Model-Driven Testing for Agile Teams

Kerry Kimbrough
Cornutum Project

This Workshop Is About...

- **A design technique...**
  - Input space modeling
- **Using a tool...**
  - Tcases, a model-driven test case generator
- **To tackle test design issues...**

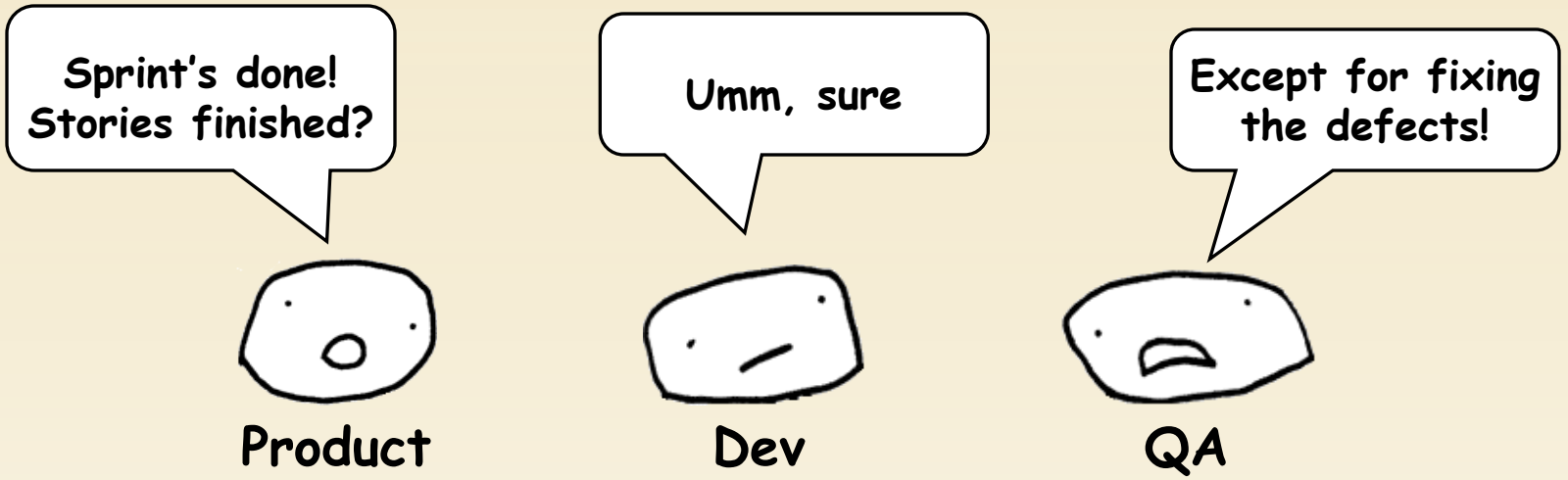
Test Design Issues

- **How do I pick test cases?**
  - Define an input model. Then generate test cases.
- **How many test cases do I need?**
  - Define a coverage model. Then generate a minimal test set.
- **Are my tests good enough?**
  - Is my input model good enough?
  - Is my coverage model good enough?

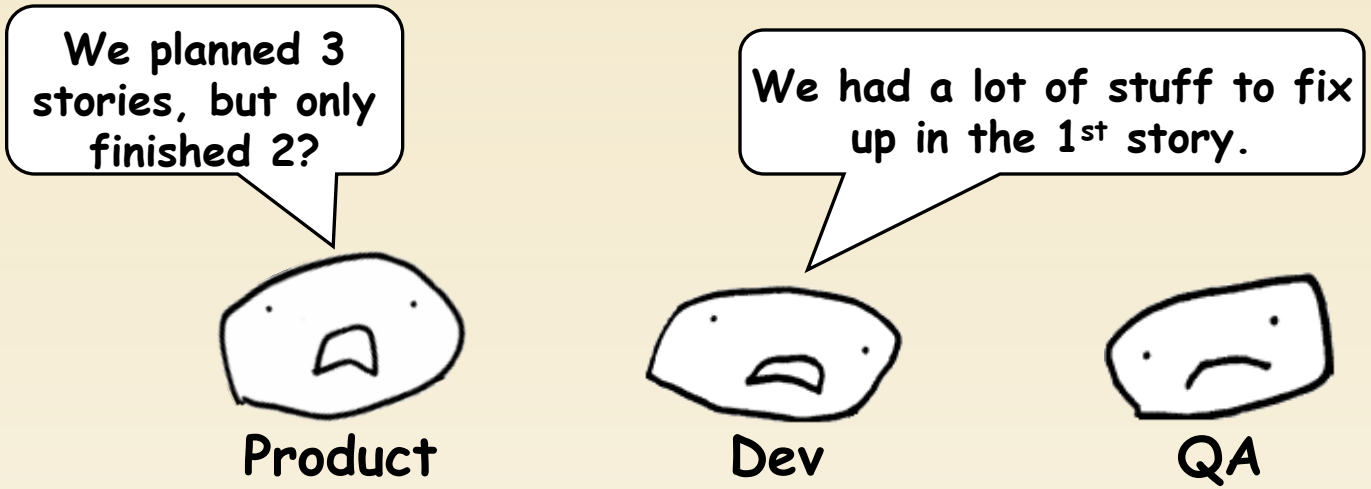
Why Does It Matter?

- **Defects are a drag!**
  - The friction that slows delivery
- **The Agile way**
  - Test early
  - Test continuously
- **And yet...**

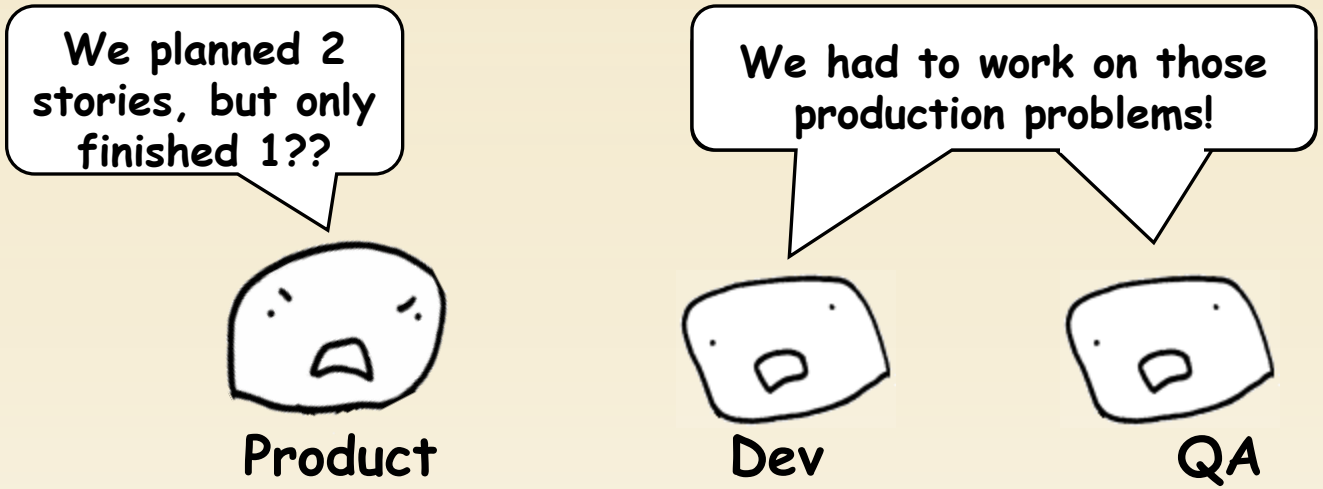
The Curse of the Undone



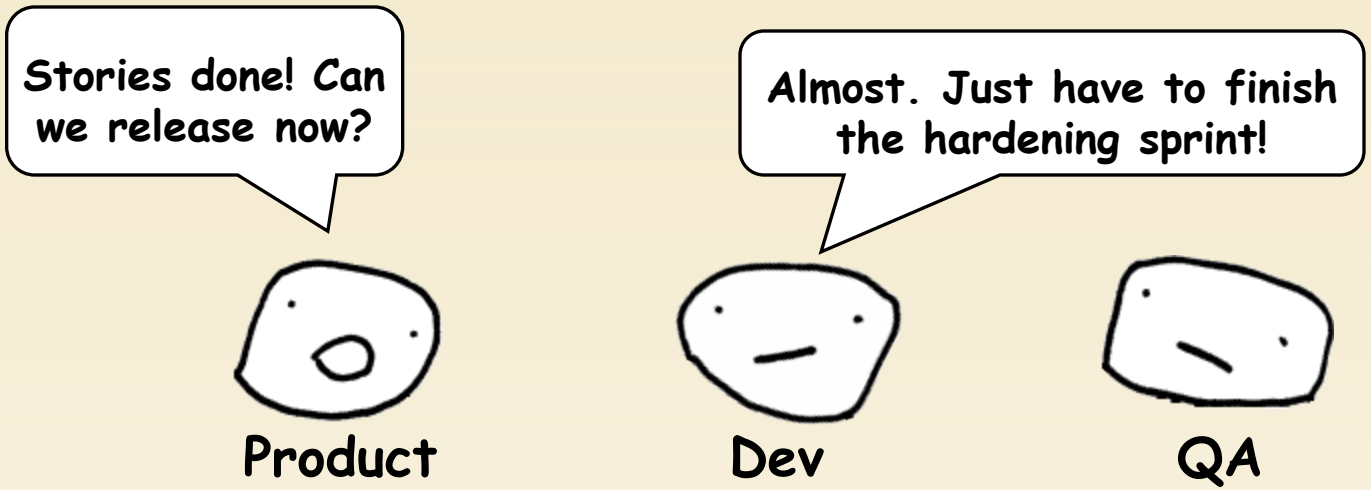
The Busted Sprint



The Sprint That Never Was



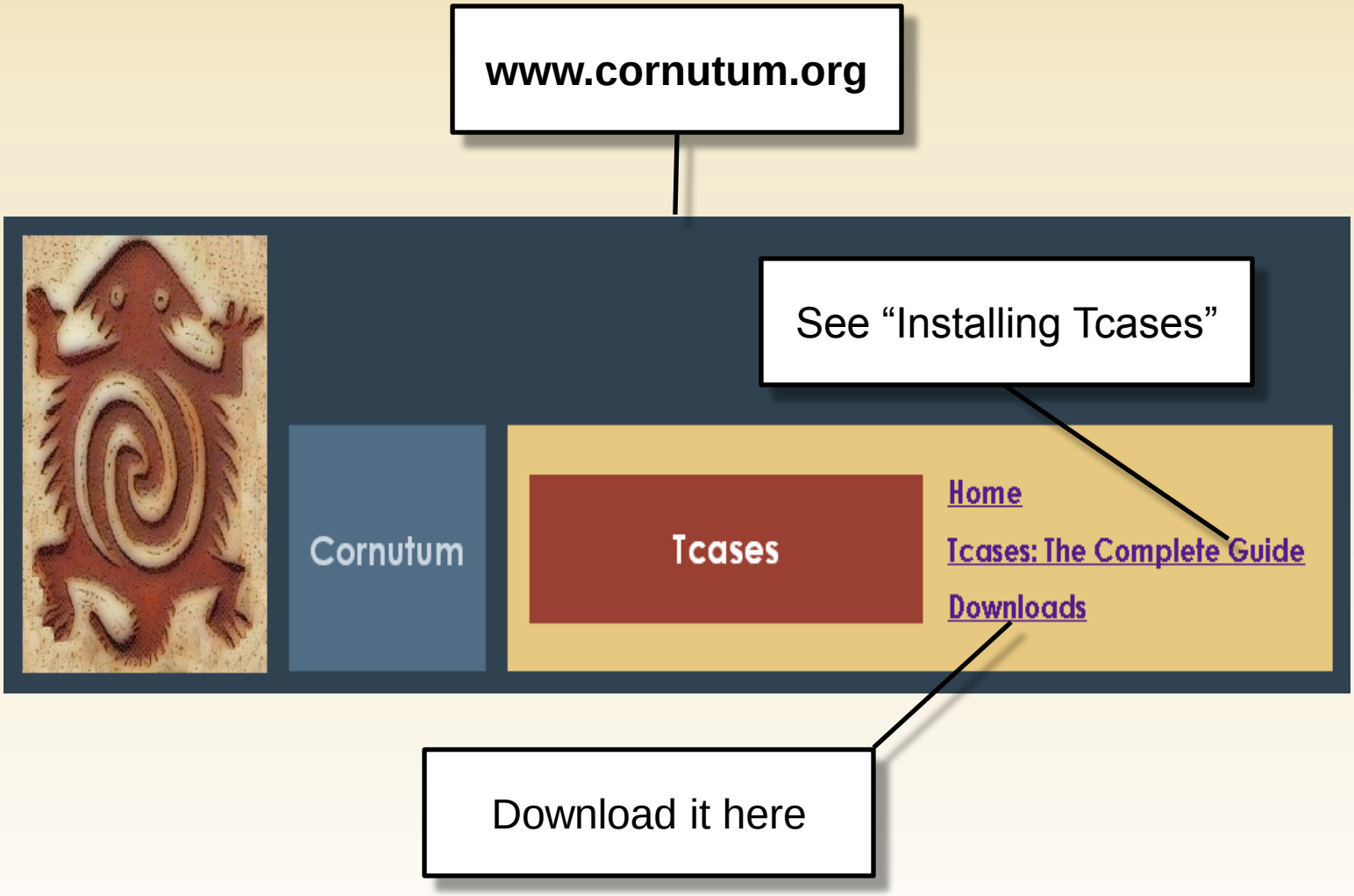
The Hardening (?!) Sprint



Why Model-Driven Testing?

- **Much more powerful tests**
- **For little or no extra effort**
- **At every step of the way**
  - Before dev
  - During dev
  - During system test

Tcases: How To Get It



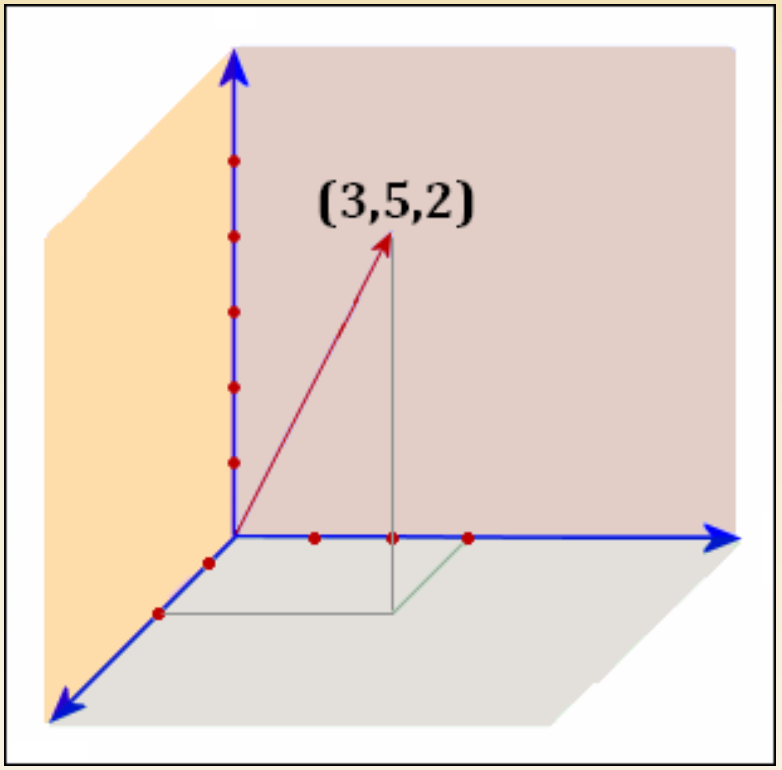
Example: find <*pattern*> <*file*>

Locates one or more instances of a given pattern in a text file.

All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the number of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file. To include a blank in the pattern, the entire pattern must be enclosed in quotes. To include a quotation mark in the pattern, two quotes in a row ("") must be used.

Modeling The Input Space



- All combinations of values
- For each dimension of variation

Defining System Functions

```
<System name="Examples"
    <!-- All system function definitions go here -->

    <Function name="find"
        <!-- All input definitions for "find" go here -->
    </Function>

</System>
```

Defining Input Variables

```
<Function name="find“>
   <Input type="arg">
      <!--arg: Direct input arguments (the default)-->
      <Var name="fileName">
        ...
      </Var>
      ...
   </Input>
   <Input type="env">
      <!--env: Environment state variables-->
      ...
   </Input>
</Function>
```

Defining Input Values

```
<Function name="find"
   <Input type="arg">
      <Var name="fileName"
         <!-- The required file name is defined -->
         <Value name="defined"/>
         <!-- The required file name is missing: an error -->
         <Value name="missing" failure="true"/>
      </Var>
      ...
   </Input>
</Function>
```

To Define Input Values...

- **Enumerate all values**
- **Or identify "equivalence classes"**
  - A subset of values that are "test-equivalent"
  - No need to test all -- any one will do
  - Examples
    - Integer: Negative, Zero, Positive
    - Bounded: Below-Min, In-Range, Above-Max
    - List: Empty, One, Many

Modeling Complex Inputs

find <*pattern*> <*file*>

Locates one or more instances of a given pattern in a text file.

All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the number of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file. To include a blank in the pattern, the entire pattern must be enclosed in quotes. To include a quotation mark in the pattern, two quotes in a row ("") must be used.

Defining Variable Sets

```
<VarSet name="file">

    <!-- Does the file exist? -->
    <Var name="exists"> ... </Var>

    <!-- Does the file contain... -->
    <VarSet name="contents">

        <!-- ... any matching lines? -->
        <Var name="patterns"> ... </Var>

        <!-- ... multiple matches in a line? -->
        <Var name="patternsInLine"> ... </Var>

    </VarSet>
</VarSet>
```

Modeling Complex Inputs

find <*pattern*> <*file*>

Locates one or more instances of a given pattern in a text file.

All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the number of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file. To include a blank in the pattern, the entire pattern must be enclosed in quotes. To include a quotation mark in the pattern, two quotes in a row ("") must be used.

Defining Variable Sets

```
<VarSet name="pattern">

  <!-- How many chars in pattern? -->
  <Var name="size"> ... </Var>

  <!-- Is it quoted? -->
  <Var name="quoted"> ... </Var>

  <!-- Does it contain blanks? -->
  <Var name="blanks"> ... </Var>

  <!-- Does it contain embedded quotes? -->
  <Var name="embeddedQuotes"> ... </Var>

</VarSet>
```

System Input Model Basics

```
<System ...> <!-- A system has... -->
  <Function ...> <!-- functions, having... -->
    <Input ...>  <!-- different types of inputs: -->
      <Var ...>  <!-- simple variables, having... -->
        <Value ...> ...</Value>  <!-- (classes of) values -->
      </Var>
      <VarSet ...>  <!-- or complex inputs, composed of... -->
        <Var ...> ...</Var>  <!-- many variables... -->
        <VarSet ...> ...</VarSet>  <!-- and many levels -->
      </VarSet>
    </Input>
  </Function>
</System>
```

Defining Constraints

- **Values can have properties**
  - Assumed by any test case that includes this value
- **Values can have conditions**
  - Properties that must appear in any test case that includes this value.
  - Properties that must not appear in any test case that includes this value.

When Variables Are Irrelevant

```
<Function name="find">
  <Input type="arg">
    <VarSet name="pattern" when="fileExists">
      <Var name="blanks" whenNot="empty">
        ...
      </Var>
      ...
    </VarSet>
    ...
  </Input>
  ...
</Function>
```

Input Model vs. Coverage Model

- **Input model**
  - Defines the input space
  - Defines the input combinations possible
- **Coverage model**
  - Defines the input combinations tested

What Is Input Space Coverage?

- **Measures input combinations tested**
- **A "black box" coverage metric**
  - Specification-based
- **Complement to "white box" coverage**
  - Code-based
  - Measures lines/branches/conditions tested
- **Basic unit: an N-tuple**
  - Combination of values from N input variables

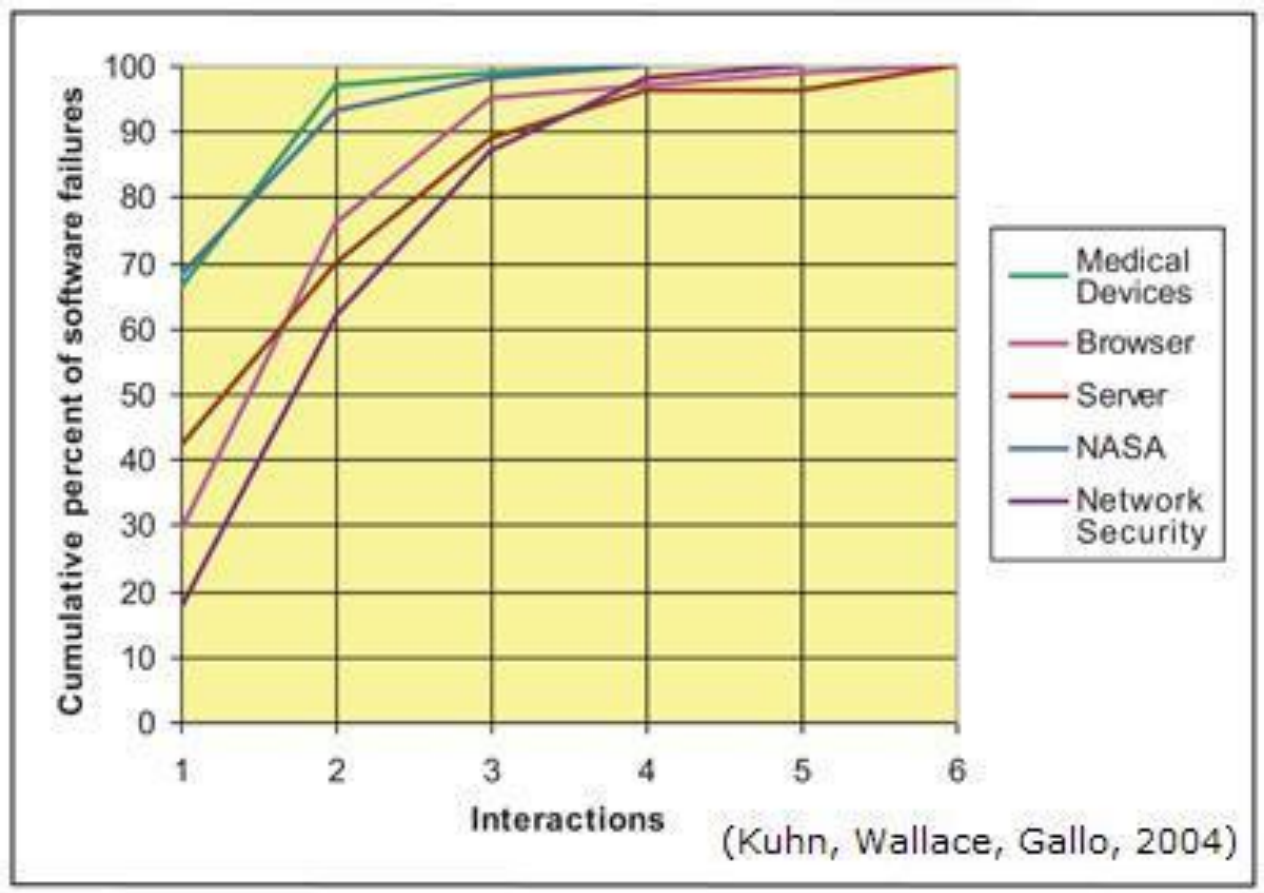
Default: 1-Tuple Coverage

```
<TestCase id="2">
 <Input type="arg">
  <Var name="pattern.size" value="manyChars"/>
  <Var name="pattern.quoted" value="yes"/>
  <Var name="pattern.blanks" value="many"/>
  <Var name="pattern.embeddedQuotes" value="none"/>
  <Var name="fileName" value="defined"/>
 </Input>
 <Input type="env">
  <Var name="file.exists" value="yes"/>
  <Var name="file.contents.linesLongerThanPattern" value="many"/>
  <Var name="file.contents.patterns" value="one"/>
  <Var name="file.contents.patternsInLine" value="one"/>
 </Input>
</TestCase>
```

2-Tuple (Pairwise) Coverage

```
<TestCase id="2">
 <Input type="arg">
  <Var name="pattern.size" value="manyChars"/>
  <Var name="pattern.quoted" value="yes"/>
  <Var name="pattern.blanks" value="many"/>
  <Var name="pattern.embeddedQuotes" value="none"/>
  <Var name="fileName" value="defined"/>
 </Input>
 <Input type="env">
  <Var name="file.exists" value="yes"/>
  <Var name="file.contents.linesLongerThanPattern" value="many"/>
  <Var name="file.contents.patterns" value="one"/>
  <Var name="file.contents.patternsInLine" value="one"/>
 </Input>
</TestCase>
```

Failures From Multiple Factors



Source: NIST http://csrc.nist.gov/groups/SNS/acts

Example: Web-Client.Config

```
<Function name="Config">
  <!-- Set up the configuration for a Web client test -->
  <Input>
    <Var name="Browser">
      <Value name="Firefox"/>
      <Value name="Chrome"/>
    </Var>
    <Var name="OS">
      <Value name="Linux"/>
      <Value name="Windows"/>
      <Value name="Mac-OS-X"/>
    </Var>
    <Var name="Flash">
      <Value name="Yes"/>
      <Value name="No"/>
    </Var>
  </Input>
</Function>
```

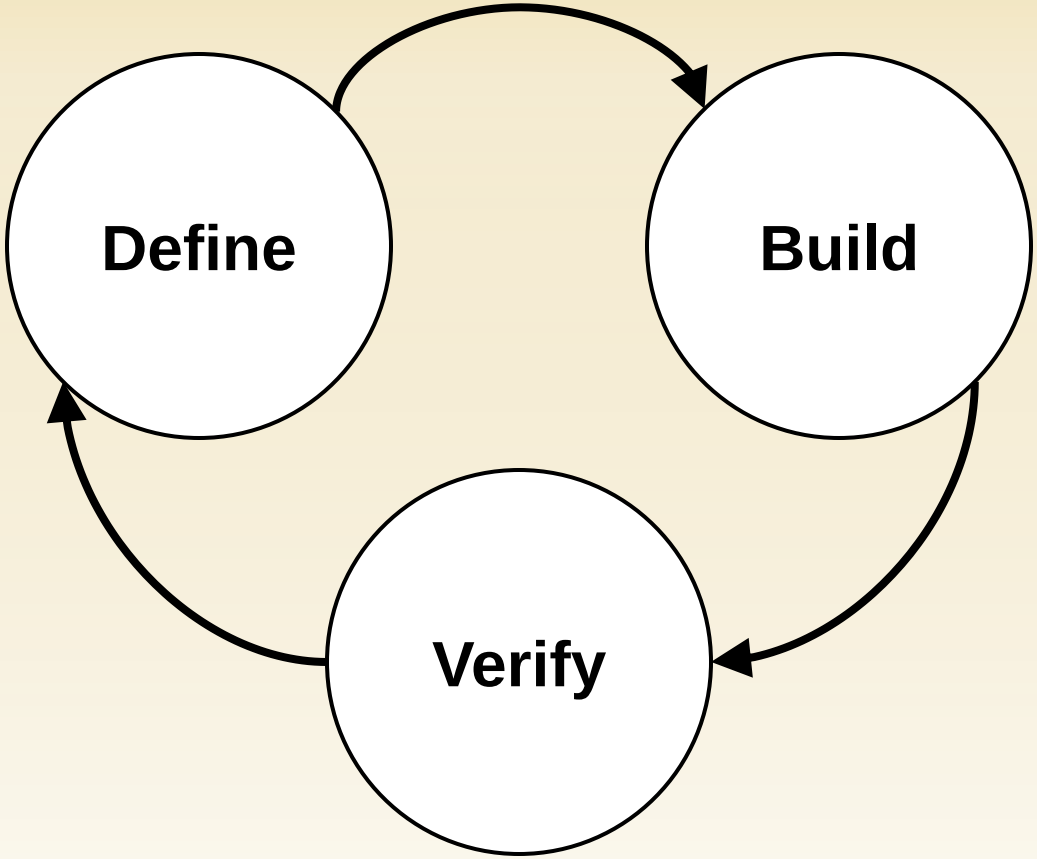
Example: Web-Client.Config

- **Default coverage (1-tuple)**
  - 7 tuples
  - 3 test cases
- **2-tuple coverage**
  - 16 tuples
  - 6 test cases (minimum)
- **All permutations**
  - 12 tuples
  - 12 test cases

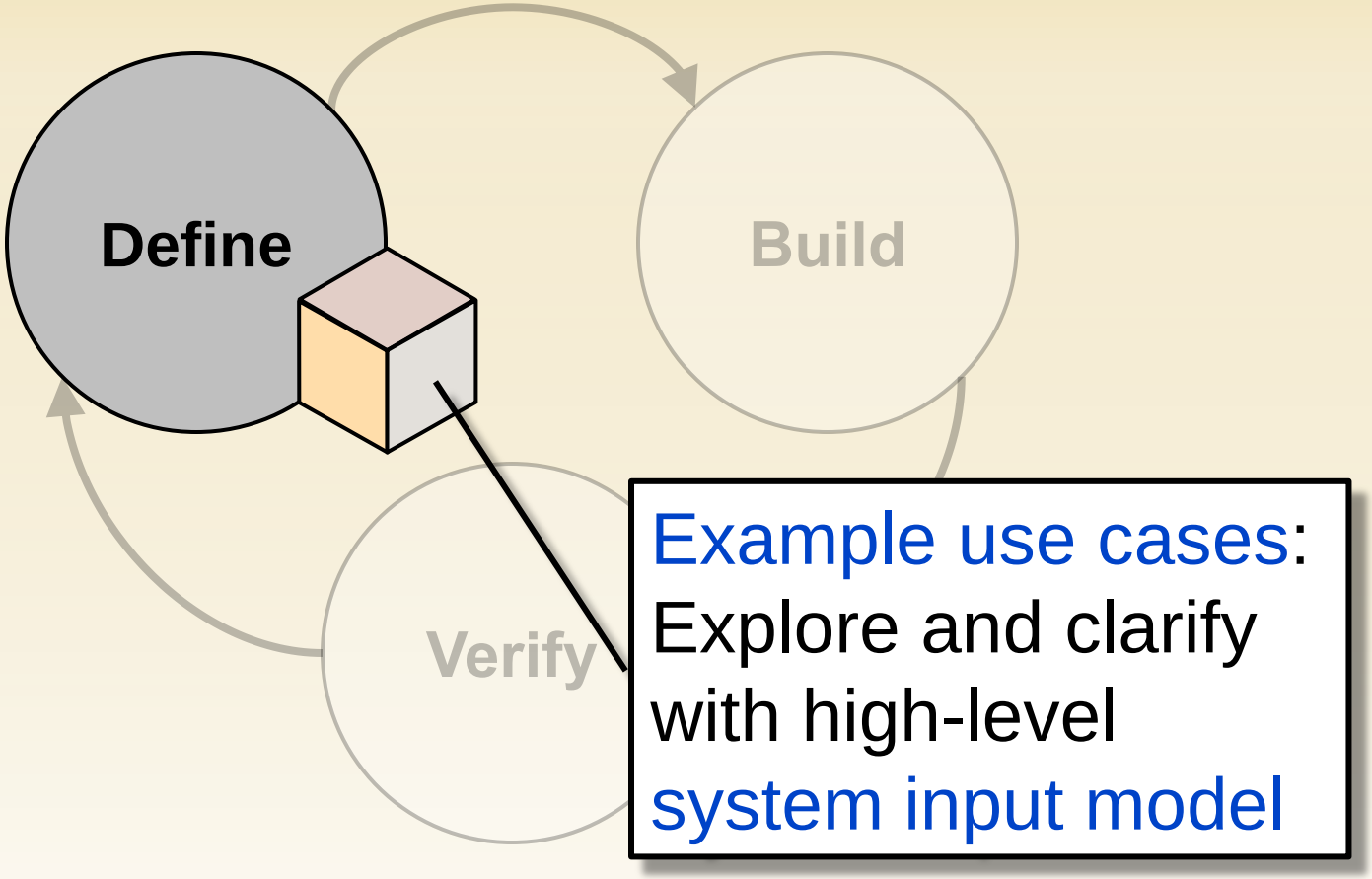
Tool Tips: Use Tcases To...

- **Define multiple coverage levels**
- **Reuse and extend previous test cases**
- **Create random combinations**
- **Simplify tests with once="true"**
- **Transform test case output**

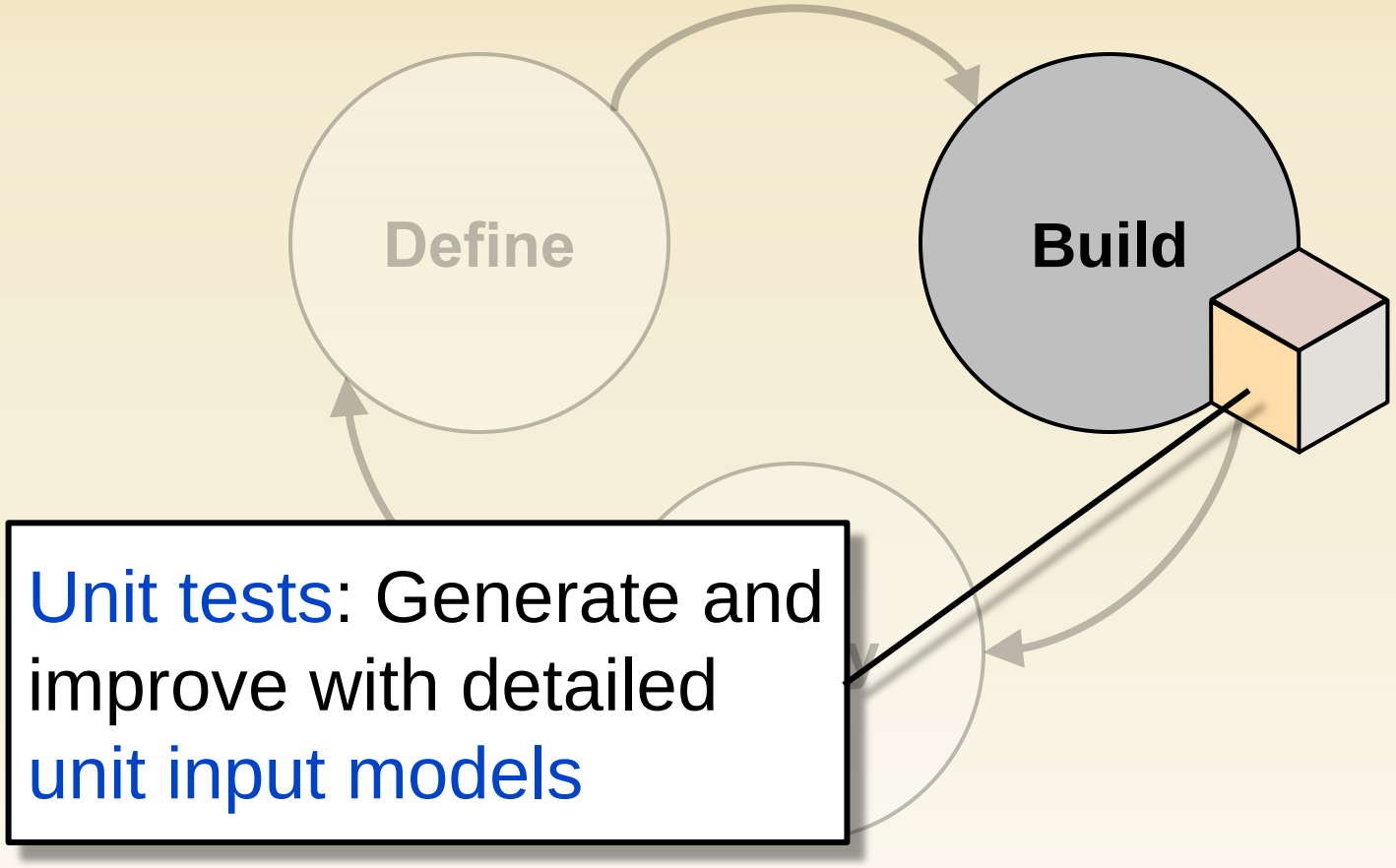
Model-Driven Testing Workflow



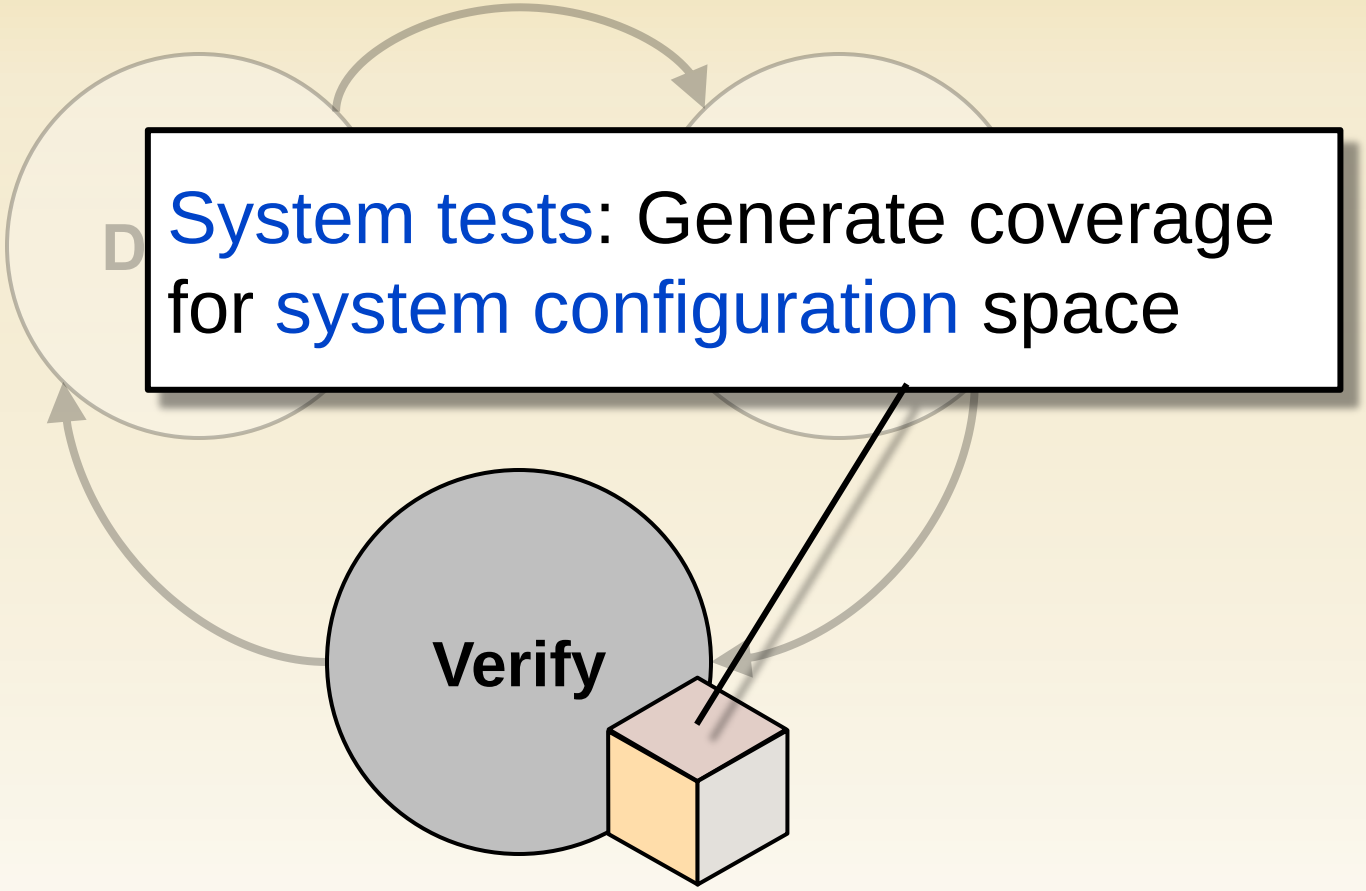
Model-Driven Testing Workflow



Model-Driven Testing Workflow



Model-Driven Testing Workflow



The End