YOU Don't Need No Stinking Test Cases

Robin F. Goldsmith, JD
GO PRO MANAGEMENT, INC.
SYSTEM ACQUISITION & DEVELOPMENT
QUALITY/TESTING
PRODUCTIVITY

22 CYNTHIA ROAD
NEEDHAM, MA 02494-1461
INFO@GOPROMANAGEMENT.COM
WWW.GOPROMANAGEMENT.COM
(781) 444-5753

“We don’t need no stinking badges”

“Badges? We ain’t got no badges! We don’t need no badges! I don’t have to show you any stinking badges!”

Alfonso Bedoya, as the Mexican bandit “Gold Hat”

In the classic film The Treasure of the Sierra Madre (1948)

http://www.quotecounterquote.com/2010/10/we-dont-need-no-stinking-badges-or.html
Objectives, You Should Be Able to

- Understand test case essentials and four keys to making them effective
- Gain the benefits of low-overhead test case formats
- Recognize you no longer have to overlook so many important test cases

Is getting enough, effective test cases an issue for you?

Awareness is a start. I’ll show you how to go further.

A Test Case Is Testing’s Basic Work Unit
What Is a Test Case?

- Essential: Inputs/conditions and expected results (outputs and changes to stored data, environment, state)
- Test case identification
  - ID, version number, name, description, resp. person
  - Cross-refs to features, requirements; category
- Pre-conditions (system/configuration, repeatable initial state, environment)
- Test procedure [advisable to keep separate]
  - Set-up, environment, tools and facilities, execution steps, results capture, environment restoration
  
Estimation Rules of Thumb: Get Consensus—

What is a Test Case? 1 of 2

Which level is the test case: A, B, C, D, or E? e.g., B=7 test cases

1. Enter an order for a customer.
   1.a Existing customer.
      1.a.1 Valid existing customer ID, customer is found.
      1.a.2 Invalid customer ID, customer is not found.
   1.b New customer.
      1.b.1 Valid name and address, added.
         1.b.1.1 Valid state abbreviation.
            1.b.1.1.1 First state (AK).
            1.b.1.1.2 Last state (WY).
            1.b.1.1.3 Short state name (IA).
            1.b.1.1.4 Long state name (NC).
            1.b.1.1.5 Delete and re-enter (MI,MN).
         1.b.1.2 Invalid state abbreviation (MM).
         1.b.2 Valid name and address, not added.
            1.b.2.1 Customer already exists.
            1.b.2.2 No disk space.
            1.b.2.3 Violates business rule, e.g., won’t sell to PO Box.
   1.c Cancel the transaction, nothing ordered.
   1.d Order an item (valid item number and quantity).
   1.e Fail to order an item.
      1.e.1 Invalid item number.
      1.e.2 Invalid quantity.
      1.e.3 Valid item number and quantity, none on hand.
      1.e.4 Cancel the transaction.
   1.f Submit the completed order (valid customer and item/quantity), ordered.
   1.g Fail to complete the order.
      1.g.1 Submit without valid item/quantity.
      1.g.2 System crashes.
A Test Case = Inputs and/or Conditions and Expected Results

- Whether or not written
- Does not need to be in any specific format

Writing has benefits:
- Don’t forget
- Can share
- Can repeat, re-use
- Can review
- Can refine
- Guide action
- Track what happened

How Much to Write: Keystroke-Level Procedure Embedded Within Test Case

Pro
- Enables execution by low-priced people with negligible knowledge
- Increases chances of precise repetition

Con
- Lots of high-priced time to create and maintain
- Time spent writing reduces number of tests and time for executing tests
- Impedes automation
- Forces execution unlike a user’s use

An automated test execution tool can do both: faster, cheaper, and more reliably
- Virtually assures finding the least amount of errors
Exploratory Testing, Error Guessing
(Only Method When No Structure)

- Experienced testers find two-three times as many errors with same script (Cem Kaner)
- Test Manager’s challenges
  - Focuses at tail-end on what was written (mainly code), not what should have been (design)
  - Write enough to be helpful—no more, but no less
  - Use as a supplementary, not primary, test technique

Often believe they don’t have test cases

Keys to Effective Testing

- Define Correctness Independently of Actual Results
- You Must Know What the “Right Answer” Is
- Follow Independent Guidelines to Avoid Overlooking Things
- Systematically Compare Actual to Expected Results

<table>
<thead>
<tr>
<th>Test Input</th>
<th>Actual Results</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cust. #123</td>
<td>John P. Jones</td>
<td>Jones, John P</td>
</tr>
<tr>
<td>New Cust's</td>
<td>Redisplays screen with</td>
<td>“Added”</td>
</tr>
<tr>
<td>name,address</td>
<td>fields cleared</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$ .75 tax</td>
</tr>
</tbody>
</table>
Defining Test Cases

Test Case Specification

Input and/or Condition
Operator enters customer number at location X.

Expected Result
System looks up customer in database and displays customer name at location Y.

What else do you need to perform this test?

Defining Test Cases This Way ...

Test Case Specification

Input and/or Condition
Operator enters customer number at location X.

Expected Result
System looks up customer in database and displays customer name at location Y.

- Interruptions, delay to find data
- Possibility of errors
  - Finding input data values
  - Checking validity of results
- Limitations on who can find data
Specify Exact Input, Expected Result

Low-overhead test cases. No procedure specified.

<table>
<thead>
<tr>
<th>Test Case Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input and/or Condition</strong></td>
</tr>
<tr>
<td>Operator enters customer number at location X.</td>
</tr>
<tr>
<td><strong>Expected Result</strong></td>
</tr>
<tr>
<td>System looks up customer in database and displays customer name at location Y.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Customer Number</strong></td>
</tr>
<tr>
<td>C123</td>
</tr>
<tr>
<td>C124</td>
</tr>
</tbody>
</table>

Test Script—Good to Test Navigation

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Result</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu=Find Customer</td>
<td>Customer entry screen</td>
<td>Can be viewed as several simple test cases</td>
</tr>
<tr>
<td>Cust. No. = C123</td>
<td>Cust. Name Jones, John P.</td>
<td>Or as a single complex test case—series of inputs/expected results</td>
</tr>
<tr>
<td>Cancel button</td>
<td>Menu</td>
<td></td>
</tr>
<tr>
<td>Menu=Find Customer</td>
<td>Customer entry screen</td>
<td></td>
</tr>
<tr>
<td>Cust. No. = C124</td>
<td>Cust. Name Not Found</td>
<td></td>
</tr>
<tr>
<td>Cancel button</td>
<td>Menu</td>
<td></td>
</tr>
</tbody>
</table>
Test Matrix

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Input Cust. No.</th>
<th>Type</th>
<th>Active</th>
<th>Expected Results Cust. Name</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C123</td>
<td>10</td>
<td>A</td>
<td>Jones, John P.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C124</td>
<td>10</td>
<td>A</td>
<td>not found</td>
<td></td>
</tr>
</tbody>
</table>

Low-overhead test cases. No procedure specified.

Any Issues with Typical Risk Approach?

- Create test cases
- Analyze and prioritize risks they address
- Run the higher risk ones

Say you create 100 test cases and have time to run 10 of them.

- What’s the value of the time spent on the other 90 that you don’t run?
- Where did you prioritize the other test cases you didn’t think of?
- Were these 100 test cases even testing the most important things?
Testware--Test (Plan) Documentation per ANSI/IEEE Std. 829-2008

- Controversial standard
- Frequently interpreted as mandating lots of documentation apparently for its own sake
  - Instead, view it as a way to organize thinking
  - Write just enough to be helpful, but no less
  - See how it can enhance agile without excess effort
- Prior version hard to read, no diagrams
  - My diagram, phrase not in standard but fit it

What must we demonstrate to be confident it works?
Testing Structure’s Advantages 1 of 3

Reactive and Proactive

- Systematically decompose large risks into smaller, more manageable pieces
- Pick view for size of need
- Organize and manage large set of test cases
- Facilitate thorough test data recreation

Testing Structure’s Advantages 2 of 3

Proactive

- Show the choices for meaningful prioritization
- Use powerful Proactive Testing™ techniques to spot ordinarily-overlooked risks
- Test the biggest risks more thoroughly
  - And earlier
- Focus first on larger issues, drill down later to detail
### Testing Structure’s Advantages

#### Proactive
- Facilitate reuse
  - Where to find
  - Where to put
  - How to make reusable
- Test cases, typically for regression tests
- Test design specifications
  - Higher leverage

---

#### Reactive

---

#### Test Design
- Identifies a set (list) of test cases (specifications) that taken together demonstrate the feature, function, or capability works
- Can be reusable or application-specific

#### Test Case
- Input/condition and expected result
- What is executed
- Specification (in natural language) and data values (which actually are input and expected)
- Can be reusable, especially specification

#### Test Procedure
- Step-by-step instructions for executing test cases
- Includes set-up, establishing pre-conditions
- Can get to keystroke level
- Often embeds input and expected result data values, which increases maintenance difficulty

---

---

You Don't Need No Stinking Test Cases
Objectives, You Should Be Able to

- Understand test case essentials and four keys to making them effective
- Gain the benefits of low-overhead test case formats
- Recognize you no longer have to overlook so many important test cases

Is getting enough, effective test cases an issue for you?
Awareness is a start. I’ll show you how to go further.

Robin F. Goldsmith, JD
robin@gopromanagement.com www.gopromanagment.com

- President of Go Pro Management, Inc. consultancy since 1982, working directly with and training professionals in business engineering, requirements analysis, software acquisition, project management, quality and testing.
- Partner with ProveIt.net in REAL ROI™ and ROI Value Modeling™.
- Previously a developer, systems programmer/DBA/QA, and project leader with the City of Cleveland, leading financial institutions, and a “Big 4” consulting firm.
- Degrees: Kenyon College, A.B.; Pennsylvania State University, M.S. in Psychology; Suffolk University, J.D.; Boston University, LL.M. in Tax Law.
- Software Quality Group of New England (SQGNE) Vice President, past President and Director.
- Formerly International Vice President of the Association for Systems Management and Executive Editor of the Journal of Systems Management.
- Founding Chairman of the New England Center for Organizational Effectiveness.
- Member of the Boston SPIN and SEPG/95 Planning and Program Committees.
- Chair of record-setting attendance BOSCON 2000 and 2001, ASQ Boston Section’s Annual Quality Conferences.
- Member IEEE Std. 829-2008 for Software Test Documentation Standard Revision Committee.
- Member IEEE Std. 730-2014 standard for Software Quality Assurance Revision Committee.
- International Institute of Business Analysis (IIBA) Business Analysis Body of Knowledge (BABOK v2) subject expert.
- TechTarget SearchSoftwareQuality.com requirements and testing expert.
- Admitted to the Massachusetts Bar and licensed to practice law in Massachusetts.
- Author of book: Discovering REAL Business Requirements for Software Project Success
- Author of forthcoming book: Cut Creep—Put Business Back in Business Analysis to Discover REAL Business Requirements for Agile, ATDD, and Other Project Success
Proactive User Acceptance Testing

Reusable Test Designs

Feasibility Analysis
Systems Analysis
Defining and Managing Business Requirements
Writing User Story Requirements

Risk Analysis
Test Estimation
System Design
Development
Implementation
Operations Maintenance

Proactive Testing:
Risk-Based Test Planning, Design, and Management

Systems QA
Software Quality Effectiveness Maturity Model
Credibly Managing Projects and Processes with Metrics
Software, Test Process Measurement & Improvement

Systems Analysis
Design
Development
Implementation
Operations Maintenance

Feasibility Analysis
Systems Analysis
Defining and Managing Business Requirements
Writing User Story Requirements

Risk Analysis
Test Estimation
System Design
Development
Implementation
Operations Maintenance

Proactive Testing:
Risk-Based Test Planning, Design, and Management

Managing Software Acquisition and Outsourcing:
> Purchasing Software and Services
> Controlling an Existing Vendor’s Performance

Making You a Leader