Department: Reliability Society
Editor: Phil Laplante, pal11@psu.edu

Combinatorial testing for building reliable systems

M S Raunak
NIST

D R Kuhn
NIST

R N Kacker
NIST

J Y Lei
University of Texas at Arlington

***Abstract*—** Combinatorial testing is an approach where test suites are developed by efficiently covering interactions of parameter values and configuration settings. Multiple studies over the years have shown the interesting phenomenon where almost all defects in a system originate from interactions of a few specific parameters or settings. Efficient algorithms compressing these value combinations into a small number of tests have made this method practical for industrial use, providing better testing at lower cost. Through this approach, fault detection close to the level of exhaustive testing can be achieved with a 20x to 700x reduction in the test suite size. Since most defects in systems can be discovered with systematic testing using 2- to 6-way interactions of parameter values, utilizing this approach can help us develop highly reliable systems.

INTRODUCTION Stakeholders of technology-oriented systems expect some level of reliability from them. IEEE defines *Reliability* as "the ability of a system or component to perform its required functions under stated conditions for specified period of time" [10]. One of the crucial factors in determining a system's quality is based on how reliable it is. However, ensuring reliability is hard, especially for systems that are large and complex in nature. In this article, we introduce combinatorial testing and its potential usage in developing highly reliable systems.[1]

Defects, also commonly termed errors or faults are the primary cause for the lack of reliability in a software or hardware system. Although the defect can emanate from misunderstanding of requirements, or from a design flaw, a vast majority of the defects are results of implementation errors. These errors also may lead to security vulnerabilities. Testing is the prevalent mechanism to detect these errors and to remedy them before system is released for wide use. Testing is also the primary way to determine the reliability level of a software and consequently a way to measure the level of assurance one can provide on the system.

While designing a test suite, engineers aim to choose test cases that are most likely to discover potential defects in the system. A large part of testing research is thus dedicated toward identifying, analyzing, and

[1] A portion of this article is adapted from NIST special publication 800-142 [16].

characterizing the defect-discovering ability of the test cases.

Developers of large, complex systems have observed that when usage of an application increases, components that have operated for months without trouble may suddenly develop previously undetected errors. For example, the application may have been installed on a different OS/hardware/DBMS/networking platform, or newly added customers may have account records with unusual combinations of values that have not occurred before. Some of these rare combinations trigger failures that have escaped previous testing and extensive use. Such failures are known as *interaction failures* because they are only exposed when two or more specific input values or operational configuration interact to cause the system to reach a faulty state resulting in a failure.

Combinatorial testing is an approach where input values or configuration settings are enumerated, and all possible 2-way, 3-way, 4-way, or in general *t-way*, interactions of these values and settings are tested systematically and efficiently to ensure its correct and reliable operation. As we will see later in this article, covering all such t-way interactions requires a remarkably smaller test set than one might initially expect. Two major benefits arise from this approach:

1. Efficient arrangement of *t-way* interactions in test cases allow for each test case to cover many interactions. This is done by developing covering arrays, which we will explain in more detail later.
2. Systematic coverage and measurements of input value combinations and configuration settings, which can help provide a quantitative understanding of the reliability of a system.

The key insight underlying *t*-way combinatorial testing is that not every parameter contributes to every failure and most failures are triggered by a single parameter value or interactions between a relatively small number of parameters. To detect these interaction failures, software developers often use “pairwise testing”, in which all possible pairs of parameter values are covered by at least one test.

Example of interaction failure can be found, for example, when a router is observed to fail only for a particular protocol when packet volume exceeds a certain rate, a 2-way interaction between protocol type and packet rate. Figure 1 illustrates how such a 2-way interaction may happen in code. Note that the failure will only be triggered when both *pressure < 10* and *volume > 300* are true.

```
if (pressure < 10) {

    // do something

    if (volume > 300){
        faulty code!  ERROR!
    }
    else {
        good code, no problem.
    }
}
else {
    // do something else
}
```

Figure 1. Failure triggered by the interaction of two parameters.

The Power of *T-way* Testing

Pairwise or 2-way testing proposes selection of test cases such that all pairs of the parameters with specific values are included in the test suite. Applying on some Unix commands and two Bellcore products, Cohen [1] showed that combinatorial approach with pairwise values, i.e., 2-way combinations not only provide good code coverage, but also is effective in discovering defects not found through traditional test methods [2]. In some cases, a defect can only be triggered by higher number of interactions such as a combination of 3, 4, or more parameter values. Pairwise tests would be unlikely to detect those defects. But is testing all 4-way combinations enough to detect all errors? What degree of interaction occurs in real failures in real systems? This question has been studied

empirically with systems in a variety of domain. Interestingly, the studies found that all discoverable failures could be triggered by a maximum of 4-way to 6-way interactions [3, 4, 5, 6]. As shown in Figure 2, the defect detection rate increases rapidly with interaction strength (the interaction level *t* in *t*-way combinations is often referred to as *strength*) and then plateaus. With one set of applications, for example, 67% of the failures were triggered by only a single parameter value, 93% by 2-way combinations, and 98% by 3-way combinations. The detection rate curves for the other applications studied are similar, reaching 100% detection with 4 to 6-way interactions. Studies by other researchers [7, 8, 9] have been consistent with these results.

therefore provide reasonably high assurance about the reliability of the system.

Covering Arrays

Let us illustrate the process of using combinatorial testing through a couple of examples. Consider a word processing application that allow the user to select 10 ways to modify some highlighted text: *subscript, superscript, underline, bold, italic, strikethrough, emboss, shadow, small caps*, or *all caps*. The font-processing function within the application that receives these settings as input must process the input and modify the text on the screen correctly. Most options can be combined, such as bold and small caps, but some are incompatible, such as subscript and superscript.

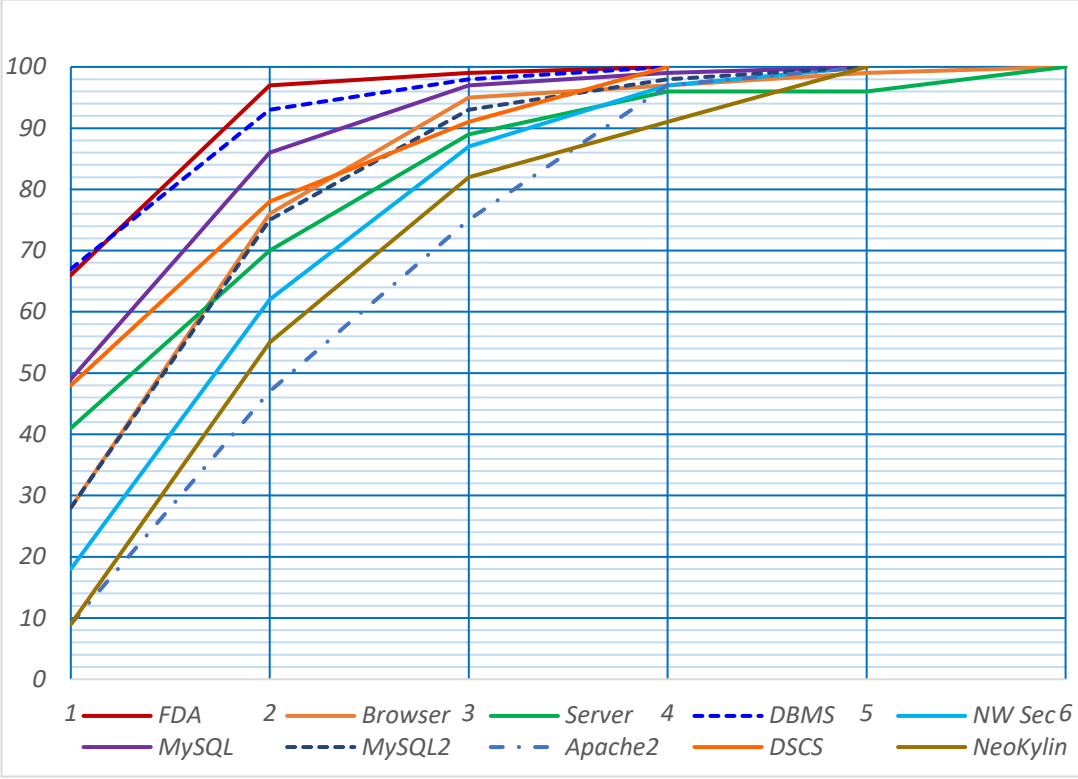

Figure 2. Defect detection rates for interaction strengths 1 though

While not theoretically proven, these results are interesting because they strongly suggest that, while pairwise testing is not sufficient, the degree of interaction involved in failures is relatively low. Testing all 4-way to 6-way combinations may

Thorough testing of this application would require checking that the font-processing function work correctly for all valid combinations of these input settings. With 10 binary inputs, there are 2^{10} = 1,024 possible combinations. We have already

noted that empirical studies have shown that failures appear to involve a small number of parameters, and that testing all 3-way combinations may detect 90% or more of bugs. For a word processing application, testing that detects better than 90% of bugs may be a cost-effective choice, but we need to ensure that all 3-way combinations of values are tested. To do this, we create a test suite to cover all 3-way combinations. This matrix of input value combinations with all 3-way interactions is known as a *covering array* [11, 12,13,14].

Figure 3 shows a 3-way covering array for 10 parameters of 2 values each. The interesting property of this array is that any three columns contain all eight possible values for three binary variables. For example, taking columns H, I, and J, we can see that all eight possible 3-way combinations (000, 001, 010, 011, 100, 101, 110, 111) occur somewhere in the three columns together. In fact, any combination of three columns chosen in any order will also contain all eight possible values. Thus, this set of tests will collectively exercise all 3-way combinations of input values in **only 13 tests, as compared with 1,024 for exhaustive coverage**.

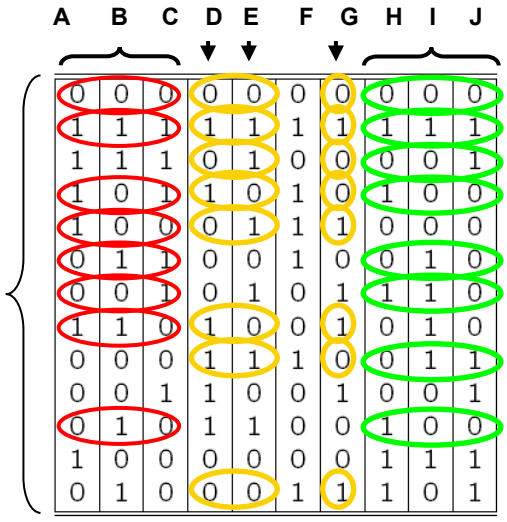

Fig. 3. *3-way* covering array of 10 variables

Similar arrays can be generated to cover up to all 6-way combinations. In general, the number of *t*-way combinatorial test cases that will be required is proportional to $\boldsymbol{v^t \log n}$, for *n* parameters with *v* possible values each.

Real World Systems

In the small example above, the advantage over exhaustive testing is not extraordinary. With larger problems, the advantages of combinatorial testing can be spectacular. Consider the problem of testing the software that processes switch settings for a cyber-physical system with 34 switches, which can each be either on or off, for a total of $2^{34} = 1.7 \times 10^{10}$ possible settings. We clearly cannot test 17 billion possible settings, but all 3-way interactions can be tested with only 33 tests, and all 4-way interactions with only 85. This may seem surprising at first, but it results from the fact that every single test case of 34 parameters contains $\binom{34}{3}$ = 5,984 3-way and $\binom{34}{4}$ = 46,376 4-way combinations. Such drastic improvement in the number of tests that need to be performed to achieve high confidence in discovering all the defects, and thus providing a high level of reliability makes combinatorial testing a must-have in reliability engineers' toolbox.

Challenges of Combinatorial Testing

To apply the combinatorial approach toward testing, one has to first design a test suite that covers *t*-way interactions of parameter values or configuration settings. Efficiently generating such combinations is a challenging mathematical problem that has been studied for over a century now. Another inherent challenge is that most parameters are continuous variables which have possible values in a very large range such as $+/- 2^{32}$ or more. These values must be discretized to a few distinct values. There are well known heuristics available for this purpose. A third challenge is to ensure the presence of test oracle, i.e., the problem of determining the correct result that should be

expected from the system under test (SUT) for each set of test inputs. Generating 1,000 test data inputs is of little help if we cannot determine what the SUT should produce as output for each of the 1,000 test cases.

With the exception of test suite generation that covers *t-way* combinations, these challenges are common to all types of software, hardware, and system testing, and a variety of good techniques have been developed for dealing with them. Over the last couple of decades, multiple efficient algorithms [15, 16] have been developed to generate *t-way* covering arrays and to design test suites based on them. Industry use shows that this approach is practical for large, real-world systems [17]. There are also well developed methods for effectively integrating the combinatorial testing approach into the overall quality and reliability assurance process.

CONCLUSION

Empirical data suggest that software failures are caused by the interaction of relatively few parameter values, and that the proportion of failures attributable to *t*-way interactions declines very rapidly with increase in *t*. A single parameter value or a pair of values are often the cause of a failure, but increasingly smaller proportions are caused by 3-way, 4-way, and higher order interactions. Because a small number of parameters are involved in failures, we can attain a high degree of assurance, and thus high confidence about a system's reliability, by testing all *t*-way interactions for an appropriate interaction strength *t* (usually between 2 to 6). The number of *t-way* tests that will be required is proportional to $\boldsymbol{v^t \log n}$, for $\boldsymbol{n}$ parameters with $\boldsymbol{v}$ values each. We have shown how combinatorial methods can be applied toward systematic testing of software system to ensure high reliability.

Combinatorial testing sometimes requires abstracting the possible values of a variable into a small set of equivalence classes. For example, if a variable is a 32-bit integer, it is not feasible to test the full range of values in +/- 2^{31}. Simple heuristics and engineering judgment are required to determine the appropriate portioning of values into equivalence classes, but once this is accomplished it is possible to generate covering arrays of a few hundred to a few thousand tests for many applications. The thoroughness of coverage will depend on resources and criticality of the application.

Over a last couple of decades, a variety of approaches and tools have been developed to make combinatorial testing a practical and effective technique for testing software and hardware systems.

DISCLAIMER: Certain equipment, instruments, software, or materials, commercial or non-commercial, may be identified in this paper in order to specify the concepts described adequately. Such identification is not intended to imply recommendation or endorsement of any product or service by NIST, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

¢ REFERENCES

1. D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton, "The combinatorial design approach to automatic test generation," Software, IEEE, vol. 13, no. 5, pp. 83–88, 9 1996
2. K. Go, S. Kang, J. Baik, and M. Kim, "Pairwise testing for systems with data derived from real-valued variable inputs," Software: Practice and Experience, vol. 46, no. 3, pp. 381–403, 3 2016.
3. D.R. Kuhn and V. Okun, "Pseudo-exhaustive Testing for Software," *Proceedings of 30th NASA/IEEE Software Engineering Workshop*, pp. 153-158, 2006
4. D.R. Kuhn, M.J. Reilly, An Investigation of the Applicability of Design of Experiments to Software Testing, *27th NASA/IEEE Software Engineering Workshop*, NASA Goddard Space Flight Center, 4-6 December, 2002 .
5. D.R. Kuhn, D.R. Wallace, and A. Gallo, "Software Fault Interactions and Implications for Software Testing," *IEEE Transactions on Software Engineering*, 30(6): 418-421, 2004.

6. D.R. Kuhn, R.N. Kacker, J.Y.Lei, "Random vs. Combinatorial Methods for Discrete Event Simulation of a Grid Computer Network", *Proceedings, Mod Sim World 2009*, Oct. 14-17 2009, Virginia Beach, pp. 83-88, NASA CP-2010-216205, National Aeronautics and Space Administration.
7. K. Z. Bell and Mladen A. Vouk. On effectiveness of pairwise methodology for testing network-centric software. *Proceedings of the ITI Third IEEE International Conference on Information & Communications Technology*, pages 221–235, Cairo, Egypt, December 2005.
8. X. Li, R. Gao, W.E. Wong, C. Yang and, D. Li, Applying combinatorial testing in industrial settings. In *2016 IEEE Intl Conf on Software Quality, Reliability and Security (QRS)*.
9. M. Grindal, J. Offutt, S.F. Andler, Combination Testing Strategies: a Survey, *Software Testing, Verification, and Reliability*, v. 15, 2005, pp. 167-199.
10. Institute of Electrical and Electronics Engineers, *IEEE Standard Glossary of Software Engineering Terminology*, ANSI/IEEE Std. 729-1983.
11. D. R. Kuhn, ACTS, https://csrc.nist.gov/acts
12. R. Bryce, A. Rajan, M.P.E. Heimdahl, Interaction Testing in Model Based Development: Effect on Model Coverage*, IEEE, 13th Asia Pacific Software Engineering Conference (APSEC'06)* pp. 259-268.
13. K. Burr and W. Young Combinatorial Test Techniques: Table-Based Automation, Test Generation, and Test Coverage, *International Conference on Software Testing, Analysis, and Review (STAR),* San Diego, CA, Oct. 1998.
14. S. R. Dalal, C. L. Mallows, Factor-covering Designs for Testing Software, *Technometrics*, v. 40, 1998, pp. 234-243.
15. J.Y.Lei, R.N. Kacker, D.R. Kuhn, V. Kuhn, J Lawrence, IPOG/IPOG-D:efficient test generation for multi-way combinatorial testing, *Jou*rnal of Software: Testing, Verification, and Reliability, Nov 2007.
16. D.R. Kuhn, R.N. Kacker, J.Y. Lei, Practical Combinatorial Testing, NIST Special Publication 800-142, 2010
17. J. D. Hagar, T. L. Wissink,, D.R.Kuhn, & R.N. Kacker (2015). Introducing combinatorial testing in a large organization. *Computer*, *48*(4), 64-72.

M S Raunak is a computer scientist at the National Institute of Standards and Technology. He has Ph.D. in computer science from University of Massachusetts Amherst. His research interest includes verification, validation, and assurance of “difficult-to-test” systems. Dr. Raunak is a senior member of IEEE. Contact him/her at raunak@nist.gov.

D R Kuhn is a computer scientist at the National Institute of Standards and Technology. He is one of the pioneering researchers in the area of combinatorial testing. Mr. Kuhn is a fellow of the IEEE. contact him at kuhn@nist.gov.

R N Kacker is a Mathematician at the National Institute of Standards and Technology. He is one of the pioneering researchers in the area of combinatorial testing. Contact him at raghu.kacker@nist.gov.

Jeff Y Lei is a professor of Computer Science and Engineering at the University of Texas Arlington. His research is in the area of automated software analysis, testing and verification, with a current focus on building trustworthy AI-based software systems. Contact him at ylei@cse.uta.edu