be executed faster and cost-efficient. Some factors affect the TCP technique's effectiveness, such as fault, redundancy, complexity, frequency, requirements, time, distance, cost, and permutation.

Generally, a complex structure of a system is always a primary challenge in software development. However, the more complex the software, the more significant number of defects will be found. This is because the complexity factor always one of the essential factors in determining the cost, time, and effort in software development. Numerous complexity metrics have been proposed and introduced in previous research in the different software development areas. The typical complexity metric measurement that has been used by the industries and researchers [5], [6] are Mc Cabe's Cyclomatic Complexity, Halstead metric, Lines of Codes (LOC), Control Flow Graph (CFG), and Function Point (FP).

A study to determine the best metric for measuring event sequences' complexity has been conducted [7]. The study

has shown that Unique Complexity Metric (UCM) is one of the best metrics in measuring event sequences' complexity compared to others. However, UCM still has some restrictions where UCM does not assign the upper and lower bound complexity values. In which, lack of this will lead to inaccuracy in finding the weightage value for complexity. Since this value will be used to determine the test case prioritization, it must be as accurate as possible. Therefore, a different complexity metric measurement is proposed to determine the weightage value. In this paper, the work focusing on determining the test case prioritization technique in event sequences, and the factor used to prioritize the test case is the complexity factor. This paper also presents TCP work using complexity factor that aims to get better accuracy in prioritizing the test cases for event sequence.

A. Related Work

To date, various approaches and techniques of TCP have been developed and introduced. For example, a system-level TCP method was proposed from the required specification [8]. It is mentioned that this method can reduce cost, increase fault detection, thus improves user satisfaction. This method prioritizes the test cases based on six factors like work by Abraham *et al*. [9]. Although the factors used are the same, but the proposed method is different. The algorithm of the proposed TCP method is as shown in Fig. 1.

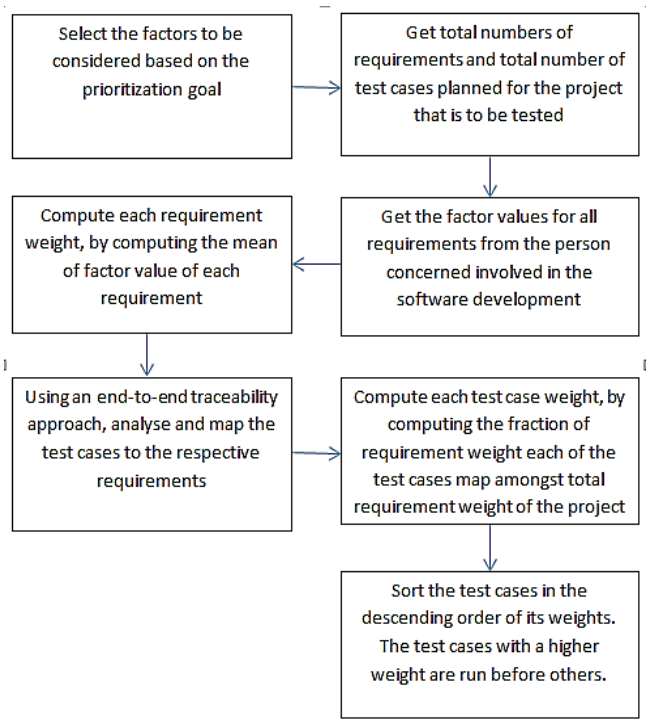

Fig. 1 Algorithm for TCP method [8]

Chaurasia *et al.* [10] suggest five metrics to be used to determine the order of the test cases in the new clustering-based approach. The five metrics are code coverage, test case failure rate, fault detection ratio, execution time, and code complexity metric. The metrics and related formulas are summarized in Table 1.

Table 1 shows that as for the execution time, it is taken while performing the prioritization. Based on the research done by Huang et al. [11], they found that the TCP approach of "fixed-strengths prioritization" does not consider multiple strengths. Therefore, Huang *et al.* [11] proposed a new technique called "aggregate-strength prioritization," which overcomes the technique's limitation. Previous work by Huang *et al*. [6] proposed TCP method named as weight-based GUI test-cases prioritization method (WGTCP) in which weightage value from control flow graph (CFG) is used as a factor. This method can solve the problem of executing all GUI test cases that consume a long time to identify test cases with higher fault. The weighted values were ordered based on the value from high to low or accustomed value.

TABLE I
METRIC AND RELATED FORMULA FOR CLUSTERING-BASED TCP TECHNIQUE [10]

Metric	Formula
Code Coverage Metric	Statements covered by a test case / Total number of covered statements
Test-Case Failure Rate	The number of the items test case has failed / No. of times it has been executed
Fault Detection Ratio	Number of detected faults by a test case / Total number of faults
Code Complexity Metric	The complexity of a test case is assigned as the average of the complexities of classes covered by that test case

In 2016, a weighted TCP technique called Modified Particle Swarm Optimization (MPSO) was introduced to determine the most disclose fault found in the lowest execution time [9]. They proposed this technique with six prioritization factors which are, implementation complexity (IC), execution time (ET), requirement complexity (RC), fault impact in requirement (FI), completeness (CT), and traceability complexity (TC). The weightage values in IC used in their proposed technique is given by the developer, in which it ranges from value 1 to 10. This value range indicates that the more value is given, the more complex it will be implemented. However, the developer's value might be subjective and not consistent if different developer allocates the values. This is because humans might have different perspectives and opinions regarding the degree of complexity in executing the requirements.

TABLE II
SEVERITY VALUE [12]

Severity Rank	Severity Code	Severity Weighted Value
Very High	VHS	32
High	HS	16
Medium	MS	08
Less	LS	04
Least	VLS	02

Nayak *et al*. [12] introduced a new prioritization technique, in which they proposed two factors, which are fault rate (FR) and the severity value of the fault (SVF). This technique aims to prioritize those test cases that expand its viability for identifying faults. The fault rate (FR) is characterized as the most extreme sum of faults situated by a TC for each unit time or the total implementation time. For TC, TCj, FRj have been ascertained utilizing all number of faults, Nj,

situated by TCj, and the aggregate implementation time, time, required by TCj to reveal those faults. It can be communicated in the equation form as follow.

$$FRj = Nj / Timej \quad (1)$$

While, for severity value is weighted as in Table 2.

Fig. 2 illustrates their proposed approach in the form of a flow chart.

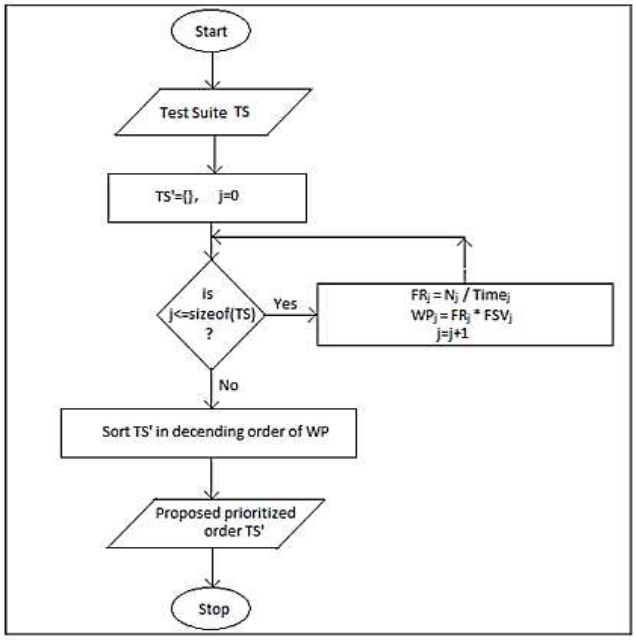

Fig. 2 Flow chart of the proposed approach (Nayak *et al.* [12])

Some studies Hettiarachchi *et al*. [13] and Marchetto *et al*. [14] used the complexity factor to prioritize requirements or test cases where they argued that complexity is an essential factor to determine the priority of the requirements or test cases. They used the Mc Cabe Complexity Metric to measure the complexity. Hettiarachchi *et al*. [13] used Mc Cabe to measure requirements complexity values of the system's codes using Eclipse IDE tool. Besides the complexity factor, Hettiarachchi *et al*. [13] also combined requirements size (RS), requirements modification status (RMS), and potential security threats (PST) in their proposed approach. Hence, an improved risk-based TCP was proposed to thoroughly evaluate requirements risks with a fuzzy expert system [13]. Whereas, Marchetto *et al*. [14] combined it with the size factor. Marchetto *et al*. [14] also argued that the previous study on the TCP approach focuses only on the single objective, either to minimize the cost or to detect the fault earlier. Therefore, they [14] proposed an approach to fulfill the multi objectives technique for prioritizing the test cases.

A recent study by Kumar and Chauhan, [15] mentioned four factors that affect the TCP approach; fault detection (C1), feedback (C2), reliability (C3), and cost (C4). This study also suggests four alternatives TCP technique as below:

- Control structure weighted test case prioritization (A1)
- Total Statement Coverage Prioritization (A2)
- Random prioritization (A3)
- Additional statement coverage prioritization (A4)

However, this study's limitation is the alternative TCP technique, and the criteria only consist of four choices while there are much other techniques and criteria that can be included in this study.

Noor and Hemmati [16] proposed an enhanced similarity-based approach using improved quality metrics. Those metrics are Basic Counting (BC), Hamming Distance (HD), and Edit Distance (ED). These metrics or measurements are used to prioritize the test cases in their proposed approach. Whereas, Wang, Zhao, and Ding [17] developed a TCP approach based on severity fault value that will elevate test cases' priority. This fault severity is divided into four categories or types: fatal fault, serious fault, general fault, and minor fault. This severity fault value becomes the weightage value where each category was assigned with a quantitative value that consists of fatal fault (2^3), serious fault (2^2), general fault (2^1) and minor fault (2^0). This approach is proposed to overcome the test cases with the same maximum coverage rate since the random selection will affect the priority of the test cases.

Based on the studies, most of the research aims to improve the fault's detecting rate and optimize the testing phase's cost and time. This is also the main objective in the software industries, hence, become the main reason why the research on TCP is exceeding. Thus, most of the previous research efforts combined two or more factors in determining the test cases' priority. Besides the fault factor, complexity is also one of the important factors focused on by the researchers. The complexity can be divided into two; the developer's complexity value and the value that came from the code by using the complexity metric such as Mc Cabe's Complexity and Control Flow Graph. We also understand that complexity is one of the important factors because the more complex the program is, the more intensive test needs to be carried out.

Branch Coverage Expectation (BCE) is a complexity measure introduced by Ferrer *et al.* in 2013 [18]. Their research has evaluated BCE with the existing complexity measures and have theoretically proved that BCE is a promising way of measuring complexity. Therefore, we applied BCE complexity measure in our proposed approach to measure the complexity of the sequence of events program for prioritizing the test cases.

II. Material and Method

The proposed approach focuses on developing a TCP using the complexity factor for sequences of events. A TCP using complexity factor solely is proposed as there is no study previously done on using only the complexity factor. In this approach, the complexity factor value is based on the weightage value of prioritizing the test cases. It highlights the activity for prioritizing the test cases of event sequences using the complexity factor where the weightage value is assigned to each event in the program.

- Provide input files: the codes (computer program), the test suite, and events.
- Calculate all the related BCE complexity measurements. It consists of a transition matrix, P, Stationary Probabilities, π, frequency of appearance E(BBi), and the BCE value. BCE calculation and its properties are based on Ferrer et al. [18], [19].

- Assigned the weightage value for all the events. Calculation of TC weightage. In this step, the weightage of all the TCs in the test suite are sum up. The summation is based on the weightage value for each event that been assigned previously.
- Prioritize the TCs – the prioritization is based on the weightage values of the TCs. The TCs is ordered in descending order from the most considerable weightage value to the lowest weightage values.

Fig. 3 illustrates the proposed approach of implementation BCE complexity measurement and TCP.

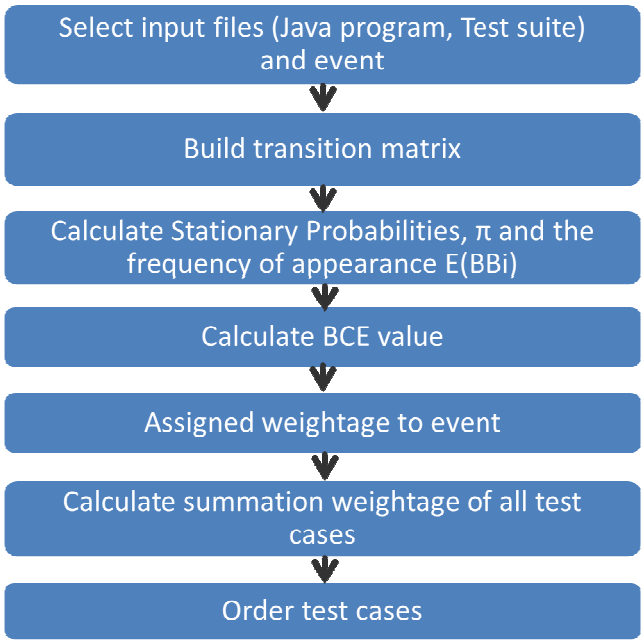

Fig. 3 Flow of the proposed approach

The sub-sections below present each step carried out in the approach.

A. Calculation of Branch Coverage Expectation (BCE)

BCE measure is based on the Markov Chain Model by Andrey Markov, a Russian Mathematician. Markov Chain is mentioned as a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. The conditional probabilities of first-order Markov Chain P (X_{t+1} = - j|Xt = i) = $P_{ij}(t)$ are called one-step transition probabilities and the matrix P(t) = [$P_{ij}(t)$] is the so-called transition probability matrix. Two properties of the transition probability matrices are:

$$Pij \geq 0, \quad (2)$$

$$Xnj=1\ Pij = 1 \quad (3)$$

Ferrer et al. [18] mentioned that if every state in a Markov chain can be reached from every other state, then it is said that the Markov chain is irreducible. For irreducible Markov chains, having only positive-recurrent states, the probability distribution of the states *q(t)* tends to a given probability distribution *p* as the time tends to infinite. This probability distribution *p* is called the stationary distribution and can be computed by solving the following linear equations:

$$\pi\ T\ P = \pi\ T, \quad (4)$$

$$\pi\ T\ 1 = 1 \quad (5)$$

Markov model is built from Control Flow Graph (CFG) where a state of Markov chain is the basic blocks (BB) of the program. BB is a portion of the code that is executed sequentially without any disruptions. The transition probabilities of all branches are computed according to logical expressions that appear in each condition. Once the transitions probabilities completed, the stationary probabilities, π, and the frequency of appearance E(BBi) are computed as:

$$E[BBi\ ] = \pi i/\pi 1 \quad (6)$$

Where, π1 is the stationary probability of the entry basic block, BB_1.

The next step is where the expectation of traversing a branch (i, j) is computed from the frequency of appearance of the previous basic block and the probability to take the concrete branch from the previous basic block as:

$$E[BBi, BBj] = E[BBi] * Pij \quad (7)$$

Then, Branch Coverage Expectation (BCE) is defined as the average of E[BBi , BBj] with a value lower than 1/2. The BCE is bounded in the interval (0, 1/2]. Formally, let A be the set of edges with E [BBi , BBj] < 1/2:

$$A = \{(i, j)|E|[BBi , BBj\ ] < 1/2\ \} \quad (8)$$

BCE is defined as below:

$$BCE = 1\ /|A|\ X\ (i,j)\in A\ E[BBi , BBj\ ]. \quad (9)$$

B. Weightage Value Based on Complexity of the Events

The event weightage value method applied in this approach is based on Huang, *et al.* [6] method called an event-weightage assignment. This method applied in this research based on the suitability of the proposed approach using the BCE complexity measurement. The weightage value given is based on the BCE complexity value for each event. The bigger of BCE value, the bigger the weightage value given to the event based on the rank.

C. Weightage Value for Test Case (TC)

After the weightage value is assigned on each event, each TC's calculation is required to sum up the weightage value for each TC. This value is important to prioritize the TC. The summation of the weightage value is an example below: Test Case: _.add (1). add (1). add (1). add (1). add (1). add (1). front ()

Summation weight= (Event Weight 1 * No of Event 1) +
(Event Weight 2 * No of Event) +
(Event Weight 3 * No of Event 3)
= (3x6) + (1x1)
= 19

Then the TC will be prioritized accordingly based on the summation of TC's weightage value.

In this phase, the proposed approach is applied in a case study of sequence of events program and validated and evaluated using APFD metric. The experiment uses a case study of the Circular Queue Program. A circular queue is one of the sequences of an event program. Currently, we only cover codes in Java programming language. The program and test suite for this study are taken from a previous research done by Baharom and Shukur [20]. The experimental setup in brief as below:

Case Study: Circular Queue Java program
Test Suite: 79 test cases
Event: Add, Remove, Front (Display).

III. Results and Discussion

A prototype tool was developed based on the proposed approach. All the calculations and properties in the BCE complexity measurement were calculated automatically to ease the calculation process, except for the evaluation part where the testing and calculation were using Junit. This tool is limited to calculate the BCE measurement for Java programs only. The tool was developed using Visual Basic as a programming language and Microsoft Access as a temporary database to keep the measurement values.

There are two inputs needed: Java program and test suite, as shown in Fig. 4. The Java program consists of program event sequences, while the test suite consists of multiple test cases. There are four main modules: a module to compute BCE value for each event, a module to give weightage for each event, a module to compute weightage for each TC, and a module to prioritized TC order.

Fig. 4 The input page

The user needs to insert two input files: the Java programming file and the test suites file. The Java programming files must be in the java extension and the test suites in the .txt extension. Once the input files have been selected, the user needs to choose the Java program's event function. For example, if the user needs to choose Add, Remove and Front events in the circular queue program. For a bounded stack program, the user needs to select *Push*, *Pop*, *Top,* and *Depth* events. This event selection is based on the program that the tool needs to measure the BCE complexity. Then, the user can click the CALCULATE button to calculate all the measurements for the BCE complexity. It consists of a transition matrix, P, Stationary Probabilities, π, frequency of appearance E (BBi), and the BCE value.

The selected event is listed in the form of a dropdown list, as shown in Fig. 5 for the ADD event and Fig. 6 for the FRONT event. The tool will automatically calculate the BCE complexity measurement and display all the measurement that consists of stationary probabilities, frequency of appearance, and the BCE complexity value for each event. The value will be stored temporarily in the database for display purposes.

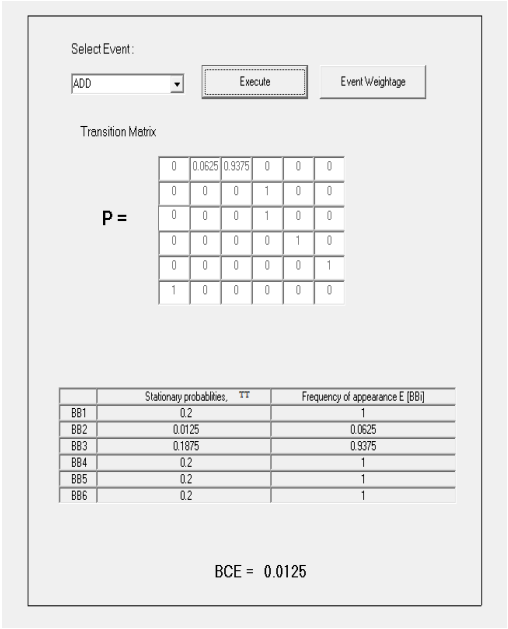

Fig. 5 BCE complexity measurement for ADD event

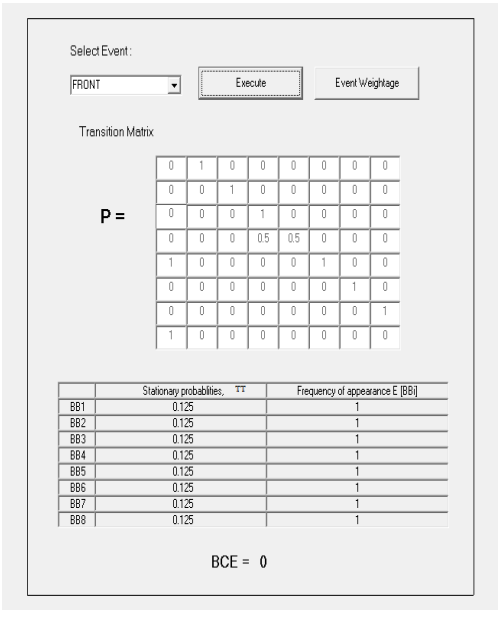

Fig. 6 BCE complexity measurement for FRONT event

The weightage value is then assigned based on the BCE complexity value that had been calculated. Fig. 7 shows the event weightage. The weightage value range is between 1 to many events selected. For example, three events function selected for the circular queue, so the weightage value is the range in between 1 to 3. Value 3 will be given to the highest BCE, value 2 to the second-highest, and value 1 to the lowest BCE. However, if there is the same value, the weightage will be assigned accordingly.

As shown in Fig. 7, the BCE value of the event *adds* and *removes* the same. So, the weightage values are given with the value of 3, which is the maximum value. This shows that both events have the same complexity, while the event *front* has given less BCE value with the minimum weightage value of 1.

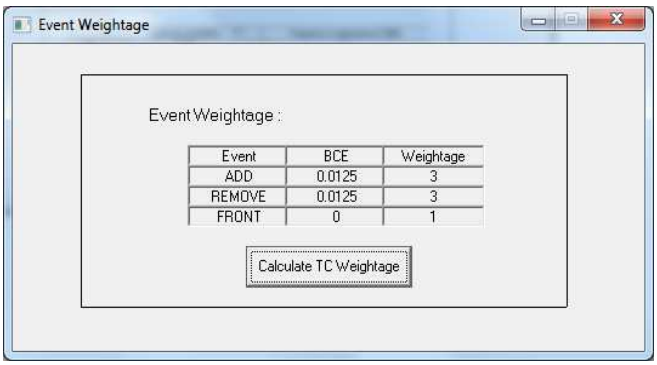

Fig. 7 Event Assign-Weightage

Fig. 8 shows a sample of the weightage calculation for each test case in the test suites that have been selected in the input interface. This interface will be displayed after the Calculate TC weightage button has been clicked. This interface will display all the test cases in the test suites that been input earlier. It will also calculate each test case's weightage based on the event in the test cases and sum up all the values. The example of the calculation is shown in the previous section.

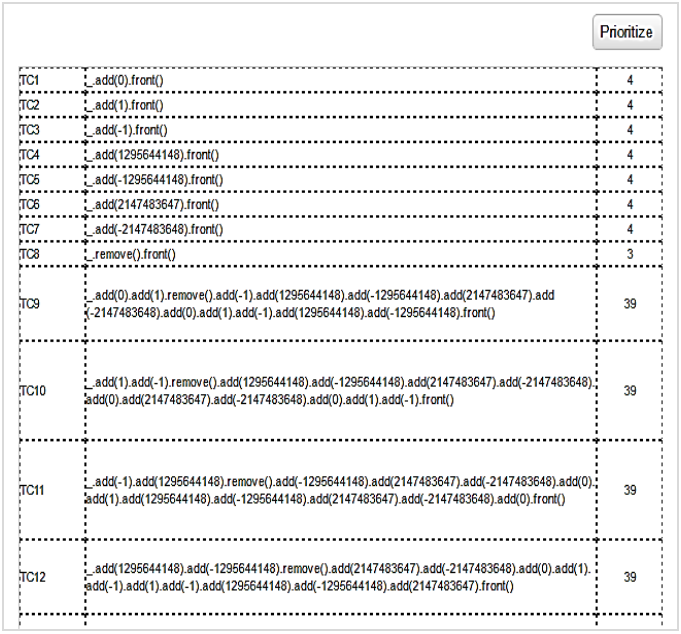

Fig. 8 Test Cases Weightage Calculation

Figure 9 above shows a sample of the orderly test cases. The test cases were sorted and ordered based on the weightage value of each TC. The order is called the prioritized TC. The test cases are sorted in descending order from most significant weightage value to lowest weightage value.

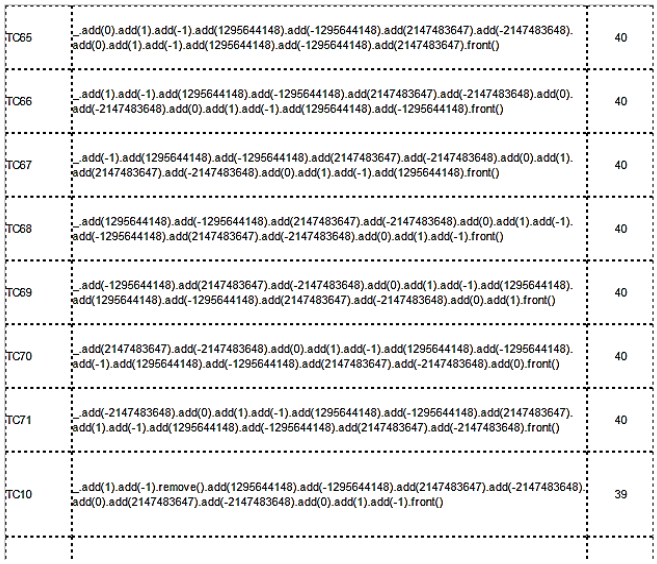

Fig. 9 The Order of Test Cases Prioritization

From the result, the new prioritization order for 79 test cases is as: TC65, TC66, TC67, TC68, TC69, TC70, TC71, TC10, TC11, TC12, TC13, TC14, TC15, TC73, TC74, TC75, TC76, TC77, TC78, TC79, TC9, TC16, TC17, TC18, TC19, TC20, TC21, TC22, TC30, TC31, TC32, TC33, TC34, TC35, TC36, TC44, TC45, TC46, TC47, TC48, TC49, TC50, TC23, TC24, TC25, TC26, TC27, TC28, TC29, TC37, TC38, TC39, TC40, TC41, TC42, TC43, TC51, TC52, TC53, TC54, TC55, TC56, TC57, TC58, TC59, TC60, TC61, TC62, TC63, TC64, TC72, TC1, TC2, TC3, TC4, TC5, TC6, TC7, TC8.

Average Percentage of Fault Detection (APFD) is then used to prove the BCE complexity measurement's effectiveness. The average percentage of fault detected (APFD) metric was introduced to measure the average rate of fault detection per percentage of test suite execution [21]. Most of the previous researchers [22], [23] used the APFD metric on determining the effectiveness of their proposed techniques. Hence, the same metric is used to evaluate the effectiveness of the proposed approach based on the prioritized and non-prioritized test cases.

APFD values range from 0 to 100, where higher numbers imply faster fault detection rates. As discussed earlier, the test cases' prioritization is based on the weightage value of the complexity measurement. APFD produces statistically and shown a significant result. It is significant to software testing's objectives, which is to detect a fault as quick as possible. The formula to find the APFD value is as below:

$$APFD = 1- \{(Tf1+Tf2+.... +Tfm)/mn\} +(1/2n) \quad (10)$$

Where:

T be a test suite containing n test cases
F be a set of m mutants revealed by T
n is a few test cases
m is the number of mutations detect fault

TF_i be the first test case in ordering T' of T, which reveals fault *i*.

In order to calculate the APFD, a mutation must be determined first. In recent years, numerous mutation tools have been developed [24]. In this research, the Jester Mutation Operators [25] was applied where the operators are as shown in Table 3.

TABLE III
JESTER MUTATION OPERATOR [20]

No	Mutation Operator
1	Change numerical constants. Mutate 0 to 1
2	Flip Boolean values. Mutate true to false and vice versa
3	Mutate if(condition) to if (true||condition)
4	Mutate if(condition) to if (false&&condition)
5	Mutate ++ to – and vice versa
6	Mutate! = to == and vice versa

There are 24 mutations found in the CQ program. These mutations are injected into the original CQ Java program and test suite using the JUnit in Eclipse. Each of the mutants is tested in 79 test cases. The total testing for CQ program is 1896 tests. A fault matrix is built from this testing. Fig. 10 shows some parts of the fault matrix. The fault that was detected is marked with 'x.'

APFD value for Non-Prioritization Test Cases (NPTC) is the benchmark for this work compared with the prioritization TCs (PTC). The APFD value can be calculated based on the complete fault matrix and the APFD formula. From the fault matrix, there are 8 faults detected in the CQ program, which are:

TF1 is in first TC = 1
TF2 is in ninth TC = 9
TF3 is in first TC = 1
TF4 is in first TC = 1
TF5 is in eight TC = 8
TF6 is in ninth TC = 9
TF7 is in fifty-one TC = 51
TF8 is in ninth TC = 9

The APFD value for NPTC

= 1-0.1408+0.0063
= 0.8529
APFD = 85.29%

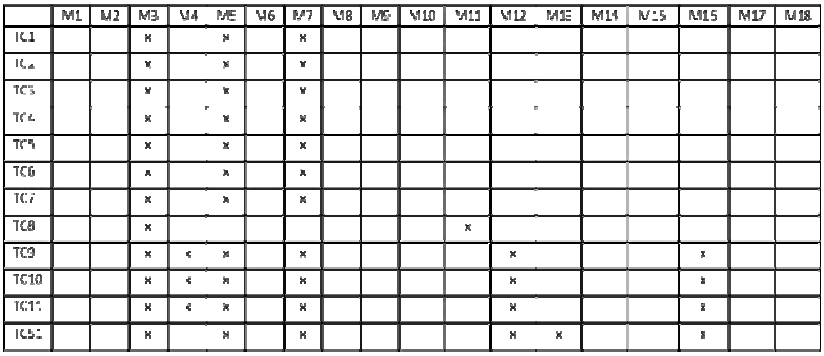

Fig. 10 Part of fault matrix

The APFD for Prioritized Test Cases (PTC) value is based on the TC prioritized order in the implementation section. The APFD is calculated as below:

TF1 is in first place = 65
TF2 is in ninth place = 11
TF3 is in first place = 65
TF4 is in first place = 65
TF5 is in eight places = 10
TF6 is in ninth place = 11
TF7 is in fifty-one place = 46
TF8 is in ninth place = 11

The APFD value for PTC

= 1-0.4494+0.0063
= 0.5443
APFD = 54.43%

The proposed approach's effectiveness is evaluated by comparing the APFD value of non-prioritization test cases (NPTC) and prioritization test cases (PTC) for the CQ program. The result is displayed in a graph form as in Fig. 11.

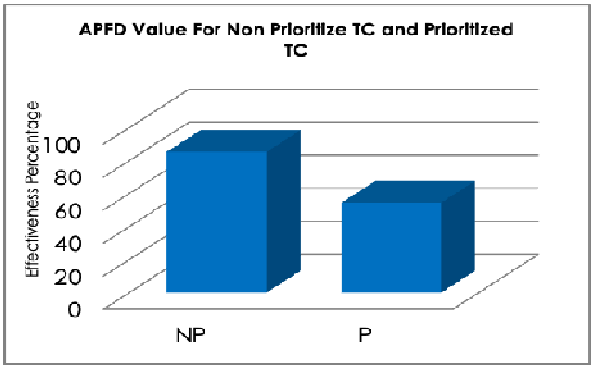

Fig. 11 Graph on Comparison APFD value for NPTC and PTC

The graph in Fig. 11 shows that the percentage of non-prioritized test cases is higher than the prioritized test cases, in which NPTC is 85.29% while PTC is 54.43%. NP value is 36.82% higher than P-value. This value shows that the prioritized test cases are less effective than the non-prioritized test cases. In PTC, the test cases are sorted and prioritized based on the weightage value of the event's complexity value. It can be concluded that if the test cases consist of many events, the weightage value becomes higher and more complex as compared to the test cases with fewer events.

IV. Conclusion

This paper focuses on the test case prioritization-based BCE complexity measure approach (TCP-BCE) for events. An experiment using a case study (Circular Queue program) is conducted to evaluate TCP-BCE's effectiveness. APFD metric is used to calculate the effectiveness of NPTC and TCP-BCE. Based on the result, it can be concluded that TCP-BCE is less effective. This is due to the test cases that are prioritized based on complexity weightage event order. The more events involved in one test case, the more complicated it will be, and the weightage value will be higher. It is shown that using only a complexity factor as a factor to prioritize the test cases is not suitable and not comprehensive enough. Wherein the weightage is based on the complexity of the codes. The more complex the codes, the higher the test case's weightage value, and the value of the APFD will be lower; regardless of any complexity measure approach is implemented. It is recommended in the future that this proposed approach can be enhanced by including more factors to be combined with the complexity factor. However, remain to use BCE, since BCE calculation is proven to be a right measurement of complexity, and it also can estimate the number of relevant test cases needed for the program.

Acknowledgment

We are grateful to Universiti Putra Malaysia and the Ministry of Education Malaysia through the Fundamental Research Grant Scheme (FRGS) that funded this research.

References

[1] G. Duggal and B. Suri, "Understanding Regression Testing Techniques," *COIT*, 2008, India.

[2] C. Catal and D. Mishra, "Test case prioritization: A systematic mapping study," *Software Quality Journal*, vol. 21(3), pp.445–478, 2013.

[3] H. Srikanth, M. Cashman, and M. B. Cohen, "Test case prioritization of build acceptance tests for an enterprise cloud application: An industrial case study," *J. Syst. Softw*., vol. 119, pp. 122–135, 2016.

[4] X. Zhang, X. Xie, and T. Y. Chen, "Test case prioritization using adaptive random sequence with category-partition-based distance," *2016 IEEE Int. Conf. Softw. Qual. Reliab. Secur.*, 2016, p. 374–385.

[5] A. Marchetto, M. Islam, and W. Asghar, "A multi-objective technique to prioritize test cases," *IEEE Transactions*, 42(10), pp. 918–940, 2016.

[6] C. Y. Huang, J. R. Chang, and Y. H. Chang, "Design and analysis of GUI test-case prioritization using weight-based methods," *Journal of Systems and Software*, vol. 83(4), pp.646–659, 2010.

[7] J. Ahmad and S. Baharom, "Comparision of software complexity metrics in measuring the complexity of event sequences," *Information Science and Applications*, vol. 424, 2017.

[8] R. Krishnamoorthi and S. A. Sahaaya Arul Mary, "Factor oriented requirement coverage-based system test case prioritization of new and regression test cases," *Information and Software Technology*, vol. *51*(4), pp. 799–808, 2009.

[9] A. K. Joseph, G. Radhamani, and V. Kallimani, "Improving Test Efficiency through Multiple Criteria Coverage based Test Case Prioritization using Modified Heuristic Algorithm," in *International Conference on Computer and Information Sciences*, 2016, p. 430–435.

[10] G. Chaurasia, S. Agarwal, and S. S. Gautam, "Clustering based novel test case prioritization technique," *2015 IEEE Students Conference on Engineering and Systems (SCES)*, 2015, pp.1–5.

[11] R. Huang, J. Chen, D. Towey, A. T. S. Chan, and Y. Lu, "Aggregate-strength interaction test suite prioritization," *Journal of Systems and Software*, *99*, pp. 36–51, 2015.

[12] S. Nayak, C. Kumar, and S. Tripathi, "Effectiveness of prioritization of test cases based on Faults," *2016 3rd International Conference on Recent Advances in Information Technology*, 2016, p. 657–662.

[13] C. Hettiarachchi, H. Do, and B. Choi, "Risk-based test case prioritization using a fuzzy expert system," *Information and Software Technology*, vol. 69, pp. 1–15, 2016.

[14] A. Marchetto, M. M. Islam, W. Asghar, A. Susi, and G. Scanniello, G, "A Multi-Objective Technique to Prioritize Test Cases," *IEEE Transactions on Software Engineering*, *42*(10), pp. 918–940, 2016.

[15] K. H. Priyanka and N. Chauhan, "A Novel Approach for Selecting an Effective Regression Testing Technique," *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE, 2016, p.1122–1125.

[16] T. B. Noor, and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, 2015, p. 58–68.

[17] Y. Wang, X. Zhao, and X. Ding, "An effective test case prioritization method based on fault severity," in *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 23-25 Sept. 2015, p.737-741.

[18] J. Ferrer, F. Chicano, and E. Alba, "Estimating software testing complexity," *Information and Software Technology*, vol. *55*(12), pp. 2125–2139, 2013.

[19] J. Ferrer, "Optimization Techniques for Automated Software Test Data Generation," PhD Thesis, University of Malaga, 2016.

[20] S. Baharom and Z. Shukur, "The conceptual design of module documentation-based testing tool," *Journal of Computer Science*, vol. 4 (6), pp.454-462, 2008.

[21] S. Elbaum, A. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in *Proc. Int'l. Symp. Softw. Testing and Analysis,* Aug. 2000, p. 102-112.

[22] A. Ansari, A. Khan, A. Khan, and K. Mukadam, "Optimized regression test using test case prioritization," *Procedia Comput. Sci*., vol. 79, pp. 152–160, 2016.

[23] P. Mahapatra and S. Tripathy, "Code based test case prioritization using APFD metric," *Global J. of Mech., Eng. & Comp. Sciences*, vol.3(2), pp.3-5, 2013.

[24] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," *IEEE Transactions on Software Engineering*, vol. 7, no. 2, pp.77-84, 2006.

[25] I. Moore, (2001). "Jester and Pester," http://jester.sourceforge.net/