Test Case Prioritization For Regression Testing Based On Cc Metric Analyzer

J. Satish Babu, Krishna Mohan G., MSR Prasad, G. Vamsi Krishna, R. Nithish Kumar, T. Viswanath

Abstract: Regression testing is the method of retesting the updated components of the code and checking that no new defects are created into already existing code. The regression test suite is very large and an intelligent technique is needed to select a few test cases which will reduce the test cost. In these scenarios, the test case prioritization techniques are to be used effectively by ordering the test cases from high priority to low priority based on a valid methodology to increase the efficiency of regression testing without decreasing the rate of fault detection. Cyclomatic complexity is software metric used to find the flow graph methods with high risk are generated and test cases are to be applied to such methods first.Applying Test cases after ordering them according to the priority will take considerably less time to execute to detect more faults and average percentage of faults detection (APFD) metricis used to measure test case prioritizations efficiency. A priority based execution for regression testing with high fault detection in less amount of time and cost can achieved by using a priority basedtechnique.

Keywords: Cyclomatic Complexity, Regression Testing, Prioritization, Metric

I. INTRODUCTION

Cyclomatic Complexity is software utilized and is developed by Thomas J, McCabe Sir in the year 1976 by utilizing the complication of the program. It will be a quantitative measure of the multifaceted nature about modifying guidelines. It specifically measures that number of linearly autonomous ways through a program's source book. It will be a standout amongst the metric In view of not system size yet the entire greater amount on information/control stream.[5] The Cyclomatic Complexity is product metric that gives quantitative measures of legitimate multifaceted nature of a system. Cyclomatic Complexity of a segment about source book is those check of the amount for linearly freeways through the source book. For instance, if the source book held no choice focuses for example, such that in through those code. Whether those code needed An single if proclamation holding a solitary condition, there might a chance to be two ways through the code: you quit offering on that one way the place the on proclamation will be assessed similarly as accurate Cyclamatic multifaceted nature may be an programming metric used to measure the multifaceted nature of a project. These metrics, measures autonomous ways through system source book. Autonomous way will be characterized Likewise a way that need no less than one threshold which need not been traversed When for whatever viable ways. Cyclomatic unpredictability has a chance to be ascertained with admiration to functions, modules, techniques alternately classes inside a project. This metric is formed by Thomas j. McCabe over 1976 and it is In view of an control stream representational of the project. Control stream depicts a system by a chart which comprises of hubs. We argue that structural coverage may not be the bestcriteria to guide the prioritization of dynamic executions [2]. In this paper, we intend to propose a new TCP technique that is based on Cyclomatic complexity along with few other metrics. When our proposed system is compared with the rest of the methods we can strongly believe that TCP technique based on CC metric is much better in detecting faults and complexity of the program.

- ***J. Satish Babu,*** *Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur, Vaddeswaram, A.P., India.*
- ***Krishna Mohan.G,*** *Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur, Vaddeswaram, A.P., India.*
- *M.S.R. Prasad, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur, Vaddeswaram, A.P., India.*
- ***G.Vamsi Krishna,*** *Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur, Vaddeswaram, A.P., India.*
- ***R.Nithish Kumar,*** *Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur, Vaddeswaram, A.P., India.*
- ***T.Viswanath,*** *Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur, Vaddeswaram, A.P., India.*

II. RELATED WORK

The complete definition of the TCP problem was first proposed by Rothaermel et al.[1]Retest All-It is the most

simple and easy approach in theory for regression testing but in practical it is very costly and not a optimal solution[7]. This test provides a very basic functionality to execute all the test cases that are present in the test suite respectively the backdrop of this would be it is very costly and time consuming.[4] Regression test selection and the other simple technique Retest all technique takes time and effort as all test cases are executed atleast once and are used to test the modified program after updating based on the requests of the client. Selective regression testing tries to attempt to reduce the cost but it is not very effective and the fault detection rate also decreases drastically. The whole point of testing is to find thedetection rate if that is compromised then there is less probability to use such technique except in some exceptional situationsThe third one in the available approaches is Test Suite Reduction, this technique uses the available information about source skeleton program and test suite to remove the test cases which have become redundant and useless over the course of time , when the code is updated according to the needs of the customer to add new functionality. It is quite different from regression test selection as regression test selection does not permanently remove test cases but quite oppositely it does select few those and execute them. The only advantage of this technique over the regression test selection as is that it reduces cost of testing, executing, and also adds on a unique feature of managing test suites over

future release of software or when a customer requests to add a custom functionality but the downside of Test Suite Reduction is it drastically reduces the fault detection capability as the test suite size is reduced. Both are directly proportional. Test case prioritization is a unique technique where each available test case is assigned a priority which is defined based on some algorithm or methodology. After the priority is identified and assigned the test cases are arranged according to the priority from the highest priority to lowest priority and the highest priority test cases are applied first reducing the cost-efficiency. When compared to the rest of the three techniques this one is very optimal as the efficiency is much higher when compared to them and even the cost of executing is very less. The rate at which the faults are detected is very important and this technique will satisfy the requirements. Test Case Prioritization, using criteria, sorts test cases to detect the most faults as fast as possible in order to improve testing efficiency[3]. Mondal et al. proposed a different kind of technique to decrease the amount of time taken to test the source program by using a analysis called alpha-shape analysis. But this method of analysis uses method sequence for analysis purpose. We have implementedtheir approach with the hope to compare itagainst ourapproach. However, our initial empirical study shows that it isnot scalable so wedecide to exclude it from our experiments.[6]

III. OUR APPROACH

Our approach is to find the average CC number of the entire package source code, even the incomplete underdevelopment code can also be given as input to find the CC number to make alterations accordingly thus reducing the cost.

3.1 Cyclomatic Complexity Number

Cyclomatic Complexity number M is defined by no of test cases required to achieve the branch coverage. M can also be the number of paths that are linearly independent throughout the whole graph.Branch coverage is equivalent to Upper Bound and No of paths are equivalent to Lower Bound.

Cyclomatic Complexity Number M for the above program when calculated theoretically will generate

M=E-N+2P

E- edges generated in the control graph
N- nodes generated in the control graph
P- connected components

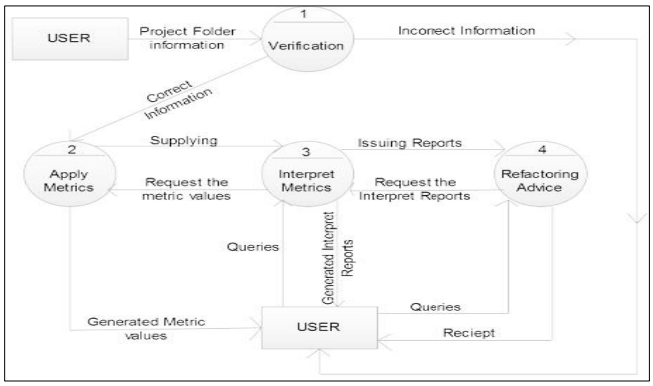
Our approach mainly consists of three important steps.The following steps are to be followed in order to calculate the cyclomatic complexity of the source program.

Step 1: A graph has to be constructed from the skeleton source program

Step 2: Independent Paths, Nodes, and edges are to be identified

Step 3: From the number of independent paths the cyclomatic complexity can be calculated.

Once the cyclomatic complexity is calculated the Test Cases can be written. If multiple test cases are already written and available priority can be assigned and test cases can be ordered and executed.The value of Cyclomatic Complexity M in the above example source code is calculated is “x” x- any positive integer that x number of unique test cases are very much required to complete the whole coverage.



***3.1** Framework of CC metric Analyzer*

The CC metric Analyzer takes information from the user as shown in Fig 3.1, and it goes through a verification phase if the information is correct it the information is forwarded to the Apply Metric phase where we apply the metric and generated metric values are again forwarded to the user as the output if any queries are needed by the user phase 3 comes into play called as Interpret Metrics with the help of refactoring phases queries are answered and values are generated. Generated the reports are again sent back to the user. The value of Cyclomatic Complexity M in the source code is calculated as ‘x’ x being any positive integer, that implies ‘x’ unique test cases are very much required to complete the whole coverage. Once the number of test cases that are required to execute the test suite without decreasing the average fault detection rate are found out, test cases are written and executed Cyclomatic complexity number M can be calculated manually if the program is too small implying the lines of code (LOC) are very less or even if the functionalities the program has to offer are minimal. Automated tools which can used to calculate the cyclomatic complexity of the program need to be used only if the program is very complex like huge lines of code, more number of functionalities, more number of methods , more number of inherited classes, more number of recursive calls as these always involves more flow graph generation. The source program can be used as input to the automated tool and the complexity number of each module, method is generated including the complexity number of the entire source code.

3.2 Complexity Number and its corresponding meaning

If Complexity number is between 1-10 that would mean the source program is well structured and is well written. It would also mean the source code can be tested efficiently and easily with less cost. Testing cost and effort required for such source codes with fewer complexity numbers are very less. If the Complexity number is between 10-20 that would mean the source program is complex and is not well written. It would also mean the source code takes an effort more than the previous scenario where the testing is very easy. The cost of testing is a bit higher when compared to previous code but not very high it can be defined as the cost and effort

required for source programs with complexity numbers between 10 and 20 is medium. If the Complexity number is between 20-40 that would mean the source program is very complex and is not well written. It would also mean the source code takes much more effort than the previous scenario where applying the required test case to detect the defect and fault rate is medium. The cost of testing is very high when compared to the previous scenario. The final conclusion would be the cost and effort to execute a test suite on these types of source programs is very high, When the complexity number is above 40 (M>40) that would mean the source code is not even fit for testing and written without any proper structure. The cost and effort required to even try to test are very high. Generally speaking, if the complexity number m is above 40 it is better not to deploy but take time to rebuild it from scratch.

Complexity Number	Meaning
1-10	Structured and well written code High Testability Cost and Effort is less
10-20	Complex Code Medium Testability Cost and effort is Medium
20-40	Very complex Code Low Testability Cost and Effort are high
>40	Not at all testable Very high Cost and Effort

TABLE 1 *Complexity Number and its corresponding meaning*

IV. EMPIRICAL STUDY

In this section, we conduct an empirical study to answer the research questions that are listed below. All the experiments arecarried out on a Custom PC with Intel Core i7-8600K 3.60GHz processor and HyperX Dual Channel 8X2GB DDR4 RAM at 3200MHz and ACER Laptop with Intel Core i5-6200U 2.40GHz processor and a Samsung Single Channel 8GB DDR4 RAM at 2133MHz.

Q1) Is the developed source program testable?

Q2)Is the developed application reliable enough?

As an answer to the Q1, we can use this technique to identify the “level” of our testing. It is a practice that if the result of Cyclomatic complexity is a very large number than expected , we consider this as a piece of unique functionality to be of complex nature and hence we conclude as a tester; that the piece of code / functionality requires in-depth testing or we can include it as program is not testable and must be rebuilt from the scratch. On the other hand, if the result of the Cyclomatic Complexity is a smaller number, we conclude as QA that the functionality is of less complexity and decide the scope accordingly.

We gave our own code as the input and achieved the following results.

Demographic Information				Conditional and Looping Statements							Output		
Sr. No.	Release Name	Module Name	Class Name	Count of "IF..THEN"	Count of "IF	Count of "DO..WHIL	Count of "FOR.."	Count of "CATCH"	Count of "CONDITIONAL AND	Count of "CONDITIONAL OR i.e.	No of Decision Points (P)	CCN (P+1)	Number of Test Cases
1	main project	testing	CCGeneratorService	30			5			1	36	37	37
2	main project	testing	CFileParserClass	14		1	1	2	4	2	24	25	25
3	main project	testing	CSFileParserClass	14		1	1	2	4	2	24	25	25
4	main project	testing	CyclomaticComplexity	3	1	2	1	1	2		10	11	11
5	main project	testing	CyclomaticComplexityExcel	5	1			2			8	9	9
6	main project	testing	FileParserClass	14		1	1	2	4	2	24	25	25
7	main project	testing	PLSQLFileParserClass	13		1	1	2	3		20	21	21
8	main project	testing	TestClass	1					1		2	3	3
9	main project	testing	VbFileParserClass	13		1	1	2	4		21	22	22

SCREENSHOT

4.1 Output of the Source Program

From Screenshot 4.1, the total number of classes,methods and functionalities present in the whole package are generated along with the count of the following statements “IF..THEN” “IF THEN..ELSE” “DO..WHILE” “FINALLY”CATCH” “BITWISE AND i.e '&' "CondtionalANDi.e. &&" "CONDITIONAL OR i.e. “No of Decision Points CCN (P+1) ,Number of Test Cases are generated successfully. No ofDecisionPoints(P) implies total count of all the conditional statements which creates a decision scenario in flow graph. Decision points in control graph are denoted by a diamondshape.CCN(Cyclomatic Complexity Number) is calculated by adding one to decision points present throughout the control graph. In our experiment the highest No of Decision Points (P) was 36 for the class CCGeneratorService and the lowest was 2 for TestClass.CCN(Cyclomatic Complexity Number) was highest for the class CCGeneratorService,which was 37 and the lowest was 3 forTestClassThe count for the number of unique test cases required to apply them to the respective methods are found individually. If the number of test cases required to test the particular method is small that does not mean the source code is small it means there are fewer decision points in the method decreasing the complexity number which in return decreases the number of test cases required. Test cases can be given a priority to be applied to the methods based on the size decreasing the time and cost required drastically. It can be altered according to the situation if the time is not abundant methods that require fewer test cases can be prioritized first and executed to find the fault detection.

(A)Cyclomatic Complexity of top 3 programs/methods in the release with the highest complexity		
Sr. No.	*Method/Program Name (Optional)*	*Cyclomatic Complexity*
1	*CCGeneratorService*	*37*
2	*CFileParserClass*	*25*
3	*CSFileParserClass*	*25*

TABLE

4.2 Cyclomatic Complexity of Top 3 Methods

The CC of the top 3 methods of the package are listed in Table 4.2,No of DecisionPoints(P)implies total count of all the conditional statements which creates a decision scenario in flow graph. Decision points in control graph are denoted by a diamondshape.
CCN(Cyclomatic Complexity Number) is calculated by adding one to decision points present throughout the control graph.

In our experiment the highest No of Decision Points (P) was 36 for the class CCGeneratorService and the lowest was 2 for TestClass. CCN(Cyclomatic Complexity Number) was highest for the class CCGeneratorServicewhich was 37 and the lowest was 3 forTestClassCCN implies that number of unique test cases are very much required to complete the whole coverage. Once the number of test cases that are required to execute the test suite without decreasing the average fault detection rate are found,test cases are written and executed

(B) Average Cyclomatic Complexity			
No. of programs/methods in the release	*9*	*Sum of Cyclomatic Complexity of all methods/programs in the release*	*178*
Average Cyclomatic Complexity	*19.78*	*Number of Unit Test Cases*	*178*

TABLE
4.3 Average Cyclomatic Complexity

The total average complexity of the entire source code has came out as 19.78 which means the source program is complex but can be tested without much effort and cost of testing is moderate.

CONCLUSION

The Cyclomatic Complexity Number(M) can be used to find the number of test cases that required to run the test with effective average fault detection by prioritizing the available test cases. Developers can assure that all the paths in the control flow graph have been tested atleastonce.It can also be used improve code coverage during development phase. Evaluation of the risk associated with the application or program can be done parallelly. Following these metrics early in the cycle which ever that is being followed by the organization reduces the risk factor in the program.

REFERENCES

[1] Elbaum, S., Malishevsky, A. G. and Rothermel, G. 2000.Prioritizing test cases forregression testing. In Proceedings of the 2000 ACM SIGSOFT International Symposium
[2] J. Chi et al., "Test Case Prioritization Based on Method Call Sequences," 2018 IEEE42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, 2018, pp.
[3] Yoo, S. and Harman, M. 2012. Regression testingminimization, selection andprioritization: a survey. Softw.Test. Verif. Reliab. 22, 2 (Feb. 2012), 67-120.
[4] Satyaprasad, R., Suryakiran, C., &Krishnamohan, G. (2018). SPC based software reliability using modified genetic algorithm: Rayleigh model. Journal of Advanced Research in Dynamical and Control Systems, 10(4 Special Issue), 500-506. Retrieved from www.scopus.com
[5] Chaitanya Krishna, B., Yeshwanth Srinath, A., Bhavani, N., & Jaya Sai, G. (2018). Analysing software quality using CMMI-2 with agile-scrum framework. International Journal of Engineering and Technology(UAE), 7(1.1 Special Issue 1), 290-293. Retrieved from www.scopus.com
[6] D. Mondal, H. Hemmati, and S. Durocher, “Exploring test suite diversificationand codecoverage in multi-objective test case selection,” inICST. IEEE, 2015, pp. 1–10. 9
[7] B. Jiang and W. Chan. Bypassing code coverage approximationlimitations via effectiveinput-based randomized test case prioritization.In Proc. IEEE International Conference on Computers, Software andApplications, COMPSAC 2013, pages 190–199, 2013.