Introduction to Software Testing
(*2nd edition*)
Chapter 5

Criteria-Based Test Design

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

20 September 2015

Changing Notions of Testing

- Old view focused on testing at each software development phase as being very different from other phases
  - Unit, module, integration, system …
- New view is in terms of structures and criteria
  - input space, graphs, logical expressions, syntax
- Test design is largely the same at each phase
  - Creating the model is different
  - Choosing values and automating the tests is different

```
char getLetterGrade( int grade)
{
    If  (grade >=80)
        return ‘B’;
    Else if(grade>= 70)
        return C;
    Else
        return D;
    }
```

New : Test Coverage Criteria

A tester's job is simple : Define a model of the software, then find ways to cover it

- Test Requirements : A specific element of a software artifact that a test case must satisfy or cover

- Coverage Criterion : A rule or collection of rules that impose test requirements on a test set

Source of Structures

- These structures can be extracted from lots of software artifacts
  - Graphs can be extracted from UML use cases, finite state machines, source code, …
  - Logical expressions can be extracted from decisions in program source, guards on transitions, conditionals in use cases, …
- This is not the same as "*model-based testing*," which derives tests from a model that describes some aspects of the system under test
  - The model usually describes part of the behavior
  - The source is explicitly ***not*** considered a model

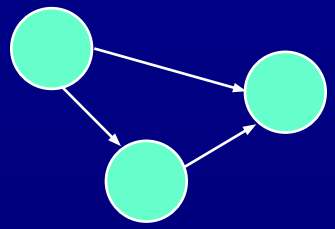
Criteria Based on Structures

Structures : Four ways to model software

1. Input Domain Characterization (sets)

   A: {0, 1, >1}
   B: {600, 700, 800}
   C: {swe, cs, isa, infs}

2. Graphs



3. Logical Expressions

   (not X or not Y) and A and B

4. Syntactic Structures (grammars)

```
if (x > y)
    z = x - y;
else
   z = 2 * x;
```

Example : Jelly Bean Coverage

Flavors :

1. Lemon
2. Pistachio
3. Cantaloupe
4. Pear
5. Tangerine
6. Apricot

Colors :

1. Yellow (Lemon, Apricot)
2. Green (Pistachio)
3. Orange (Cantaloupe, Tangerine)
4. White (Pear)

- Possible coverage criteria :
  1. Taste one jelly bean of each flavor (How many test requirements are there?)
  2. Taste one jelly bean of each color (How many test requirements are there?)

- **Criteria 1:**
- Cover all colors
- How many test requirements? 4 TR
- Criteria 2:
- Cover all flavours
- How many test requirements? 6 TR

- Omar created Test Suite 1: {Orange(Cantalatope), Green (Pistachio), Orange (Tangarine), White (pear)}
- Calculate coverage % for Omar's test suite against Criteria 1: ¾ = 0.75
- Calculate coverage % for Omar's test suite against Criteria 2: 4/6 = 0.66

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*

- Infeasible test requirements : test requirements that cannot be satisfied
  - No test case values exist that meet the test requirements
  - Example: Dead code
- Thus, 100% coverage is impossible in practice

```
if (x > y)
{
    cout<<“Say Hello”;
    if (y > x)
    {
        cout<<“Say Good Bye”;//DEAD code
    }
}
```

More Jelly Beans

T1 = { three Lemons, one Pistachio, two Cantaloupes, one Pear, one Tangerine, four Apricots }

- Does test set T1 satisfy the flavor criterion ?
- Does T1 suffer from redundancy? (**Minimal test sets**)
- Can T1 be minimal?

T2 = { One Lemon, two Pistachios, one Pear, three Tangerines }

- Does test set T2 satisfy the flavor criterion ?
- Does test set T2 satisfy the color criterion ?

Coverage Level

The ratio of the number of test requirements satisfied by *T* to the size of *TR*

- T2 on the previous slide satisfies 4 of 6 test requirements

Two Ways to Use Test Criteria

1. Directly generate test values to satisfy the criterion
   - Often assumed by the research community
   - Most obvious way to use criteria
   - Very hard without automated tools
2. Generate test values externally and measure against the criterion
   - Usually favored by industry
   - Sometimes misleading
   - If tests do not reach 100% coverage, what does that mean?

Test criteria are sometimes called metrics

Generators and Recognizers

- Generator : A procedure that automatically generates values to satisfy a criterion
- Recognizer : A procedure that decides whether a given set of test values satisfies a criterion

- Both problems are provably undecidable for most criteria
- It is possible to recognize whether test cases satisfy a criterion far more often than it is possible to generate tests that satisfy the criterion

- Coverage analysis tools are quite plentiful

Comparing Criteria with Subsumption (5.2)

- Criteria Subsumption : A test criterion *C1* subsumes *C2* if and only if every set of test cases that satisfies criterion *C1* also satisfies *C2*
- Must be true for every set of test cases
- *Examples* :
  - The flavor criterion on jelly beans subsumes the color criterion … if we taste every flavor we taste one of every color
  - If a test set has covered every branch in a program (satisfied the branch criterion), then the test set is guaranteed to also have covered every statement

Criteria Summary

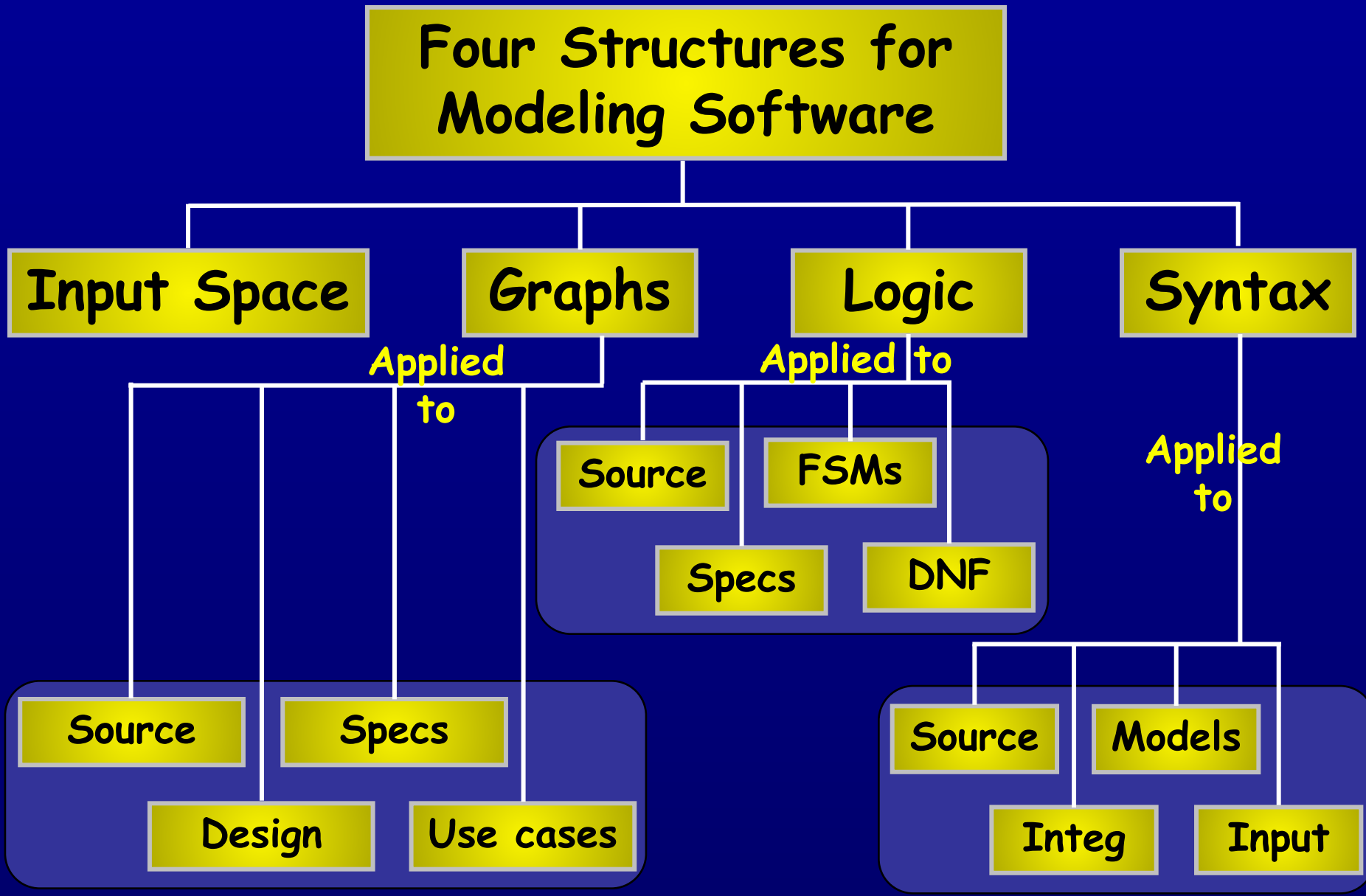
- Many companies still use "monkey testing"
  - A human sits at the keyboard, wiggles the mouse and bangs the keyboard
  - No automation
  - Minimal training required
- Some companies automate human-designed tests
- But companies that use both automation and criteria-based testing

Save money

Find more faults

Build better software

Structures for Criteria-Based Testing



Required Reading

- Chapter 5 from the course's textbook: "Introduction to Software Testing", Cambridge University Press. P. Amman and J. Offutt, Second Edition, 2017.