Overcoming Requirements-Based Testing's Hidden Pitfalls

Robin F. Goldsmith, JD

GO PRO MANAGEMENT, INC.

SYSTEM ACQUISITION & DEVELOPMENT

QUALITY/TESTING

PRODUCTIVITY

BUSINESS ENGINEERING

TRAINING

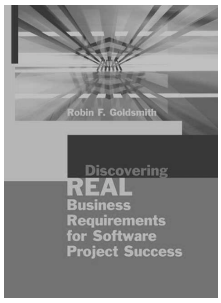
22 CYNTHIA ROAD

NEEDHAM, MA 02494-1412

INFO@GOPROMANAGEMENT.COM

WWW.GOPROMANAGEMENT.COM

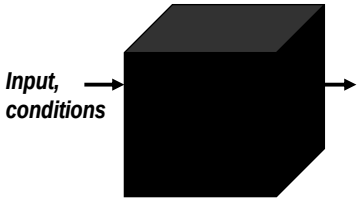
(781) 444-5753 *VOICE/FAX*



Objectives

- Identify strengths, and often unrecognized weaknesses, of requirements-based tests.
- Emphasize the importance of testing based on business, as well as system, requirements.
- Describe ways of determining how many tests are needed.

Requirements-Based Tests Usually Considered Black Box (Functional)



- ✚ Strengths
  - ✚ Intuitive
  - ✚ **Most important, relates to what software is supposed to do**
- ✖ Weaknesses
  - ✖ Potential for inconsistency
  - ✖ No single, certain right answer
  - ✖ Illusory sense of adequacy
    - ✖ Often done ad hoc, inefficiently
    - ✖ Overlooks things, input-driven

Key Requirements-Based Test Issues

- ❑ What are the requirements?
  - ❑ Incomplete, unknowable, and changing requirements
  - ❑ Testability—won't catch wrong or overlooked reqs
  - ❑ Business requirements vs. system/software requirements and use cases
- ❑ Emphasis on documentation formats (including use cases) and techniques to define test case detail can cause **overlooking important tests**
- ❑ Relevance to (developers') unit testing

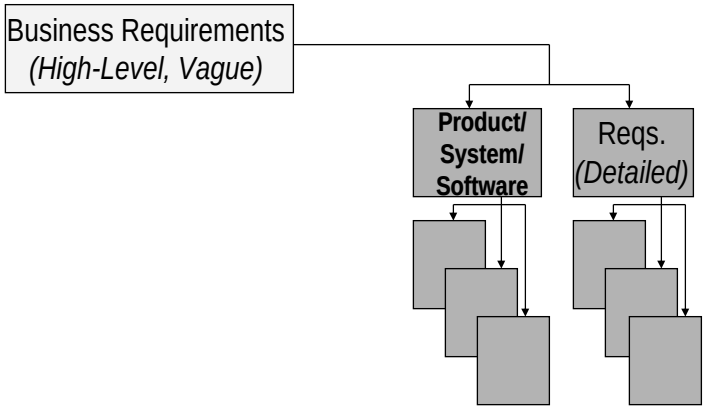
Premise: Requirements Changes Make Documenting Impractical, Too Late

- Becomes a self fulfilling prophecy rationalizing desire to leap blindly into coding
- ***Awareness*** of the REAL requirements changes much more than the REAL requirements themselves
- Even if not perfect, can know much more; tests help us know; *write enough to be helpful but no more*, iterate to detail

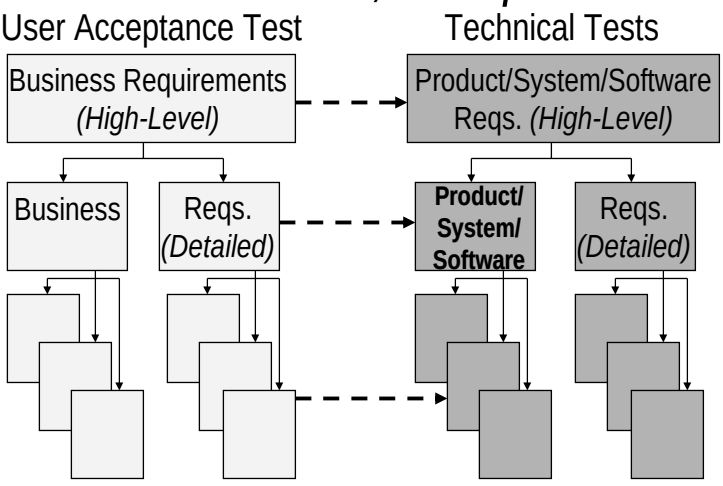
Two Types of Requirements:

Business/User	Product/System/Software
• Business/user language & view, conceptual; *exists* within the business environment • Serves business objectives • ***What*** business results must be delivered to solve a business need (problem, opportunity, or challenge) and provide value when delivered/satisfied/met **Many possible ways to accomplish**	• Language & view of a *human-defined product/system* • **One of the possible ways *How*** (design) presumably to accomplish the presumed business requirements • Often phrased in terms of external functions each piece of the product/system must perform to work as designed (Functional Specifications)

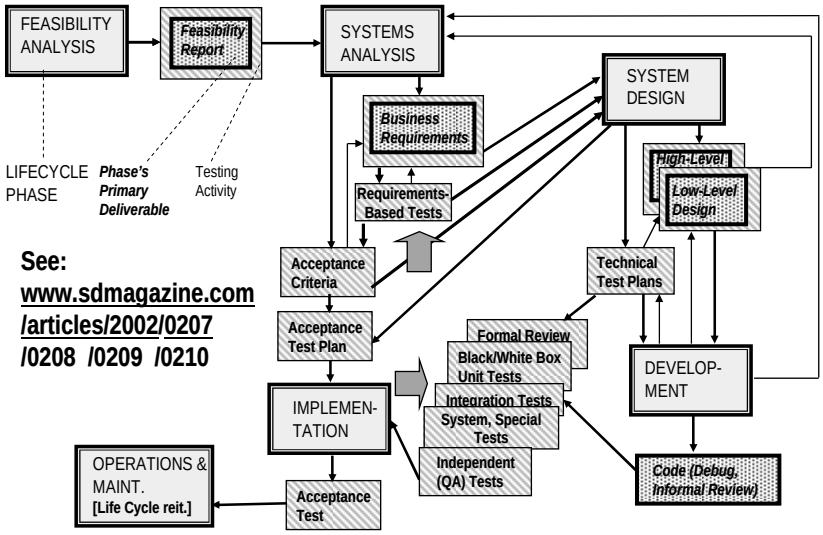
Even Requirements "Experts" Think the Difference is Detail



When Business/User Requirements Are Detailed First, Creep Is Reduced



Proactive Testing™ Life Cycle



Key Proactive Testing™ Methods Involving Requirements-Based Tests

- Define as part of Proactive User Acceptance Testing™
  - Minimum for acceptability
  - Detects requirements which are not testable→ problems
  - Unlikely to detect incorrect and overlooked requirements (defining Acceptance Criteria and more than 21 other more powerful methods detect these important issues)
- Define as part of Proactive planning/design of technical tests
  - Traditional black box test design techniques
  - Conventional black box Use Case execution, plus white box
  - Black and white box design of test cases for business rules

Requirement: Calculate 5% Sales Tax on the Order

- Requirements-Based Test:
  - Enter one item at $99.99, Sales tax should be $5.00
- Acceptance Criteria
  - Create an order consisting of both taxable and nontaxable items
  - Create an order that covers 2 pages
  - Produce a credit voucher for a returned item

Expected results, including proper sales tax amounts, would be defined when acceptance criteria are driven to lower detail

Premise: Requirements-Based Tests Just Involve Test Case Design Techniques

- Systematic techniques to thoroughly elaborate business rules are *necessary* and identify one test per scenario, e.g.,
  - Decision trees and tables
  - Business logic mapping (including Cause and Effect Graphing) is like white box

- But *not sufficient*, don't reveal overlooked
  - Unit, integration, special, system tests
  - Features, functions, capabilities

Exercise: Find/Add Customer

When a customer wants to place an order, first the customer has to be in the system and identified. Once in the system, each customer has a unique Customer Number. When the Customer Number is entered, the specific customer's record should be retrieved. Some customers have registered a credit card which they prefer to use. If the Customer Number is not known, these customers' records can be located by entering the credit card number.

If the customer's record cannot be found by exact match of Customer Number or credit card number, the customer's record can be searched for alphabetically by the customer's name. The name should be entered in last, first, middle name sequence. The name to be searched for (search argument) can be full or partial. The program will display a list of customers starting with the customer whose name is equal to or next greater than whatever has been entered as a search argument. One can scroll backward and forward alphabetically through the list of names and addresses and select the record that is the customer's.

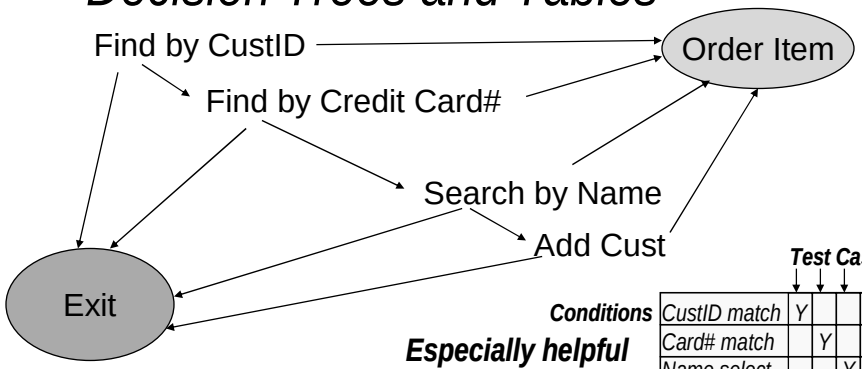
If the customer's record cannot be located, the customer may be added to the database and assigned the next sequential Customer Number. When adding a customer, the customer's name, address, home/business phone numbers, and (optionally) a credit card number must be entered. Phone numbers should be 10 digits. The address should have a five- or nine-digit postal Zip code and a valid two-character state abbreviation.

Once the customer's record has been retrieved/created and confirmed, go to the item entry routine.

Functionality Matrix

		Technical View								
User View (Use Cases)	Create	Retrieve	Update	Delete	Commun.	Interface	Logic	ChgState	PerfLevel	Constraint
Find by Cust. No. (exact match)		X			X				X	
Customer is not found *					X	X		X		
Cust. is found and confirmed*					X	X		X		
Cust. is found, not confirmed*					X	X		X		
Find by credit card no. (exact)		X			X				X	
Search by cust. Name (partial)		X			X		X		X	
Select cust. from search list					X	X		X		
Quit the search					X	X		X		
Add new customer to database	X				X	X	X	X	X	X
Quit					X	X		X		

✓Designing Tests: Decision Trees and Tables



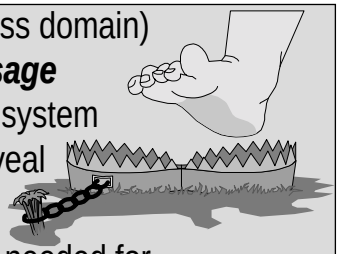
		Test Cases			
Conditions	*CustID match*	*Y*			*N*
	Card# match		*Y*		*N*
	Name select			*Y*	*N*
	Add Cust				*Y*
Actions	***Order***	***Y***	***Y***	***Y***	***Y***

Especially helpful for business rules. Is this correct?

Helps with combinations of conditions, to a point

Premise: User Requirements Are Use Cases and Translate Directly into Tests

- Use Cases can be user (business domain) requirements but usually are ***usage*** requirements of the anticipated system (i.e., high level design); may reveal business requirements issues



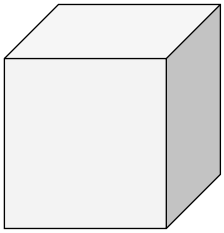
- Premise is that one test case is needed for each Use Case path/scenario, but
  - A path usually has multiple conditions to test
  - "Happy Path" is most commonly exercised
  - Alternative paths are often missed

Will One Test Suffice to Give Confidence Finding a Customer by Credit Card Works?

-
-
-
-
-
-
-
-

What conditions need to be demonstrated to be confident it works?

White Box (Structural) Tests Usually Based on Code



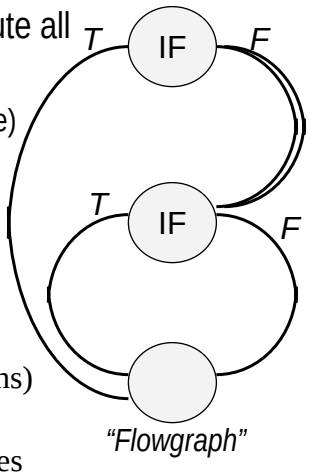
- Demonstrates system as written has been tried out
- Approach developers generally use (informally and unconsciously)
- Test cases generally more complex

Degrees of Structural Coverage

1. Execute module/routine from entry to exit

   Within a module/routine, execute all

   2. Lines of code
   3. Branches (basis paths, complete)
   4. Logic paths (exhaustive)



"Flowgraph"

- Flowgraph: ***Node***
  - Module's Entry, Exit
  - Where logic branches
  - Junctions (logic returns)
- Flowgraph: ***Edge***
  - all logic between nodes

White Box (Structural) Tests

✚ Strengths
- ✚ "Engineering Approach" yields consistent, quantifiable answer to "How many tests?"
- ✚ Assures code is tested
- ✚ Can use prior to coding

✖ Weaknesses
- ✖ **Does not test what ought to be**
- ✖ Can be thwarted by bugs, changes
- ✖ Tedious, time-consuming, error-prone

Can be used at higher levels based on:
- ✓ Business logic
- ✓ Use case flow
- ✓ Work flow (buttons and menu choices are nodes)

Developers need to test from requirements too

Test Each Use Case Path/Scenario

Defined as "How an actor interacts with the system."
The *actor* is usually the user, and the *system* is what the developers expect to be programmed. Therefore, use cases really are white box/design rather than black box/business requirements. ***Flowgraph this Use Case.*** *Path=Test Case*

U1. Enter customer number	R1.1. Customer is found (U4)
	R1.2 Customer is not found (U2)
U2. Enter customer name	R2.1 Select customer from list (U4)
	R2.2 Customer is not in list (U3)
U3. Add customer	R3 Customer is added
U4. Enter order	R4 Order is entered (Exit)

Exercise: Use Cases

In your group, write all the different inputs/conditions (in words, not with data values) that cause a specified use case path to be executed (instructor will assign a path to each group). Note: taken together all these inputs/conditions need to be demonstrated to give confidence that the specified use case path works.

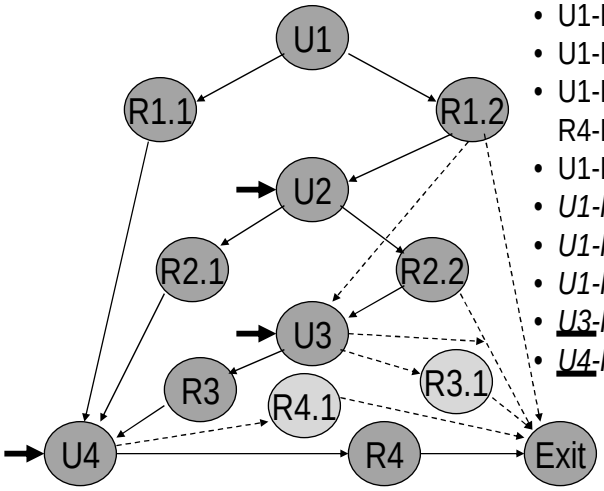
What does this tell us about the premise that you need only a single test case for each use case scenario (path)?

Exercise: Mapping Logic Structure

Read each record in a file of customers.
Print the customer's name and account number.
For each customer
    Read each accounts receivable record for them
        (could be 0 to many A/R records for a customer,
        each of which could still be due or already be paid)
        If an amount is due, print the date and amount due
    Print the total due for the customer.
Print the grand total due for all customers.

Working together in your group, flowgraph the above logic and create a set of tests (inputs and expected results --e.g., Customer 1, Customer Name 1) that will exercise all branches. Are there any specific conditions (paths) you also want to make sure are executed?

Flowgraph of Use Case



- U1-R1.1-U4-R4-Exit
- U1-R1.2-U2-R2.1-U4-R4-Exit
- U1-R1.2-U2-R2.2-U3-R3-U4-R4-Exit
- U1-R1.2-U3-R3-U4-R4-Exit
- *U1-R1.2-Exit*
- *U1-R1.2-U2-R2.2-Exit*
- *U1-R1.2-U2-R2.2-U3-Exit*
- *U3-R3.1-Exit*
- *U4-R4.1-Exit*

Exercise: Find Customer by Credit Card

What else needs to be demonstrated to be confident it works?

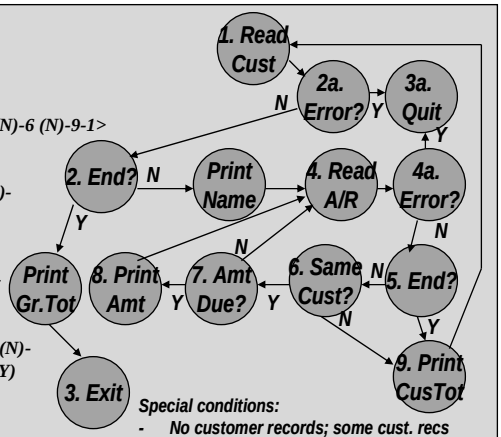
Valid	Invalid
• VISA, 16 digits, "4…"	• Leading digit wrong
• Mastercard, 16 digits, "5…"	• 16 digits for Amex
• Amex, 15 digits, "3…"	• 15 digits for VISA, MC
• Entered as only digits	• Wrong check digit
• Entered with dashes	• Expired card
• Card held by multiple custs	• Card cancelled, stolen
• Edit and re-enter	• Customer not on file
	• Card not on file for cust

What does this tell us about requirements and their tests?

Exercise: Mapping Logic Structure

CustRec	A/R Cust#	Amt	Custot	GrandTot
1-2a (N)-2 (Y)-3				
No Cust records				
1-2a (N)-2 (N)-4-4a (N)-5 (Y)-9-1->				
Cust 1	No A/R records			
1-2a (N)-2 (N)-4-4a (N)-5 (N)-6 (Y)-7 (Y)-8-4-5 (N)-6 (N)-9-1>				
Cust 2	A/R 2	20		
	A/R 3		20	
1-2a (N)-2 (N)-4-4a (N)-5 (N)-6 (Y)-7 (N)-4-5 (N)-6 (N)-9-1>				
Cust 3	A/R 3			
	A/R 4		0	
1-2a (N)-2 (N)-4-4a (N)-5 (N)-6 (N)-9-1-2 (Y)-3>				
Cust 5	A/R 6		0	20

1-2a (N)-2 (N)-4-4a (N)-5- (N)-6 (Y)-7 (Y)-8-4-5 (N)-6 (Y) -7 (N)-4-5 (N)-6 (Y)-7 (N)-4-5 (N)-6 (Y)-7 (Y) -8-4-5 (N)-6 (Y)-7 (Y)-8-4-5 (N)-6 (N)-1-2 (Y)-3				
Cust 6	A/R 6	11		
	A/R 6			
	A/R 6			
	A/R 6	22		
	A/R 6	33		
	A/R 7		66	
No Cust record				86



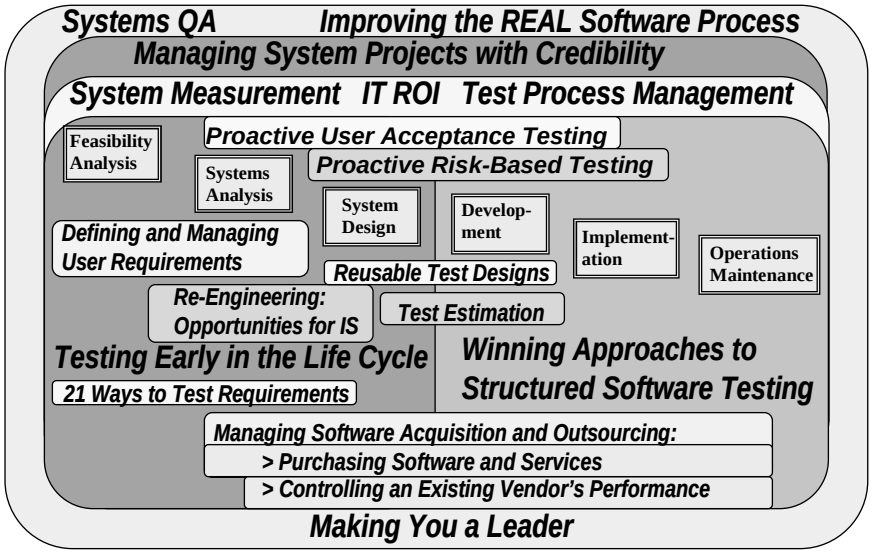
Special conditions:

- *No customer records; some cust. recs*
- *Cust. with no A/R records*
- *Cust. with some A/R records,*
  - *with 1 and 2, neither having an amount due*
  - *with 1 and 2, all having an amount due*
  - *with a mix of amount due and not due*

Summary

- Requirements-based tests are essential but also often have unrecognized weaknesses, especially due to failure to define requirements adequately.
- At least two tests per requirement are needed; but overly emphasizing test case detail can cause other requirements to be overlooked.
- Proactive Testing™ test design techniques can reveal overlooked and inaccurate requirements

Go Pro Management, Inc. Seminars--Relation to Life Cycle



Robin F. Goldsmith, JD

robin@gopromanagement.com (781) 444-5753

www.gopromanagement.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.
- Previously a developer, systems programmer/DBA/QA, and project leader with the City of Cleveland, leading financial institutions, and a "Big 5" 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.
- Published author and frequent speaker at leading professional conferences.
- 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 BOSCON 2000 and 2001, ASQ Boston Section's Annual Quality Conferences.
- Member ASQ Software Division Methods Committee.
- Admitted to the Massachusetts Bar and licensed to practice law in Massachusetts.
- Author of book: ***Discovering REAL Business Requirements for Software Project Success***