International Journal of Hybrid Information Technology
Vol. 9, No. 5 (2016), pp. 47-60
http://dx.doi.org/10.14257/ijhit.2016.9.5.05

Test-Case Prioritization Using Adaptive Approach

Sarabjit Kaur and Bhanu Priya

Computer Science and Engineering department, CT Group of Institution, Jalandhar (PB) 144008
er_sarabjitkaur35@rediffmail.com, thakurpriya552@gmail.com

Abstract

Test-case prioritization provides the way to organize the order of test cases according to the priority. There are many approaches which are used for test case prioritization. Adaptive approach can be used for test case prioritization. At the execution time adaptive approach gives the all information regarding to current execution or previous execution of program. The adaptive approach evaluating the effectiveness of the test cases. This also provides the information regarding to errors. The adaptive approach arrange the priority of each test cases in decreasing order. We have conducted results which are based upon java programs. We are generating 100 test cases based upon java language. In, this paper we are using APSC metric which describe the statement coverage of test cases. The adaptive approach select the test case then execute it, after execution the priority of the test case will be calculated. This paper also gives the information of statement coverage as well as gives information regarding to those statement which are not covered or unselected. The error rate of test case execution will be also described using adaptive approach. As well as we can say that this paper presents the detail study regarding to APSC metric with the help of adaptive approach.

Keywords*: software testing; regression testing; test-case prioritization; adaptive approach; APSC metric*

1. Introduction

A Software testing plays an important role to providing the quality of software system. Software testing is very costly or it will consume maximum cost of any software [1]. There are many types of software testing techniques. Software testing is performed by the testers. Each software testing is different than others because some are suitable for small code or some are suitable for large code of software. Software testing is always performs after the coding phase of any software. In testing code of the software will be check. Testing will be always performing with the help of some tools. There are large numbers of tools which are now used for software testing. Therefore, many researchers work on to improve the efficiency of software testing.

Regression testing is also an important technique of software testing. In regression testing same code will be used for finding the faults from the existing code.

Regression testing is very time consuming process. Regression testing helps to finding the faults from the code. Test-case prioritization 4 is firstly proposed in regression testing whose aim is to test the software by applying the test cases of the previous version. For performing the regression testing, test case prioritization will be used. Test case prioritization schedules the execution order of test cases to finding the faults in software. There are large numbers of test cases will be used in test case prioritization approach. The test case will be generated with the help of any programming language like java *etc*. If we are collecting a large number of test cases then it is known as test suite. Test case prioritization is used for calculating the priority of the test cases. Priority will be calculated according to the fault occurrence in test cases. The highly faulty test case will

ISSN: 1738-9968 IJHIT

be placed of the last of the test case prioritization approach. There are many algorithms which are used for finding the priority of test cases. The original sequence of test cases will be generated. The test case executes one by one each test case. After execution a new sequence of test cases will be generated which is based on the priority. The original test case sequence or priority based test case sequence will not match with each other because in priority based test cases the less faulty test case will be coming on the first number of the sequence. But in the original test case sequence we do not know that which test cases is having more faults or which are having less faults. There are many metrics which are calculated with the help of test case prioritization approach. Test Case prioritization is used for scheduling the order of execution of test cases. There are many algorithms that can be used for test case prioritization. Adaptive algorithm also works for providing prioritization of test cases. In this paper, we apply adaptive algorithm for prioritizing the test cases. The scheduling of test cases are very important for executing the large number of test cases. The aim of test case scheduling is to find the faulty modules. A module means set of code. The collection of code is also known as a module. In coding we can make different modules of the code. Test case prioritization is also a very difficult to find out the faults. But now many approaches can be used which are very helpful for finding the priority of the test cases. Test case prioritization provide the effectiveness of the software. It will be help to execute the test cases in order. In adaptive approach a fault detection of test cases will be calculated while selecting test cases. In this one test case will be select firstly, and then execute it. When the first test case will be complete or finish its execution it will be store its output. The selection procedure of test cases will be continuing until test cases will be finished. Some statements are unselected because these will not execute. So, adaptive approach is best approach for test case prioritization. The existing approaches are not useful as compare to adaptive approach. Adaptive approach will be providing best and effective results. Test case prioritization approaches based on general approaches and the version specific approaches.

2. Related Work

In present work, we are discussing about the test case prioritization approaches. We are defining the most work which is related to the APSC metric. This paper focuses on the statement coverage for checking the more test cases for find the errors. It will also define the execution time of test cases.

2.1. Test Case Prioritization

Many authors presented different views on the test case prioritization approach. There are large numbers of studies which are implemented on the test case prioritization approach. But test case prioritization approaches are divided into two categories.

2.1.1. General Test Case Prioritization

In this approach, test case prioritization schedules the order of test cases. In this approach modified test case prioritization is based upon previous approach. But in this paper, we can use the general test case prioritization approach. This approach is suitable for the purpose of prioritization.

2.1.2. Version Specific Test Case Prioritization

This approach is used for scheduling the order of test cases. In this prioritization the modified version can be used. Test case priority will be modified with the help of different versions. Then, it will check the priority of the test cases and find the order of the priority without losing the previous information of the priority of the test cases.

Version specific approach is further classified into two approaches cod based approaches and model base approaches.

In code based approach the source code will be modified for executing the new code which is derived with the help of previous code. In this code version of the code will not be specified.

Model based approaches schedule the order of the test cases based upon the modification on the system model and its execution information. In this approach modified code can be used.

In our present work, we are studying various general and version specific approach but after analyzing both we identify that version specific approach is better as compare to general approach. So, we can use apply the version specific approach in our present work.

2.2. Adaptive Test-Case Prioritization

In this section, we firstly define the adaptive process for the test case prioritization. It will also define the process of working of adaptive approach. The detail study of adaptive algorithm can be also identified in this section. We can also describe the algorithm of the adaptive approach. In our present work, we are implementing the adaptive approach on the statement coverage metric. It can be calculates the number of statement will be execute according to the time limit. The statement coverage will be vary according to the different values of P and Q factors.

2.2.1. Adaptive Process

In the present work, we define an adaptive test case prioritization approach which provides the scheduling of the execution order of test cases. In this Figure it will show that the whole process of defining the adaptive algorithm. This algorithm also describes the process of working of adaptive approach. Adaptive test case prioritization approaches will be implemented on different metrics. Each metric will be based upon its specific functionality.

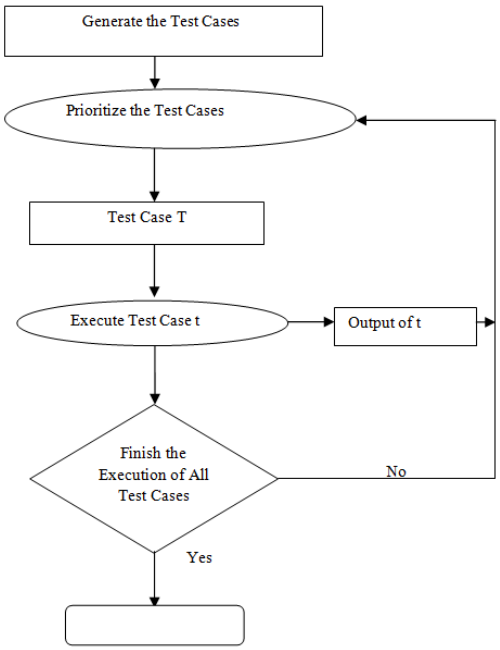

Figure 1. Adaptive Test Case Prioritization Approach

The existing test-case prioritization approaches, including the general approaches and the version-specific approaches, view test-case prioritization and test-case execution as two separate processes. In the above Figure 1, it define the procedure of executing number of test cases. A large number of test cases will be generated which are executed with the help of APSC metric. In APSC metric it will define that how many statements are covered. It will show the number of covered statements as well as define how many statements are uncovered. Priority of test cases can by determined with the help of formulas. Adaptive approach is used for prioritize the test cases. So it is better technique as compare to existing approach. It also defines the process of fault detection. It defines the number of faults are calculated during execution of test cases. It is based upon adaptive algorithm approach. This algorithm defines the process of calculating the fault detection capability of test cases as following section.

Algorithm 1 Algorithm of the Adaptive Approach

Input: Test suite T

Output: Prioritized test suite T'

Declaration: T_S represents the latest selected test case

General Process:

Begin

for each test case t in T

calculate initial Priority(t)

end for

select the test case (i.e., t_S) with the largest Priority in t

add t_S to T'

$T \leftarrow T - \{t_S\}$

run t_S

while T is not empty **do for each** test case t in T

modify Priority(t) based on the output of t_S

end for

select the test case (i.e., t_S) with the largest Priority in T

add t_S to T'

$T \leftarrow T - \{t_S\}$

run t_S **end while return** T'

End

3. Research Method

Our research method is used for finding the prioritization of test cases and calculating the error detection capability.

3.1. Error-Detection Capability

The adaptive approach defines that how many statements are executed completely. Priority of the test cases can be checked using equation (1).

$$Priority\ (t) = \sum_{s \text{ is executed by } t} Potential(s) \quad (1)$$

Where priority (t) defines the priority of test cases. The value of Potential(s) lies between 0 to 1. The Potential(s) represents the statements are executed or not. Potential(s) is based upon equation (2).

$$Potential(s) = \begin{cases} Potential'(s) & s \text{ is not executed by } t' \\ Potential'(s)*q, & s \text{ is executed by } t' \wedge \\ & t' \text{ is passed} \\ Potential'(s) & s \text{ is executed by } t' \wedge \\ & t' \text{ is failed} \end{cases} \quad (2)$$

Here (s) represents the statements are executed or not. If the statements of test cases will executed completely then test case will be passed. If statement will not executed completely the it become failed. Here q represent the values of p and q factor whose result must become 1 in each time. The values of p and q will be vary but result always become 1. It also represents like p+q=1.

3.2. Measurement

The adaptive approach is also determines the execution order of test cases based upon error-detection capability. It also gives the result of error detection time period that how many statements are executed with in different values of input. In adaptive approach following equation can be used for error detection capability.

$$APSC = 1 - \frac{(TS1 + TS2 + \ldots\ldots\ldots\ldots + TSm)}{nm} + \frac{1}{2n} \quad (3)$$

Where n represents the total number of test cases, m represents the total number of detected errors and TS_i $(1 \leq i \leq m)$ denotes the smallest number of test cases programmers will go until error I is exposed. APSC values range lies between 0 to 1. These values must be very important because it will provides help for calculating the statement coverage.

Table 1. Results of Generating Test Cases

Test Case No.	Test Case Name
1	AntClassLoaderPerformance.java(t0)
2	DispatchTaskTest.java(t1)
3	DirectoryScannerTest.java(t2)
4	ExecutorTest.java(t3)
5	ImmutableTest.java(t4)

6	IncludeTest.java(t5)
7	ProjectTest.java(t6)
8	ProjectComponentTest.java(t7)
9	ProjectHelperRepositoryTest.java(t8)
10	PropertyFileCLITest.java(t9)
11	EscapeUnicodeTest.java(t10)
12	TokenFilterTest.java(t11)
13	StripJavaCommentsTest.java(t12)
14	LineContainsTest.java(t13)
15	DynamicFilterTest.java(t14)
16	HeadTailTest.java(t15)
17	ConcatFilterTest.java(t16)
18	ReplaceTokensTest.java(t17)
19	ReplaceTest.java(t18)
20	TouchTest.java(t19)
21	LoadFileTest.java(t20)
22	XmlPropertyTest.java(t21)
23	PropertyTest.java(t22)
24	PreSetDefTest.java(t23)
25	MultiMapTest.java(t24)
26	MkdirTest.java(t25)
27	ZipTest.java(t26)
28	ExecTaskTest.java(t27)
29	JarTest.java(t28)
30	UnzipTest.java(t29)
31	UntarTest.java(t30)
32	ParallelTest.java(t31)
33	SQLExecTest.java(t32)
34	TypeAdapterTest.java(t33)
35	AntStructureTest.java(t34)
36	UpToDateTest.java(t35)
37	MacroDefTest.java(t36)
38	FilterTest.java(t37)
39	PathConvertTest.java(t38)
40	DeleteTest.java(t39)
41	BUnzip2Test.java(t40)
42	GzipTest.java(t41)
43	DeltreeTest.java(t42)
44	SleepTest.java(t43)
45	ConcatTest.java(t44)

46	InputTest.java(t45)
47	AvailableTest.java(t46)
48	ExecuteJavaTest.java(t47)
49	WarTest.java(t48)
50	ConditionTest.java(t49)
51	AntLikeTasksAtTopLevelTest.java(t50)
52	XmlnsTest.java(t51)
53	BZip2Test.java(t52)
54	ZipExtraFieldTest.java(t53)
55	GetTest.java(t54)
56	ImportTest.java(t55)
57	ManifestTest.java(t56)
58	DynamicTest.java(t57)
59	EchoXMLTest.java(t5)
60	ManifestClassPathTest.java(t59)
61	MoveTest.java(t60)
62	NiceTest.java(t61)
63	ExecuteWatchdogTest.java(t62)
64	TStampTest.java(t63)
65	DateUtilsTest.java(t64)
66	ResourceUtilsTest.java(t65)
67	UnPackageNameMapperTest.java(t66)
68	UnPackageNameMapperTest.java(t66)
69	PackageNameMapperTest.java(t68)
70	VectorSetTest.java(t69)
71	ClasspathUtilsTest.java(t70)
72	LinkedHashtableTest.java(t71)
73	FileUtilsTest.java(t72)
74	LayoutPreservingPropertiesTest.java(t73)
75	JAXPUtilsTest.java(t74)
76	SymlinkUtilsTest.java(t75)
77	XMLFragmentTest.java(t76)
78	ReaderInputStreamTest.java(t77)
79	DOMElementWriterTest.java(t78)
80	DeweyDecimalTest.java(t79)
81	LoaderUtilsTest.java(t80)
82	LazyFileOutputStreamTest.java(t81)
83	StringUtilsTest.java(t82)
84	LineOrientedOutputStreamTest.java(t83)
85	FileListTest.java(t84)

86	ZipFileSetTest.java(t85)
87	FileSetTest.java(t86)
88	PatternSetTest.java(t87)
89	ResourceOutputTest.java(t88)
90	AddTypeTest.java(t89)
91	FlexIntegerTest.java(t90)
92	DescriptionTest.java(t91)
93	DirSetTest.java(t92)
94	RedirectorElementTest.java(t93)
95	IsReferenceTest.java(t94)
96	IsFileSelected.java(t95)
97	ContainsTest.java(t96)
98	AntVersionTest.java(t97)
99	EqualsTest.java(t98)
100	IsReachableTest.java(t99)

Table 2. Results of Priority of Test Cases

Test Case No.	Test Case Name	Test Case Priority
Test 1	ExecuteJavaTest.java(t47)	187
Test 2	AvailableTest.java(t46)	159.2498
Test 3	DispatchTaskTest.java(t1)	142.92722
Test 4	TypeAdapterTest.java(t33)	134.11157
Test 5	AntClassLoaderPerformance.java(t0)	126.0342
Test 6	PropertyTest.java(t22)	109.266014
Test 7	AddTypeTest.java(t89)	107.55554
Test 8	LineContainsTest.java(t13)	102.29387
Test 9	StripJavaCommentsTest.java(t12)	98.64918
Test 10	TokenFilterTest.java(t11)	94.06605
Test 11	EscapeUnicodeTest.java(t10)	92.21553
Test 12	PropertyFileCLITest.java(t9)	89.39593
Test 13	ProjectHelperRepositoryTest.java(t8)	84.989174
Test 14	ProjectComponentTest.java(t7)	83.22448
Test 15	ProjectTest.java(t6)	83.22448
Test 16	IncludeTest.java(t5)	80.6499
Test 17	ImmutableTest.java(t4)	33.67293
Test 18	ExecutorTest.java(t3)	3.8109636
Test 19	DirectoryScannerTest.java(t2)	3.6393287
Test 20	AntLikeTasksAtTopLevelTest.java(t50)	3.567732
Test 21	ConditionTest.java(t49)	3.4612107

Test 22	WarTest.java(t48)	3.2662015
Test 23	InputTest.java(t45)	3.1202614
Test 24	ConcatTest.java(t44)	2.9488444
Test 25	SleepTest.java(t43)	2.81604
Test 26	DeltreeTest.java(t42)	2.6782207
Test 27	GzipTest.java(t41)	2.5461962
Test 28	BUnzip2Test.java(t40)	2.4144042
Test 29	DeleteTest.java(t39)	2.2936842
Test 30	PathConvertTest.java(t38)	2.2485604
Test 31	FilterTest.java(t37)	2.1798081
Test 32	XmlPropertyTest.java(t21)	2.0731232
Test 33	LoadFileTest.java(t20)	1.9300495
Test 34	TouchTest.java(t19)	1.8689111
Test 35	ReplaceTest.java(t18)	1.7669033
Test 36	ReplaceTokensTest.java(t17)	1.7398925
Test 37	ConcatFilterTest.java(t16)	1.6873205
Test 38	HeadTailTest.java(t15)	1.5928447
Test 39	MacroDefTest.java(t36	1.5216743
Test 40	UpToDateTest.java(t35)	1.4466661
Test 41	AntStructureTest.java(t34)	1.3651532
Test 42	SQLExecTest.java(t32)	1.3036776
Test 43	DynamicFilterTest.java(t14)	1.2150334
Test 44	PreSetDefTest.java(t23)	1.15422794
Test 45	ResourceOutputTest.java(t88)	1.1542794
Test 46	PatternSetTest.java(t87)	1.118987
Test 47	FileSetTest.java(t86)	1.096566
Test 48	ZipFileSetTest.java(t8)5ExecuteJavaTest.java(t47)	1.0634328
Test 49	FileListTest.java(t84	1.0417372
Test 50	LineOrientedOutputStreamTest.java(t83)	1.0417372
Test 51	StringUtilsTest.java(t82)	1.0417372
Test 52	XmlnsTest.java(t51)	1.0417372
Test 53	LoaderUtilsTest.java(t80)	1.0417372
Test 54	DeweyDecimalTest.java(t79)	1.0417372
Test 55	DOMElementWriterTest.java(t78)	1.0417372
Test 56	ReaderInputStreamTest.java(t77)	1.0417372
Test 57	XMLFragmentTest.java(t76)	1.0417372
Test 58	SymlinkUtilsTest.java(t75)	1.0417372
Test 59	JAXPUtilsTest.java(t74)	0.98964924
Test 60	ParallelTest.java(t31)	0.98964924
Test 61	UntarTest.java(t30)	0.96010333

Test 62	UnzipTest.java(t29)	0.911084
Test 63	JarTest.java(t28)	0.8616754
Test 64	ExecTaskTest.java(t27)	0.82225317
Test 65	ZipTest.java(t26)	0.77766013
Test 66	MkdirTest.java(t25)	0.7657723
Test 67	MultiMapTest.java(t24)	0.742084
Test 68	LazyFileOutputStreamTest.java(t81)	0.6913724
Test 69	FlexIntegerTest.java(t90)	0.65655434
Test 70	PatternSetTest.java(t87)	0.5788541
Test 71	IsReachableTest.java(t99)	0.4327941
Test 72	DirSetTest.java(t92)	0.40923685
Test 73	ManifestClassPathTest.java(t59)	0.4090237
Test 74	MoveTest.java(t60)	0.4090237
Test 75	ManifestTest.java(t56)	0.388571662
Test 76	ImportTest.java(t55)	0.37202916
Test 77	GetTest.java(t54)	0.36914378
Test 78	ZipExtraFieldTest.java(t53)	0.36914378
Test 79	BZip2Test.java(t52)	0.36914378
Test 80	DescriptionTest.java(t91)	0.35342783
Test 81	LayoutPreservingPropertiesTest.java(t73)	0.33610314
Test 82	FileUtilsTest.java(t72)	0.33315173
Test 83	LinkedHashtableTest.java(t71)	0.33315173
Test 84	ClasspathUtilsTest.java(t70)	0.33315173
Test 85	VectorSetTest.java(t69)	0.08246337
Test 86	PackageNameMapperTest.java(t68)	0.016657613
Test 87	GlobPatternMapperTest.java(t67)	0.016657613
Test 88	UnPackageNameMapperTest.java(t66)	0.016657613
Test 89	ResourceUtilsTest.java(t65)	0.016657613
Test 90	DateUtilsTest.java(t64)	0.016657613
Test 91	TStampTest.java(t63)	0.016657613
Test 92	ExecuteWatchdogTest.java(t62)	0.016657613
Test 93	NiceTest.java(t61)	0.016657613
Test 94	EchoXMLTest.java(t58)	0.05824711
Test 95	RedirectorElementTest.java(t93)	0.0150335
Test 96	IsReferenceTest.java(t94)	0.014140389
Test 97	IsFileSelected.java(t95)	0.013440465
Test 98	ContainsTest.java(t96)	0.0
Test 99	AntVersionTest.java(t97)	0.0
Test 100	EqualsTest.java(t98)	0.0

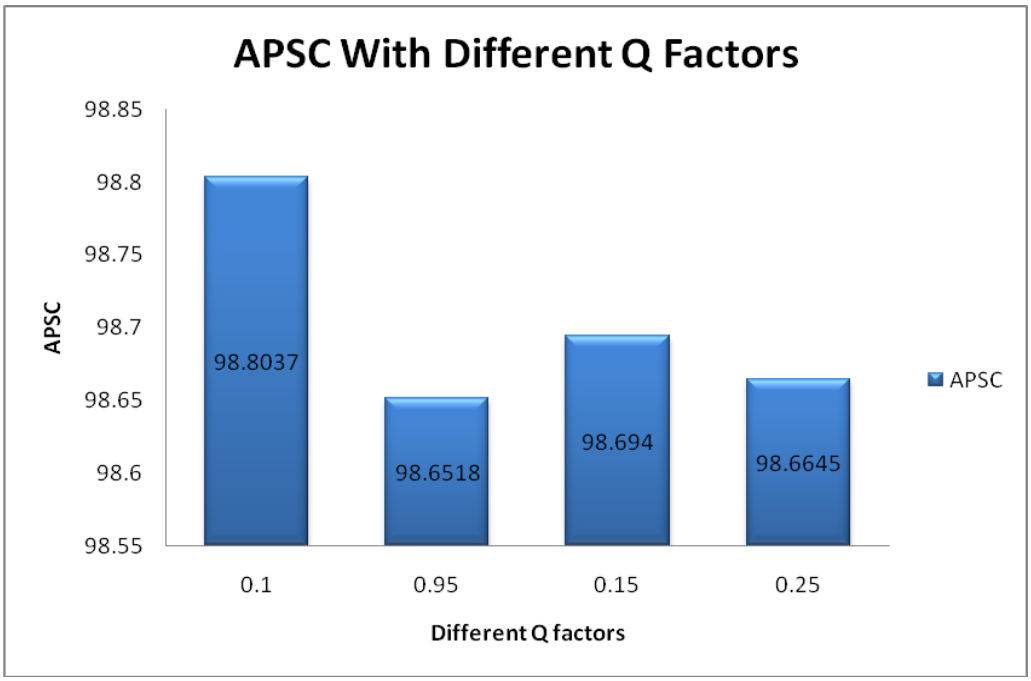

Figure 2. APSC Metric with Different Q Factors

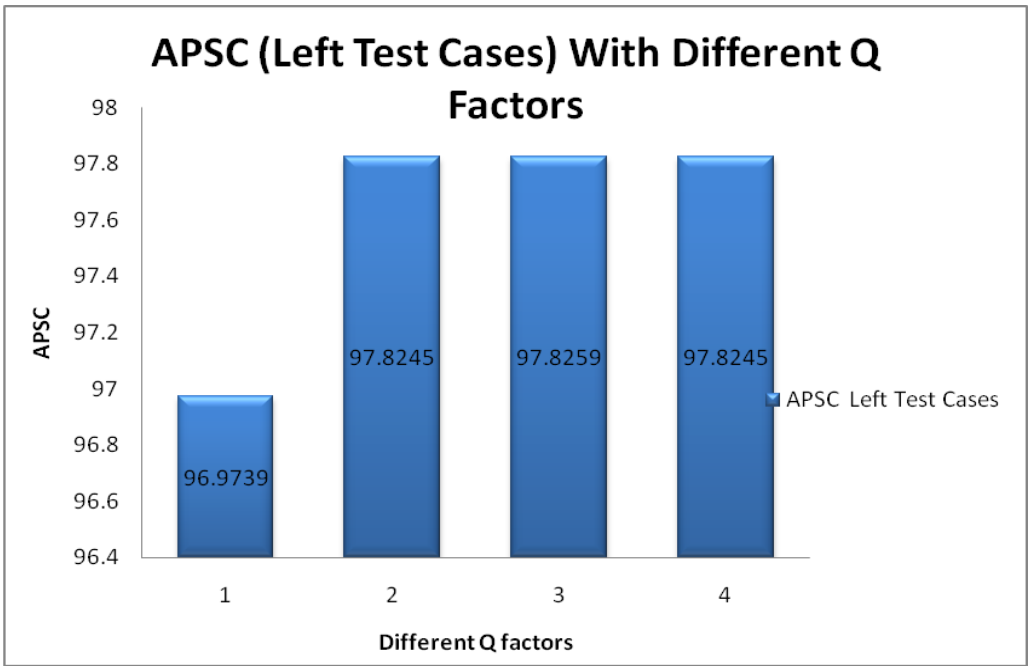

Figure 3. APSC Metric with Left Test Cases

5. Conclusion and Future Work

Test-case prioritization is provide the scheduling of execution order of test cases. An adaptive test-case prioritization approach, which calculates the execution order of test cases as well as during the execution of test cases. In particular, the adaptive approach selects test cases based on their error- detection capability, which is calculated based on the output of selected test cases. It will provide the other execution information like statement coverage can be collected during the execution of the modified program and

may improve the adaptive approach, it may consume extra cost to collect such information and increment the time cost of regression testing.

In the future, we plan to conduct experiments to compare the adaptive approach with the existing test-case prioritization approaches on their time cost. Furthermore, we will attempt to present another adaptive test-case prioritization approach based on other execution information of the modified program besides the output. In this work we only apply average percentage of statement coverage (APSC) metric but in future work we can apply many metrics together.

References

[1] J. S. Collofello and S. N. Woodfield, "Evaluating the Effectiveness of Reliability-Assurance Techniques", Journal of Systems and Software, vol. 9, no. 3, **(1989)**, pp. 191-195.
[2] H. Leung and L. White, "Insights into Regression Testing", International Conference on Software Maintenance, **(1989)**, pp. 60-69.
[3] B. Beizer, "Software Testing Techniques", New York, NY: VansNostrand Reinhod, **(1990)**.
[4] Y. F. Chen, D. S. Rosenblum and K. P. Vo, "Test-Tube: A System For Selective Regression Testing", In The 16th International Conference On Software Maintenance, **(1994)**, pp. 211-222.
[5] W. Wong, J. Horgan, S. London and H. Agrawal, "A Study Of Effective Regression Testing In Practice", In International Symposium On Software Reliability Engineering, **(1997)**, pp. 230-238.
[6] T. Y. Chen and M. F. Lau, "A New Heuristic for Test Suite Reduction", Information and Software Technology, vol. 40, no. 5-6, **(1998)**, pp. 347-354.
[7] G. Rothermel, R. Untch, C. Chu and M. Harrold, "Test-Case Prioritization: An Empirical Study", In International Conference on Software Maintenance, **(1999)**, pp. 179-188.
[8] S. Elbaum, A. Malishevsky and G. Rothermel, "Prioritizing Test Cases for Regression Testing", In International Symposium on Software Testing and Analysis, **(2000)**, pp. 102-112.
[9] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Prioritizing Test Cases for Regression Testing", IEEE Transactions on Software Engineering, vol. 27, no. 10, **(2001)** October, pp. 929-948.
[10] J.-M. Kim and A. Porter, "A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments", In International Conference on Software Engineering, **(2002)** May, pp. 119-129.
[11] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test-Case Prioritization: A Family of Empirical Studies", IEEE Transactions on Software Engineering, vol. 28, no. 2, **(2002)**, pp. 159-182.
[12] J. A. Jones and M. J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage", IEEE Transactions on Software Engineering, vol. 29, no. 3, **(2003)** March, pp. 195-209.
[13] B. Korel, L. H. Tahat and M. Harman, "Test Prioritization Using System Models", In the 21st IEEE International Conference on Software Maintenance, **(2005)**, pp. 559-568.
[14] K. R. Walcott and G. M. Kapfhammer, "Time-Aware Test Suite Prioritization", In International Symposium on Software Testing and Analysis, **(2006)**, pp. 1-12.
[15] B. Korel, L. H. Tahat and M. Harman, "Model-Based Test Prioritization Heuristic Methods and their Evaluation", In the 3rd ACM Workshop on Advances in Model Based Testing, **(2007)**, pp. 34-43.
[16] Z. Li, M. Harman and R. Hierons, "Search Algorithms for Regression Test-Case Prioritization", IEEE Transactions on Software Engineering, vol. 33, no. 4, **(2007)**, pp. 225-237.
[17] S. S. Hou, L. Zhang, T. Xie, H. Mei and J. S. Sun, "Applying Interface-Contract Mutation in Regression Testing of Component- Based Software", In the 23rd International Conference on Software Maintenance, **(2007)**, pp. 174-183.
[18] B. Korel, G. Koutsogiannakis and L. Tahat, "Application of System Models in Regression Test Suite Prioritization", In International Conference on Software Maintenance, **(2008)**, pp. 247-256.
[19] S. S. Hou, L. Zhang, T. Xie and J. S. Sun, "Quota-Constrained Test-Case Prioritization for Regression Testing of Service-Centric Systems", In the 24th International Conference on Software Maintenance, **(2008)**, pp. 257-266.
[20] B. Jiang, Z. Zhang, W. K. Chan and T. H. Tse, "Adaptive Random Test-Case Prioritization", In the 24th International Conference on Automated Software Engineering, **(2009)**, pp. 257-266.
[21] L. Zhang, S.-S. Hou, C. Guo, T. Xie and H. Mei, "Time- Aware Test-Case Prioritization Using Integer Linear Programming", In International Symposium on Software Testing and Analysis, **(2009)**, pp. 213-224.
[22] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou and G. Rothermel, "A Static Approach to Prioritizing Junit Test Cases", IEEE Transactions on Software Engineering, vol. 38, no. 6, **(2012)**, pp. 1258-1275.
[23] L. Zhang, D. Marinov, L. Zhang and S. Khurshid, "Regression Mutation Testing", In International Symposium on Software Testing and Analysis, **(2012)**, pp. 331-341.G.
[24] S. Yoo and M. Harman, "Regression Testing Minimisation, Selection and Prioritisation: A Survey", Software Testing, Verification and Reliability, vol. 22, no. 2, **(2012)**, pp. 67-120.

[25] L. Zhang, D. Hao, L. Zhang, G. Rothermel and H. Mei, "Bridging the Gap Between the Total and Additional Test-Case Prioritization Strategies", In the 35th International Conference on Software Engineering, **(2013)**.
[26] D. Hao, X. Zhao and L. Zhang, "Adaptive Test Case Prioritization Guided by Output Inspection", In Proceedings 37th IEEE Conference of Computer Software and Applications, vol. 4, **(2013)** July,pp. 169-179.

