Preprint: Kuhn, R. and Raunak, M.S., 2020. Narrow Spectrum Software Testing Addressing Complexity and Trust.
IEEE Computer, 53(4), pp.15-18.

Narrow Spectrum Software Testing

Rick Kuhn (NIST) and M S Raunak (Loyola University Maryland)

Software testing is a delicate combination of art and science. A great deal of sound theory has been developed, but in practice this theory is often ignored. Much testing is ad hoc, especially for consumer level software, where time to market can be the most critical factor in development. In this case, tests are often aimed primarily at demonstrating specific functionality of the application being tested, such as showing that an inventory program accepts and saves records of new products received.

But many software functions are more generic, such as text search, set membership checking, time and date management, and many others. Showing that the software works with more common inputs and configuration fails to test the system for unexpected combinations of inputs, which are more often the causes of failure. For routine functions or common problems, well-developed generic test cases can be used. This is common in security testing, where a variety of specialized methods have been developed to test for buffer overflow, memory leaks, and other common problems. Such focused test ideas go back to the early days of software testing research; other examples include test catalogs for particular checks, such as in [1].

This focused approach, which might be called *narrow spectrum software testing or NSST*, should become more common practice. For example, a string search on a database is a specific functionality that can be tested with a narrow focus to discover a range of failure scenarios from unusual inputs. There are many reasons for NSST beyond the need for better software assurance. For example, much software testing is contracted out, with little to evaluate the thoroughness of assurance beyond basic requirements tracing and demonstration of functions (happy path testing). Structural coverage may be used in some cases, but even this is often limited to the most basic (and minimally effective) metric of statement coverage. Focused, narrow spectrum tests can be one component of improved assurance that products are free of failures. Tests designed using knowledge of different failure types and fault characteristics could be made widely available and used by the community.

A great deal of data on software problems exists and can be used for understanding and organizing problem types. Examples include the Common Weakness Enumeration (CWE) [2], which seeks to describe software flaws generically such that they can be better understood and classified, with the ultimate goal of providing methods to ensure particular weaknesses are not present in software. A more rigorous and systematic approach to this goal is the Bugs Framework [3], which is being developed to provide a precise taxonomy of flaws, a sort of 'periodic table' of software bugs. Knowledge in these data sets can be used to develop more systematic and focused tests, for all types of flaws beyond security weaknesses.

What would such focused, narrow spectrum tests include, and how can their thoroughness of coverage be evaluated? When test coverage is measured, it typically refers to structural coverage (i.e., code coverage), but if we are testing only a narrow range of functionality, it should not be expected that code is exercised outside of a few modules specific to the focus of the test, so other measures are needed. In particular, we need to define an input model for the specific functionality being tested and measure the degree to which it has been covered in tests. The input space model refers to parameters and values tested, and the definition of equivalence classes used to select the test cases. Because rare combinations of input values can trigger failures, coverage measures for the input space should include the combination coverage achieved at each combination level (2-way, 3-way combinations, etc.).

To illustrate such an approach, we developed tests for full text search in a large, heavily used public database [4]. Our objective was to test the search functionality with not only the common search strings,

but also with uncommon combinations of letters, keywords, numbers, and special characters, which may cause the database backed web application to depict a failure scenario. The other aspect of this test case selection was to model the input-fragments and use systematic combinations of them to cover a large portion of the input space. The input model we developed for the text search included five parameters with both keywords and special characters, with 10, 4, 10, 4, and 10 enumerated values for the parameters. This is designated a $4^2 10^3$ input space configuration (two parameters with four values each and three parameters with ten values). Covering all possible input strings would then require 16,000 tests, but all 2-way combinations of the input fragments resulted in only 100 tests, and all 3-way combinations produced 999 tests. The key question in such testing is whether tests have been sufficient. Comparing faults detected at different coverage strengths (2-way, 3-way etc.) showed that 49 faults were discovered with 2-way combinations, but no additional faults occurred with higher strengths of 3-way and 4-way. Using higher strength *t*-way tests provides a reasonable measure of test completeness. When no new faults are discovered with (*t*+2)-way tests, additional faults are unlikely to be found. The text search tests can be applied with little or no change to a variety of systems, since text search is a common function. By using *t*-way factor combinations, we can show the entire input space has been covered, up to suitable combination level.

Combination coverage based testing supplements basic structural coverage based test selection. This provides a sound test engineering method with defensible, quantitative measures of test completeness. We hope to develop more of these narrow spectrum tests and encourage others to do so as well.

Disclaimer: Products may be identified in this document, but identification does not imply recommendation or endorsement by NIST, nor that the products identified are necessarily the best available for the purpose.

References

[1] Marick, B. (1994). *The Craft of Software Testing*, Prentice-Hall, Inc.
[2] Common Weakness Enumeration. https://cwe.mitre.org
[3] https://www.nist.gov/publications/bugs-framework-bf-structured-approach-express-bugs
[4] Raunak, M. S., Kuhn, D. R., & Kacker, R. (2017, July). Combinatorial testing of full text search in Web applications. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (pp. 100-107). IEEE.