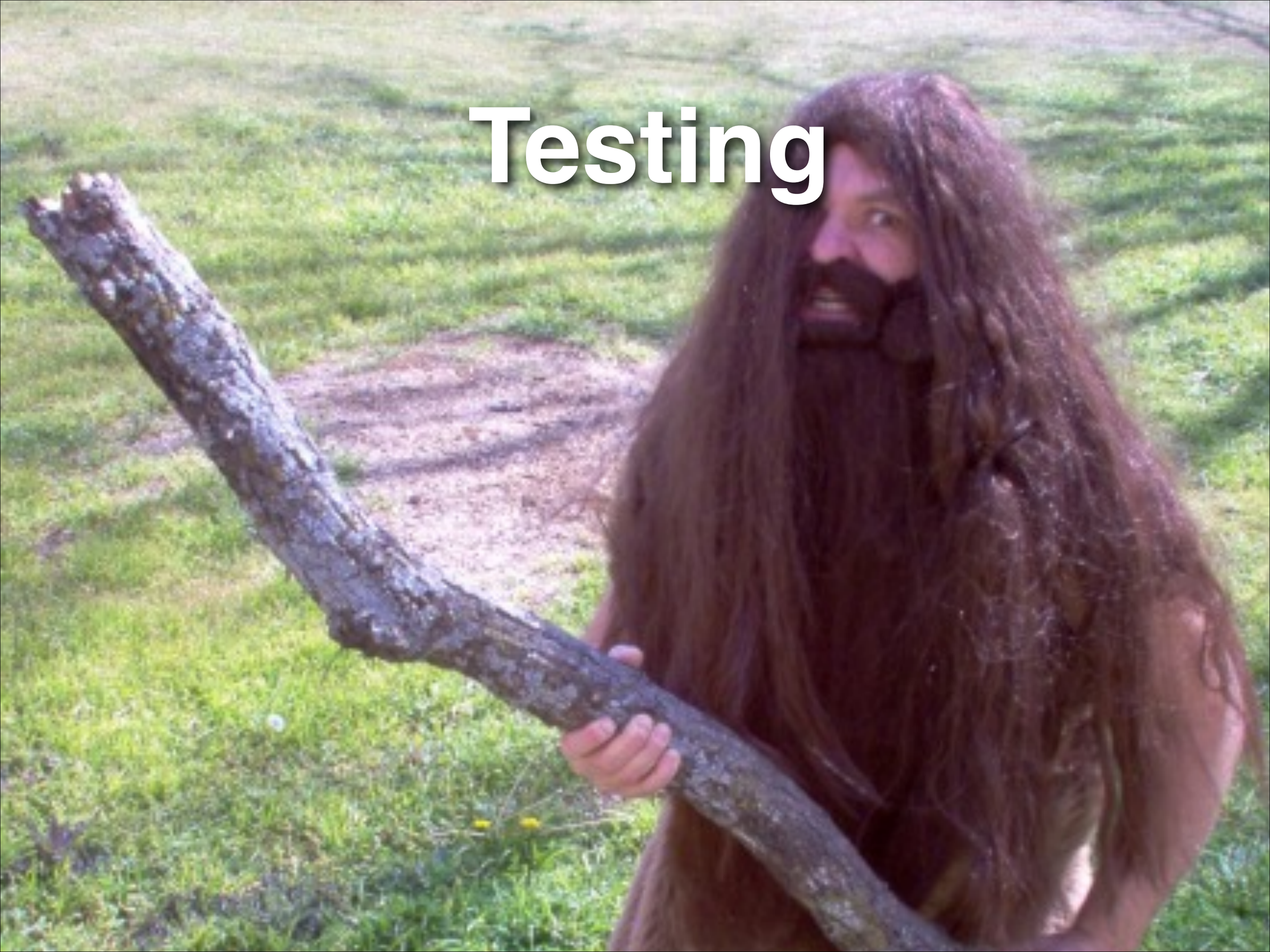
Functional Testing

Software Engineering

Andreas Zeller • Saarland University

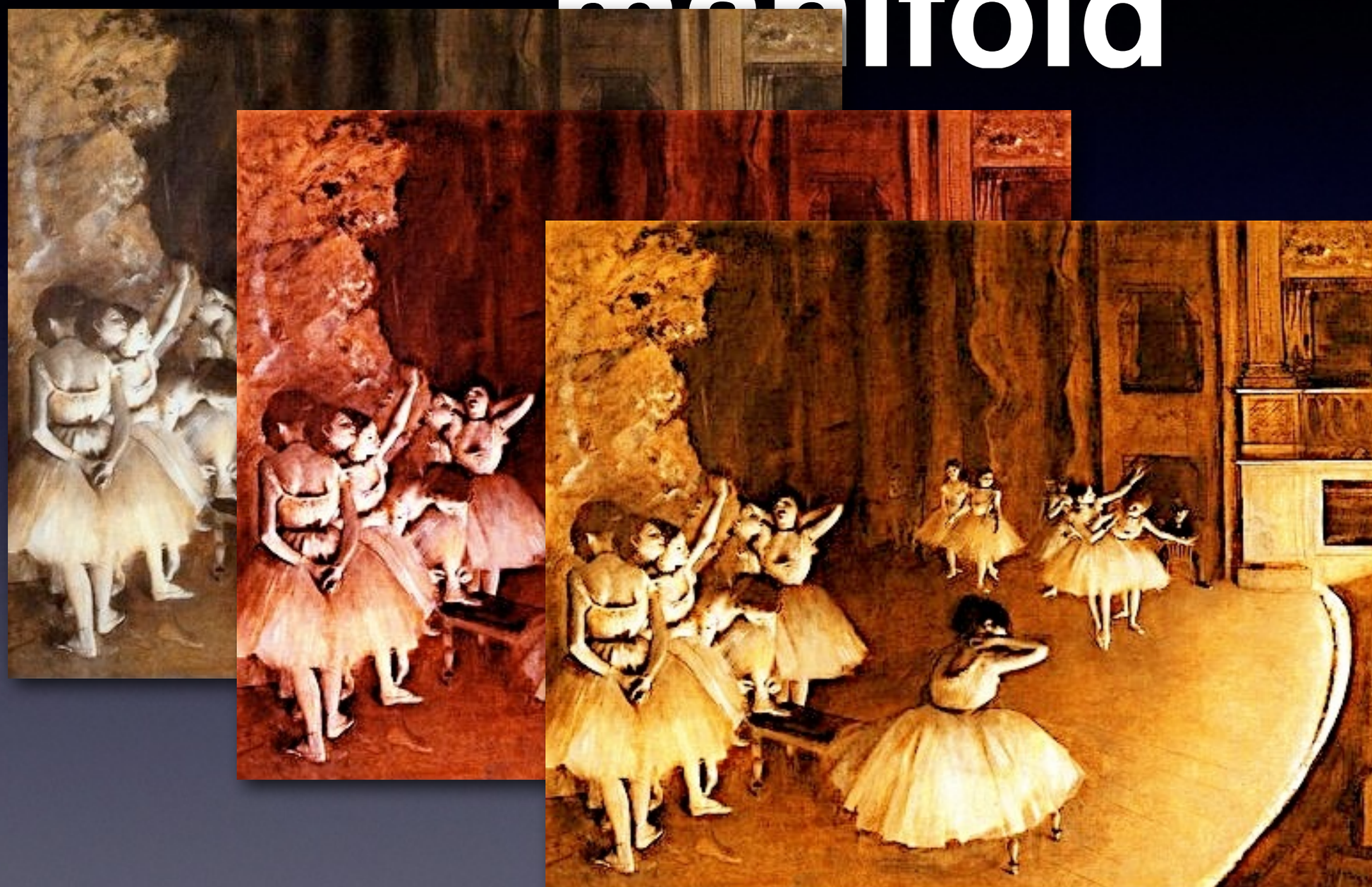

Testing

Testing

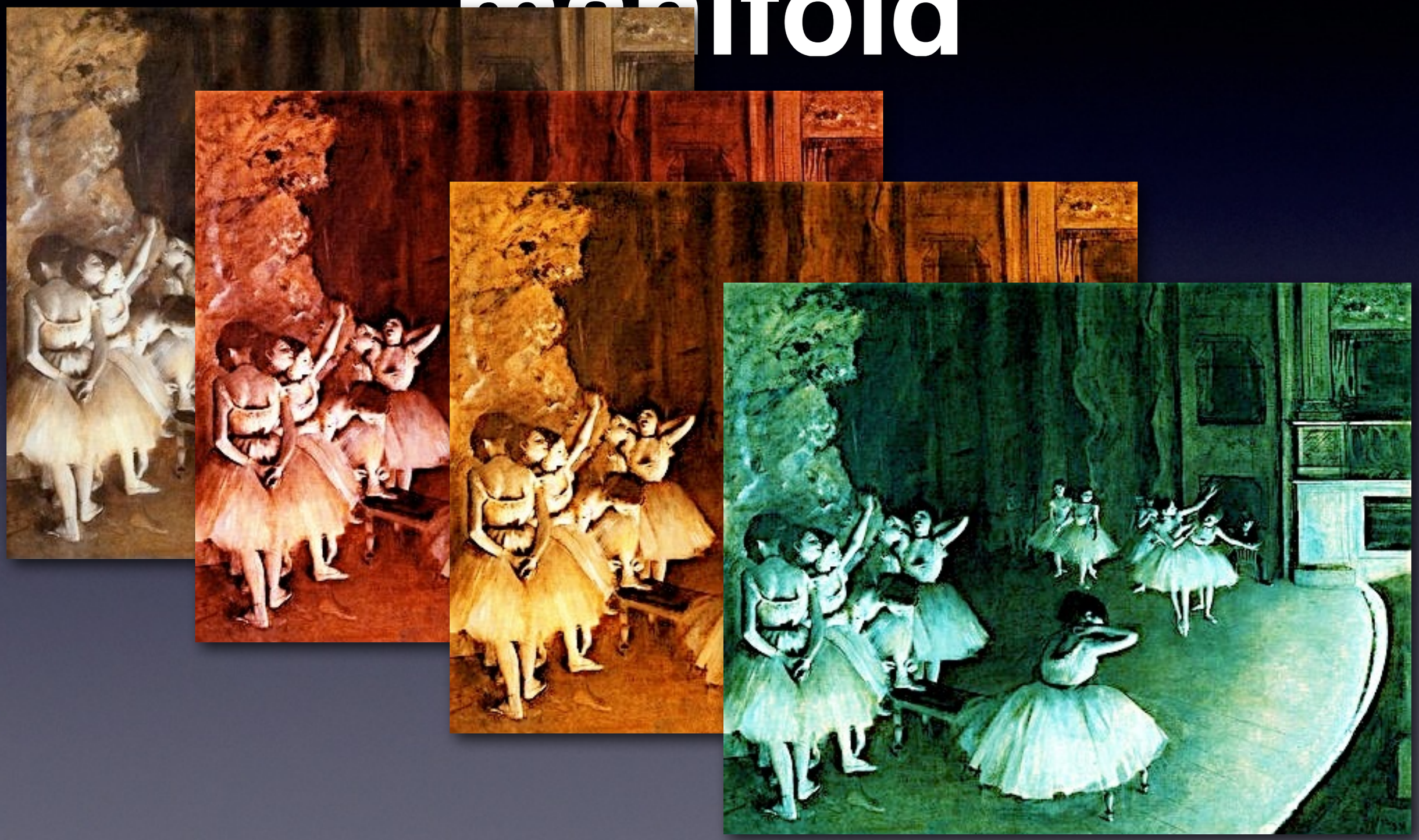
Even more Testing

Testing

Software is manifold

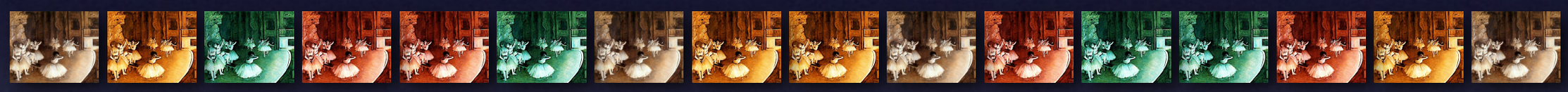


Software is manifold

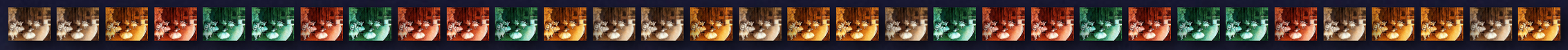


Software is manifold

Software is manifold



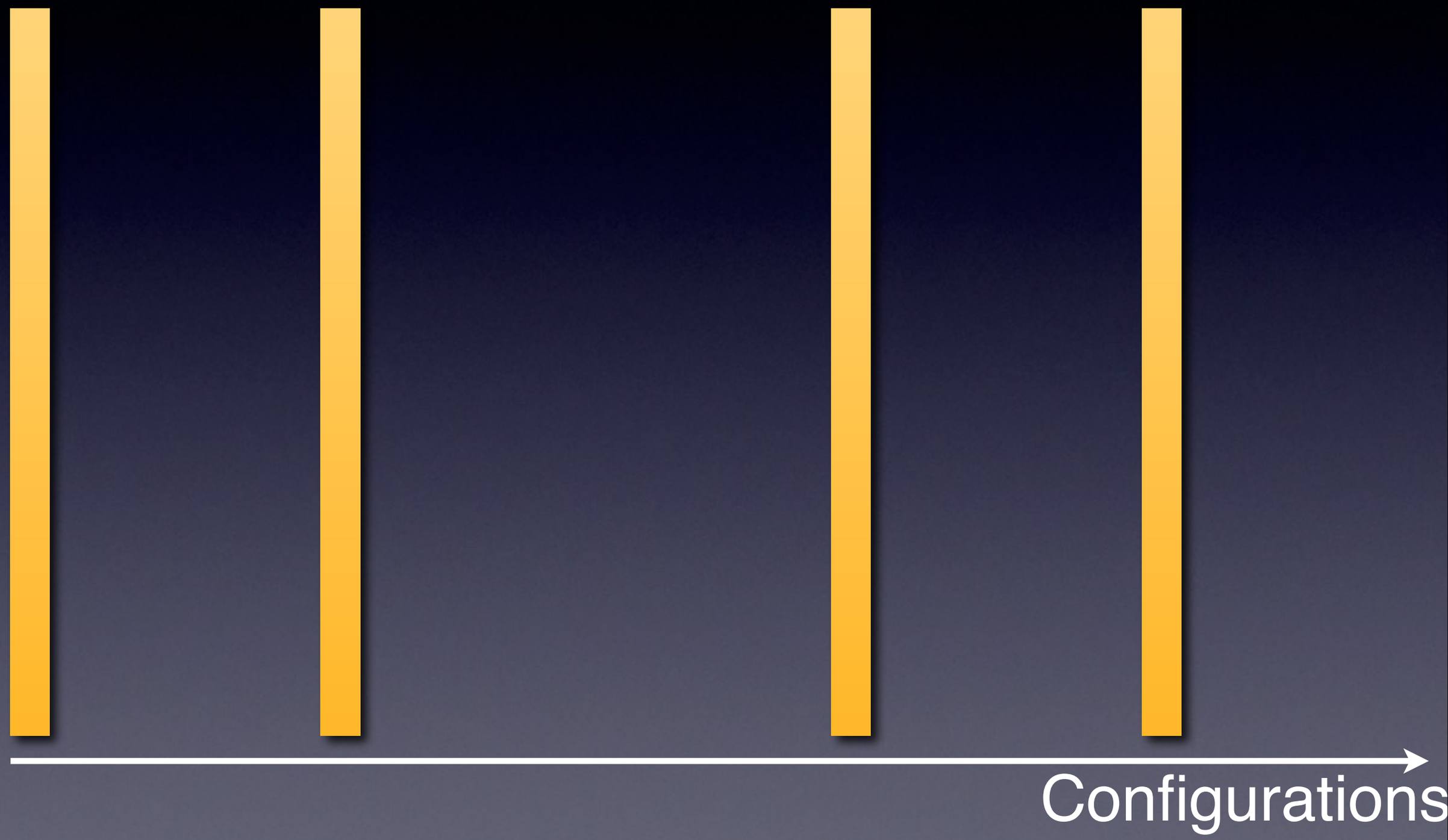
Software is manifold



Software is manifold

Testing

What to test?



Dijkstra's Curse

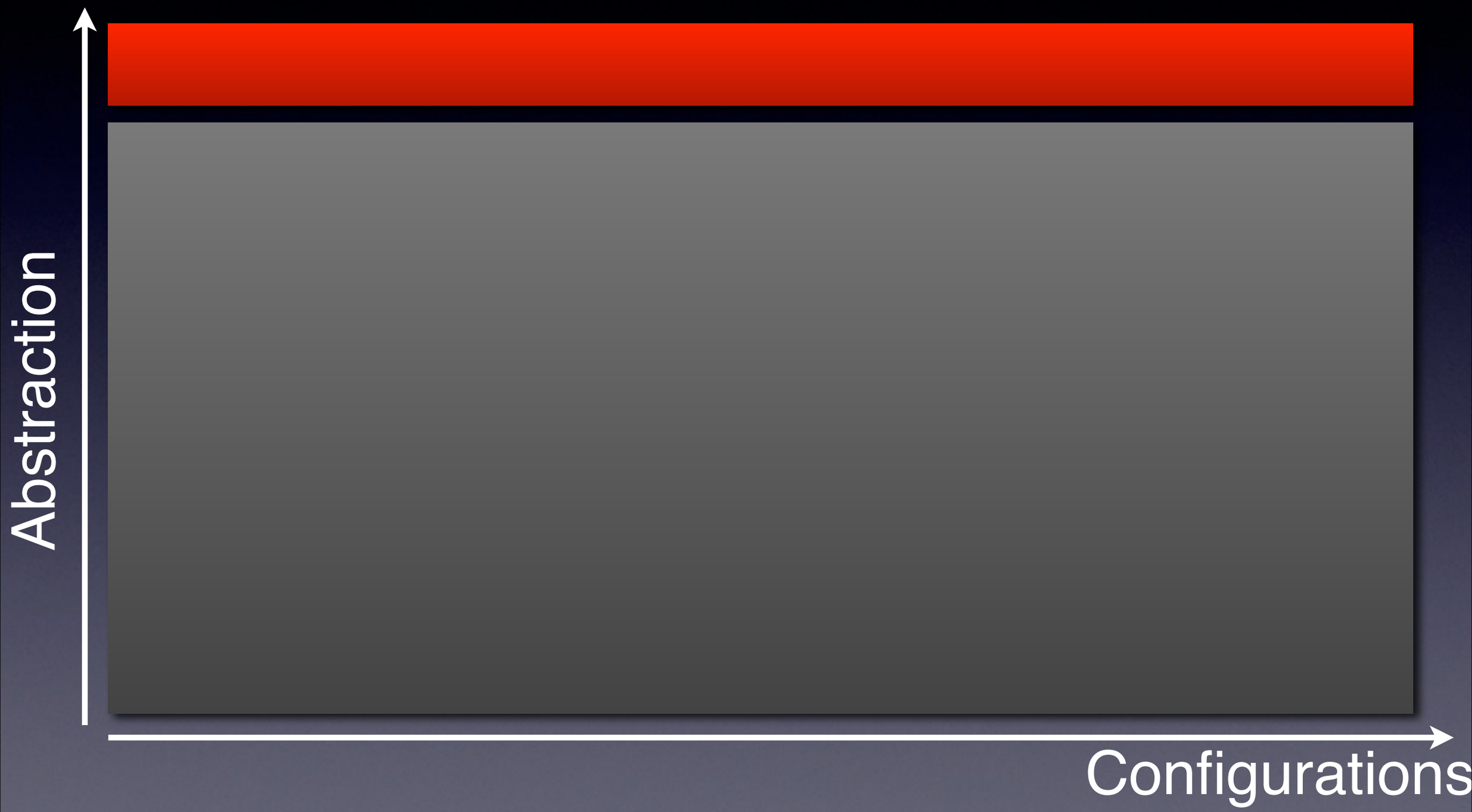
Testing can only find the *presence* of errors, not their *absence*

Configurations

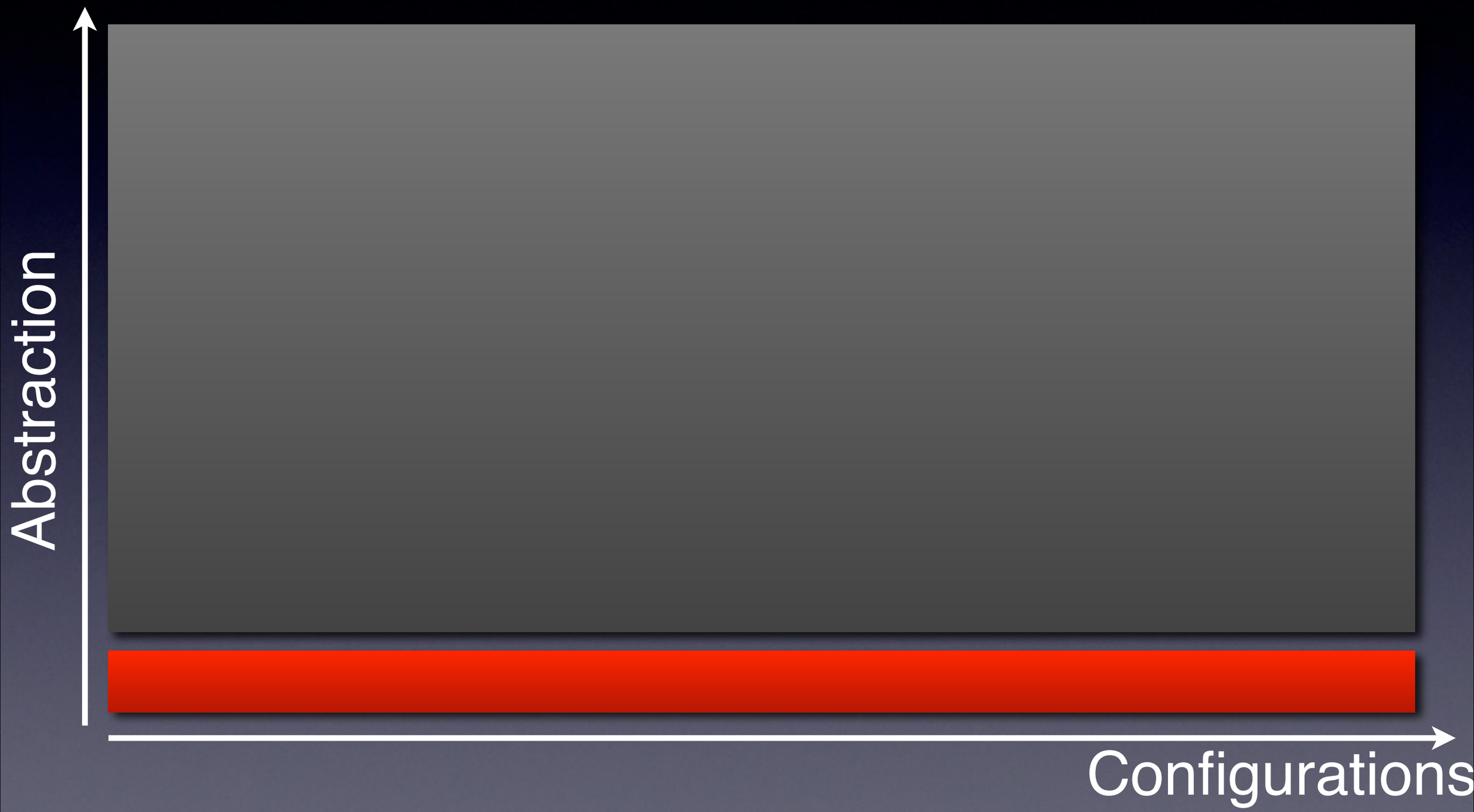
Formal Verification

Formal Verification

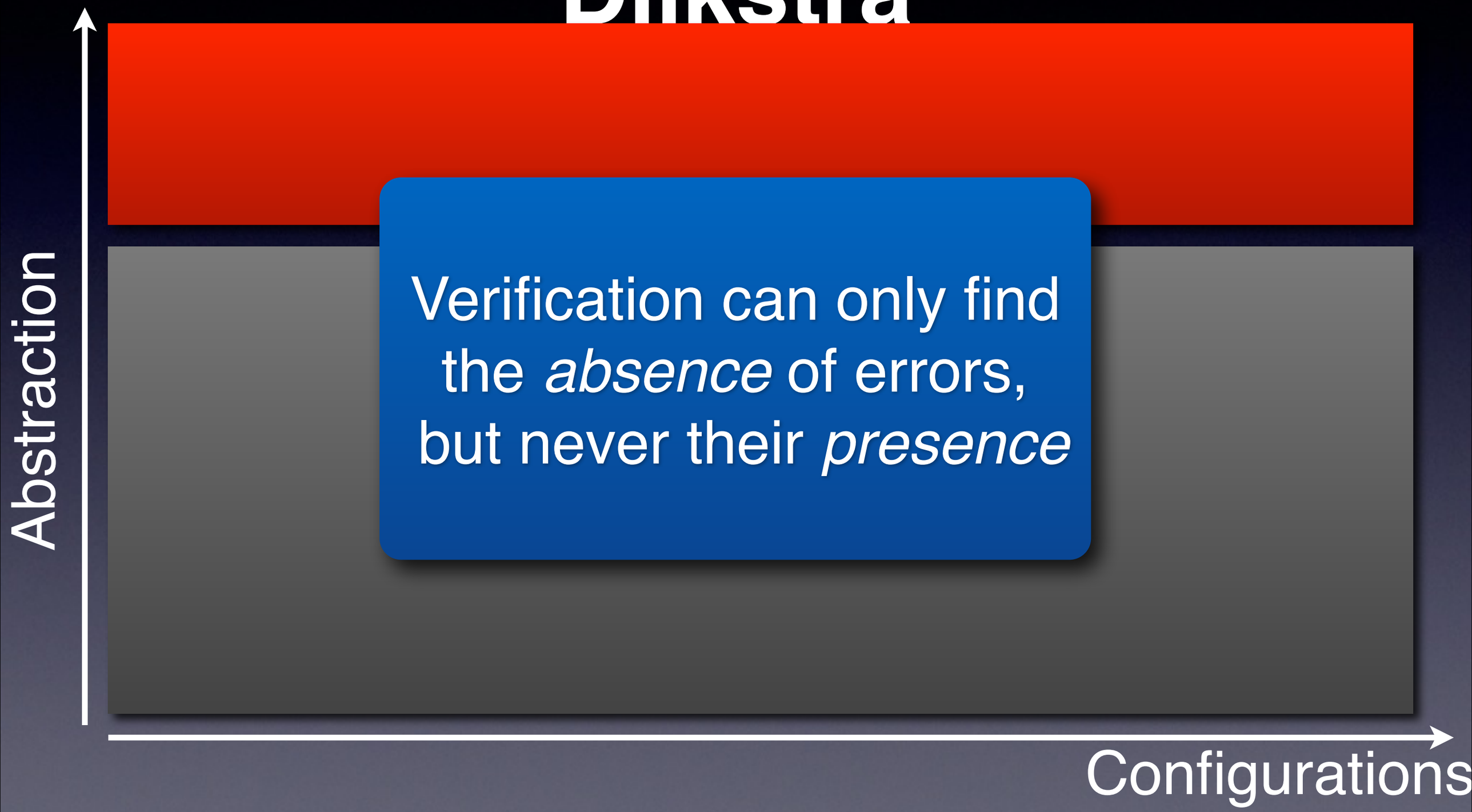
Formal Verification



Formal Verification

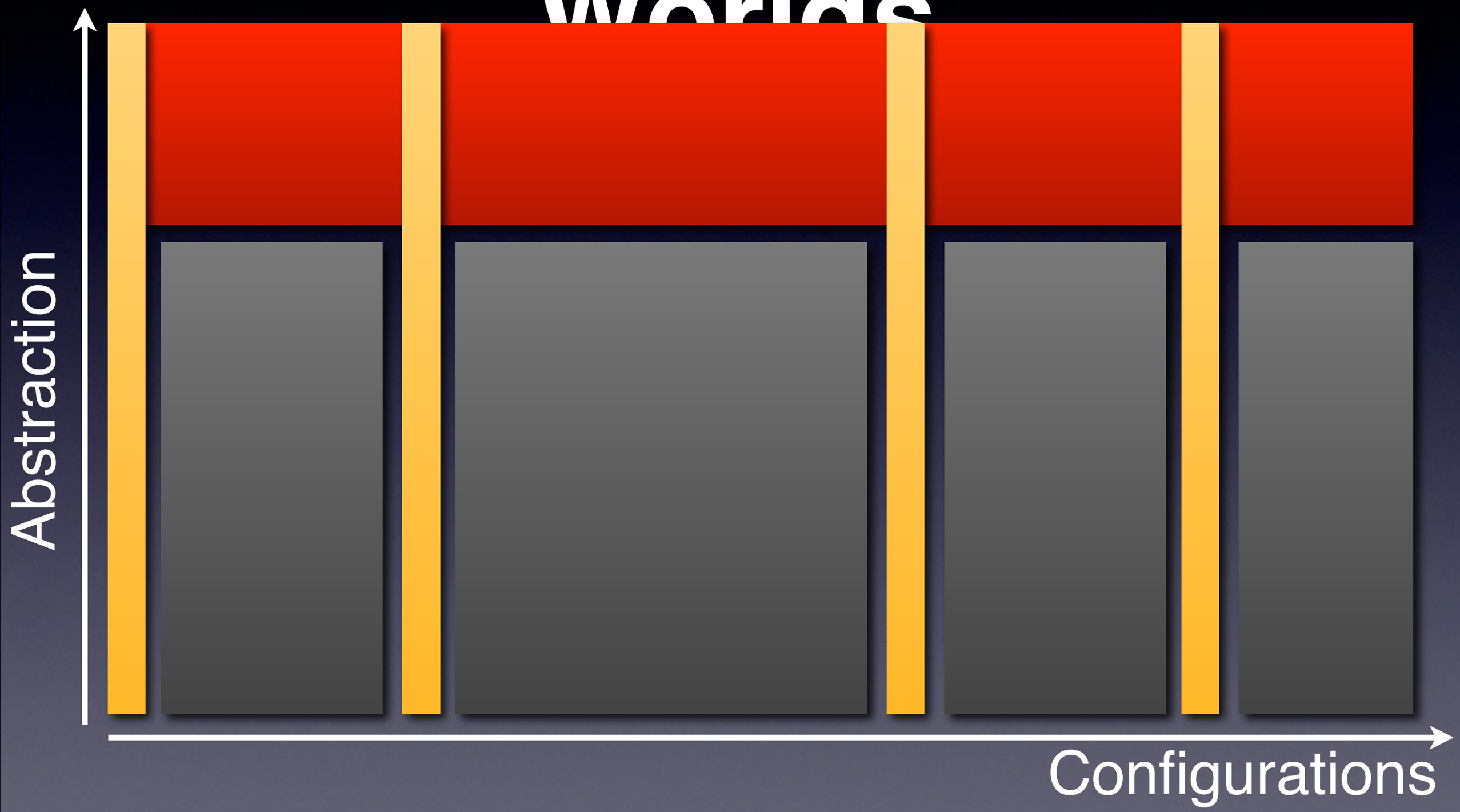


Zeller's Variation on Dijkstra

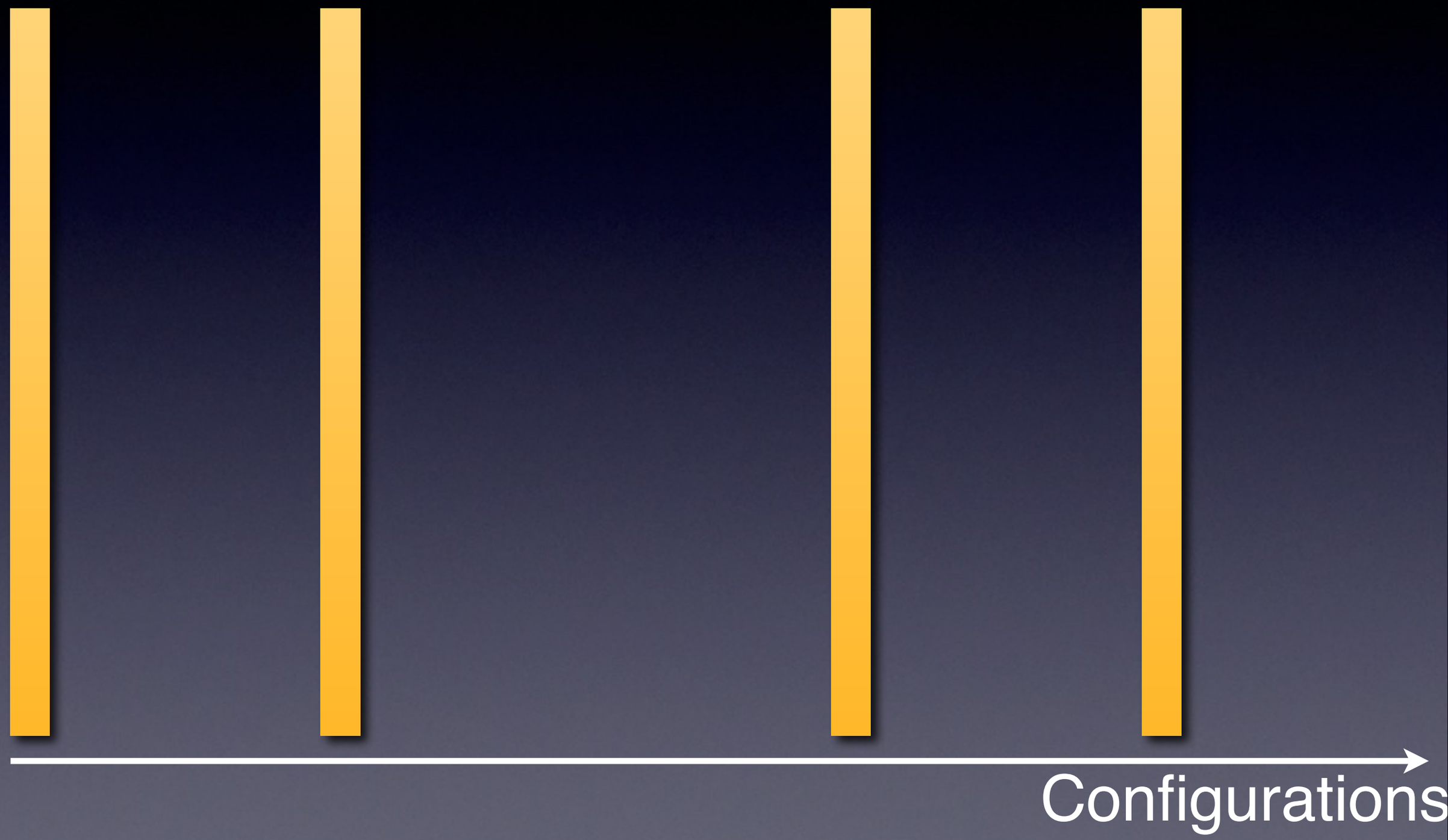


Verification can only find the *absence* of errors, but never their *presence*

The Best of two Worlds



What to test?

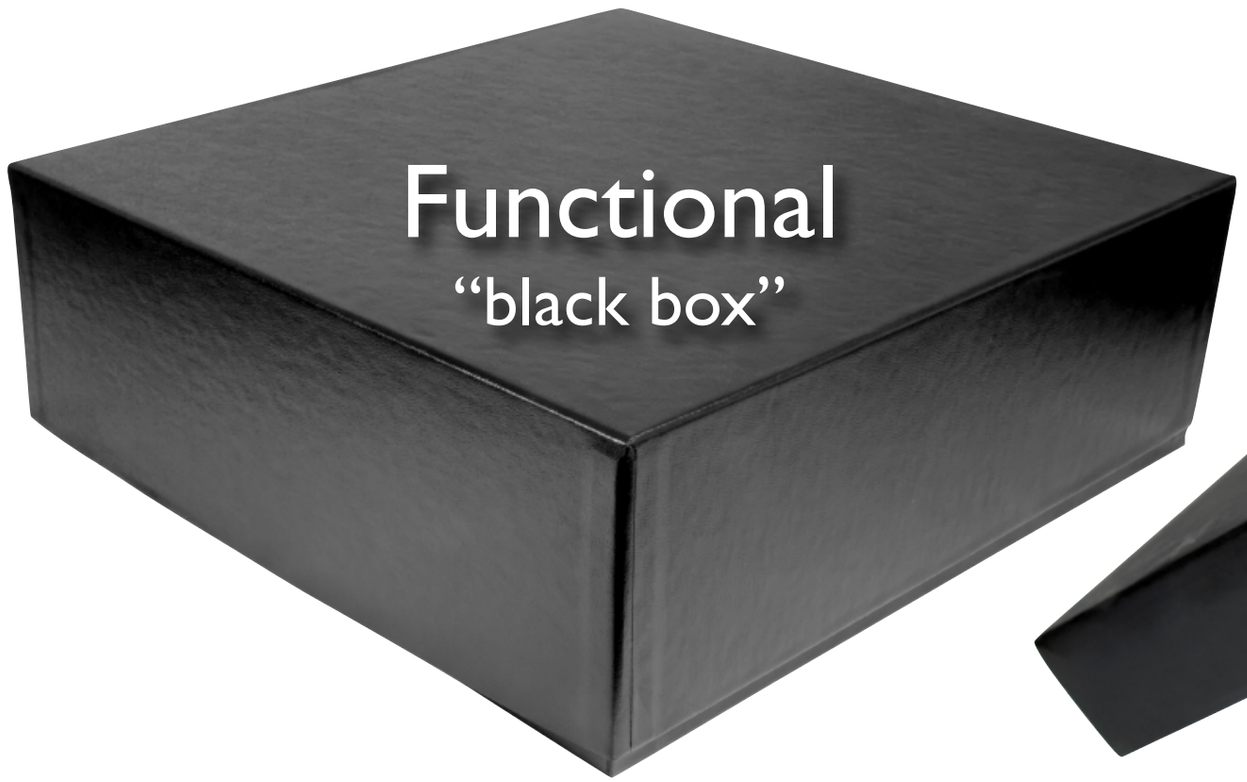


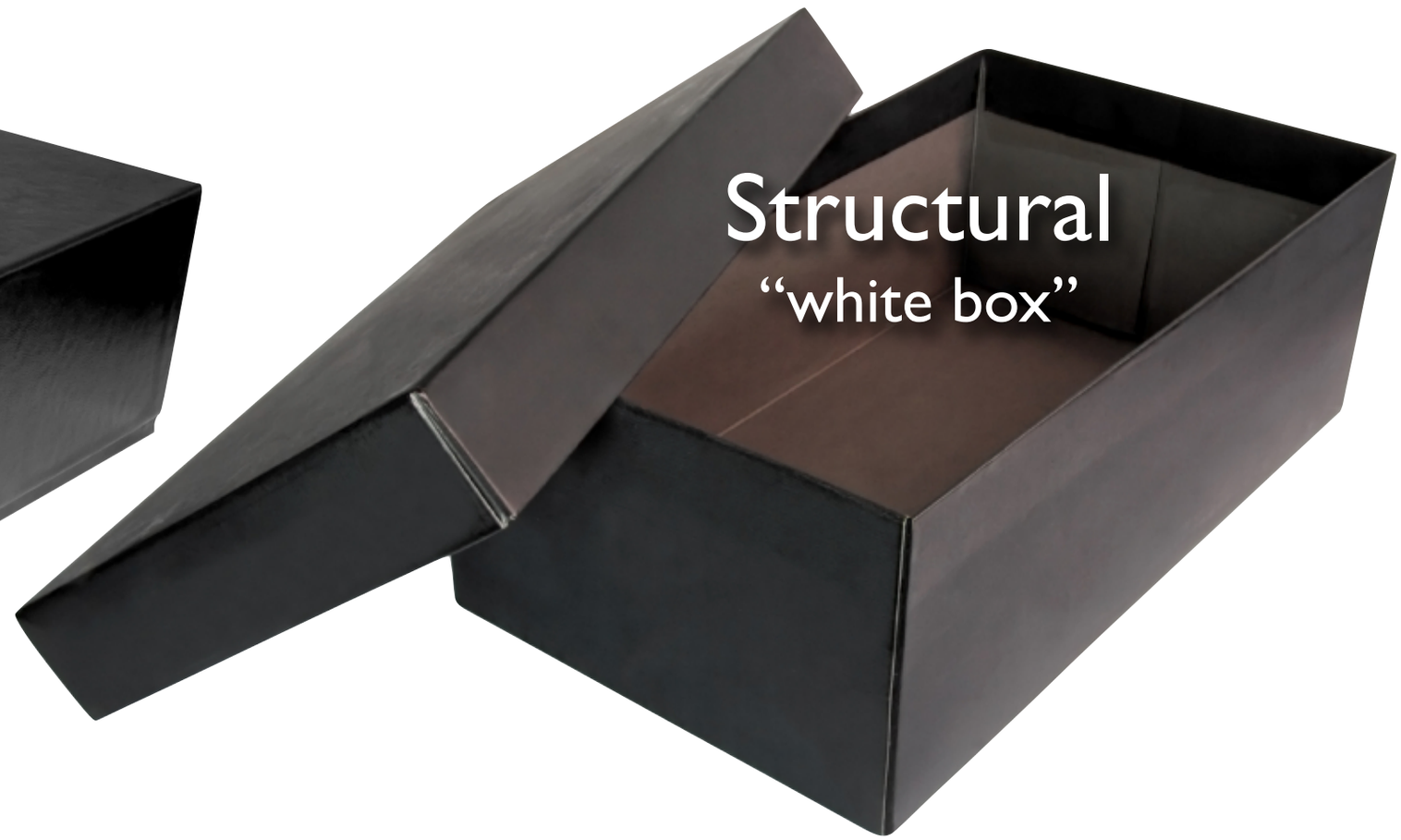
Functional Testing

Software Engineering

Andreas Zeller • Saarland University

Testing Tactics

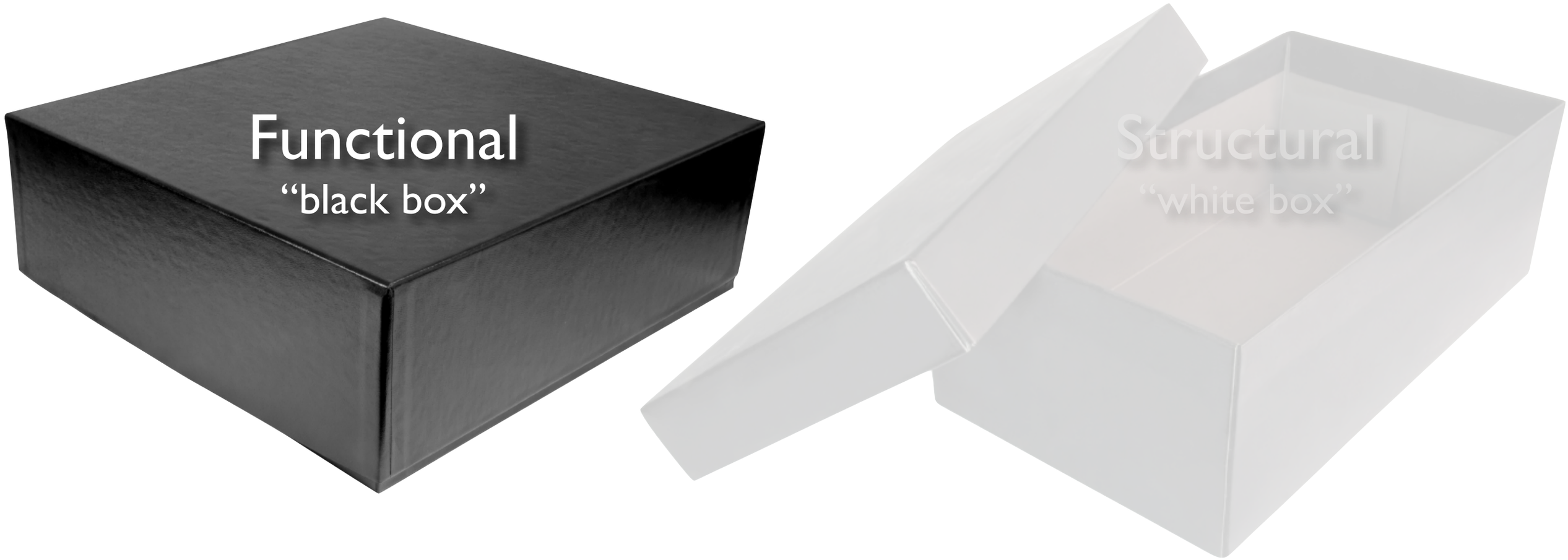




- Tests based on *spec*
- Test covers as much *specified* behavior as possible

- Tests based on *code*
- Test covers as much *implemented* behavior as possible

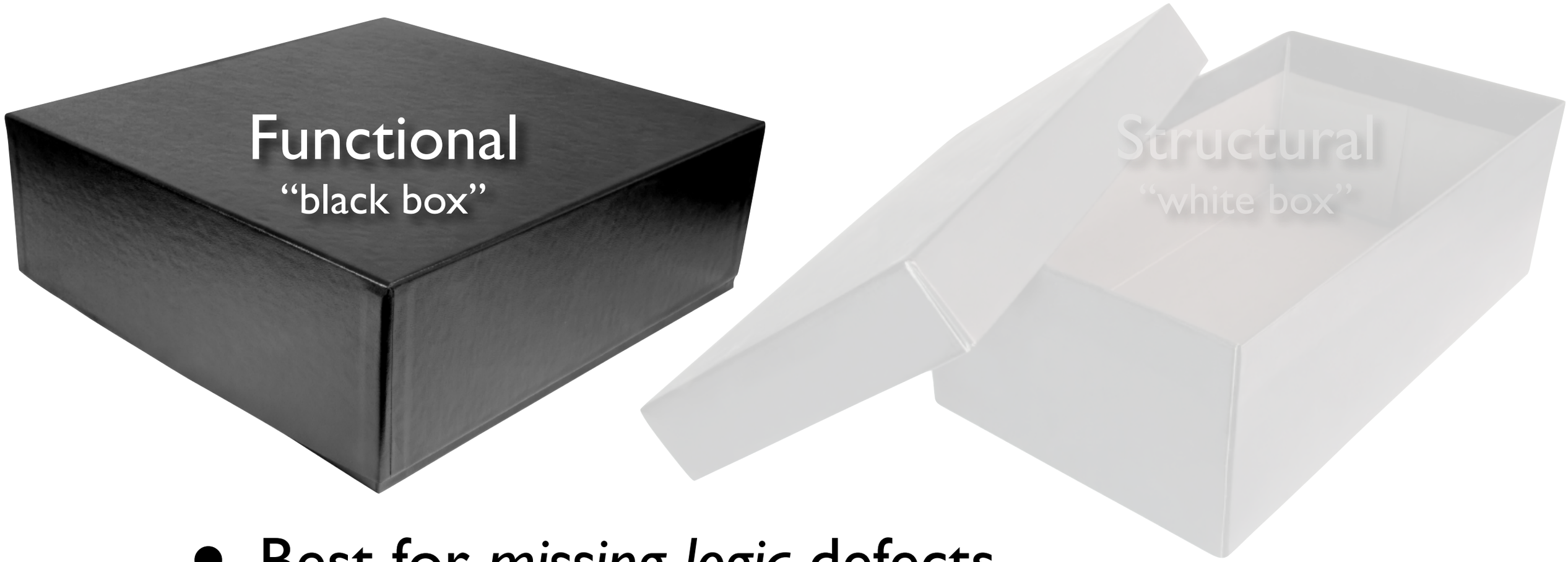
Why Functional?



- Program code not necessary
- Early functional test design has benefits

  reveals spec problems • assesses testability • gives additional explanation of spec • may even serve as spec, as in XP

Why Functional?



- Best for *missing logic* defects

  Common problem: Some program logic was simply forgotten

  Structural testing would not focus on code that is not there

- Applies at all granularity levels

  unit tests • integration tests • system tests • regression tests

Random Testing

- Pick possible inputs uniformly
- Avoids designer bias

  A real problem: The test designer can make the same logical mistakes and bad assumptions as the program designer (especially if they are the same person)
- But treats all inputs as equally valuable


PUNKTE
0


Angle
Force


PUNKTE
0

Infinite Monkey Theorem


Youtube


Angle
2^{32} = 4.294.967.296
different values
Force
2^{32} = 4.294.967.296
different values

2^{32} = 4.294.967.296
different values

2^{32} = 4.294.967.296
different values

2^{64} = 18.446.744.073.709.551.616
different runs

18.446.744.073.709.551.616
Minutes

gadgets-club.com

9.223.372.036.854.775.808 Minutes



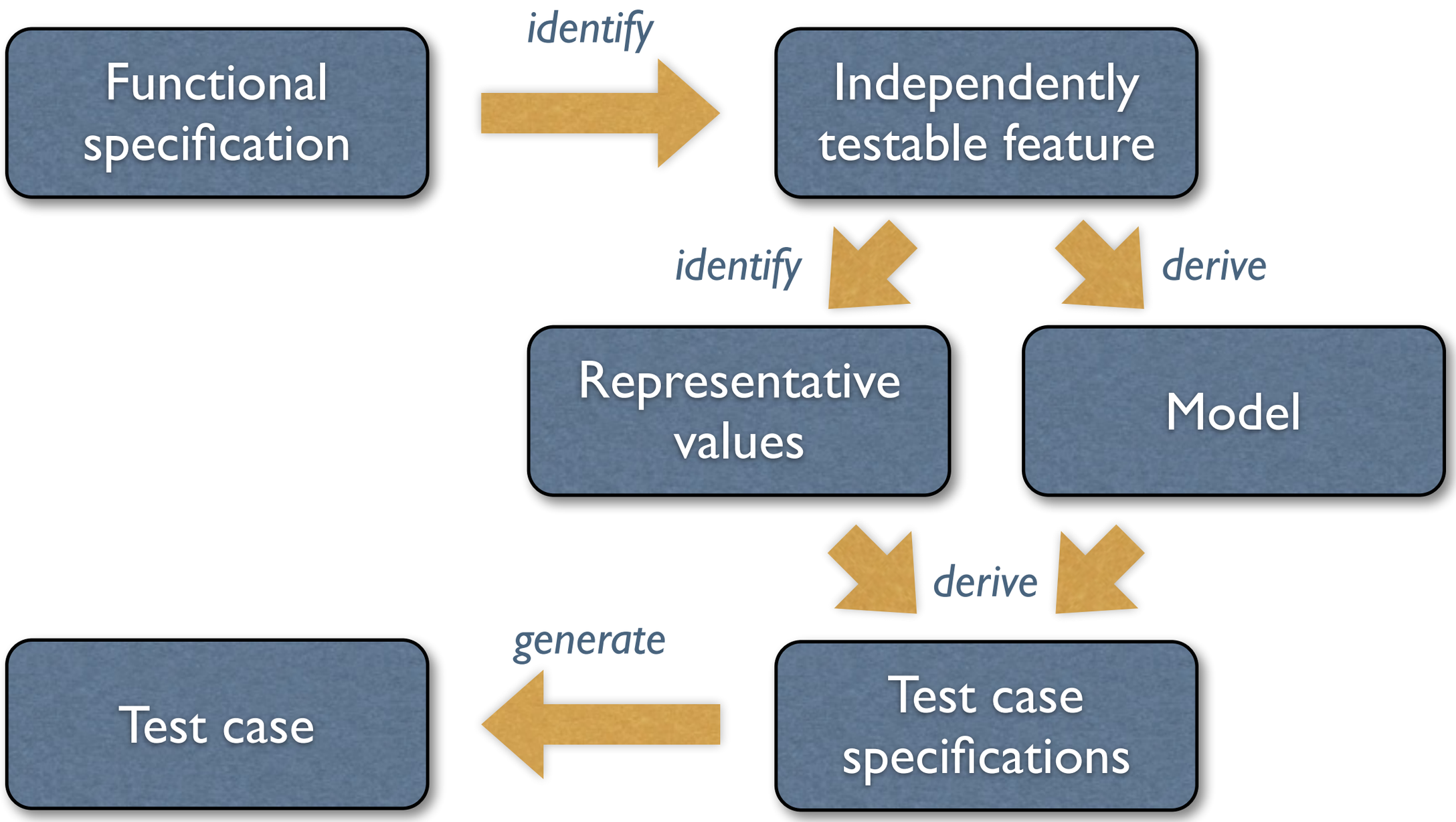

4.611.686.018.427.387.904 Minutes

18.446.744.073.709.551.616

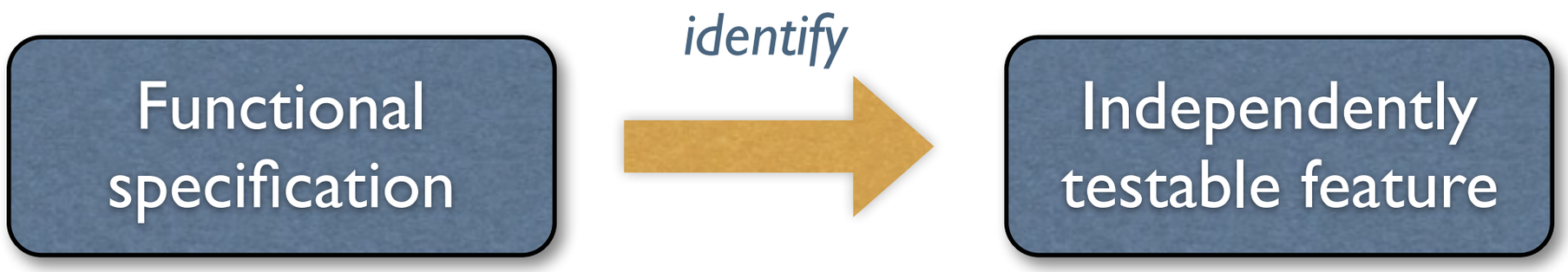
×

1 Minute

Systematic Functional Testing



Testable Features



- Decompose system into *independently testable features* (ITF)
- An ITF need not correspond to units or subsystems of the software
- For system testing, ITFs are exposed through user interfaces or APIs

Testable Fatures

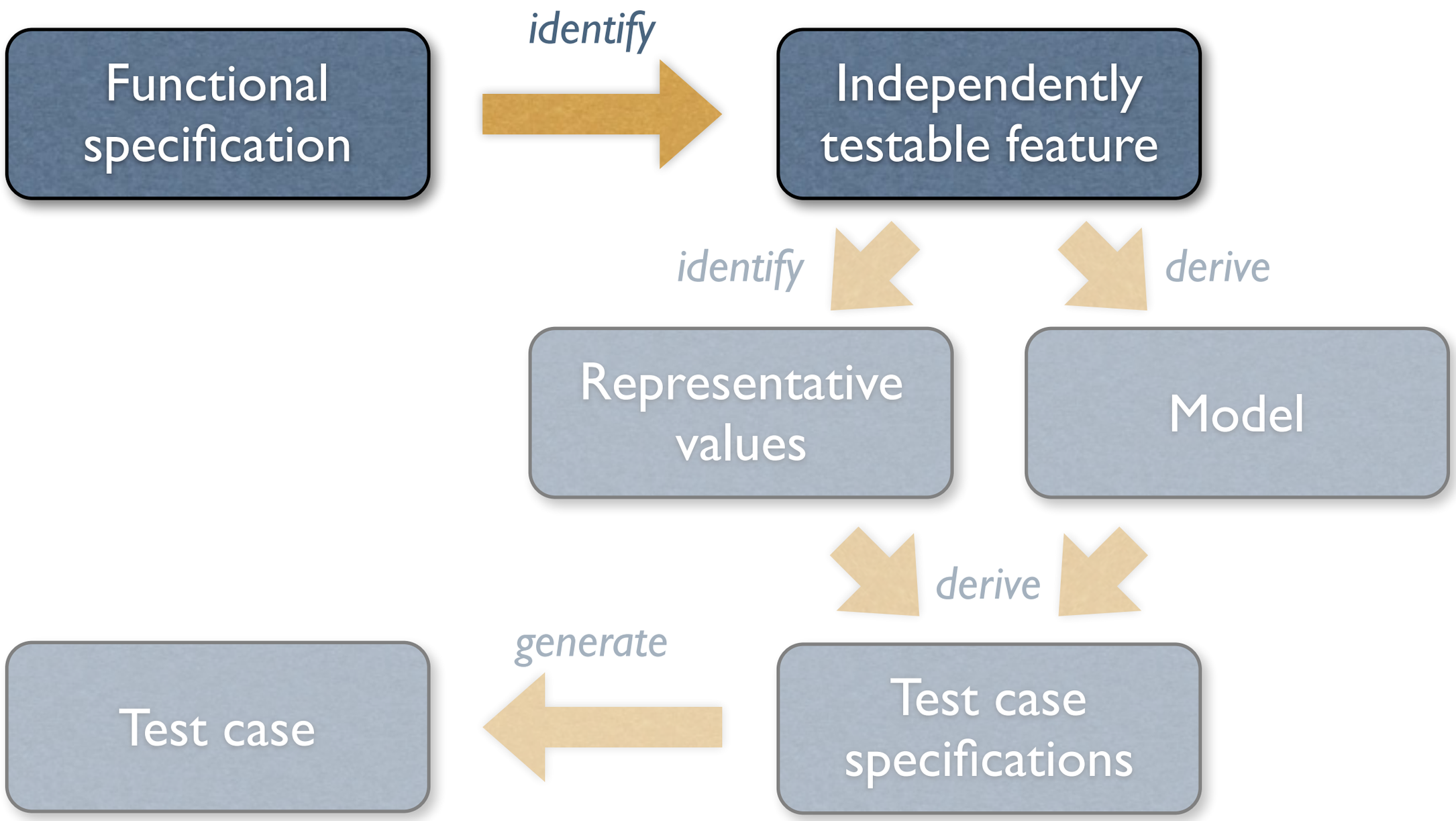
```
class Roots {
    // Solve ax² + bx + c = 0
    public roots(double a, double b, double c)
    { … }

    // Result: values for x
    double root_one, root_two;
}
```

- What are the independently testable features?

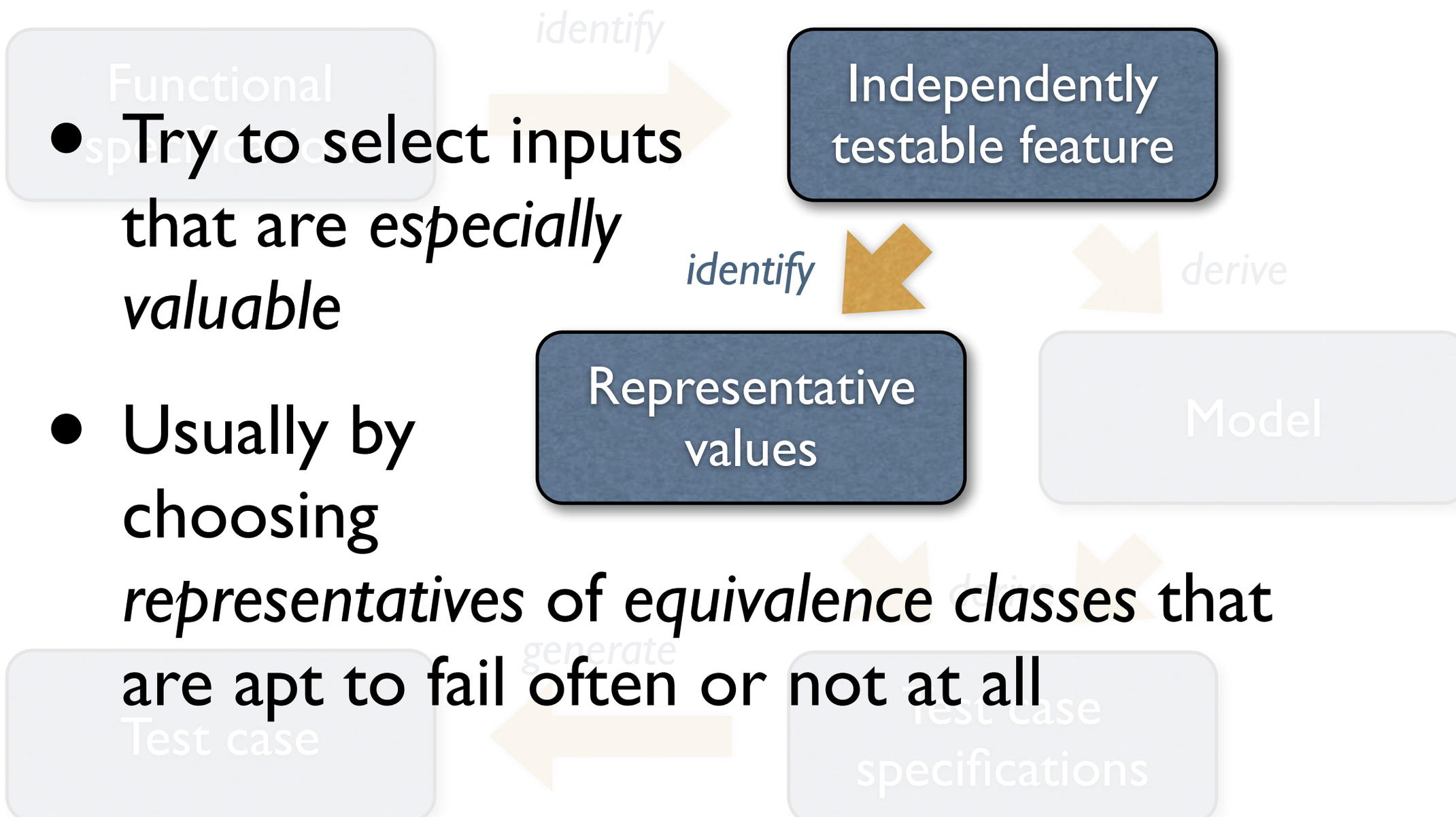
Testable Fatures

- Consider a multi-function calculator
- What are the independently testable features?

Testable Features



Representative Values

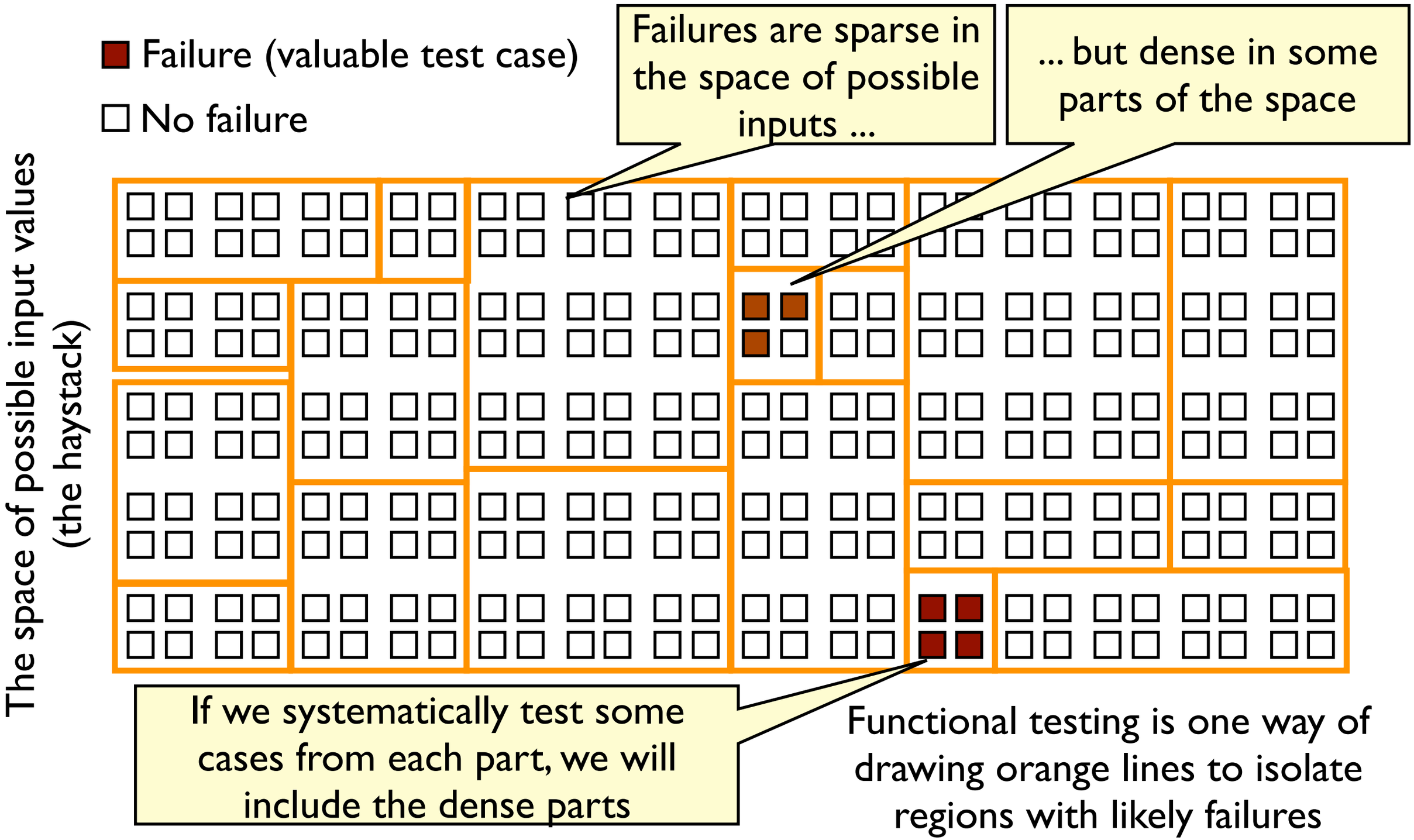


- Try to select inputs that are *especially valuable*
- Usually by choosing *representatives* of *equivalence classes* that are apt to fail often or not at all

Needles in a Haystack

- To find needles, look systematically
- We need to find out *what makes needles special*

Systematic Partition Testing

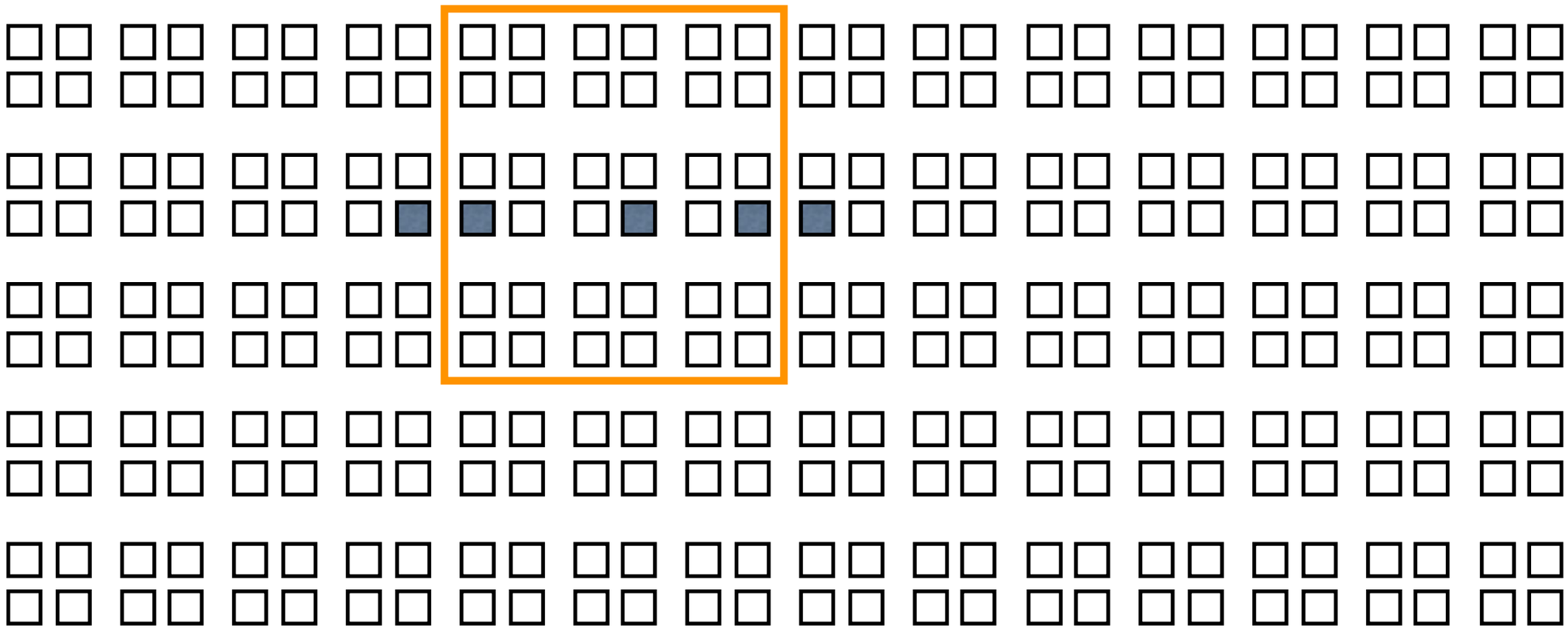


Equivalence Partitioning

Input condition	Equivalence classes
range	one valid, two invalid (larger and smaller)
specific value	one valid, two invalid (larger and smaller)
member of a set	one valid, one invalid
boolean	one valid, one invalid

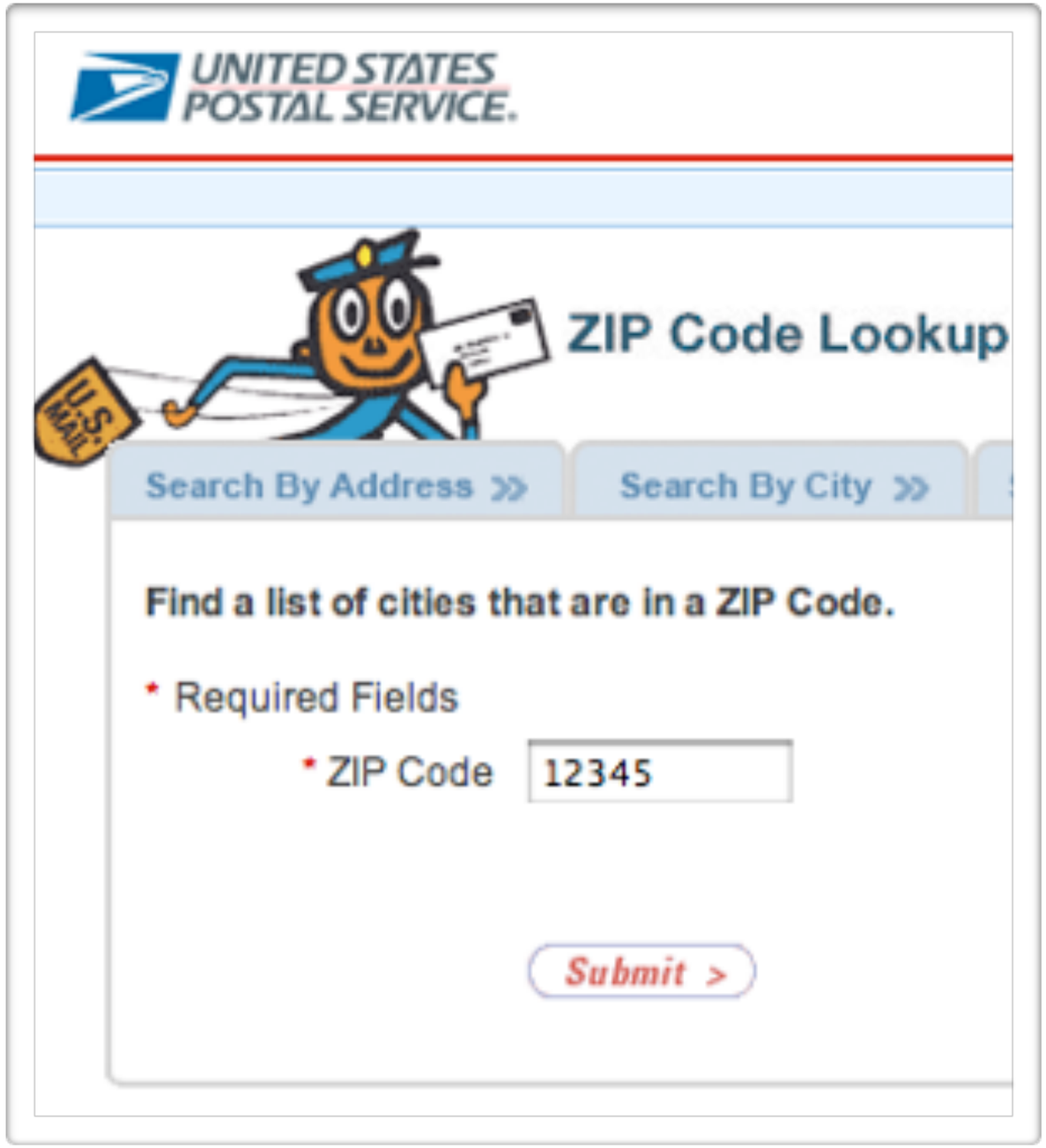
Boundary Analysis

- Possible test case



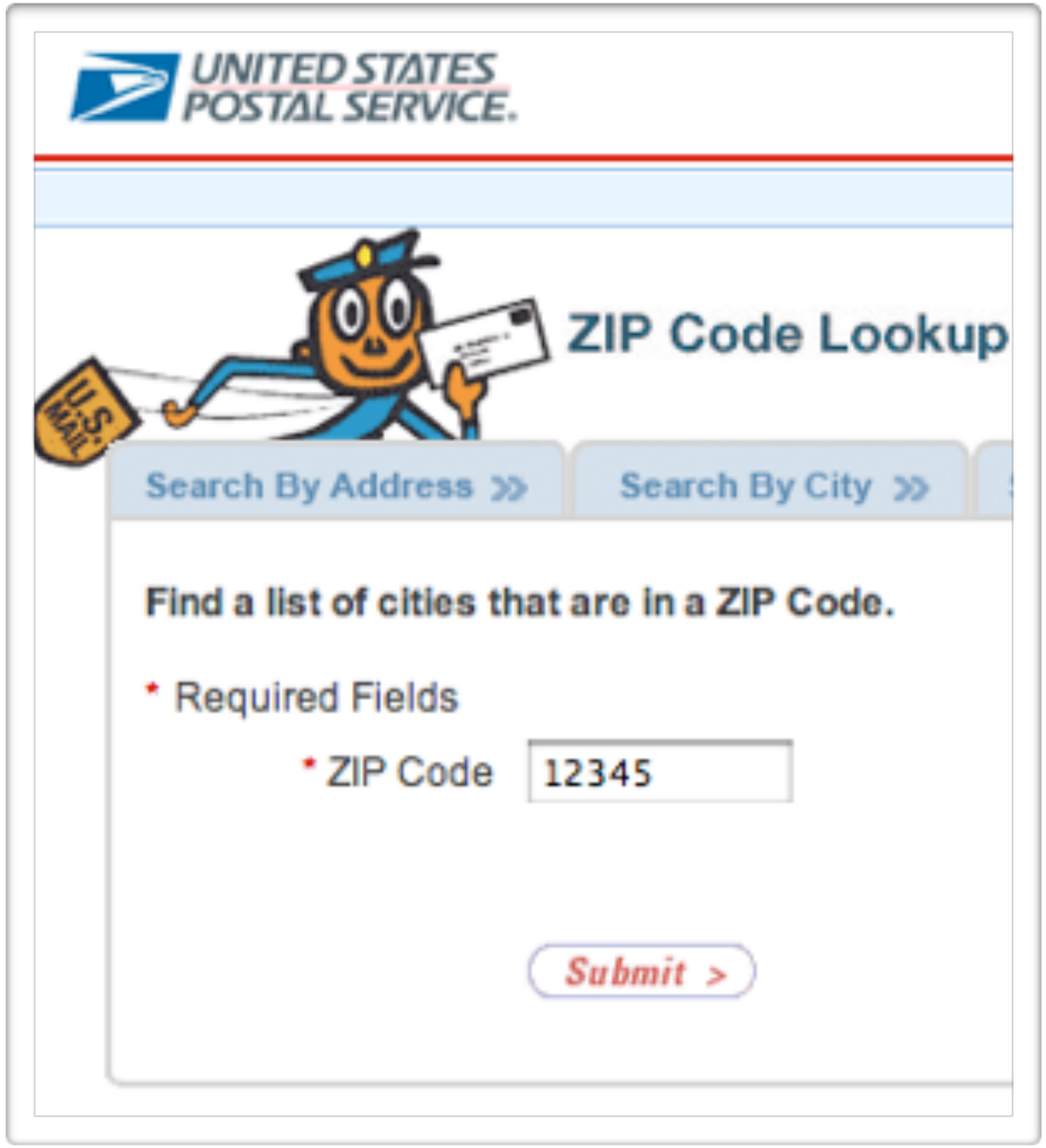
- Test at *lower range* (valid and invalid), at *higher range* (valid and invalid), and at *center*

Example: ZIP Code



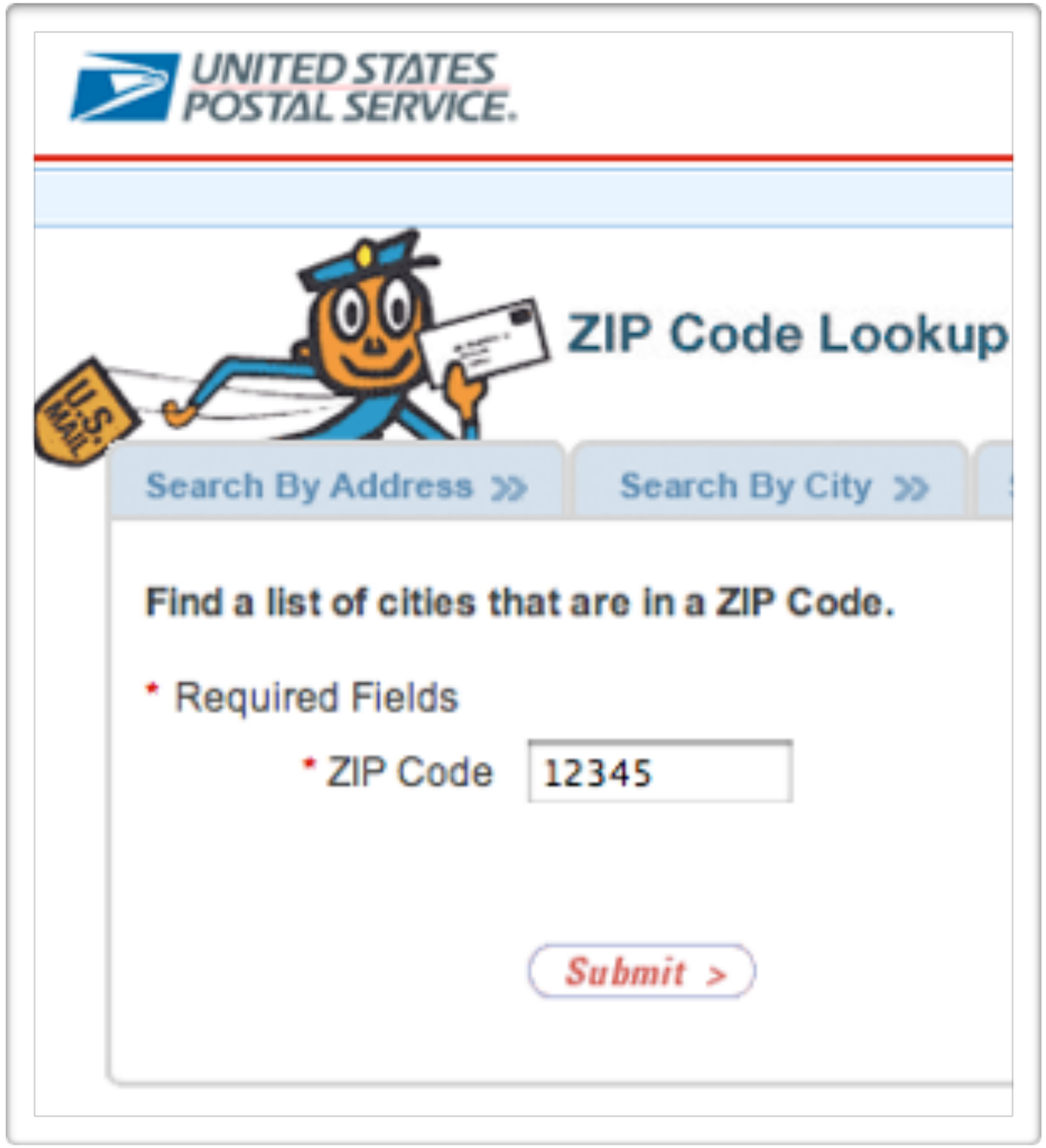
- Input:
  5-digit ZIP code
- Output:
  list of cities
- What are representative values to test?

Valid ZIP Codes



1. with 0 cities as output
   (0 is boundary value)
2. with 1 city as output
3. with many cities as output

Invalid ZIP Codes



4. empty input
5. 1–4 characters
   (4 is boundary value)
6. 6 characters
   (6 is boundary value)
7. very long input
8. no digits
9. non-character data

“Special” ZIP Codes

- How about a ZIP code that reads

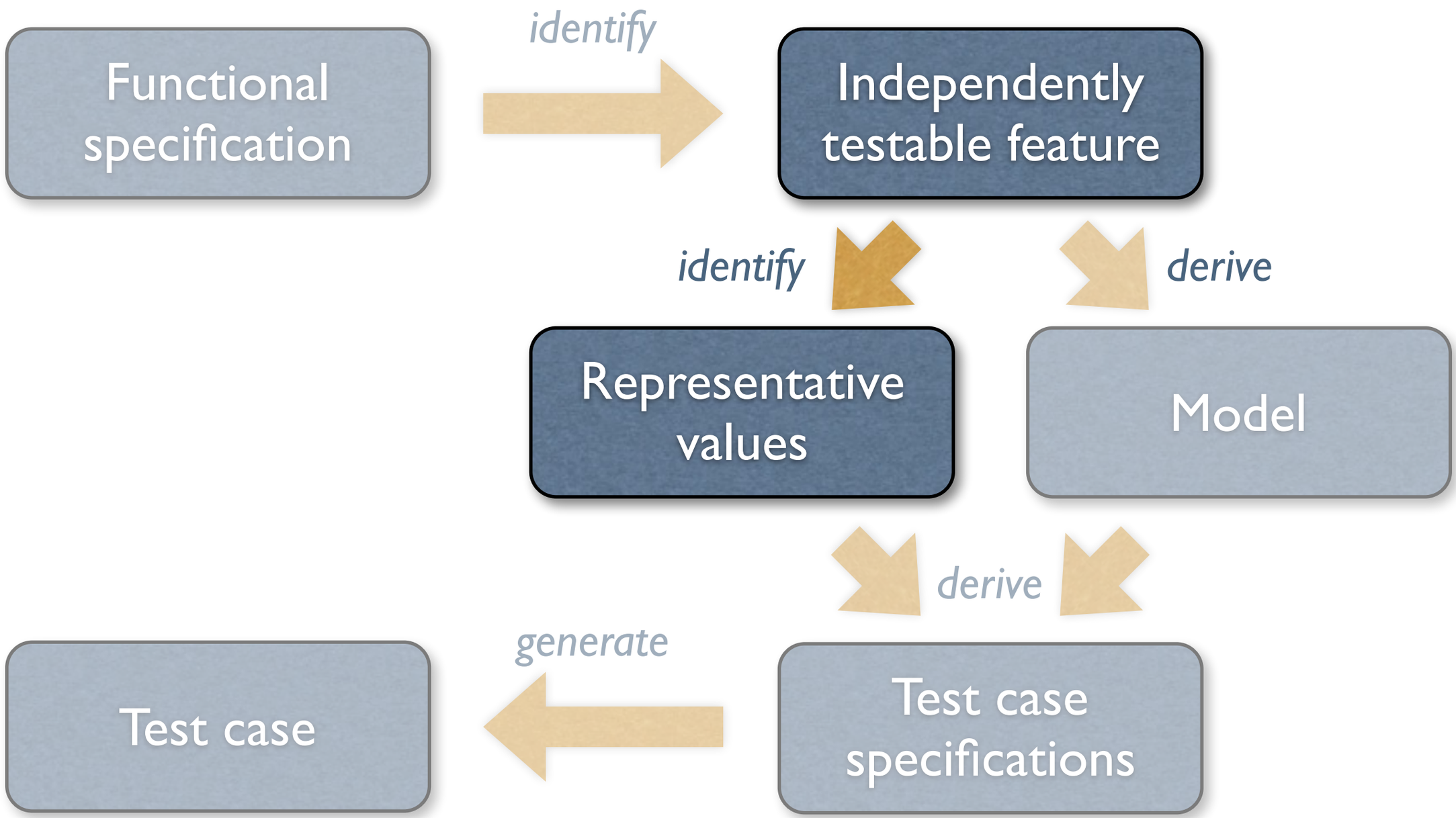
```
12345‘; DROP TABLE orders; SELECT
* FROM zipcodes WHERE ‘zip’ = ‘
```

- Or a ZIP code with 65536 characters…
- This is security testing

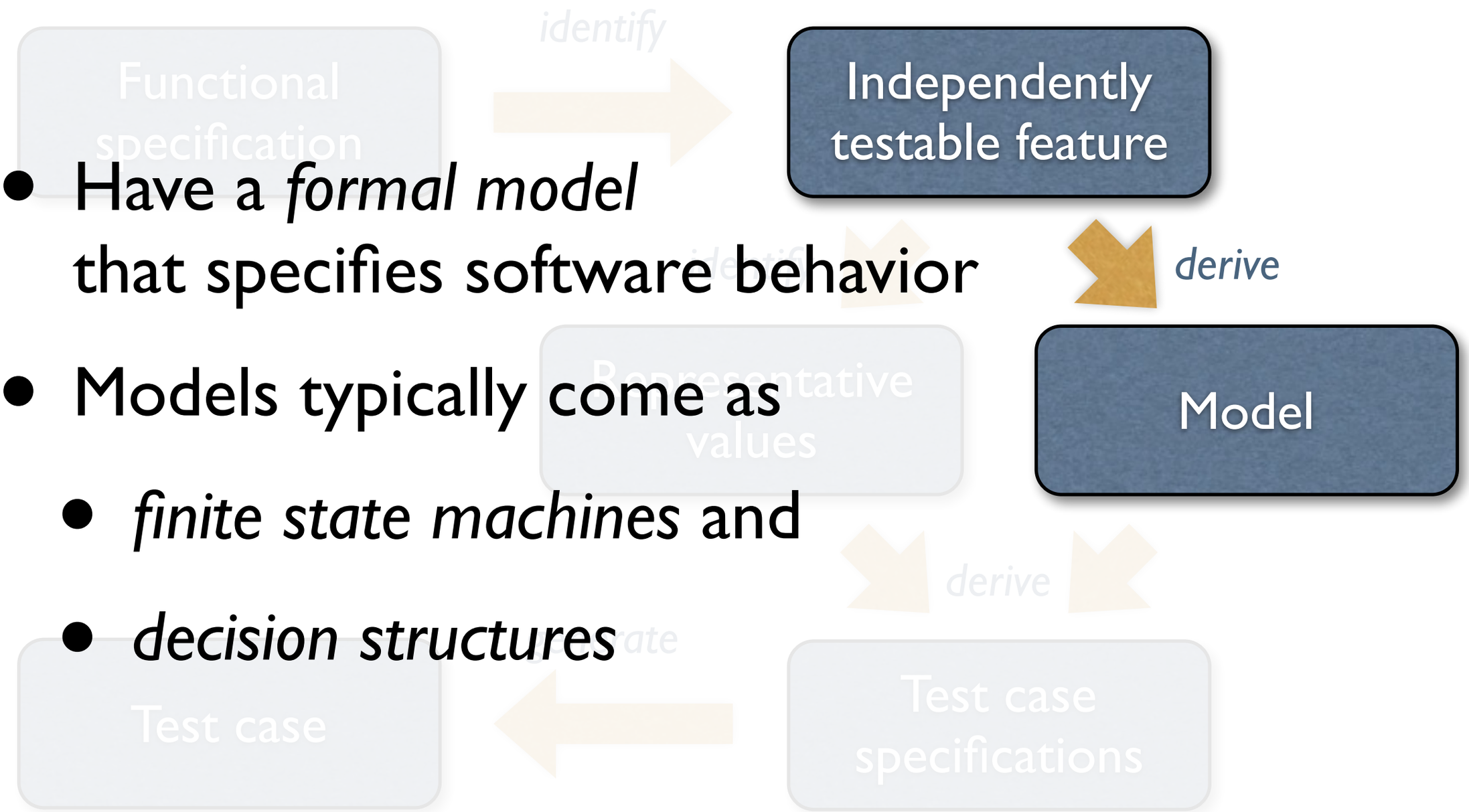
Gutjahr's Hypothesis

Partition testing
is more effective
than random testing.

Representative Values

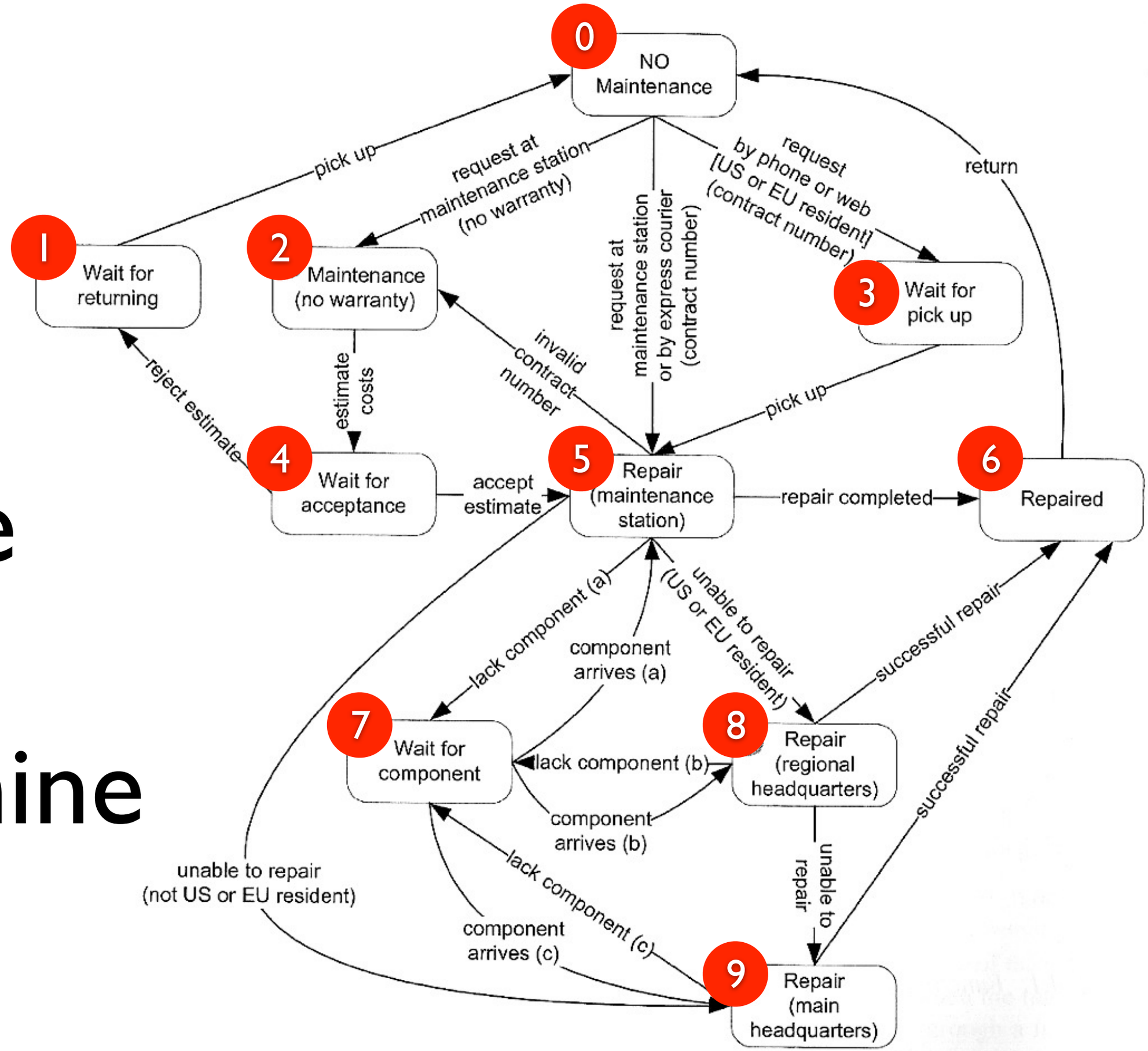


Model-Based Testing



- Have a *formal model* that specifies software behavior
- Models typically come as
  - *finite state machines* and
  - *decision structures*

Finite State Machine



Maintenance: The *Maintenance* function records the history of items undergoing maintenance.

If the product is covered by warranty or maintenance contract, maintenance can be requested either by calling the maintenance toll free number, or through the Web site, or by bringing the item to a designated maintenance station.

If the maintenance is requested by phone or Web site and the customer is a US or EU resident, the item is picked up at the customer site, otherwise, the customer shall ship the item with an express courier.

If the maintenance contract number provided by the customer is not valid, the item follows the procedure for items not covered by warranty.

If the product is not covered by warranty or maintenance contract, maintenance can be requested only by bringing the item to a maintenance station. The maintenance station informs the customer of the estimated costs for repair. Maintenance starts only when the customer accepts the estimate. If the customer does not accept the estimate, the product is returned to the customer.

Small problems can be repaired directly at the maintenance station. If the maintenance station cannot solve the problem, the product is sent to the maintenance regional headquarters (if in US or EU) or to the maintenance main headquarters (otherwise).

If the maintenance regional headquarters cannot solve the problem, the product is sent to the maintenance main headquarters.

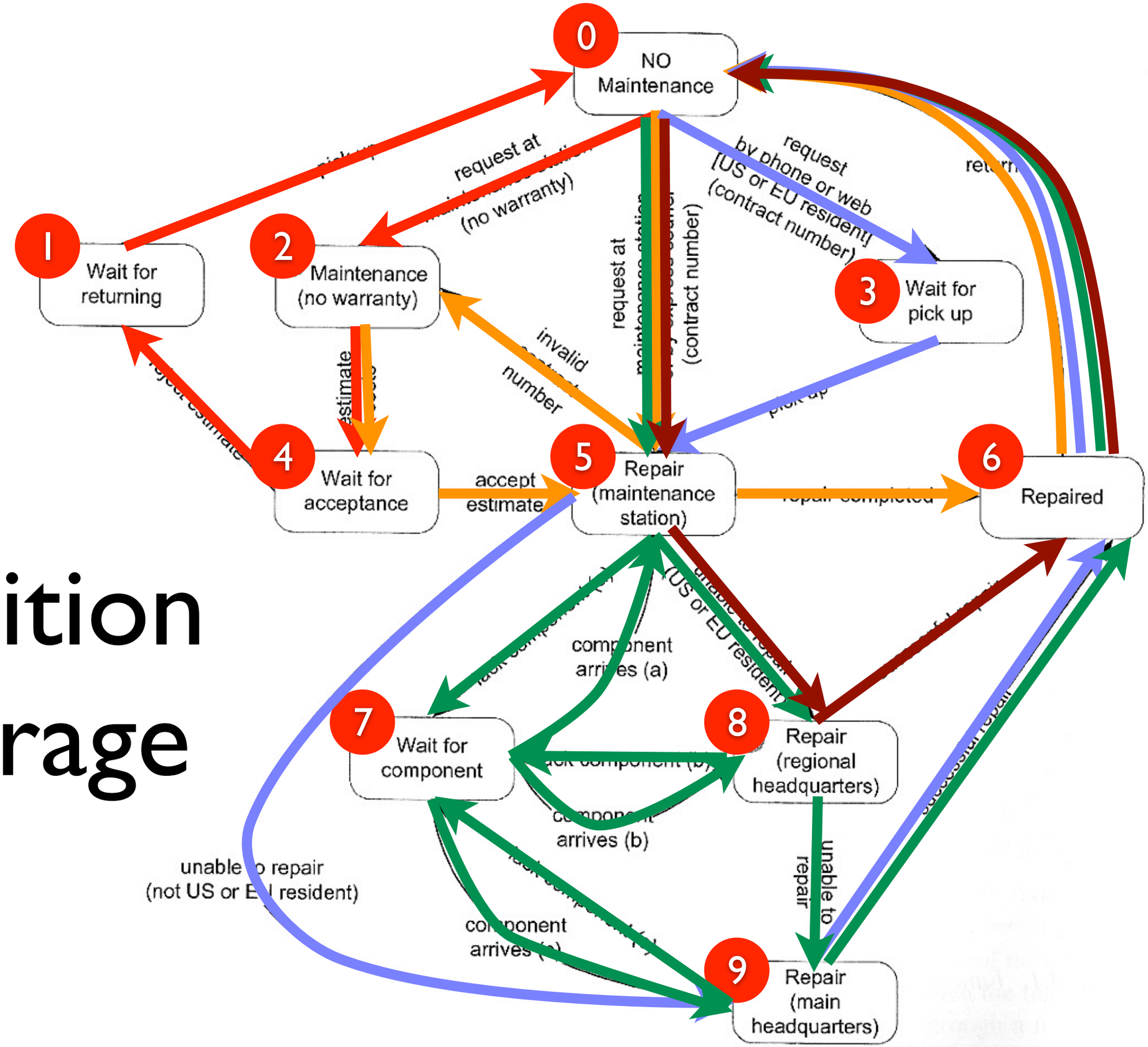
Maintenance is suspended if some components are not available.

Once repaired, the product is returned to the customer.

Coverage Criteria

- *Path coverage:* Tests cover every path
  Not feasible in practice due to infinite number of paths
- *State coverage:* Every node is executed
  A minimum testing criterion
- *Transition coverage:* Every edge is executed
  Typically, a good coverage criterion to aim for

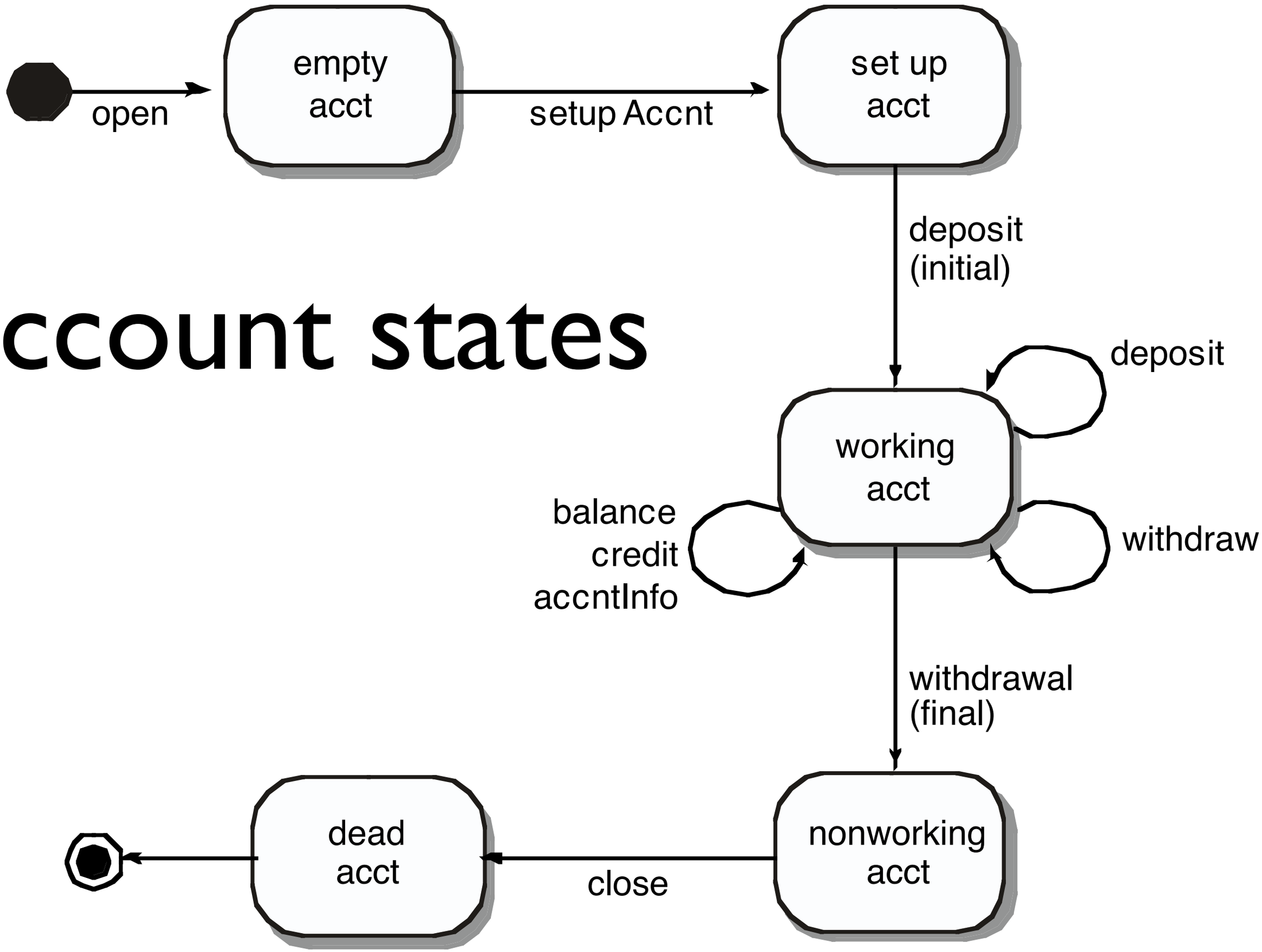
Transition Coverage



State-based Testing

- Protocols (e.g., network communication)
- GUIs (sequences of interactions)
- Objects (methods and states)

Account states



Decision Tables

	Education		Individual					
Education account	T	T	F	F	F	F	F	F
Current purchase > Threshold 1	–	–	F	F	T	T	–	–
Current purchase > Threshold 2	–	–	–	–	F	F	T	T
Special price < scheduled price	F	T	F	T	–	–	–	–
Special price < Tier 1	–	–	–	–	F	T	–	–
Special price < Tier 2	–	–	–	–	–	–	F	T
Out	Edu discount	Special price	No discount	Special price	Tier 1 discount	Special price	Tier 2 discount	Special Price

Condition Coverage

- *Basic criterion:* Test every column
  "Don't care" entries (–) can take arbitrary values
- *Compound criterion:* Test every combination
  Requires 2^n tests for n conditions and is unrealistic
- *Modified condition decision criterion (MCDC):* like basic criterion, but additionally, modify each T/F value at least once *such that the outcome changes*
  Again, a good coverage criterion to aim for

MCDC Criterion

	Education		Individual					
Education account	F	T	F	F	F	F	F	F
Current purchase > Threshold 1	–	–	F	F	T	T	–	–
Current purchase > Threshold 2	–	–	–	–	F	F	T	T
Special price < scheduled price	F	T	F	T	–	–	–	–
Special price < Tier 1	–	–	–	–	F	T	–	–
Special price < Tier 2	–	–	–	–	–	–	F	T
Out	Edu discount	Special price	No discount	Special price	Tier 1 discount	Special price	Tier 2 discount	Special Price

MCDC Criterion

	Education		Individual					
Education account	T	T	F	F	F	F	F	F
Current purchase > Threshold 1	–	–	F	F	T	T	–	–
Current purchase > Threshold 2	–	–	–	–	F	F	T	T
Special price < scheduled price	T	T	F	T	–	–	–	–
Special price < Tier 1	–	–	–	–	F	T	–	–
Special price < Tier 2	–	–	–	–	–	–	F	T
Out	Edu discount	Special price	No discount	Special price	Tier 1 discount	Special price	Tier 2 discount	Special Price

MCDC Criterion

	Education		Individual					
Education account	T	F	F	F	F	F	F	F
Current purchase > Threshold 1	–	–	F	F	T	T	–	–
Current purchase > Threshold 2	–	–	–	–	F	F	T	T
Special price < scheduled price	F	T	F	T	–	–	–	–
Special price < Tier 1	–	–	–	–	F	T	–	–
Special price < Tier 2	–	–	–	–	–	–	F	T
Out	Edu discount	Special price	No discount	Special price	Tier 1 discount	Special price	Tier 2 discount	Special Price

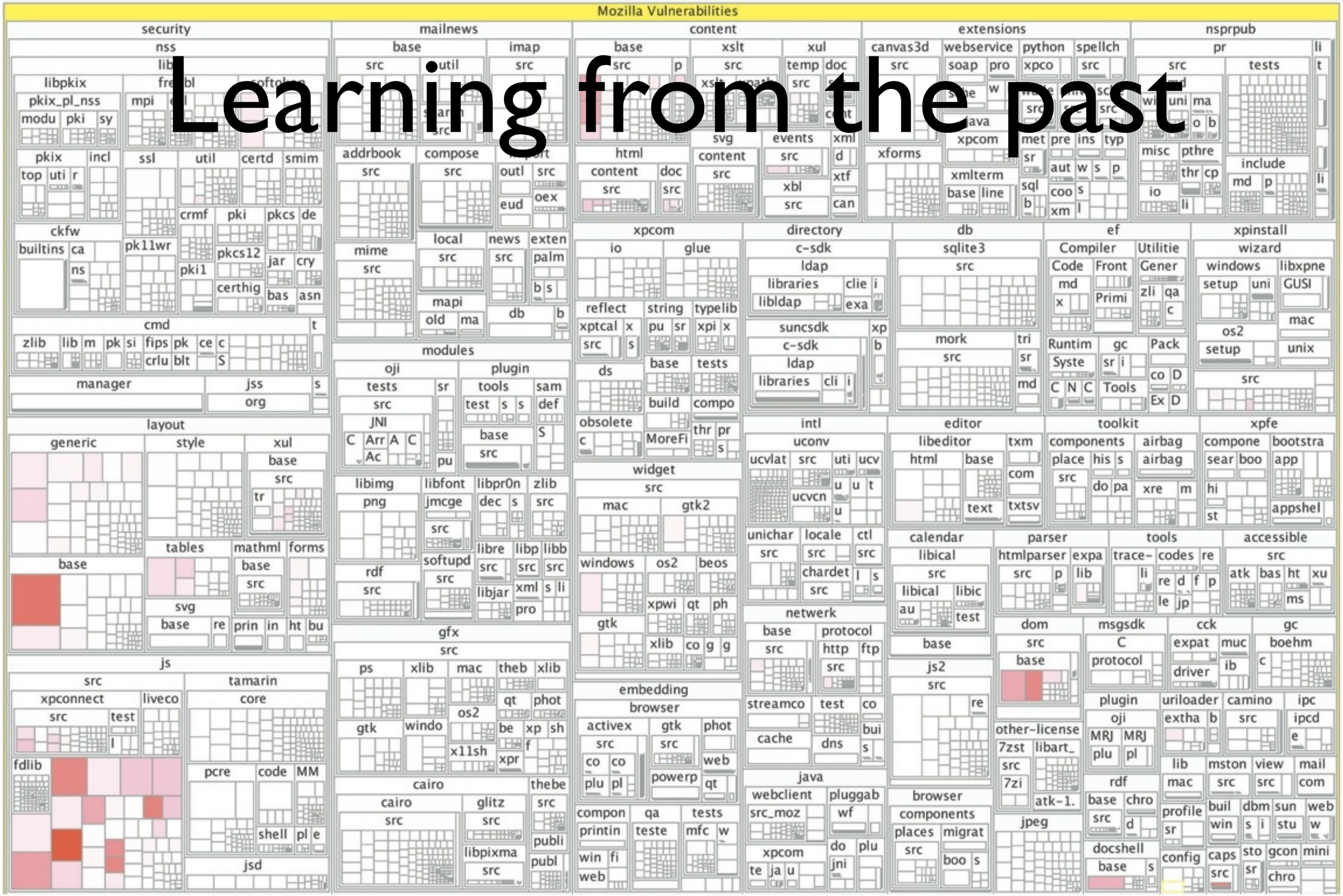
MCDC Criterion

	Education		Individual					
Education account	T	T	F	F	F	F	F	F
Current purchase > Threshold 1	–	–	F	F	T	T	–	–
Current purchase > Threshold 2	–	–	–	–	F	F	T	T
Special price < scheduled price	F	F	F	T	–	–	–	–
Special price < Tier 1	–	–	–	–	F	T	–	–
Special price < Tier 2	–	–	–	–	–	–	F	T
Out	Edu discount	Special price	No discount	Special price	Tier 1 discount	Special price	Tier 2 discount	Special Price

Weyuker's Hypothesis

The adequacy of a coverage criterion can only be intuitively defined.

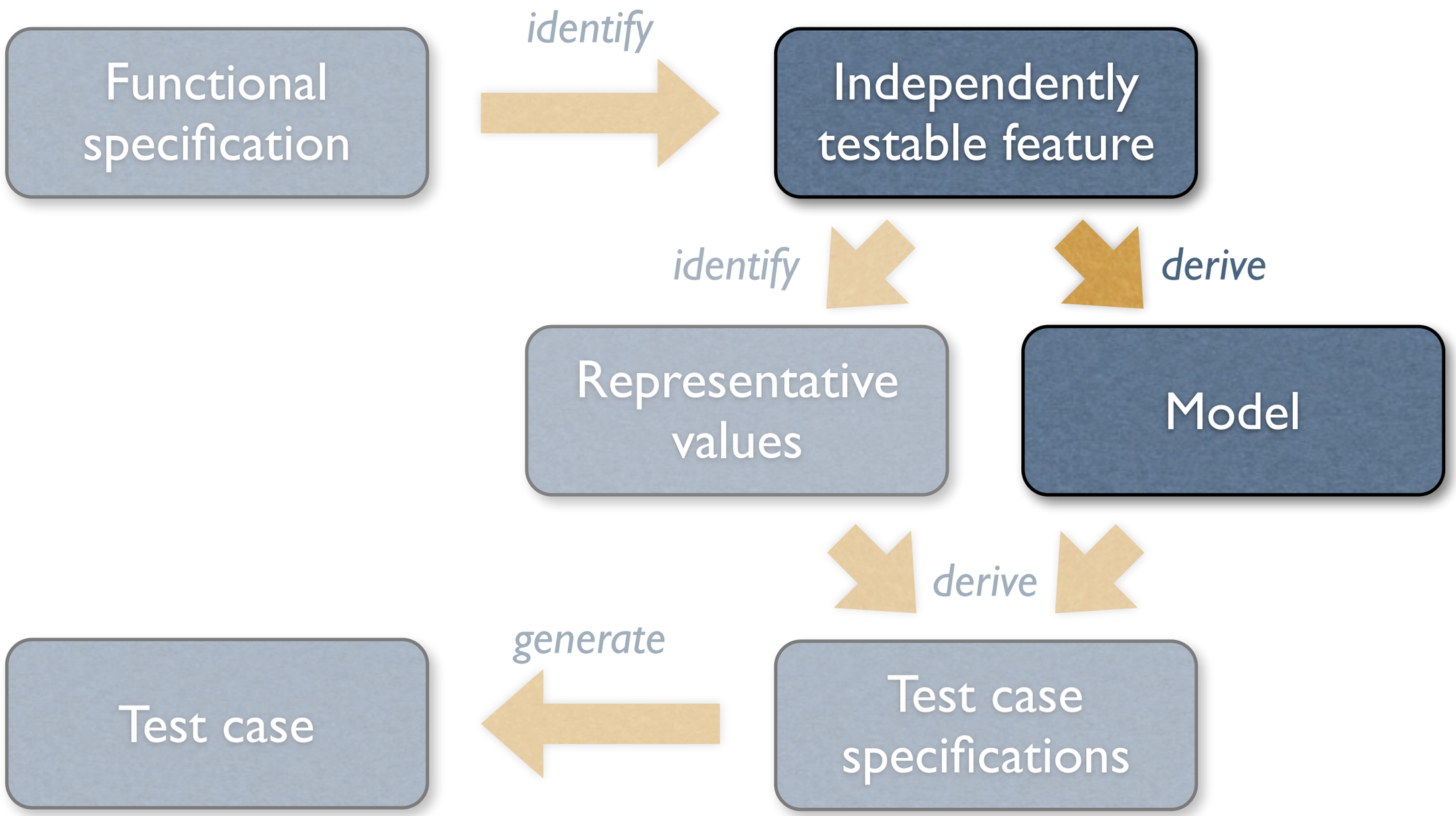
Learning from the past



Pareto's Law

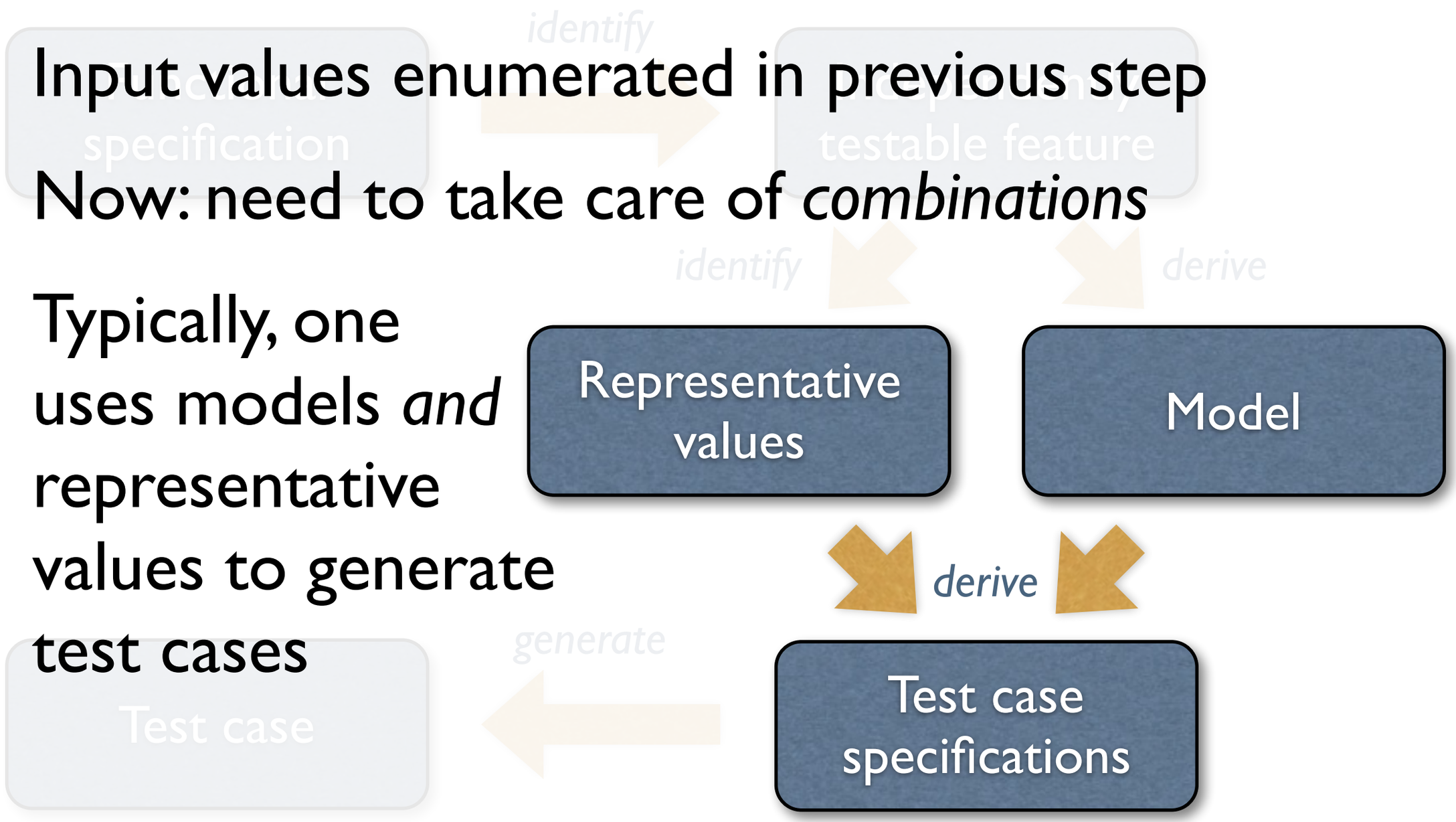
Approximately 80% of defects come from 20% of modules

Model-Based Testing

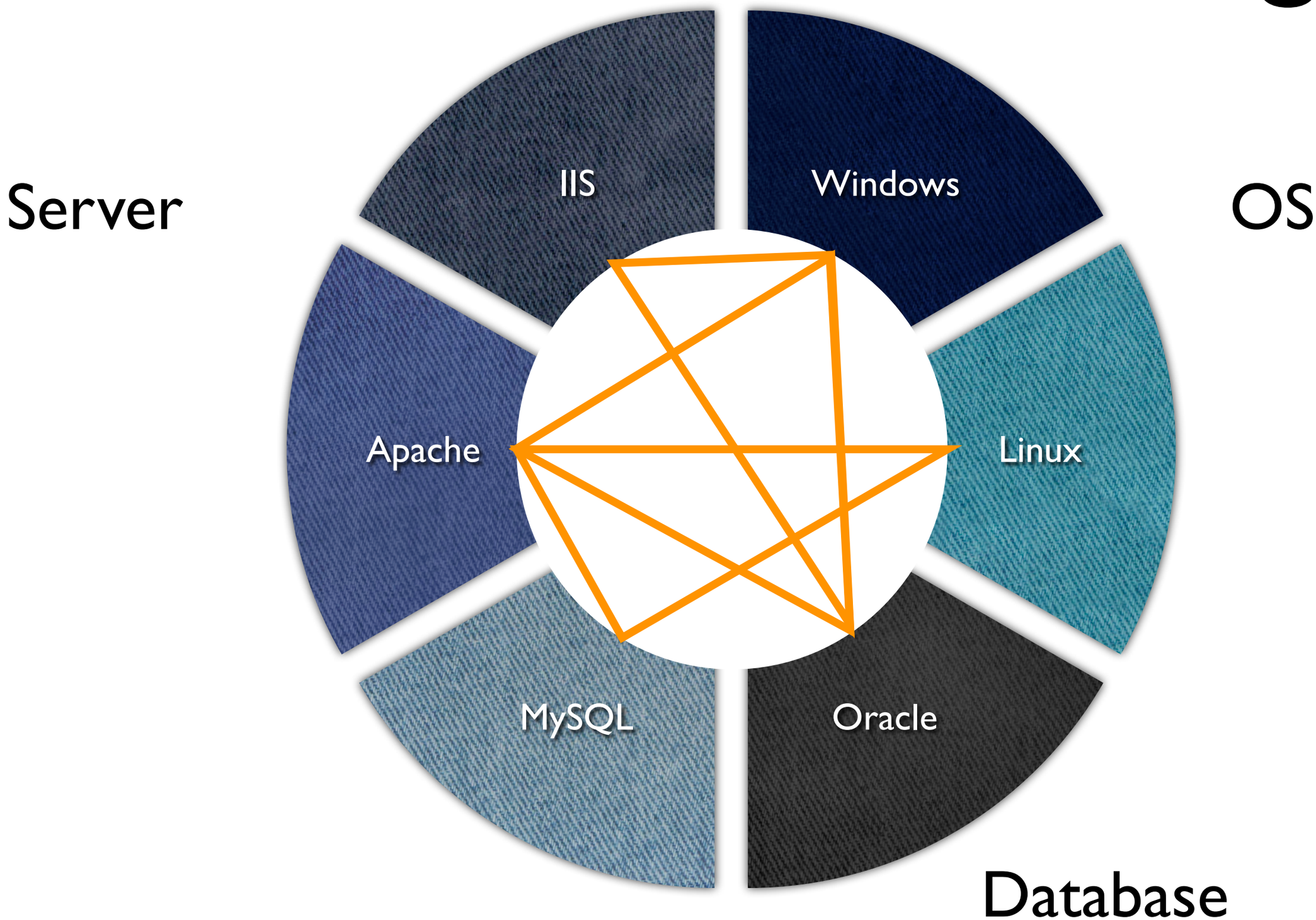


Deriving Test Case Specs

- Input values enumerated in previous step
- Now: need to take care of *combinations*
- Typically, one uses models *and* representative values to generate test cases



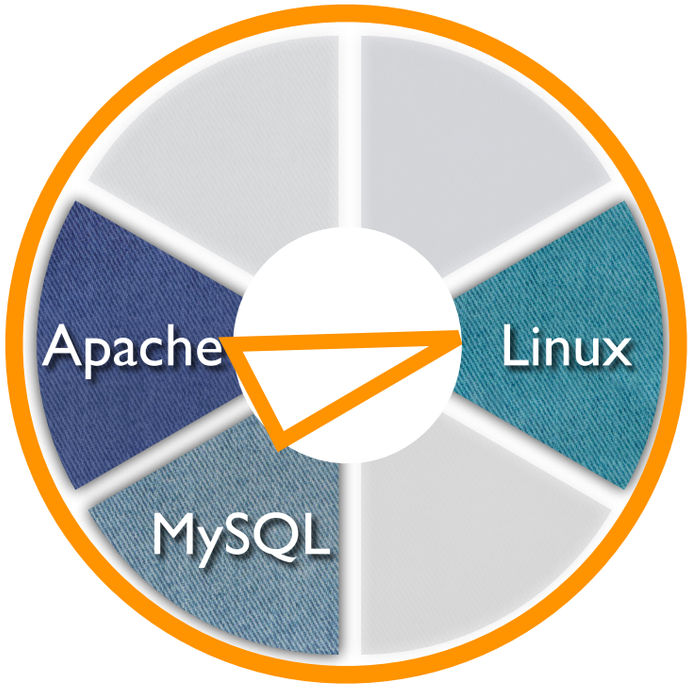
Combinatorial Testing

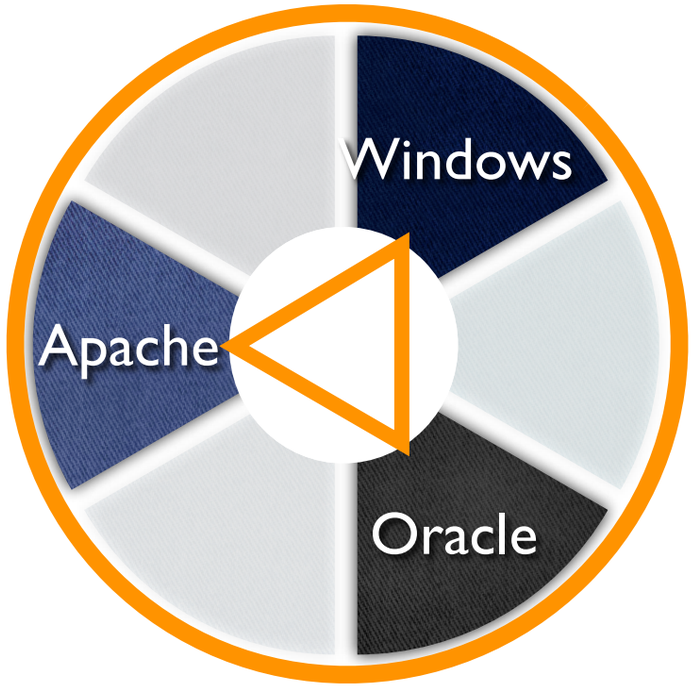


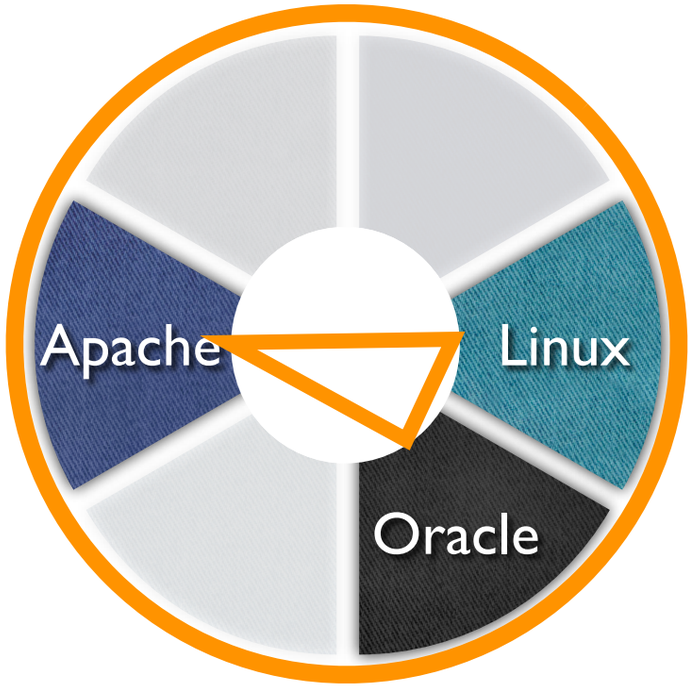
Combinatorial Testing

- Eliminate invalid combinations
  IIS only runs on Windows, for example
- Cover *all pairs* of combinations
  such as MySQL on Windows and Linux
- Combinations typically generated automatically
  and – hopefully – tested automatically, too

Pairwise Testing



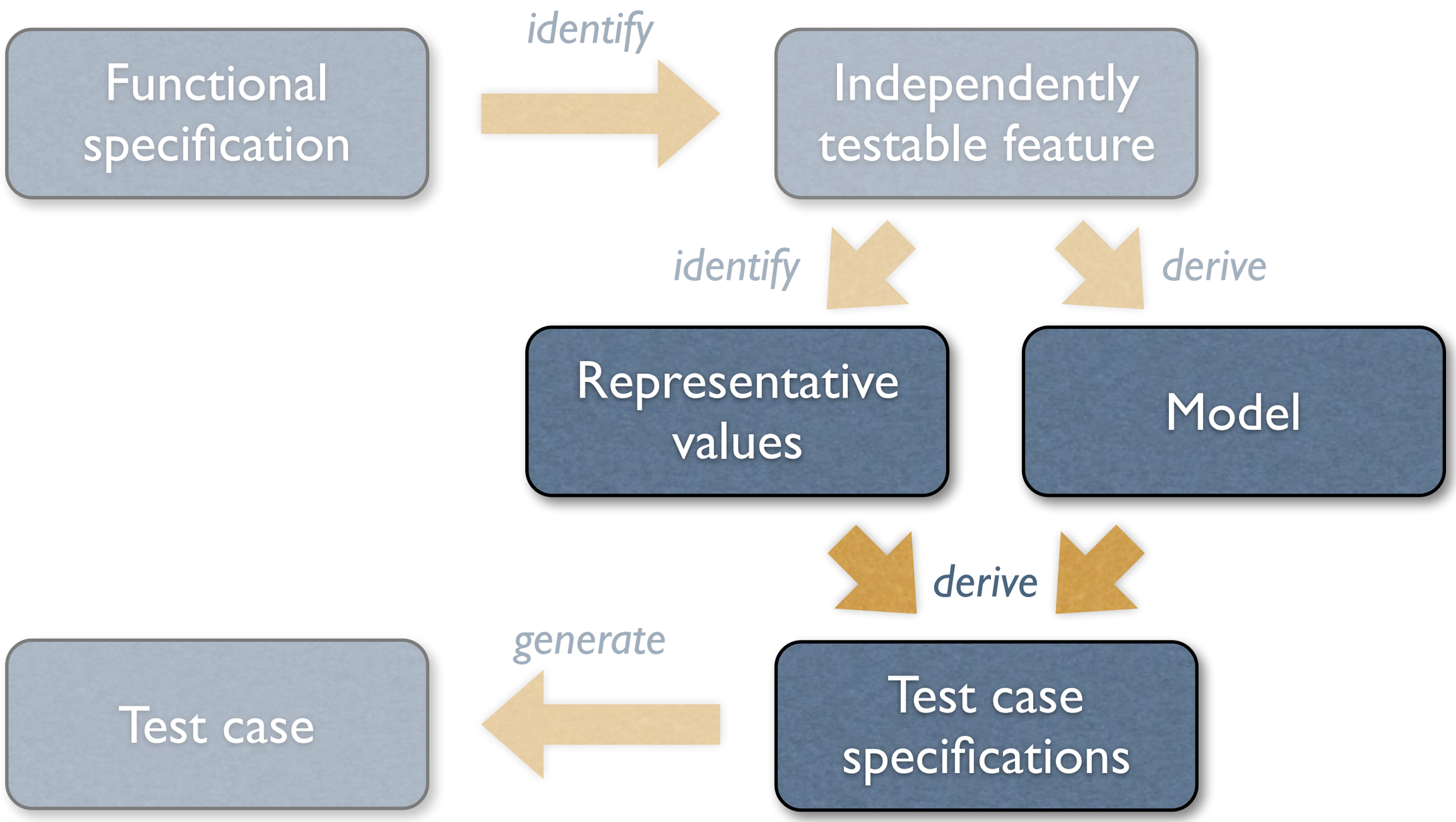




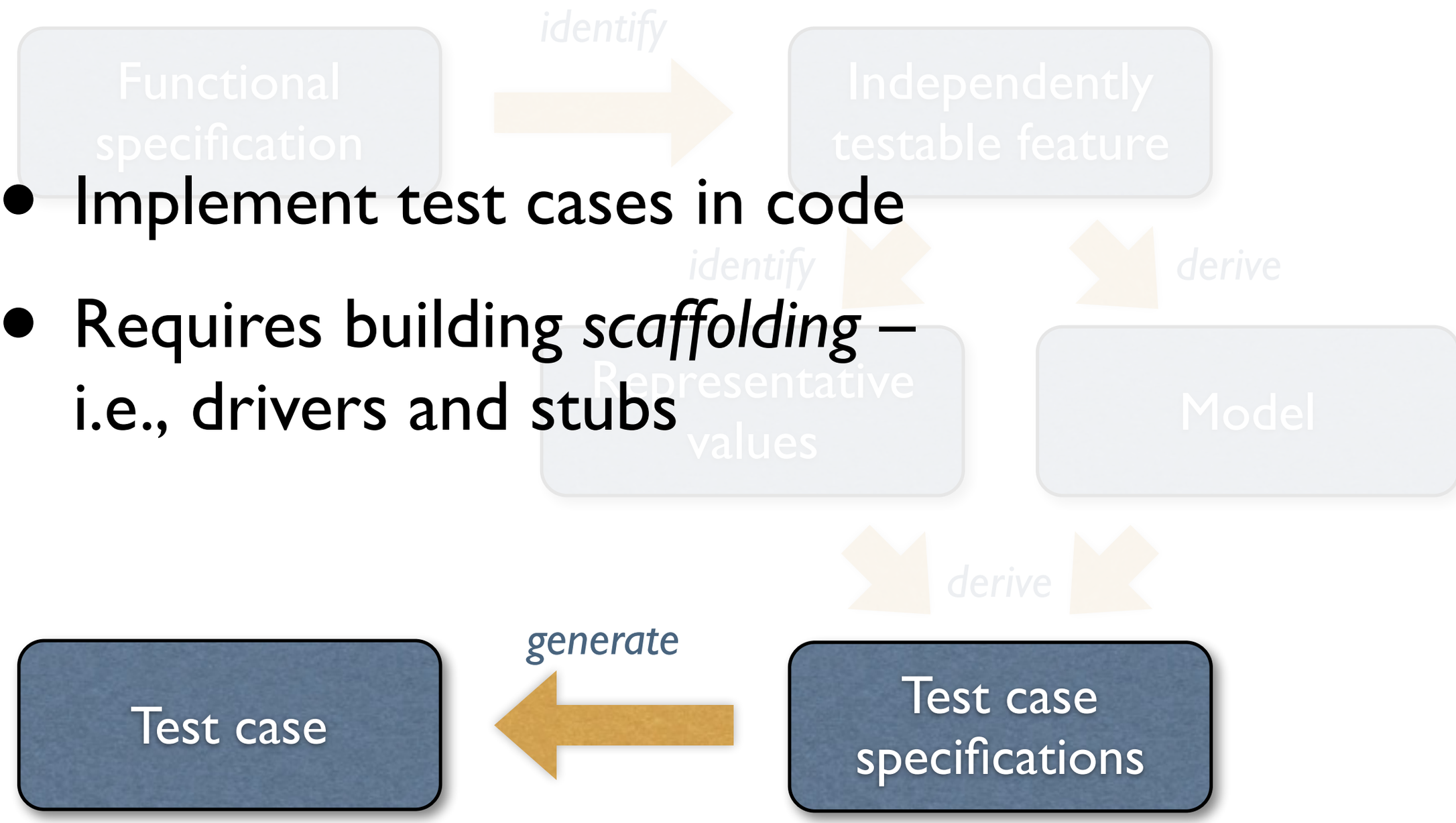
Testing environment

- Millions of configurations
- Testing on dozens of different machines
- All needed to find & reproduce problems

Deriving Test Case Specs



Deriving Test Cases



- Implement test cases in code
- Requires building *scaffolding* – i.e., drivers and stubs

Unit Tests

- Directly access units (= classes, modules, components…) at their programming interfaces
- Encapsulate a set of tests as a single syntactical unit
- Available for all programming languages (JUNIT for Java, CPPUNIT for C++, etc.)

Running a Test

A test case…

1. *sets up an environment for the test*
2. *tests* the unit
3. *tears down* the environment again.

Testing a URL Class

http://www.askigor.org/status.php?id=sample

Protocol Host Path Query

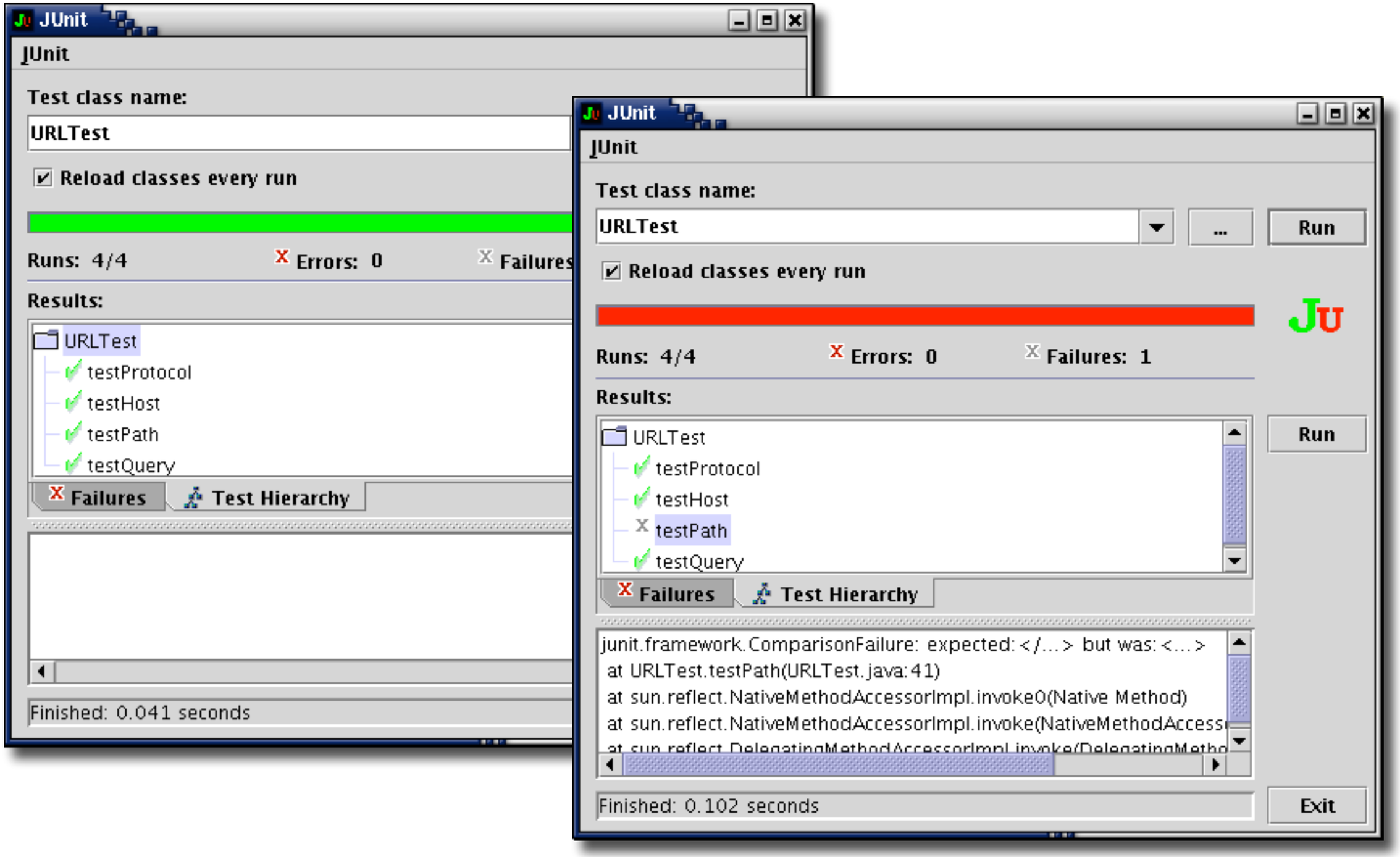
```java
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class URLTest extends TestCase {
    private URL askigor_url;

    // Create new test
    public URLTest(String name) { super(name); }

    // Assign a name to this test case
    public String toString() { return getName(); }

    // Setup environment
    protected void setUp() {
        askigor_url = new URL("http://www.askigor.org/" +
                              "status.php?id=sample"); }
    // Release environment
    protected void tearDown() { askigor_url = null;}
```

```java
// Test for protocol (http, ftp, etc.)
public void testProtocol() {
  assertEquals(askigor_url.getProtocol(), "http");
}

// Test for host
public void testHost() {
  int noPort = -1;
  assertEquals(askigor_url.getHost(), "www.askigor.org");
  assertEquals(askigor_url.getPort(), noPort);
}

// Test for path
public void testPath() {
  assertEquals(askigor_url.getPath(), "/status.php");
}

// Test for query part
public void testQuery() {
  assertEquals(askigor_url.getQuery(), "id=sample");
}
```

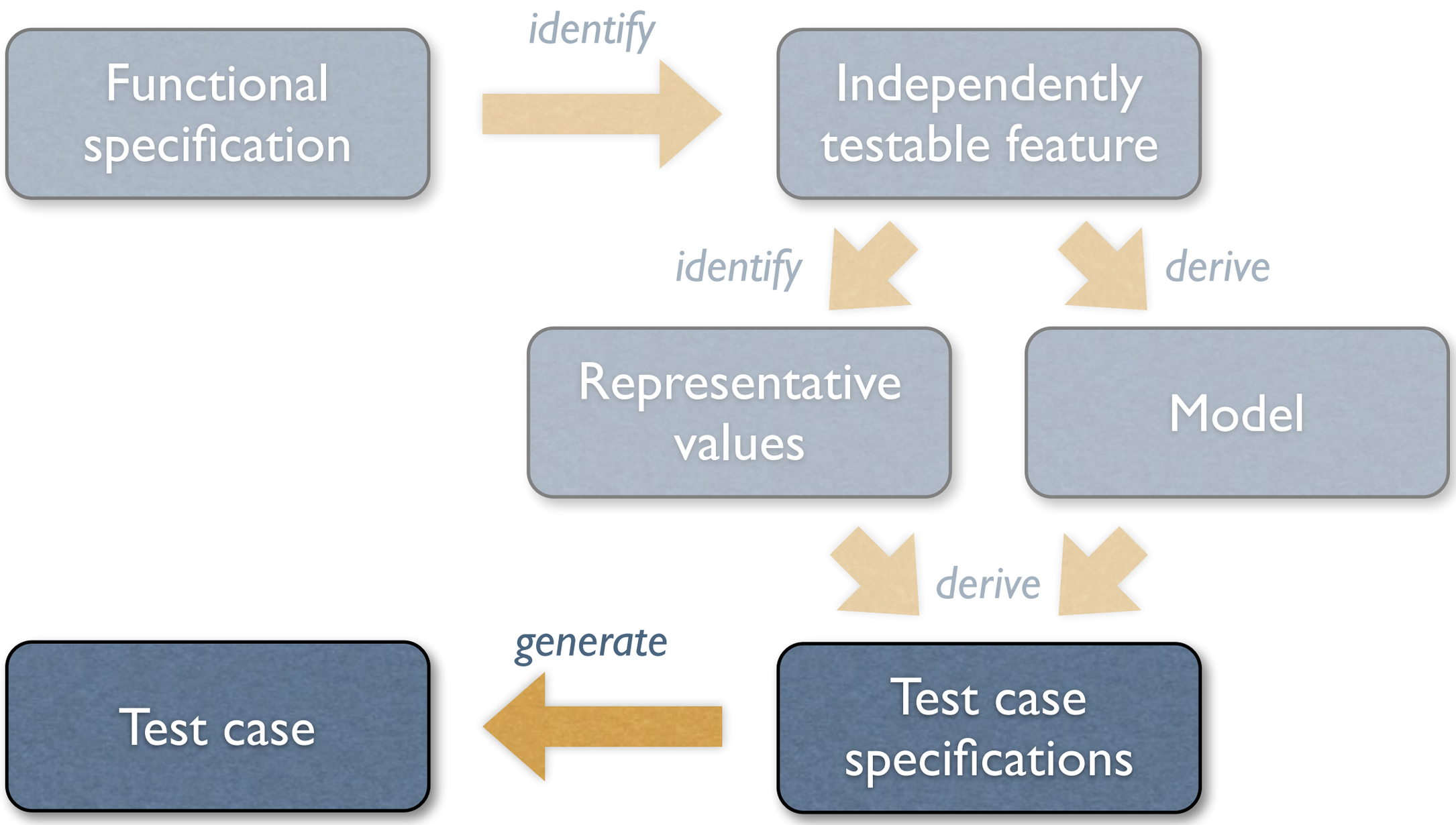
This functional test can be used as a *specification!*

```
    // Set up a suite of tests
    public static Test suite() {
        TestSuite suite = new TestSuite(URLTest.class);
        return suite;
    }

    // Main method: Invokes GUI
    public static void main(String args[]) {
        String[] testCaseName =
            { URLTest.class.getName() };
        // junit.textui.TestRunner.main(testCaseName);
        junit.swingui.TestRunner.main(testCaseName);
        // junit.awtui.TestRunner.main(testCaseName);
    }
}
```

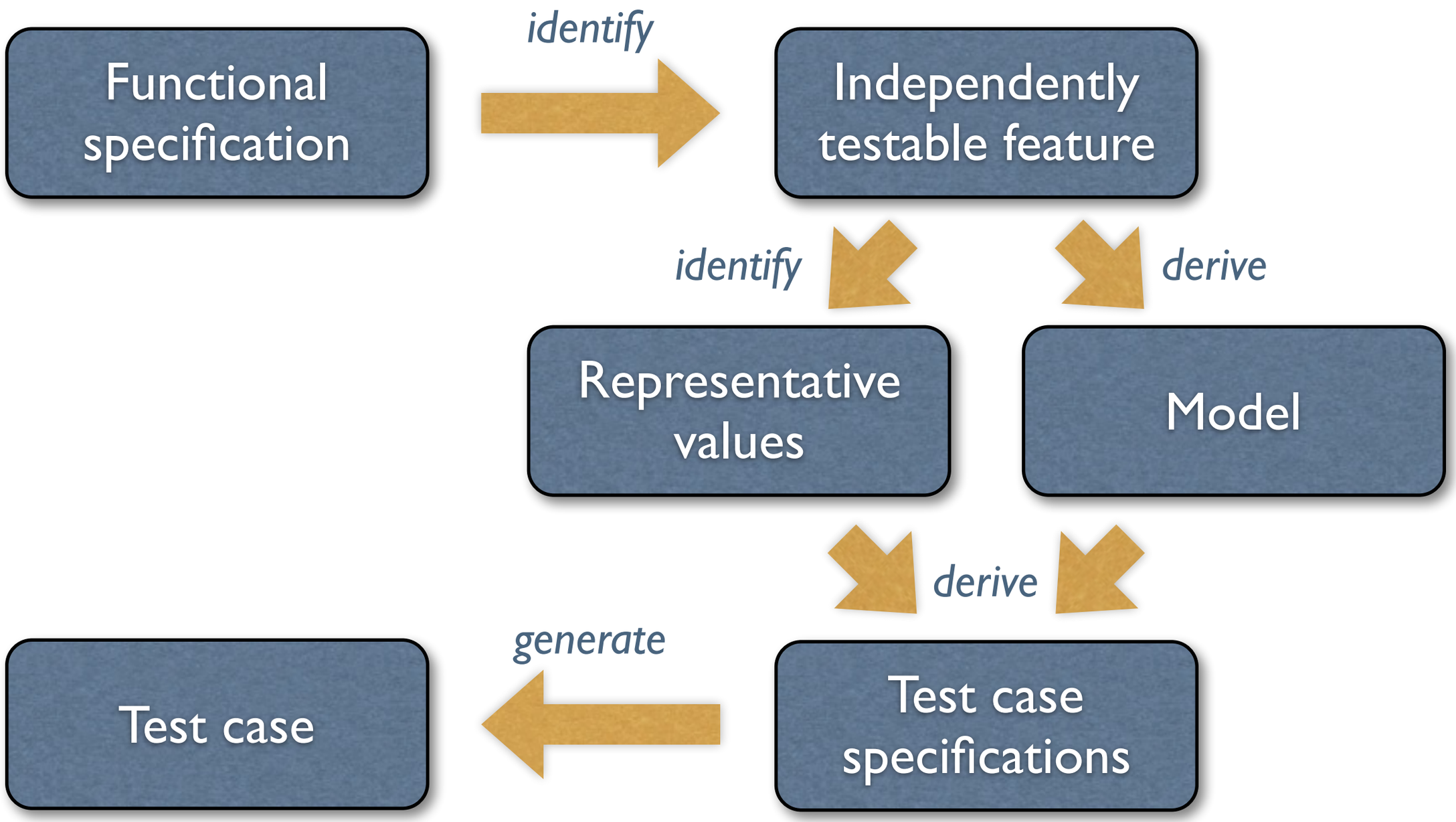
JUnit


JUnit
JUnit
Test class name:
URLTest
Reload classes every run
Runs: 4/4
Errors: 0
Failures
Results:
URLTest
testProtocol
testHost
testPath
testQuery
Failures
Test Hierarchy
Finished: 0.041 seconds
JUnit
JUnit
Test class name:
URLTest
...
Run
Reload classes every run
Runs: 4/4
Errors: 0
Failures: 1
Results:
URLTest
testProtocol
testHost
testPath
testQuery
Run
Failures
Test Hierarchy
junit.framework.ComparisonFailure: expected:</...> but was:<...>
at URLTest.testPath(URLTest.java:41)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccess
Finished: 0.102 seconds
Exit

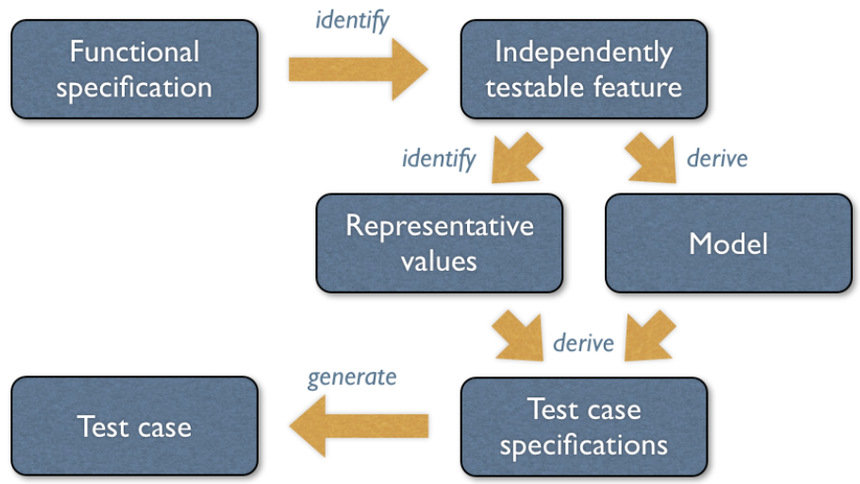
Deriving Test Cases



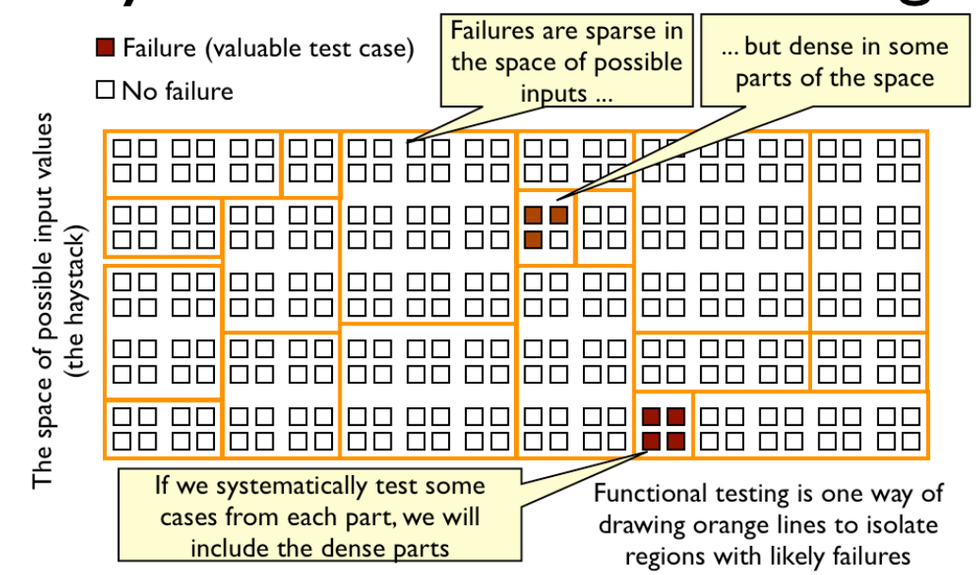
Systematic Functional Testing



Systematic Functional Testing



Systematic Partition Testing



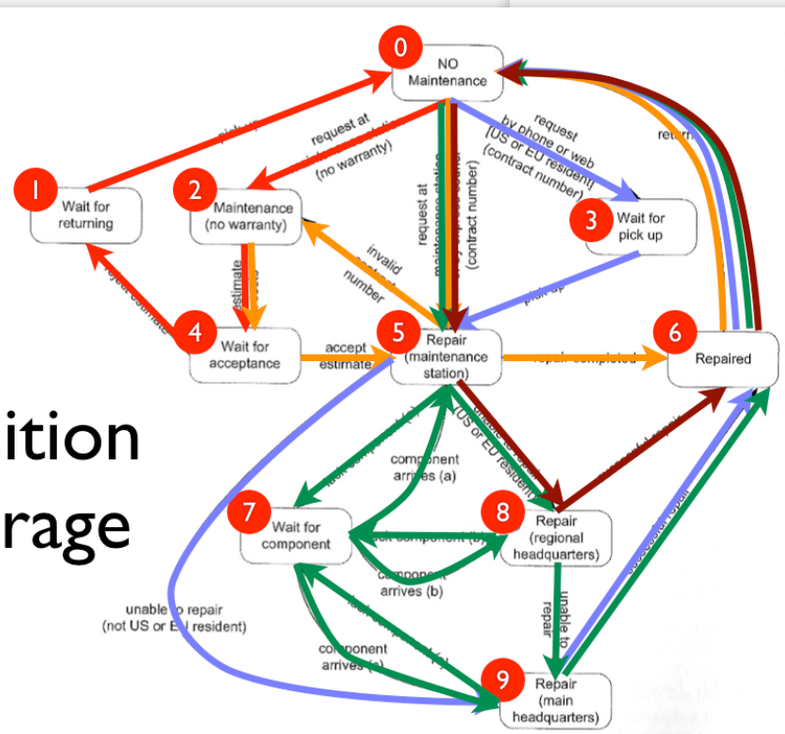
esting Tactic

on *spec*

as much
havior

- Tests bas
- Test cove
implemen
as possib

Transition Coverage



MC/DC Criterion

	Education		Individual					
Education account	F	T	F	F	F	F	F	F
Current purchase > Threshold 1	–	–	F	F	T	T	–	–
Current purchase > Threshold 2	–	–	–	–	F	F	T	T
Special price < scheduled price	F	T	F	T	–	–	–	–
Special price < Tier 1	–	–	–	–	F	T	–	–
Special price < Tier 2	–	–	–	–	–	–	F	T
Out	Edu discount	Special price	No discount	Special price	Tier 1 discount	Special price	Tier 2 discount	Special Price