# Test Gruplama ile Zaman Tasarrufu Sağlama*

Mehmet Engin Saçan[1]**, Müjde Ceylan[1], Aylin Kuşku[1], Rainer Heck[2], Michael Koestner[2], and Daniel Schertler[2]

[1] Siemens, AG
Esentepe Mahallesi, Yakacık Yolu No:111, 34870 Kartal/Istanbul, Türkiye
[2] Siemens
Siemensallee 75, 76187 Karlsruhe, Almanya
{engin.sacan; mujde.agra; aylin.kusku;
rainer.heck; koestner.michael; daniel.schertler}@siemens.com

**Özet** Test, yazılım kalitesinin büyük bir parçasıdır. Testin önemi gün geçtikçe artmaktadır. Özellikle büyük uygulamaları test etmek için zaman gerçekten önemlidir. Test mimarisinde, gömülü sistemleri test etmek için takip edilecek birçok teknik vardır. Regresyon testi bunlardan biridir. Regresyon testi, modifiye edilmiş ve düzeltilmiş ortamı yeniden test eden test tekniği türüdür. Bu teknikle, her bir yazılım versiyonuna yeni fonksiyonlar eklenir ve eklenen her fonksiyonu test etmek zaman gerektirir. Bu noktada, yeni eklenen fonksiyonların uygun zaman periyotlarında mevcut fonksiyonları bozmadığından emin olmak için her versiyonda regresyon testinin uygulanması önemlidir.
Bu yazıda regresyon testinin verimliliğini artırmak için yeni teknikler sunulmaktadır. Gömülü sistemlerin regresyon testi için olan yaklaşımımız, zamandan tasarruf etmek ve daha fazla test yapmak için hem test seçim tekniklerini hem de zaman tasarrufu tekniklerini kullanmaktadır. Bu teknikler, test durumlarını benzer özelliklerine göre gruplandırmayı ve test durumlarında gerçekleştirilen yapılandırmayı geri almayı içerir. Sonuç olarak, önemli bir zaman tasarrufu sağlanır. Bu sonuç geliştirmeye ve düzeltmeye yüksek fayda sağlar.

**AnahtarKelimeler:** Regresyon Testi, Yazılım Kalitesi, Test Seçim Teknikleri, Zaman Tasarrufu Teknikleri

* Supported by organization Siemens.
** Corresponding Author

# Affects of Grouping Test Cases for Time Saving*

Mehmet Engin Saçan[1]**, Müjde Ceylan[1], Aylin Kuşku[1], Rainer Heck[2], Michael Koestner[2], and Daniel Schertler[2]

[1] Siemens, AG
Esentepe Mahallesi, Yakacık Yolu No:111, 34870 Kartal/Istanbul, Turkey
[2] Siemens
Siemensallee 75, 76187 Karlsruhe, Almanya
{engin.sacan; mujde.agra; aylin.kusku;
rainer.heck; koestner.michael; daniel.schertler}@siemens.com

**Abstract.** Testing is a big part of software quality.The importance of testing is increasing day by day. Time is really important especially for testing big applications. In a test architecture there are many techniques to follow for testing embedded systems. Regression testing is one of them. Regression test is the type of testing technique, which is retesting the modified and corrected environment. With this technique, new functions are added to each software version and every function requires time for testing each one. At this point it will be important to apply the regression test in each version to make sure that the newly added functions do not corrupt existing functions at appropriate time periods.
This paper presents new techniques to increase the efficiency of regression testing. Our approach which is for regression testing of embedded systems uses both test selection techniques and time saving techniques to save time and run more tests. These techniques include grouping test cases based on their similar features and undo configuration which is performed in test cases. As a consequence, significant time saving has been achieved.

**Keywords:** Regression Testing · Software Quality · Test Selection Techniques · Time Saving Techniques

* Supported by organization Siemens.
** Corresponding Author

## 1 INTRODUCTION

Software development lifecycle(SDLC) is the process which includes all phases of software development from the beginning to end like planning, analysis, design, production and maintenance [1]. Defined process makes software development more orderly and foreseeable. There are many different types of SDLC models like waterfall, agile, spiral etc. All of these models should follow 6 steps to build a good software. These steps are requirement analysis and gathering, system analysis, system design, coding, testing and deployment.

In testing step of SDLC, defects and problems are found and testers inform developers with details of the issue. If it is a valid defect which meets a set of confirmed defect definitions by the development team, the developer will fix defect and create a new software version for retesting. All defects are fixed until product reaches specified requirements and quality measurements [2].

Software regression testing is one of the types of testing levels. In software, a regression testing is performed after a new feature is implemented and finds if this new implementation causes unexpected results. It is necessary to perform regression testing after version change is made in software, not just to find defects. Because these changes made in system may have distorted the structure. Thus, it is tested whether the functions lose their functionality or are harmed by the newly added functions.When enough time is allocated for regression testing, project can be tracked more closely and the success of the project is ensured [3,4,5].

In today's world, large number of embedded applications are used in industrial communication area. Their size and sophistication are increasing as a result of their wide usage. So that, effective regression testing of embedded applications have great significance[6,7].

Regression testing can be used for each level of software testing. It is used in integration part of our software development. After build is released, automated regression testing starts and validates that unchanged parts of code are not affected by the code changes. Plenty of time is spent setting up the test environment, analyzing test results, adding new features to test automation system and dealing with the environmental issues. In this approach, maintenance effort is decreased for existing test suite and more effort can be used for new test cases. These all test cases will be finished earlier and results will be covered faster. This result, not only improves the suitability of the developments, but also it can provide great contributions to software quality [8,9,10].

# 2 TEORETICAL BACKGROUND

## 2.1 INTEGRATION TESTING

There are three test levels as the White-box, the Black-box and the Grey-box testings. Integration test is related with the Grey-box testing. Software mostly consists of many modules. Integration testing shows how different modules of software product work together.

## 2.2 REGRESSION TESTING

Regression testing is a type of testing ensures that changed or updated code has not broken any functionality of software. Tests of existing functionalities have been running again to be sure that they have not damaged anything. Regression testing can be done at any type of testing (e.g., unit, integration, system, acceptance etc.)

Regression testing is crucial for software development lifecycle; at the same time it has some major challenges. Since it involves running all old tests, it can consume much time comparibly other type of tests while running. It can be complicated to maintain big amount of tests. Moreover, when the number of tests increases, designing regression test environment is a hard task to achieve.

Test automation is an efficient way of running regression tests since manually running same tests repeatedly can make testers bored and less motivated. Moreover, updating tests regularly is a good way of running regression tests since it can be more difficult and complex later on. Furthermore, reviewing test design and test environment periodically to fix errors and make improvements increase efficiency of testing.

In our testing module, we applied regression test into integration test which means early feedbacks are given by daily builds.

## 2.3 QUALITY IN TESTING

According to ISO 9126-1 [ISO/IEC, 2011], while evaluating the quality of software products, internal, external, quality in use and process attributes are measured.

- Internal quality of a product is measured with the internal quality requirements. Internal quality of a product stays stable except design of product is not changed.
- External quality of a product is measured with the testing of software after it is execution. During the testing period most errors can be found and fixed.

Process quality helps improving product quality and product quality helps improving quality in use. External and internal quality of software product can be classified into characteristics which are functionality, reliability, usability, efficieny, maintainability, portability[11].

Regression testing is not only about the suitability of the developments, it can also provide great contributions to software quality like reducing time consuming and getting quick feedbacks.

## 2.4 REGRESSION TESTING TECHNIQUES

In the software development life cycle, maintenance is the most crucial phase of the software delivered to the clients[3]. Software maintenance results are like error recovery, extension or deletion of capabilities, and optimization of the system. Regression testing is defined[12] as "the process of retesting the modified parts of the software and ensuring that no new errors have been introduced into previously tested code". There are several regression testing techniques exist;

- Retest all
- Regression Test Selection
- Test Case Prioritization
- Hybrid Approach

Definitons of these techniques are given below:

### 2.4.1 Retest All

Retest all is a method for regression testing that all the tests are rerun in the existing test suite. So the retest all method is much more expensive than other regression testing methods which requires more time and budget[13,14] .

### 2.4.2 Test Case Prioritization

This technique of regression testing prioritizes the test cases so as to increase a test suite's rate of fault detection that is how quickly a test suite detects faults in the modified program to increase reliability.This is classified as general prioritization and version specific prioritization. The meaning of general prioritization is to select a sort of test cases for keeping the test coverage for every versions of software orderly whereas the meaning of version specific prioritization does concern a specific version of the software[13,15,16].

### 2.4.3 Hybrid Approach

There are many Hybrid Approaches and different algorithms are proposed by researchers for both Regression Test Selection and Test Case Prioritization.

1. Leon and Podgurski proposed a new hybrid approach which is combining coverage-based and distribution based prioritisations. It is based on observing the effects of basic coverage maximisation and repeated coverage maximisation. Distribution based filtering techniques are more effective methods to reveal errors according to coverage-based filtering techniques. These two techniques are complementary in terms of finding different defects [17].
2. Wong et al developed hybrid technique which combines modification, minimization and prioritization-based selection using software changes and previous versions of test history [4].
3. Varsha and Babita proposed [18] , a new hybrid approach for regression test case prioritization based on source code coverage, branch coverage and mandatory user requirement coverage.

4. Silva et al [19], introduced a hybrid approach for the test case prioritization and selection. Instead of running a full test, they proposed to create different test suite alternatives classified as Prioritization, Selection and Minimization techniques with existing test cases that run in a shorter time.

#### 2.4.4 Regression Test Selection

In regression test selection technique, tests are selected that are deemed necessary from the existing test suite to verify the validity of the modified software. To speed up and increase efficiency of regression testing there are many algorithms [8,20,21,22,23,24,25,26,27,28,29,30] have been proposed related to regression test selection, but it is not possible to compare and evaluate them because the goal of each of them is different [28]. Re-running of full regression test suites every day can be extremely time consuming. This paper outlines grouping the tests and re-running these selected test groups in existing test suite with implemented time-saving methods. This approach not only provides time saving but also reduces the cost of regression testing.

## 2.5 REGRESSION TESTING PROCESS COST ANALYSIS

Regression testing is a repeating and growing process so that especially for large software programs it can be complex and costly in progress of time. Testers spend time for the following steps in a regression test process:

– Testers develop new test cases to test new functionalities and requirements added to software. It sometimes takes weeks to add one test to regression test suite.
– Testers spend time in the execution of test suite. While software is becoming larger, this costs more time and effort. Test automation is a good way to decrease effort for this step. Moreover, test automation can be executed in not working hours.
– Testers investigate test results from execution. Investigating failures manually may take much time. Failure may be related with both software development and software testing.
– Testers track failures until they are fixed. This step related with issues such as the affect of failure, the experience of developer and it can be time consuming.
– Testers sometimes have to execute all test suites and validate all functionality of software. Execution and investigation of results can take long time.

Since regression testing is a time consuming process, there are many techniques to decrease development and maintenance part of regression testing [5,31,32].

In the literture, there are many studies to increase efficiency of regression testing in terms of time, memory and cost. Data flow technique has been used to specify tests for retesting after a change [33,34,35,36]. Data flow technique finds relations which change affects and select all tests to cover these relations [8]. Semantic differencing technique uses system dependence graphs instead of data flow graphs to determine semantic differences between the code before and

after a change. Dependency graphs not only decrease the number of tests to rerun and but also make code less complex by ignoring redundant relations [37]. History-based technique is based on the old test execution data. Test execution history is analyzed to find tests to re-run [38]. Slicing-based technique proposes a slicing algorithm to decrease memory and time consume. This algorithm finds all associations affected by the change or modification in a program without using all data flow history. Partial data flow is enough for implementing algorithm [39].
Unlike previous studies, our approach uses both test techniques and time saving method to enhance efficiency of regression system.

## 3 RESULTS AND DISCUSSIONS

The testing process begins in parallel with the requirement analysis in the V-model software development projects. Testing is a continuous activity throughout the software development process. Each software process has a corresponding test process. For this reason, the quality of test is as important as quality of software development process.

In our testing module, we applied regression test into integration test which means early feedbacks are given by daily builds. Re-running of full regression test suites every day can be extremely time consuming. This paper outlines grouping the tests and re-running these selected test groups in existing test suite with implemented time-saving methods. This approach not only provides time saving but also reduces the cost of regression testing. Unlike previous studies, our approach uses both test techniques and time saving method to enhance efficiency of regression system.

In this study, we propose a test automation system which provides time savings without compromising the quality of the test. In the automation system, there are many scripts that have been automated to run every day. At the beginning of every script, there is a factory restart, which means that the configuration is completely erased so that the next script is not affected by the previous configuration in anyway. On average, a factory restart time lasts about 3-4 minutes, and it covers a considerable amount of time in many automation systems. For a quality testing process, if we get the same quality results in a short period of time, we will save time by removing the factory restart from the scripts. If there is a problem to configure device in scripts, we use again factory restart to avoid configuration failures. That is why factory restart could not be removed in automation system.

At this point, scripts which are related with the same content are combined with an .xml file in test automation. Each feature has a `"group_order"` node which is related to the `"inside_group_order"` node. These two nodes provide us to list our test scripts and groups. When the “id” of each group changes, automation performs factory restart to device. These steps are given below.

```
<feature1>
<test> test_1</test>
...
<group_order> 1</group_order>
<inside_group_order>1</inside_group_order>
...
<test> test_2</test>
...
<group_order> 1</group_order>
<inside_group_order>2</inside_group_order>
...
</feature1>
<feature2>
<test> test_1</test>
...
<group_order> 1</group_order>
<inside_group_order>1</inside_group_order>
...
<test> test_2</test>
...
<group_order> 1</group_order>
<inside_group_order>2</inside_group_order>
...
</feature2>
```

Feedback is really important for the development perspective. Test results should be as fast as it can be. It is hard to get test results faster because of these problem. The solution for these problems are grouping them and remove configuration after test script finish and do not perform restart. This grouping gives us around 20% time reduction for running test cases more to control and work on the test stations.(See Figure 1)

Generally, features are tested with unit test after every feature test system should reboot to clear configuration in devices. This helps clean start for device to check this feature behaviour.

Perl is one of the most powerful and practical programming language that can be used in the case of intensive text processing. The Perl programming language works in almost all operating systems. This specialty helps to work in independent environment.That's why we used Perl in test automation system. It starts with reading a XML file and create a run list by order in `"group_order"` node than check `"inside_group_order"` lists that XML and runs test scripts. Automation uses Tool Command Line (TCL) to see the serial connection responses. TCL programing language uses ”expect” extension to get responses from a terminal and sending commands to terminal. Generally, testing approach for test scripts ; even test script failed or passed, test automation send device to factory restart to eliminate configuration failure in new test script. This causes a lot of time loss. during unit tests. Avoiding this problem, automation perform restart

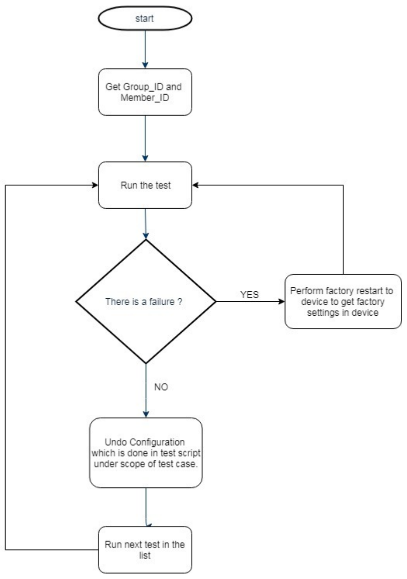
start
Get Group_ID and Member_ID
Run the test
There is a failure ?
YES
Perform factory restart to device to get factory settings in device
NO
Undo Configuration which is done in test script under scope of test case.
Run next test in the list

**Fig. 1.** Our grouping algorithm for time saving.

with factory settings between each group. If there is a failure in the script in same group of tests, factory restart is applied and total run would not be effected by this failure. There could be an alternative for this approach which is:

Never use factory restart and try to undo configuration even test is failed. Continuing all test scripts with this method, could save time on factory restart. The one big problem about this situation is undo the configuration in failed scripts. There should be unexpected configuration change and this will effect all other test scripts.

Another benefit of this method is, we can find more related errors like even undo configuration about one test script would be an effect on other features and this situation decreases our vulnerability(Table 1 and Table 2 ). Device run duration be tested also with our approach with this technique, we will observe whether there is an error in the case of continuous operation.We also find 10 failures in scripts which are in the same group because of undo configuration effect in the firmware.

**Table 1.** Before this implementation test duration time

| Device | Total Test Run | Duration |
|---|---|---|
| X device | 332 | 1 Day 06 hours 04 minutes 50 seconds |

**Table 2.** After this implementation test duration

| Device | Total Test Run | Duration |
|---|---|---|
| X device | 332 | 1 Day 21 minutes 26 seconds |

## 4 CONCLUSION

This paper describes a regression test technique that uses both test selection method and time saving method. Our professional experience demonstrates that this technique helps testers about giving feedback about failures on time and also prove that related failures could be found even in integration part of the testing. This approach significantly outperforms that easy applied time reducing technique with grouping related test cases. Testers could better understand and create better reproduction steps for defects in versions. Undo the configuration for test cases also can be a fundemental technique for our future work like mesuring software version stability without performing the factory settings to device.

## References

1. N. B. Ruparelia, "Software development lifecycle models," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8–13, 2010.
2. Y. Bassil, "A simulation model for the waterfall software development life cycle," *arXiv preprint arXiv:1205.6904*, 2012.
3. G. Duggal and B. Suri, "Understanding regression testing techniques," in *Proceedings of 2nd National Conference on Challenges and Opportunities in Information Technology*, 2008.
4. W. E. Wong, J. R. Horgan, S. London, and H. Agrawal, "A study of effective regression testing in practice," in *Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on*, pp. 264–274, IEEE, 1997.
5. A. K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," *Communications of the ACM*, vol. 41, no. 5, pp. 81–86, 1998.
6. W.-T. Tsai, L. Yu, F. Zhu, and R. Paul, "Rapid embedded system testing using verification patterns," *IEEE software*, vol. 22, no. 4, pp. 68–75, 2005.
7. P. Liggesmeyer and M. Trapp, "Trends in embedded software engineering," *IEEE software*, vol. 26, no. 3, 2009.
8. Y. Chen, R. L. Probert, and D. P. Sims, "Specification-based regression test selection with risk analysis," in *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*, p. 1, IBM Press, 2002.

9. H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *Empirical Software Engineering, 2005. 2005 International Symposium on*, pp. 10–pp, IEEE, 2005.
10. B. Korel, L. H. Tahat, and B. Vaysburg, "Model based regression test reduction using dependence analysis," in *Software Maintenance, 2002. Proceedings. International Conference on*, pp. 214–223, IEEE, 2002.
11. I. O. for Standardization and I. E. Commission, *Software Engineering–Product Quality: Quality model*, vol. 1. ISO/IEC, 2001.
12. K. Aggarwal and Y. Singh, "Software engineering programs documentation, operating procedures," *New Age international publishers*, 2001.
13. S. Kadry, "A new proposed technique to improve software regression testing cost," *arXiv preprint arXiv:1111.5640*, 2011.
14. H. K. Leung and L. White, "Insights into regression testing (software testing)," in *Software Maintenance, 1989., Proceedings., Conference on*, pp. 60–69, IEEE, 1989.
15. N. Dhamija *et al.*, "Test cases prioritization using model based test dependencies: A survey," *International Journal of Innovation and Applied Studies*, vol. 6, no. 2, p. 144, 2014.
16. R. Pradeepa and K. VimalDevi, "Effectiveness of testcase prioritization using apfd metric: Survey," in *IJCA Proceedings on International Conference on Research Trends in Computer Technologies*, pp. 1–4, 2013.
17. D. Leon and A. Podgurski, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases," in *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*, pp. 442–453, IEEE, 2003.
18. V. Kaushik and B. Yadav, "A new approach for regression test case prioritization using branch coverage, code coverage and mandatory requirement coverage,"
19. D. Silva, R. Rabelo, M. Campanha, P. S. Neto, P. A. Oliveira, and R. Britto, "A hybrid approach for test case prioritization and selection," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pp. 4508–4515, IEEE, 2016.
20. R. A. Haraty, N. Mansour, and B. Daou, "Regression testing of database applications," in *Proceedings of the 2001 ACM symposium on Applied computing*, pp. 285–289, ACM, 2001.
21. D. Binkley, "Semantics guided regression test cost reduction," *IEEE Transactions on Software Engineering*, vol. 23, no. 8, pp. 498–516, 1997.
22. D. Willmor and S. M. Embury, "A safe regression test selection technique for database-driven applications," in *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pp. 421–430, IEEE, 2005.
23. G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 6, no. 2, pp. 173–210, 1997.
24. T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel, "An empirical study of regression test selection techniques," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 10, no. 2, pp. 184–208, 2001.
25. 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, pp. 929–948, 2001.
26. M. J. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. A. Spoon, and A. Gujarathi, "Regression test selection for java software," in *ACM SIGPLAN Notices*, vol. 36, pp. 312–326, ACM, 2001.

27. A. Orso, N. Shi, and M. J. Harrold, “Scaling regression testing to large software systems,” in *ACM SIGSOFT Software Engineering Notes*, vol. 29, pp. 241–251, ACM, 2004.
28. G. Rothermel and M. J. Harrold, “Analyzing regression test selection techniques,” *IEEE Transactions on software engineering*, vol. 22, no. 8, pp. 529–551, 1996.
29. S. Bates and S. Horwitz, “Incremental program testing using program dependence graphs,” in *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 384–396, ACM, 1993.
30. G. Rothermel, M. J. Harrold, and J. Dedhia, “Regression test selection for c++ software,” *Software Testing Verification and Reliability*, vol. 10, no. 2, pp. 77–109, 2000.
31. J. A. Jones, M. J. Harrold, and J. Stasko, “Visualization of test information to assist fault localization,” in *Proceedings of the 24th international conference on Software engineering*, pp. 467–477, ACM, 2002.
32. D. S. Rosenblum and E. J. Weyuker, “Using coverage information to predict the cost-effectiveness of regression testing strategies,” *IEEE Transactions on Software Engineering*, vol. 23, no. 3, pp. 146–156, 1997.
33. M. J. Harrold and M. Souffa, “An incremental approach to unit testing during maintenance,” in *Software Maintenance, 1988., Proceedings of the Conference on*, pp. 362–367, IEEE, 1988.
34. M. J. Harrold, “An approach to incremental testing,” 1988.
35. T. J. Ostrand and E. J. Weyuker, “Using dataflow analysis for regression testing,” in *Proceedings of the Sixth Annual Pacific Northwest Software Quality Conference*, pp. 233–47, 1988.
36. A.-B. Taha, S. M. Thebaut, and S.-S. Liu, “An approach to software fault localization and revalidation based on incremental data flow analysis,” in *Computer Software and Applications Conference, 1989. COMPSAC 89., Proceedings of the 13th Annual International*, pp. 527–534, IEEE, 1989.
37. D. Binkley, “Using semantic differencing to reduce the cost of regression testing,” in *Software Maintenance, 1992. Proceerdings., Conference on*, pp. 41–50, IEEE, 1992.
38. J.-M. Kim and A. Porter, “A history-based test prioritization technique for regression testing in resource constrained environments,” in *Proceedings of the 24th international conference on software engineering*, pp. 119–129, ACM, 2002.
39. R. Gupta, M. J. Harrold, and M. L. Soffa, “Program slicing-based regression testing techniques,” *Journal of Software Testing Verification and Reliability*, vol. 6, no. 2, pp. 83–111, 1996.