USE CASE TESTING

Ross Collard

Collard & Company

Overview

- Welcome to this hands-on, learn-by-doing workshop.
- In the workshop we'll learn how to test a feature, or a set of related features, based on the system's use case(s). The question is: How can we utilize the use case to develop an effective set of test cases?

Issues We will Discuss

How to:

- Generate test cases from good use cases – *this is the primary focus of the workshop.*
- Recognize weak use cases and decide whether to clean them up for testing.
- Transform weak use cases into testable ones.
- Develop good use cases in the first place.

Topics: Testing Techniques

- **Decision-Driven Testing**
- **Equivalence**
- **Input-Based Testing**
- **Risk-Based Testing**
- **Feature Interaction Testing**

Disclaimer: we will not have the time to cover these all in depth.

So, Why Should I Get Excited?

- “So what? These techniques don’t apply to me. I test GUIs, web sites, telecom network interfaces, embedded devices or databases for performance. Don’t tell about me this stuff you listed before, tell me what I need to know.”

What WON’T We Cover Today?

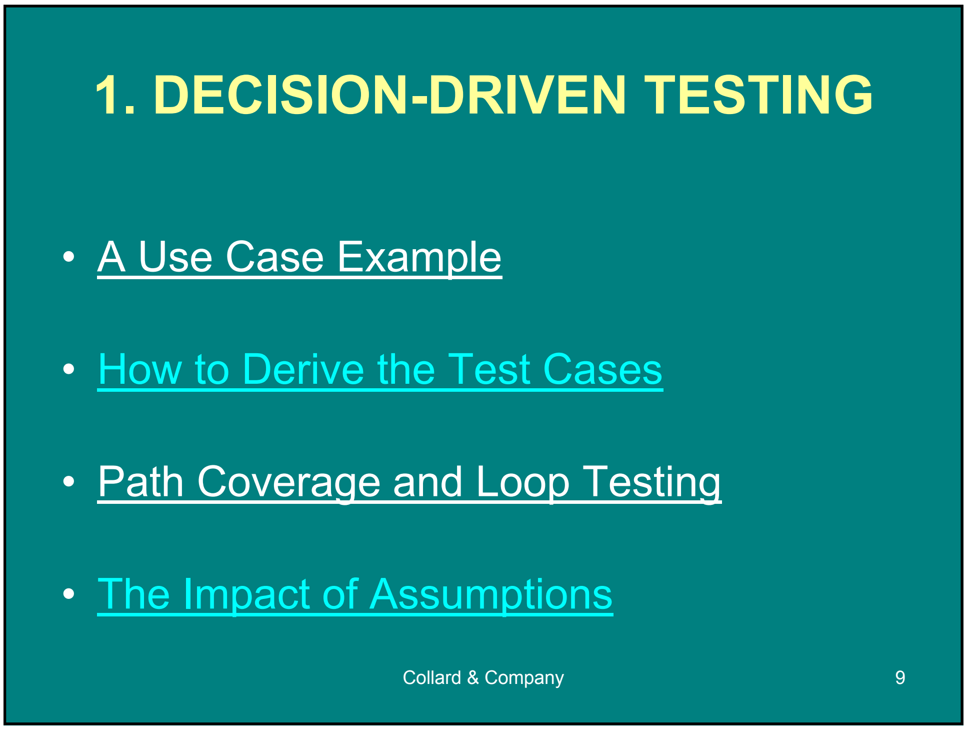
- Esoteric techniques, e.g., statistical methods.
- Applications of the techniques discussed in every possible situation.
- Every page of the book and every exercise (because of time limitations).

Getting Value

- **Continue to ask yourself during the workshop: how does each idea or technique apply in my particular situation?**
- **At the end of the workshop, plan to develop a list of your personal follow-up action items.**

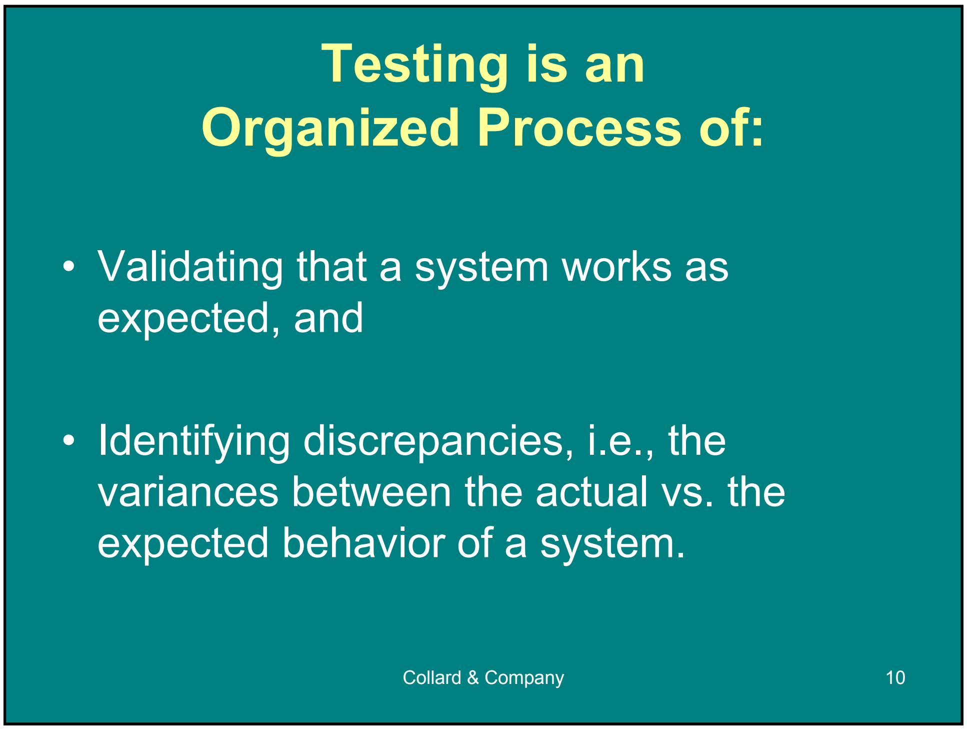
Ross Collard's Background

- Industry consulting: American Express, Boeing, Citibank, Cisco, Dell, Federal Reserve Bank, GE, HP, IBM, Intel, Merck, Microsoft, Motorola, NASA, U.S. Air Force, etc.
- International speaker: Harvard, UC Berkeley, etc.
- You can reach him at rcollard@attglobal.net.



1. DECISION-DRIVEN TESTING

- A Use Case Example
- How to Derive the Test Cases
- Path Coverage and Loop Testing
- The Impact of Assumptions



Testing is an Organized Process of:

- Validating that a system works as expected, and
- Identifying discrepancies, i.e., the variances between the actual vs. the expected behavior of a system.

What's a Test Case?

- A test case checks one particular situation or condition within the system being tested.
- Also referred to as a test condition, test script, test scenario, test spec, test assertion, etc.

What does a Test Case look like?

Typical contents:

- 1. Test case identification.
- 2. Pre-conditions.
- 3. Input data.
- 4. Test procedure.
- 5. Expected results.

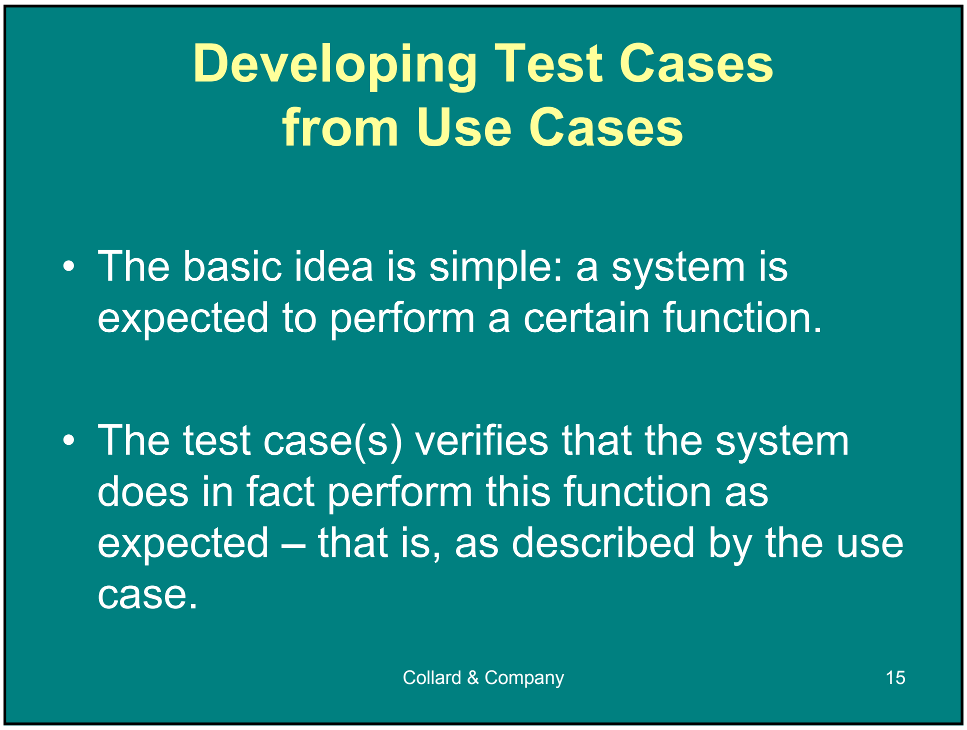
The Value of Test Case Design

- "The act of designing tests is one of the most effective error prevention mechanisms known. The thought process that must take place to create useful tests can discover and eliminate problems at every stage of development."

Boris Beizer

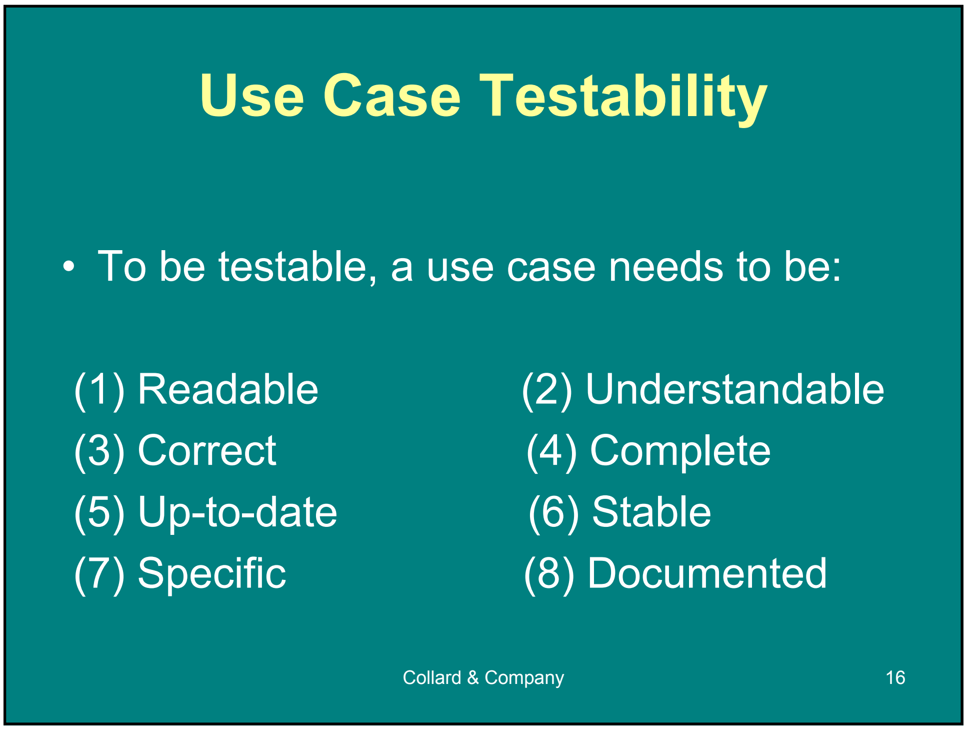
Developing Test Cases from Use Cases

- Use cases are a practical way of specifying the behavior of a system from a user's perspective.
- They can provide a powerful source for generating test cases.
- We will examine how to derive these test cases.



Developing Test Cases from Use Cases

- The basic idea is simple: a system is expected to perform a certain function.
- The test case(s) verifies that the system does in fact perform this function as expected – that is, as described by the use case.



Use Case Testability

- To be testable, a use case needs to be:

(1) Readable
(2) Understandable
(3) Correct
(4) Complete
(5) Up-to-date
(6) Stable
(7) Specific
(8) Documented

The Nature of Use Cases

- A use case is a sequence of actions performed by a system, which combined produce a result of value to a system user.
- Use cases are a type of process flow model, so they are similar in concept (and in how to test them), to classic flow models like data flow diagrams and cause-effect graphs.

Comparison with other Methods

- Use case testing offers a new perspective, and identifies test cases which other techniques have difficulty seeing.
- The value of use cases is that they focus attention on the user, rather than on the actions the system performs.

A Use Case Example

- **Use Case Name: Select Product**
- **Use Case Description**
  - This use case helps to build a purchase order (PO) by adding a new line to an order and selecting the product to order.
  - A PO can contain several line items; each line item on the PO orders a quantity of one specific product, from the vendor to whom we have addressed the PO.
- **Pre-Conditions**
- **Post-Conditions**

Exercise 1.1A: Brainstorming

- Jot down three test cases for this use case.
- What method and thought process did you utilize to identify these test cases?

Exercise 1.1B: Developing a Visual Model

- Draw a model of the use case, based on the use case flow of events and showing the main system-user interactions.
- The format and graphics conventions you employ are less important than the content.
- Perform the following steps ...

Exercise 1.1B (continued):

- First, read the description of the Select Product use case.
- Second, sketch a flowchart showing the interactions among the user and the system.
- Third, document each path as a sequence of the user and system action numbers, e.g.: [U1 -> S1.1 -> U2.1 -> …].

Exercise 1.1C: Deriving Test Cases from a Use Case

- ***A. Path-Based Test Cases***

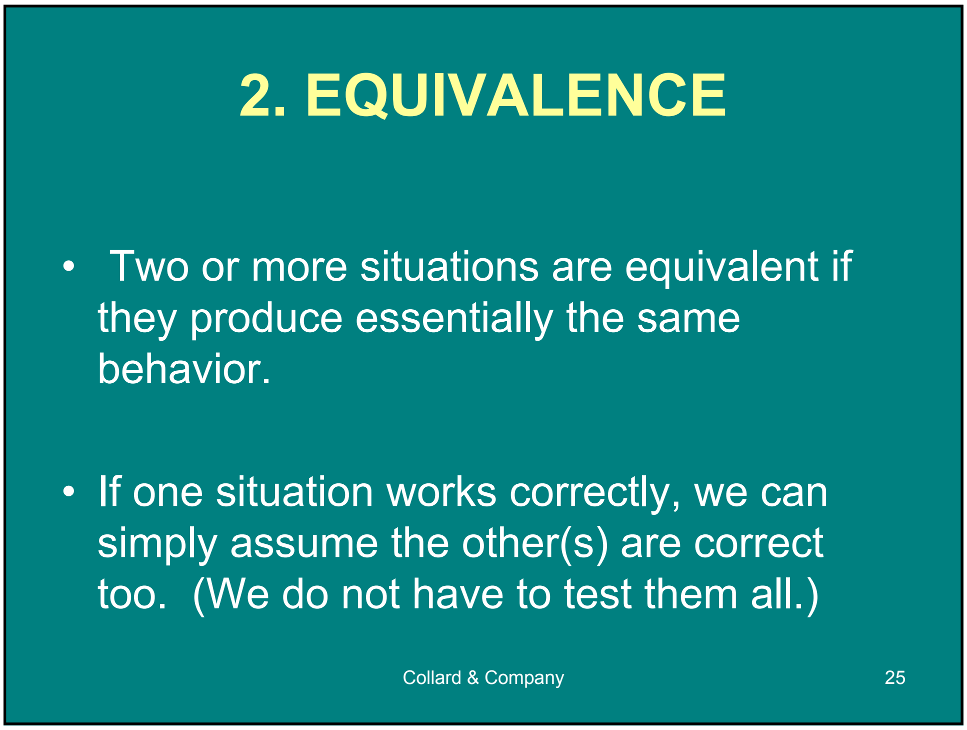
- How many paths does your model contain?
- What is the set of test cases needed to exercise the paths through the Select Product use case?

Exercise 1.1D: Supplementing Path-Based Test Cases

- Critique the list of proposed additional test cases.

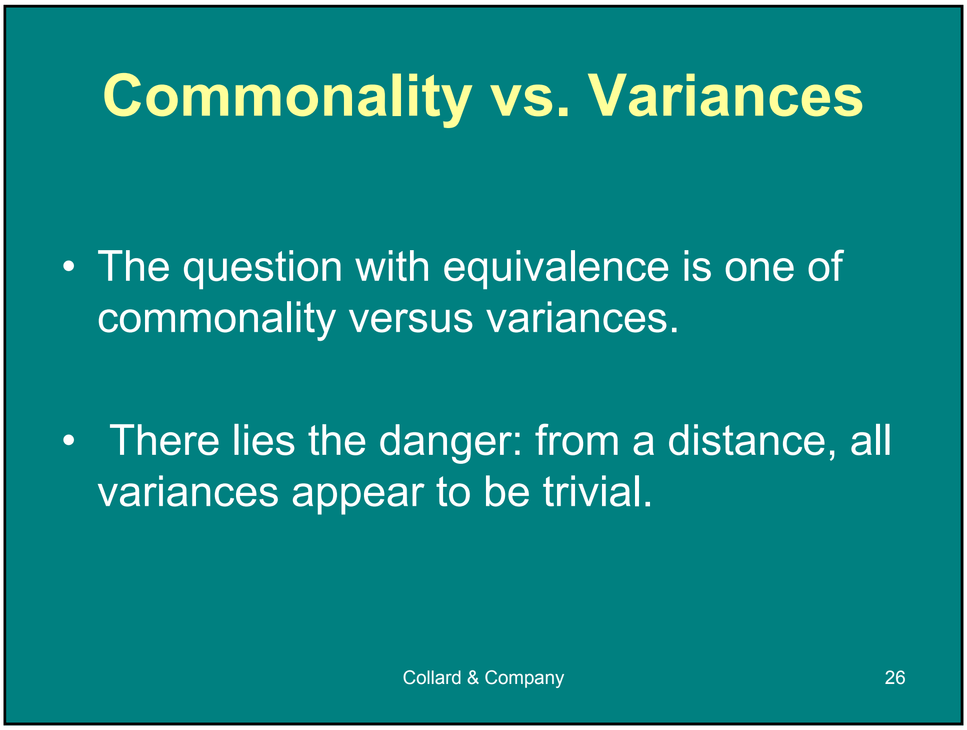
- Are these test cases feasible, justifiable and significant?

- What other test cases do you recommend?



2. EQUIVALENCE

- Two or more situations are equivalent if they produce essentially the same behavior.
- If one situation works correctly, we can simply assume the other(s) are correct too. (We do not have to test them all.)



Commonality vs. Variances

- The question with equivalence is one of commonality versus variances.
- There lies the danger: from a distance, all variances appear to be trivial.

3. INPUT-BASED TEST TECHNIQUES

Three questions must be addressed:

- (1) What are the inputs which drive the feature?
- (2) What variations of input data values, both valid and invalid, are adequate for testing? (Continued)

Input-Based (Domain) Testing

- (3) What initial conditions or initial state must be established for the input(s) to produce the expected outcomes?
  - -- Valid conditions
  - -- Invalid conditions

Boundary Value Testing

- 1. Both positive (i.e., valid) and negative (i.e., invalid) test cases are needed.
- 2. Equivalence and partitioning are used to identify test cases.
- 3. Errors tend to congregate at the boundaries.

4. RISK-BASED TESTING

- We cannot test everything.
- We need to make conscious decisions about where to focus the depth and intensity of testing.

Prioritization

- The top 10% to 15% of the test cases uncover 75% to 90% of the significant defects.
- Risk prioritization is a method of choosing the 10% to 15% which are the most critical test cases.

Major Risk Factors

- Customer or user impact
- Business operational risk
- Prior defect history
- What's new and modified

Major Risk Factors

- Heavily used features
- Complex features
- Exceptions
- Troublesome (poorly built or maintained) portions

5. FEATURE INTERACTIONS TESTING

The interactions among features, processes and events are a major source of defects. Interference can occur through these three types of dependencies:

- Timing and sequence of events
  - Lack of synchronization of synchronous processes.
  - Out-of-sequence events and violation of timing dependencies.
  - The passing of messages or transactions among systems and internally among subsystems.
  - Time-outs, the expiration of wait periods without the expected event occurring.

Types of Interactions

- Data sharing (also called data coupling)
  - Defect propagation from feature to feature through corruption of common data (which is shared among features or processes).
  - Exclusive data which should not be shared but accidentally is.
  - Resource contention and lock-out: inability to access data when needed, because it is already in use by anther feature or process.
- Resource sharing
  - Interference within a shared resource, such as a common software component. Most resource sharing problems actually fall into the first two categories: timing or data, but they are worth a separate category because they are hard to see. The occurrence of a failure is hidden within a software component or in a fleeting interaction between components.

Where do We go from Here?

- Try these techniques on your own use cases and systems testing projects.
- Send follow-up questions to me at:
  rcollard@attglobal.net.