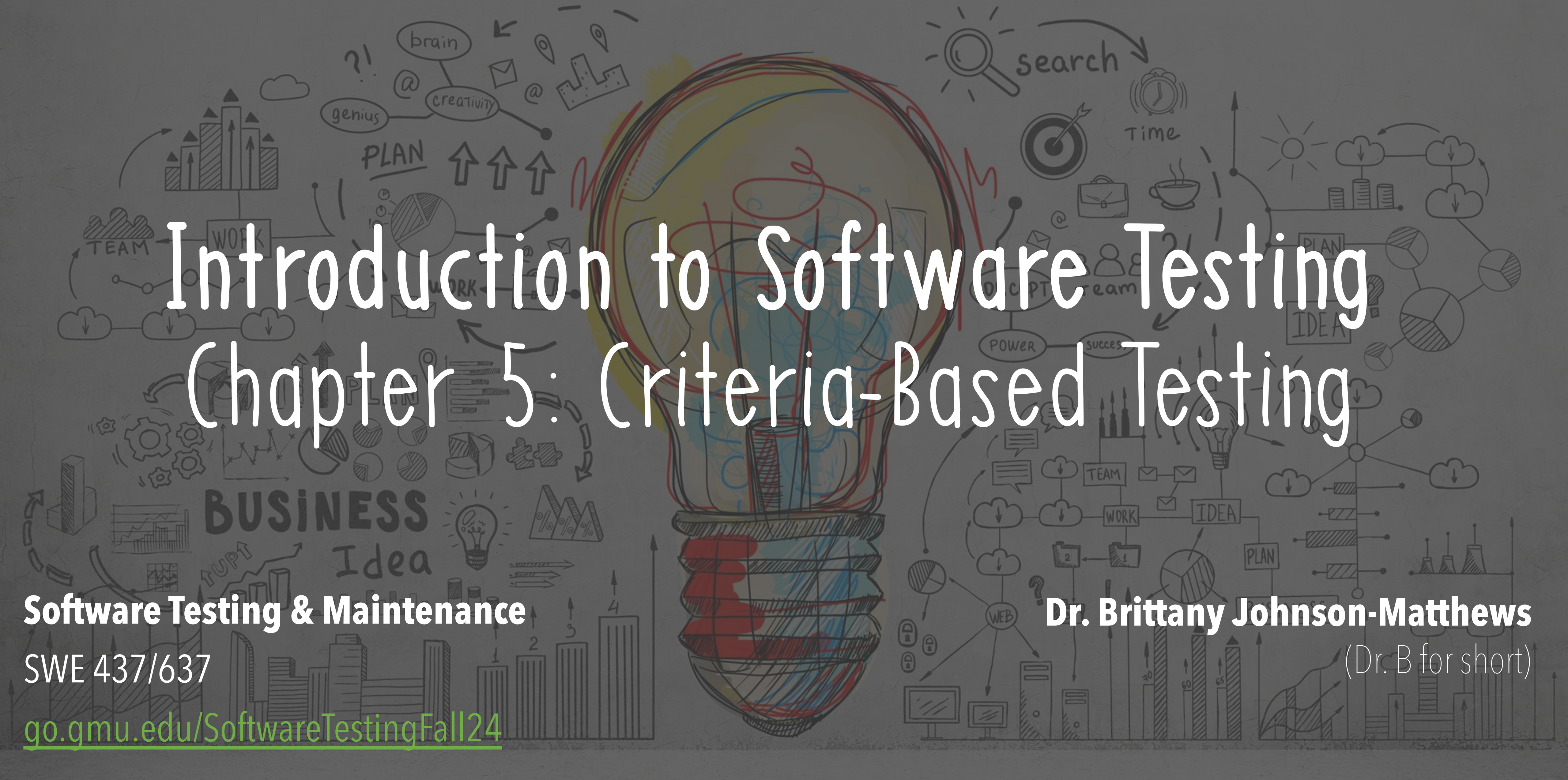


# Introduction to Software Testing
# Chapter 5: Criteria-Based Testing

**Software Testing & Maintenance**

SWE 437/637

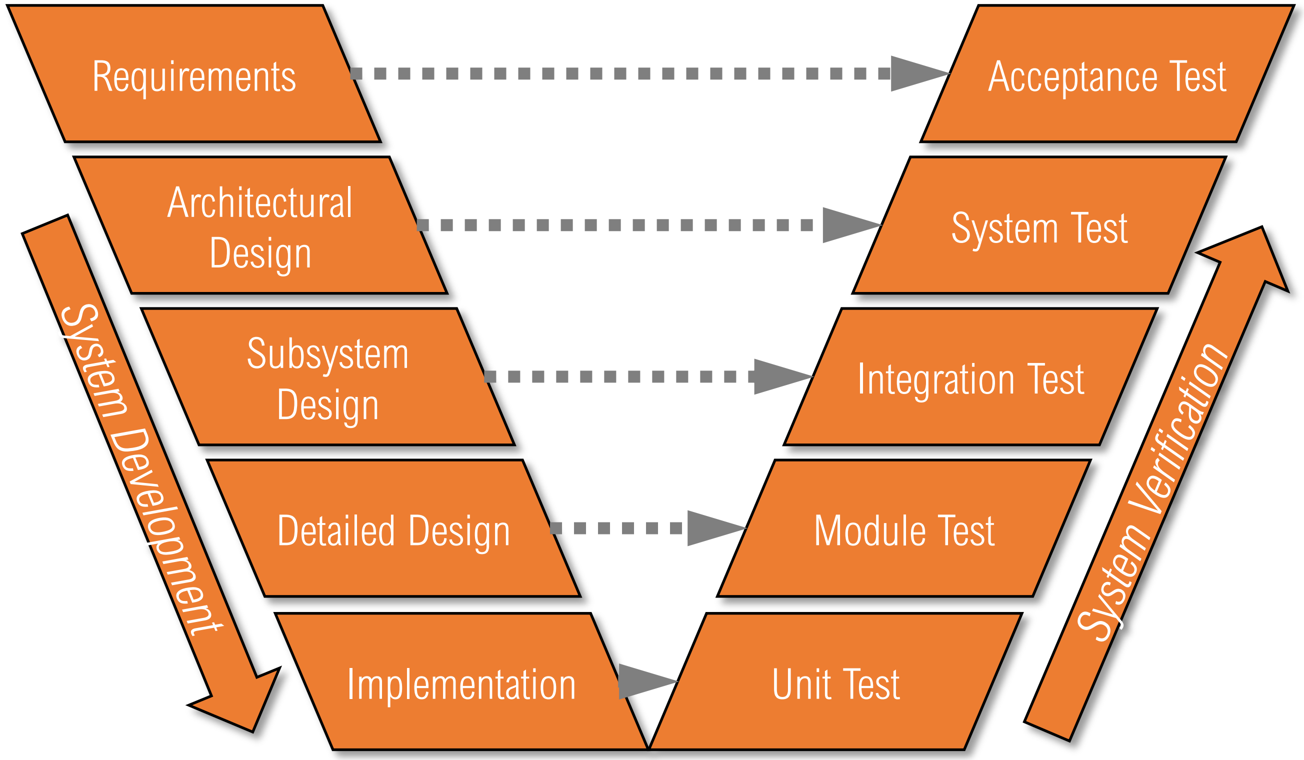
go.gmu.edu/SoftwareTestingFall24

**Dr. Brittany Johnson-Matthews**

(Dr. B for short)

# 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

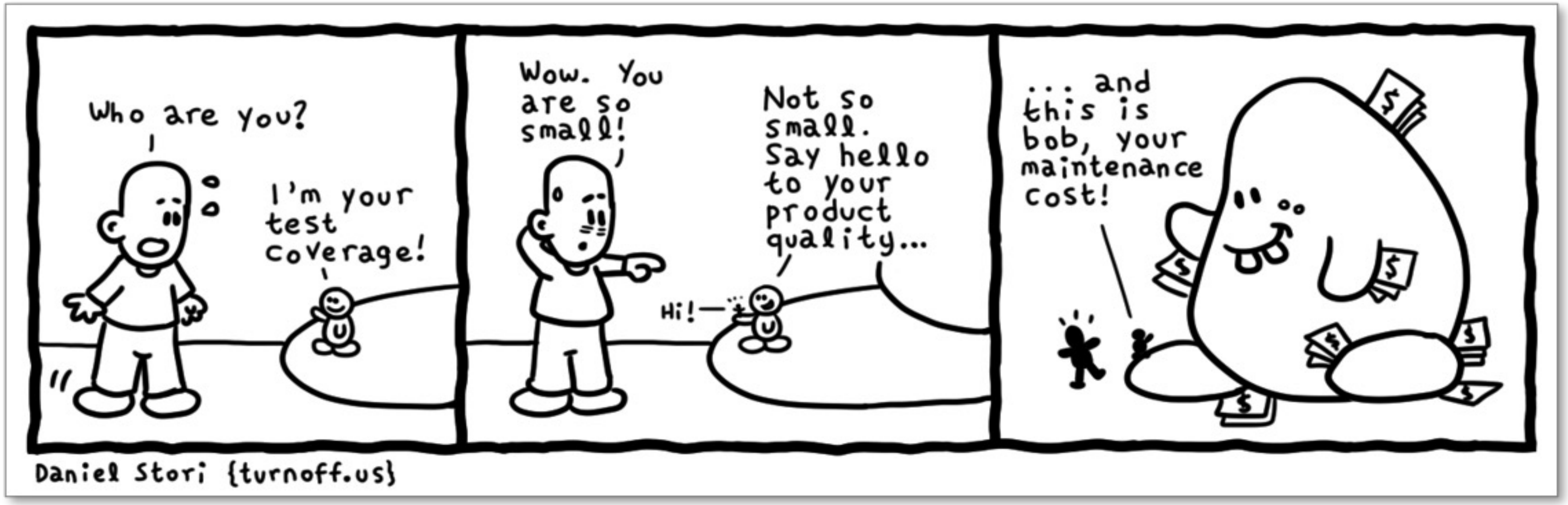
- Creating the model is different
- Choosing test values is different
- Automating the tests may be quite different

# Test Coverage

The tester's job is simple –

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

Coverage matters…



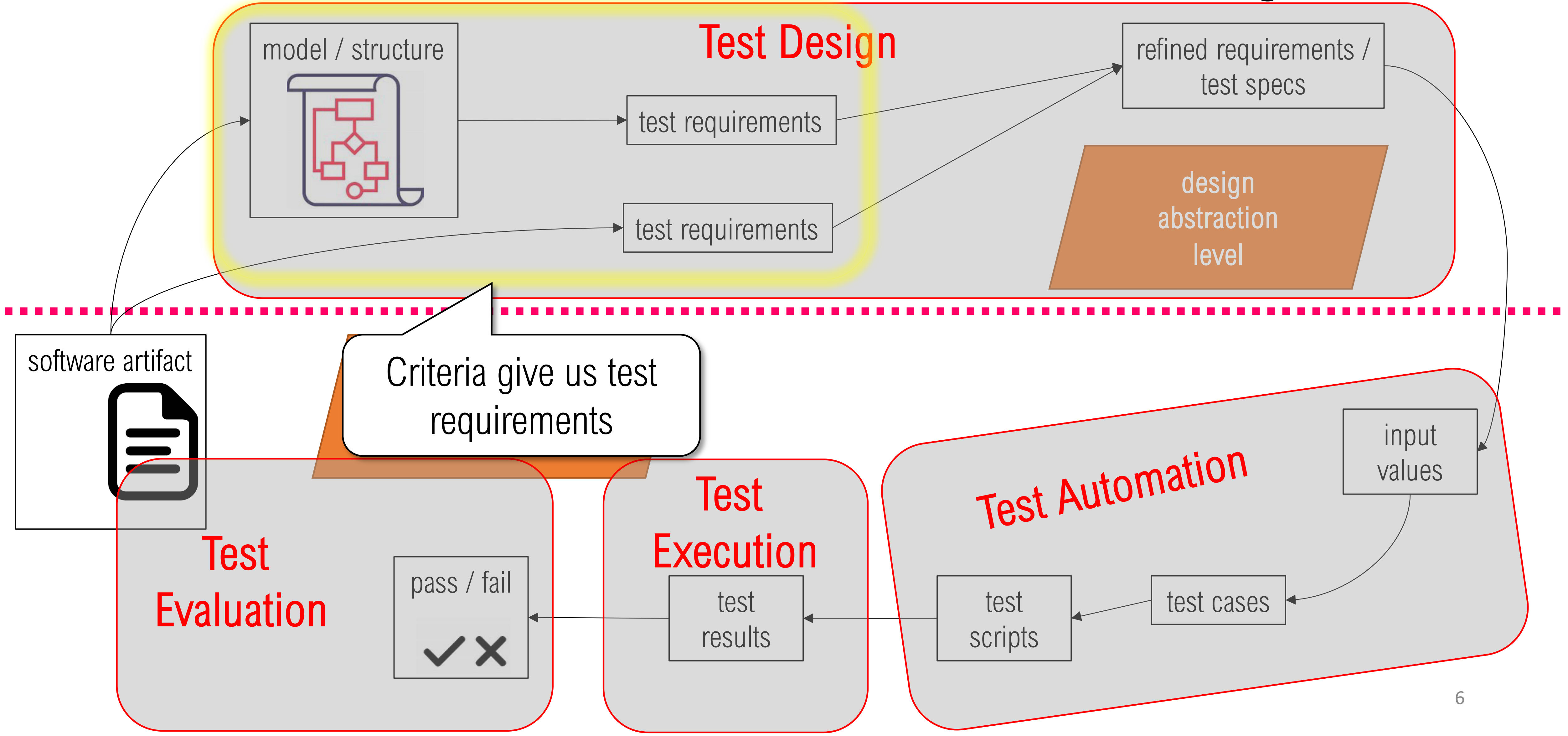
Who are you?
I'm your test coverage!
Wow. You are so small!
Hi!
Not so small. Say hello to your product quality...
... and this is bob, your maintenance cost!
$
Daniel Stori {turnoff.us}

## 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

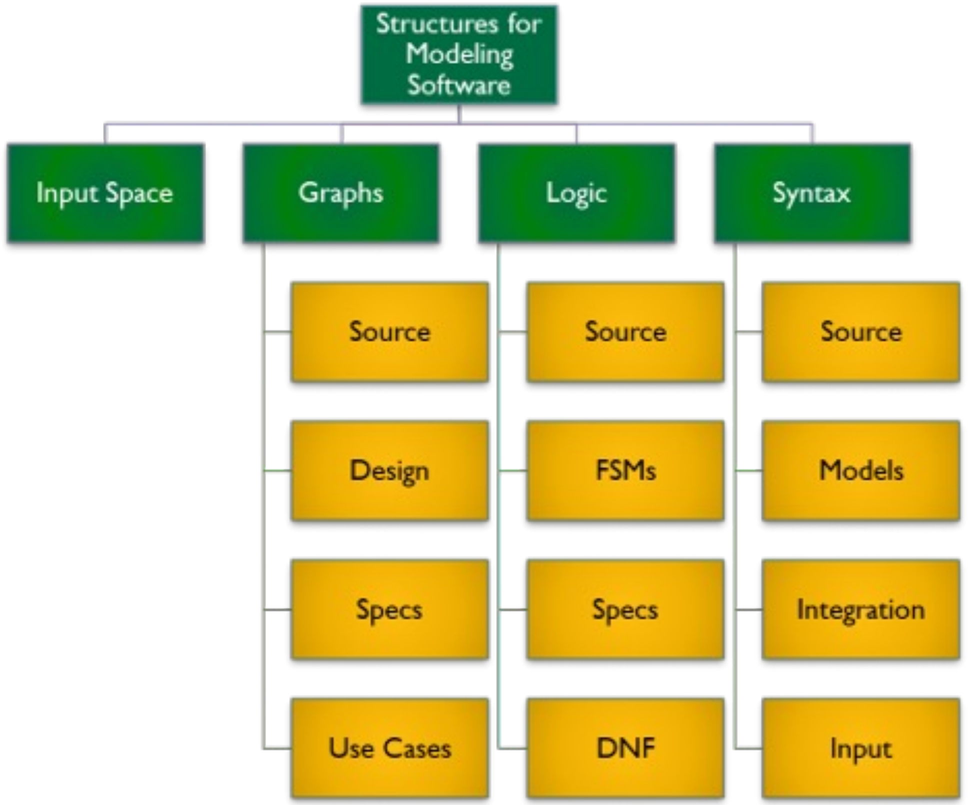
# Model-Driven Test Design



# Test 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 FSM transitions, etc.

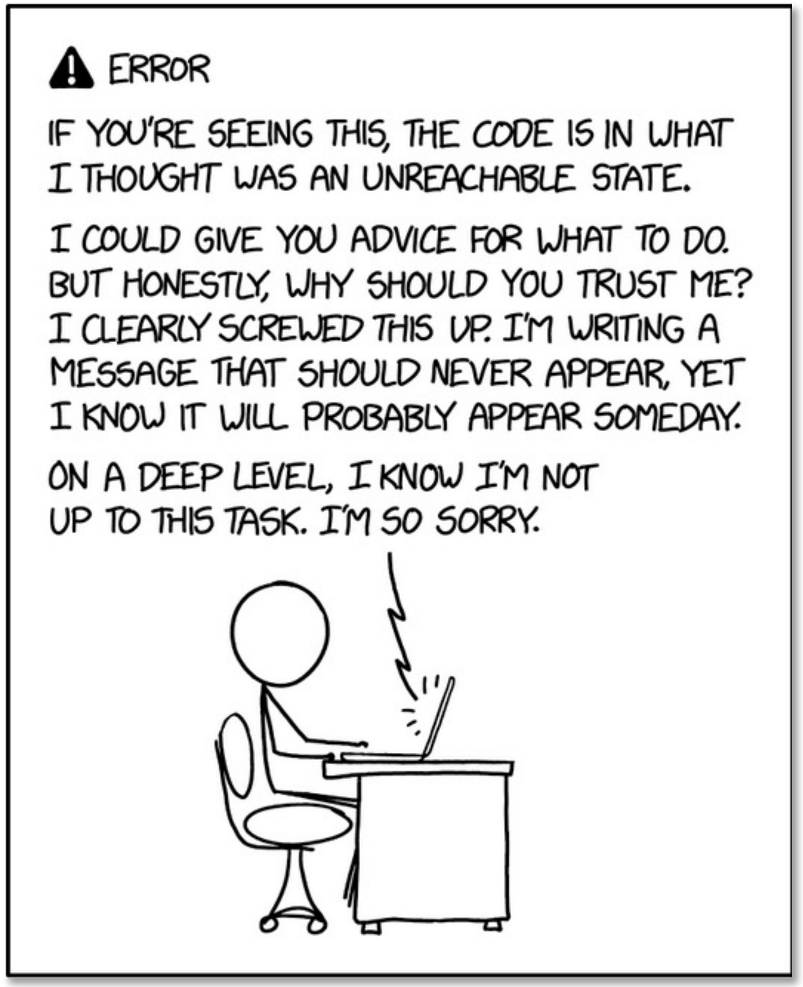
# Defining Coverage

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*

**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 <ins>cannot</ins> 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/

# Jellybean coverage (an example)

| Flavors | Colors |
|---|---|
| Lemon | Yellow<br>*lemon, apricot* |
| Pistachio | Green<br>*pistachio* |
| Cantaloupe | Orange<br>*cantaloupe, tangerine* |
| Pear | White<br>*pear* |
| Tangerine | |
| Apricot | |

## Possible coverage criteria:

- Taste one jelly bean of ***each flavor***
  - Deciding if a yellow jellybean is lemon or apricot is a *controllability* problem
- Taste one jelly bean of ***each color***

# Jellybean coverage (an example)

T1 = ( three Lemon, one Pistachio, two Cantaloupe, one Pear, one Tangerine, four Apricot }

*Does this test set T1 satisfy the* ***flavor criterion****?*

T2 = { 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*

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

T2 satisfies:

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

# Criteria Subsumption

A test criterion *C1* **subsumes** *C2* if and only if every set of test cases that satisfies criterion *C1* also satisfies *C2*

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.