

which was presented in Figure 2 and 3 respectively. Suppose after some time, the developers of service A modify the pseudocode to what is shown in figure 4. The differences are shown in italics.

```

1 order(item) -
2 if (item exists) -
3 if (item is in stock) -
4 if (customer has money) -
5 order item;
6 return successful;
} else
7 return ERROR: 103: customer lacks funds
}
} else -
8 return error("ERROR: 104: item not in stock");
}
} else
9 return error("ERROR: 110: item does not exist");
}
}

```

Figure 4: Altered pseudocode for ordering service from Fig 2.

The regression test selection approach must build the control-flow graph for this new version of the order service and that is shown with the original one in Figure 5.

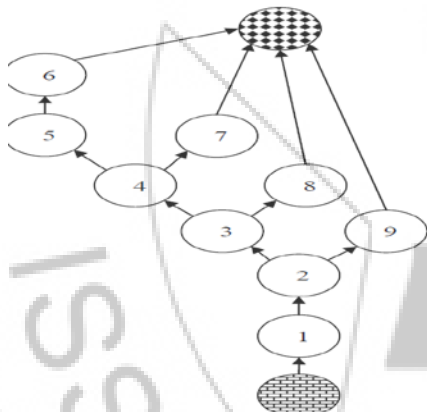


Figure 5: Control Flow Graph for pseudocode in Figure 4

The algorithm which determines the set of dangerous edges compares the two control-flow graphs by performing a dual-traversal as described. The result of the dual traversal marks the following edges dangerous: 1-2, 2-3, 2-7, and, 3-4. It selects these edges because the node corresponding to four is structurally different than the original and because the node corresponding to seven is textually different. The coverage information can easily be thought of as a table and the process is simply a table lookup using that coverage information. The technique guarantees that any test case which does not cover a dangerous edge, or entity, will behave exactly the same in both P and P', and thus can never expose a new fault in P'. Since it is guaranteed to only remove those tests which can never expose new faults in P', this technique is safe because it minimizes the number of test cases while maintaining the same level of confidence provided by selecting all test cases. For example, suppose that the original service A was augmented with test cases and coverage information which are both shown in figure 6. Note that since the code shown is pseudocode, the test cases will follow suit.

Test Cases

Inputs corresponding to three test cases

1. Order item which does not exist
2. Order item which does exist but is not in stock
3. Order item which does exist and is in stock

Expected outputs corresponding to the three test cases

1. return error
2. return error
3. return successful

Coverage Information

1. 1-2-7
2. 1-2-3-6
3. 1-2-3-4-5

Figure 6: Three test cases and their coverage information for service A

The coverage table is used as lookup table and tests numbered one and three are selected for retesting. These tests are selected because the dangerous edge list prefixes these two tests completely. Control-flow graphs are ideal for use in Web service environments for a number of reasons. Firstly, control-flow graphs can be generated from programs written in any language, or extracted from designs at any granularity. Secondly, since control-flow graphs are special cases of finite state machines, they can be composed into global finite state machines [6]. These two characteristics of control-flow graphs are essential for supporting both the interoperability and composition of web services.

5. Testing Web Services

Testing web services is more challenging compared to traditional systems for two primary reasons; the complex nature of web services and the limitations that occur due to the nature of SOA. It has been argued [7] that the distributed nature of web services based on multiple protocols such as UDDI and SOAP, together with the limited system information provided with WSDL specifications, makes web service testing challenging.

6. Test Case Prioritization

The purpose of test case prioritization is to increase the likelihood that if the test cases are used for regression testing in the given order, they will more closely meet some objective than they would if they were executed in some other order. Test cases can be prioritized in terms of the number of statements, basic blocks, or methods they executed on a previous version of the software. A second way in which prioritization techniques can be distinguished involves the use of "feedback". A third way in which prioritization techniques can be distinguished involves their use of information about code modifications. Most prioritization techniques proposed to date focus on increasing the rate of fault detection of a prioritized test suite. To measure rate of fault detection we use a metric, APFD (Average Percentage Faults Detected), introduced for this purpose in [8], which measures the weighted average of the percentage of faults detected over the life of a test suite. APFD values range from 0 to 100; higher numbers imply faster (better) fault detection rates. More

formally, let T be a test suite containing n test cases, and let F be a set of m faults revealed by T. Let TF_i be the first test case in ordering T' of T that reveals fault i.

7. Average Percentage Faults Detected (APFD) Metric

To quantify the goal of increasing a subset of the test suite's rate of fault detection, I use a metric called APFD developed by Elbaum et al. [9,10] that measures the average rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the number of faults detected during the run of the test suite. APFD can be calculated using a notation:

Let T → The test suite under evaluation

m → the number of faults contained in the program under test

n → The total number of test cases

TF_i → The position of the first test in T that exposes fault i.

$$APFD = \frac{1}{nm} (TF_1 + TF_2 + \dots + TF_m + \frac{1}{2}n)$$

So as the formula for APFD shows that calculating APFD is only possible when prior knowledge of faults is available. The APFD metric relies on two assumptions: (1) all faults have equal costs. (2) all test cases have equal costs. These assumptions are manifested in the fact that the metric plots the percentage of faults detected against the percentage of the test suite run. In practice, however, there are cases in which these assumptions do not hold: cases in which faults vary in severity and test cases vary in cost. In such cases, the APFD metric can provide unsatisfactory results, necessitating a new approach to test case prioritization that is "cognizant" of these varying test costs and fault severities.

8. Limitations of the APFD Metric

Consider the following four scenarios of cases in which the assumptions of equal test costs and fault severities are not met.

Table 1: Example Test Suite and Faults Exposed

Test	Fault									
	1	2	3	4	5	6	7	8	9	10
A	X				X					
B						X	X			
C	X	X	X	X	X	X	X			
D					X					
E								X	X	X

Example 1. Under the APFD metric, when all ten faults are equally severe and all five test cases are equally costly, orders A-B-C-D-E and B-A-C-D-E are equivalent in terms of rate of fault detection; swapping A and B alters the rate at which particular faults are detected, but not the overall rates of fault detection. This equivalence would be rejected in equivalent APFD graphs (as in Figure 7A) and equivalent APFD values

(50%). Suppose, however, that B is twice as costly as A, requiring two hours to execute where A requires one. In terms of faults-detected-per-hour, test case order A-B-C-D-E is preferable to order B-A-C-D-E, resulting in faster detection of faults. The APFD metric, however, does not distinguish between the two orders.

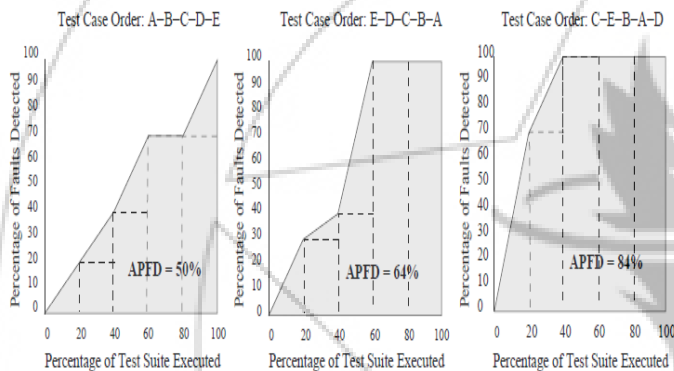
Example 2. Suppose that all five test cases have equivalent costs, and suppose that faults 2-10 have severity k, while fault 1 has severity 2k. In this case, test case A detects this more severe fault along with one less severe fault, whereas test case B detects only two less severe faults. In terms of fault-severity detected, test case order A-B-C-D-E is preferable to order B-A-C-D-E. Again, the APFD graphs and values would not distinguish between these two orders.

Example 3. Examples 1 and 2 provide scenarios in which the APFD metric proclaims two test case orders equivalent when intuition says they are not. It is also possible, when test costs or fault severities differ, for the APFD metric to assign a higher value to a test case order that would be considered less valuable. Suppose that all ten faults are equally severe, and that test cases A, B, D, and E each require one hour executing, but test case C requires ten hours. Consider test case order C-E-B-A-D. Under the APFD metric, this order is assigned an APFD value of 84% (see Figure 7C). Consider alternative test case order E-C-B-A-D. The APFD for this order is 76% lower than the APFD for test case order C-E-B-A-D. However, in terms of faults-detected-per-hour, the second order (E-C-B-A-D) is preferable: it detects 3 faults in the first hour, and remains better in terms of faults-detected-per-hour than the first order up through the end of execution of the second test case. An analogous example can be created by varying fault severities while holding test costs uniform.

Example 4. Finally, consider an example in which both fault severities and test costs vary. Suppose that test case B is twice as costly as test case A, requiring two hours to execute where A requires one. In this case, in Example 1, assuming that all ten faults were equally severe, test case order A-B-C-D-E was preferable. However, if the faults detected by B are more costly than the faults detected by A, order B-A-C-D-E may be preferable. For example, suppose test case A has cost "1", and test case B has cost "2". If faults 1 and 5 (the faults detected by A) are assigned severity "1", and faults 6 and 7 (the faults detected by B) are assigned severities greater than "2", then order B-A-C-D-E achieves greater "units-of fault-severity-detected-per-unit-test-cost" than does order A-B-C-D-E. Again, the APFD metric would not make this distinction. Consider an example program with 10 faults and a suite of five test cases, A through E, with fault detecting abilities as shown in Table 3. Suppose the test cases are placed in order A-B-C-D-E to form a prioritized test suite T1. Figure 7A shows the percentage of detected faults versus the fraction of T1 used. After running test case A, two of the 10 faults are detected; thus 20% of the faults have been detected after 20% of T1 has been used. After running test case B, two more faults are detected and thus 40% of the faults have been detected after 40% of the test suite has been used. In Figure 7A, the area inside the inscribed rectangles (dashed boxes) represents the weighted percentage of faults detected over the corresponding percentage of the test suite. The solid lines connecting the corners of the rectangles delimit the area

representing the gain in percentage of detected faults. The area under the curve represents the weighted average of the percentage of faults detected over the life of the test suite. This area is the prioritized test suite's average percentage faults detected metric (APFD); the APFD is 50% in this example.

The notion that a tradeoff exists between the costs of testing and the costs of leaving undetected faults in software is fundamental in practice and testers face and make decisions about this tradeoff frequently. It is thus appropriate that this tradeoff be considered when prioritizing test cases, and so a metric for evaluating test case orders should accommodate the factors underlying this tradeoff. There is such a metric by adapting the APFD metric; the new "cost-cognizant" metric is named APFD_C. In terms of the graphs used in Figure 7 the creation of this new metric entails two modifications. First, instead of letting the horizontal axis denotes "Percentage of Test Suite Executed", the horizontal axis denotes "Percentage Total Test Case Cost Incurred".



A. APFD for prioritized suite T1 B. APFD for prioritized suite T2 C. APFD for prioritized suite T3
Figure 7: Example test case orderings illustrating the APFD metric

Now, each test case in the test suite is represented by an interval along the horizontal axis, with length proportional to the percentage of total test suite cost accounted for by that test case. Second, instead of letting the vertical axis in such a graph denotes "Percentage of Faults Detected", the vertical axis denotes "Percentage Total Fault Severity Detected". Now, each fault detected by the test suite is represented by an interval along the vertical axis, with height proportional to the percentage of total fault severity for which that fault accounts.

9. Conclusion

Assuring the quality of Web services has become increasingly more important. Organizations which depend on Web services to fulfill their business process needs must verify that those needs are being met even as the business processes evolve especially for mission critical systems such as those which directly involve customers. Therefore, regression test selection techniques will become increasingly important to any enterprise seeking to ensure that their services remain of the highest quality. A framework was developed to perform regression test selection and regression testing for the verification of Web services based on the proposed approach which is safe, distributed, automated, end-to-end, and handles

the composability and interoperability aspects of Web services.

References

- [1] Kreger, H., et al, Web Services Conceptual Architecture: WSCA 1.0, <http://www306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, May 2001.
- [2] Bray, T., et al, Extensible Markup Language (XML) 1.0, W3C Recommendation, <http://www.w3.org/TR/2004/REC-xml-20040204/>, Feb. 2004.
- [3] Gudgin, M., et al, Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extensions, <http://www.w3.org/TR/wsdl20-extensions/>, Oct. 2004.
- [4] Clement, L., et al, UDDI Version 3.0.2, UDDI Spec, <http://www.oasis-open.org/committees/302.htm>, Oct. 2004.
- [5] Tsai, W. T., et al, "Scenario-based Modeling and its Applications", Proceedings of the 7th International Workshop on Object-Oriented Real-Time Dependable Systems, (WORDS 2002), pp.253-260, San Diego, CA, Jan. 2002.
- [6] Siblini, R., Mansour, N., "Testing Web Services", Proceedings of the ACS/IEEE Conference on Computer Systems and Applications, pp 135-142, Cairo, Egypt, Jan. 2005.
- [7] Gudgin, M., et al, SOAP Version 1.2 Part 1: Messaging Framework W3C Recommendation, <http://www.w3.org/TR/soap12-part1>, Feb. 2004.
- [8] Pfleeger, S., Software Engineering: Theory and Practice, Second Edition, Prentice Hall, 2001.
- [9] Rothermel, G., and Harrold, M. J., "A Safe, Efficient Regression Test Selection Technique", ACM Transactions on Software Engineering Methodology, vol. 6, no. 2, pp. 173-210, Apr. 1997.
- [10] Rothermel, G., and Harrold, M. J., "Analyzing Regression Test Selection Techniques", IEEE Transactions on Software Engineering, vol.22, no.8, pp.529- 551, Aug. 1996.

Author Profile

Manali Gupta received the B.Tech degree in Information Technology from Uttar Pradesh Technical University and M.Tech degree in Computer Science and Engineering from Amity University, Noida in 2007 and 2013, respectively. Her research interest includes data mining, database management systems, fundamental study of real time operating systems and regression testing.

Shweta Rathour received the B.Tech degree in Information Technology from H.N.B. Garhwal University and M.Tech degree in Computer Science and Engineering from Uttarakhand Technical University in 2009 and 2011, respectively. Her research interest includes network Security and database management system, web technology.