Intro to Software Testing
chapter 5

Criteria- based Testing

Dr. Brittany Johnson-Matthews
(Dr. B for short)

https://go.gmu.edu/SWE637
Adapted from slides by Jeff Offutt and Bob Kurtz
Changing Notions of Testing

The old (but still useful) view was based on each software development phase being very different from the others.
Changing Notions of Testing

The new view is based on **structures** and **criteria**
- Input space, graphs, logical expressions, syntax

**Test design** is largely the same at each phase
- Creating the model is different
- Choosing test values is different
- Automating the tests may be quite different
Test Coverage

The tester’s job is simple –

define a model of the software, then find ways to cover it

Coverage matters…
Test Coverage Criteria

Test Criterion: A collection of rules and a process that define test requirements

Test Requirements: Specific things that must be satisfied or covered during testing
Model-Driven Test Design

Test Design

- software artifact
  - model / structure
  - criteria give us test requirements

Test Execution
- test requirements
- test requirements
- test results
- test scripts
- test cases
- input values
- pass / fail

Test Automation
- refined requirements / test specs
- design abstraction level

Test Evaluation
- criteria give us test requirements
- software artifact
- model / structure
- test requirements
- test requirements
Test Requirements & Criteria

Testing researchers have defined *dozens of criteria*, but they are all based on four types of structures:

1. Input domains
2. Graphs
3. Logic expressions
4. Syntax descriptions
Sources of Structures

Structures can be extracted from many different artifacts

- **Graphs** can be extracted from UML use cases, source code, finite state machines, etc.
- **Logical expressions** can be extracted from conditions in use cases, decisions in source code, guards on FSE transitions, etc.
Defining Coverage

Given a set of test requirements $TR$ for coverage criterion $C$, a test set $T$ satisfies $C$ coverage if and only if for every test requirement $tr$ in $TR$, there is at least one test $t$ in $T$ such that $t$ satisfies $tr$

DEFINITION

Infeasible test requirements: test requirements that cannot be satisfied

- No test case values exist that meet the test requirement
- Example: dead code
- Detection of infeasible test requirements is formally undecidable for most test criteria
Software cannot be fully tested

⚠️ ERROR

If you're seeing this, the code is in what I thought was an unreachable state.

I could give you advice for what to do, but honestly, why should you trust me?

I clearly screwed this up. I'm writing a message that should never appear, yet I know it will probably appear someday.

On a deep level, I know I'm not up to this task. I'm so sorry.

https://xkcd.com/2200/
Example: Jellybean coverage

Flavors
- Lemon
- Pistachio
- Cantaloupe
- Pear
- Tangerine
- Apricot

Colors
- Yellow
  - lemon, apricot
- Green
  - pistachio
- Orange
  - cantaloupe, tangerine
- White
  - pear

Possible coverage criteria:
- Taste one jelly bean of each flavor
  - Choosing between lemon and apricot is a controllability problem
- Taste one jelly bean of each color
More Jellybean Coverage

T1 = \{ \text{three Lemon, one Pistachio, two Cantaloupe, one Pear, one Tangerine, four Apricot} \}

Does this test set T1 satisfy the flavor criterion?

T2 = \{ \text{one Lemon, two Pistachio, one Pear, three Tangerine} \}

Does test set T2 satisfy the flavor criterion?
Does test set T2 satisfy the color criterion?
Coverage Level

Coverage level is the ratio of the number of test requirements satisfied by $T$ to the size of $TR$.

$T_2 = \{ \text{one Lemon, two Pistachio, one Pear, three Tangerine} \}$

$T_2$ satisfies:

- 4 of 6 test requirements for the flavor criterion, or 67%
- 4 of 4 test requirements for the color criterion, or 100%
Comparing Criteria with Subsumption

A test criterion $C_1$ subsumes $C_2$ if and only if every set of test cases that satisfies criterion $C_1$ also satisfies $C_2$

The subsumption relationship must hold for every set of test cases

The flavor criterion on jelly beans subsumes the color criterion – if we taste every flavor, then we’ve tasted every color (but not vice-versa)

The branch criterion on code subsumes the statement criterion – if we execute every branch, then we’ve executed every statement (but not vice-versa)
Advantages of criteria-based design

Criteria maximize “bang for the buck”
  • Leads to fewer tests that are more effective at finding faults

Comprehensive test sets with minimal overlap

Traceability from software artifacts to tests
  • Answers “why have this test” for every test

Provides a stopping rule for testing, with advance knowledge of how many tests are needed

Natural to automate
Characteristics of good criteria

1. It should be easy to compute test requirements automatically
2. It should be efficient to generate test values
3. Resulting tests should reveal as many faults as possible

Subsumption is a rough but useful approximation of the ability of a criterion to reveal faults