

Reviewing 25 Years of Testing Technique Experiments

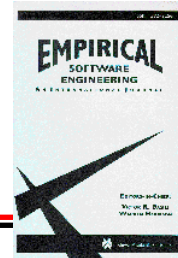
Natalia Juriso

Ana Moreno

Sira Vegas

Universidad Politecnica de Madrid

Empirical Software Engineering
Volume 9, 2004



NC STATE UNIVERSITY

Objectives

- **To compile body of knowledge on testing techniques**
 - Developers can identify applicability conditions of different testing techniques
 - find the most bugs with the least effort, in the least time, with the greatest confidence

- **To provide a picture of what aspects of testing techniques have been tested empirically**
 - Researchers can focus their research

NC STATE UNIVERSITY

Review

- List the black box, white box, and “both” box test strategies we have gone over this semester
- Classify each as black box, white box, or both.

Classification of Testing Techniques

Black Box	White Box
Random Testing	
Functional Testing	
	Control-flow Testing
	Data-flow Testing
	Mutation Testing
Regression Testing	
Improvement Testing Techniques	

Random Testing

- **Pure random:** Test cases generated at random, and generation stops when there appears to be enough.
- **Guided by number of test cases:** Test cases are generated at random, and generation stops when a given number of test cases have been reached.
- **Error guessing:** Test cases are generated guided by the subject's knowledge of what typical errors are usually made when programming. Generation stops when they all appear to have been covered.

Functional Testing

- What strategies do you use to create functional test cases?
- What artifacts do you look at to create functional test cases?

Functional

- **Equivalence partitioning:** A test case is generated for each equivalence class found. The test case is selected at random from within the class.
- **Boundary value analysis:** Several test cases are generated for each equivalence class, one that belongs to in the inside of the class and as many as necessary to cover the limits (or boundaries) of the class.

Control flow Testing

- What strategies do you use to create control flow cases?
- What artifacts do you look at to create control flow test cases?

Control-flow

- **Statement coverage:** The test cases are generated so that all the program statements are executed at least once.
- **Decision coverage (branch coverage):** The test cases are generated so that the program decisions take the value true or false.
- **Condition coverage:** The test cases are generated so that all the program conditions (predicates) that form the logical expression of the decision take the value true or false.
- **Path coverage:** Test cases are generated to execute all program paths. (This criterion is not workable in practice.)

Data flow Testing

- What strategies do you use to create data flow cases?
- What artifacts do you look at to create data flow test cases?

Data-flow

- **All-definitions:** Test cases are generated to cover each definition of each variable for at least one use of the variable.
- **All-c-uses/some-p-uses:** Test cases are generated so that there is at least one path of each variable definition to each c-use of the variable. If there are variable definitions not covered, use p-uses.
- **All-c-uses/some-p-uses:** Test cases are generated so that there is at least one path of each variable definition to each p-use of the variable. If there are variable definitions not covered, use c-uses.
- **All c-uses:** Test cases are generated so that there is at least one path of each variable definition to each c-use of the variable.
- **All p-uses:** Test cases are generated so that there is at least one path of each variable definition to each p-use of the variable.
- **All uses:** Test cases are generated so that there is at least one path of each variable definition to each use (read: p-use and c-use) of the definition.
- **All-du paths:** Test cases are generated for all the possible paths of each definition of each variable to each use of the definition.

Mutation

- **Strong (standard) mutation:** Test cases are generated to cover all the mutants generated by applying all the mutation operators defined for the programming language in question.
- **Selective (or constrained) mutation:** Test cases are generated to cover all of the mutants generated by applying some of the mutation operators defined for the programming language in questions.
- **Weak mutation:** Test cases are generated to cover a given percentage of mutants generated by applying all the mutation operators defined for the programming language in question.

Regression

- **Random:** The test cases are randomly selected from the existing test suite.
- **Retest-all:** Run all the tests in the existing suite.
- **Safe:** The test selection algorithm excludes no test from the original test suite that if executed would reveal faults in the modified program.
- **Based on modifications:** Place an emphasis on selecting existing test cases to cover modified program components and those that may be affected by the modifications.
- **Dataflow/coverage based:** Select tests that exercise data interactions that have been affected by modifications.

Improvement (for all prior strategies)

- **Minimization:** Reduces the number of test cases generated.
- **Prioritization:** Establishes an order in the set of test cases generated so that defects in the program are found as early as possible during the testing process.

General conclusions from studies on *data-flow testing*

- **If time is an issue:**
 - All **p-uses** should be used instead of **all-uses**
 - **All-uses** should be used instead of **all-du-paths**
 - » to generate fewer test cases the generally cover the test cases covered by the (stronger) criteria.
- **All-du-paths is usable in practice.**

- **Limitation: focuses on number of test cases generated – need to look at effectiveness as well.**

General conclusions from studies on *mutation testing*

- **Standard mutation more effective and more costly than mutation variants (selective, weak)**
- **Selective and weak may be suitable for non-critical systems because the effectiveness is practically the same.**

General conclusions from studies on regression testing

- For small programs and set of test cases is small, then retest-all.
- Use safe techniques for large programs and programs with a large number of test cases.

Comparative Studies: Data flow (all uses and all dus), Control Flow (branch), Random

- Time constrained use random; effectiveness similar to all-uses in 50% of the cases.
- When testing needs to be exhaustive, use all-uses.
- Relationship between coverage and effectiveness for all-uses and all-branches but not for random.
- Use branch coverage + all dus because they detect different faults.

Comparisons between Mutation and Data-flow

- **If coverage is important and time is limited, use all-uses rather than mutation because it will be just as effective in half the cases..**
- **Mutation testing appears to be more effective than all-uses, but more costly.**

Comparisons between Functional and Control-flow Testing

- **For experienced testers, use boundary value rather than statement coverage or branch coverage.**
- **For inexperienced testers and when time is an issue, use statement coverage instead of boundary value.**