Overcoming Requirements-Based Testing's Hidden Pitfalls

Robin F. Goldsmith, JD

GO PRO MANAGEMENT, INC.

SYSTEM ACQUISITION & DEVELOPMENT

QUALITY/TESTING

PRODUCTIVITY

BUSINESS ENGINEERING

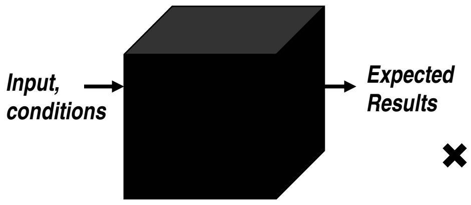
TRAINING

22 CYNTHIA ROAD
NEEDHAM, MA 02494-1412
INFO@GOPROMANAGEMENT.COM
WWW.GOPROMANAGEMENT.COM
(781) 444-5753 *VOICE/FAX*

Objectives

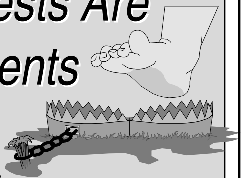
- Identify strengths, and often unrecognized weaknesses, of requirements-based tests.
- Emphasize the importance of testing based on business, as well as system, requirements.
- Describe ways of identifying more of the often-overlooked tests that are needed.

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

Pitfall: Requirements-Based Tests Are Only as Good as the Requirements



- ❑ Most organizations' aren't very good — unclear, wrong, incomplete – and *actually even worse than realized because most **miss REAL requirements***
- ❑ Some say this is because it's impossible to define requirements, writing involves too much worthless busywork, and/or number of changes is too great and causes excessive rewriting, so don't waste your time
  - ❑ Inadequate requirements becomes a self-fulfilling prophecy
  - ❑ Some denigrate requirements-based tests, all say *just go test*
  - ❑ ***Question: what exactly is one testing (or developing) if s/he doesn't know what the requirements are?***

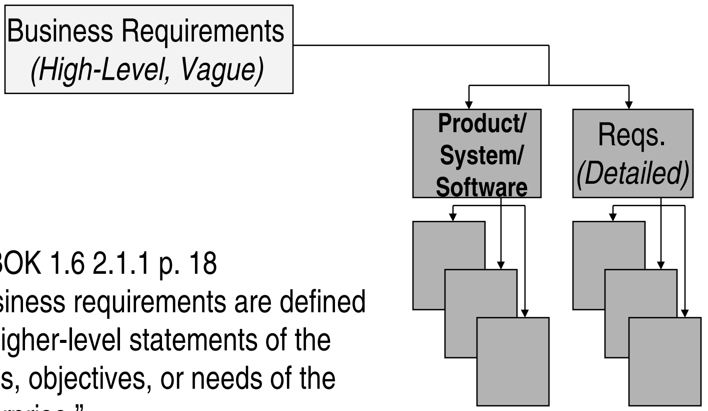
You Can *Discover a Lot More of the Requirements Than You Usually Do*

- Perfection is not the issue, adequacy is
- *Write enough to be helpful but no more*, iterate to detail as needed
- REAL requirements are business requirements
  - ***Awareness*** of the REAL, business requirements changes much more than the REAL requirements themselves; business requirements are fairly stable
  - Mindset change enables discovering more fully
  - Proactive Testing™ techniques help get them right

➲Two Types of Requirements:

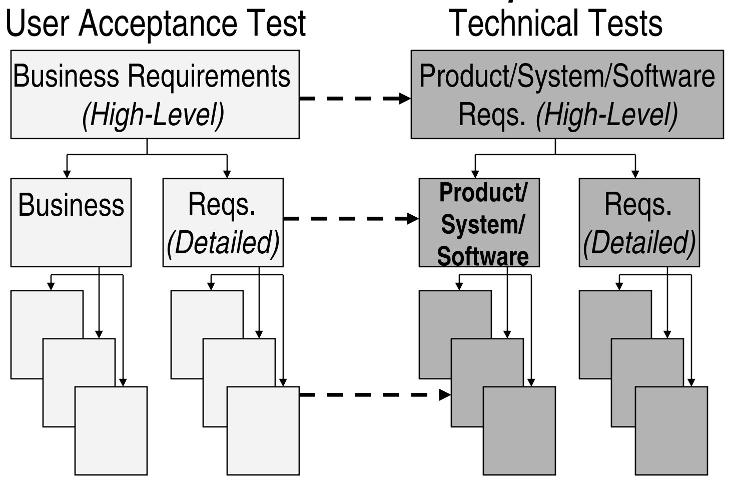
Business/User	Product/System/Software
● Business/user language & view, conceptual; *exists* within the business environment ● Serves business objectives ● ***What*** business results must be delivered to solve a business need (problem, opportunity, or challenge) and provide value when delivered/satisfied/met **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

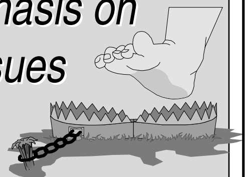


BABOK 1.6 2.1.1 p. 18
"Business requirements are defined as higher-level statements of the goals, objectives, or needs of the enterprise."

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



Pitfall: Testers' Almost Total Emphasis on Testability/Clarity Misses Major Issues



- ● Reviewing requirements is often treated as part of requirements-based testing
  - ● Ambiguous or untestable (a clarity issue) requirements are likely to be misunderstood and developed wrong
  - ● Regardless, without a test, there's no way to confirm whether or not the development meets the requirement
- ❑ A requirement can be perfectly testable and perfectly wrong, testability doesn't assure correctness
- ❑ Testability is irrelevant for overlooked requirements

Proactive Testing™ Reviews Catch Wrong, Overlooked, Unclear Reqs

- ➢ *Requirements*, 21+ ways to evaluate
  - ➢ Formats (including clarity, ambiguity, testability)
  - ➢ Finding overlooked requirements
  - ➢ Detecting incomplete and incorrect requirements
- ➢ *Design*, 15+ ways to evaluate
  - ➢ Requirements errors are most common cause of design issues
- ➢ *Code*, generally reflects design errors (see above)

Key Proactive Testing™ Methods Involve Requirements-Based Tests

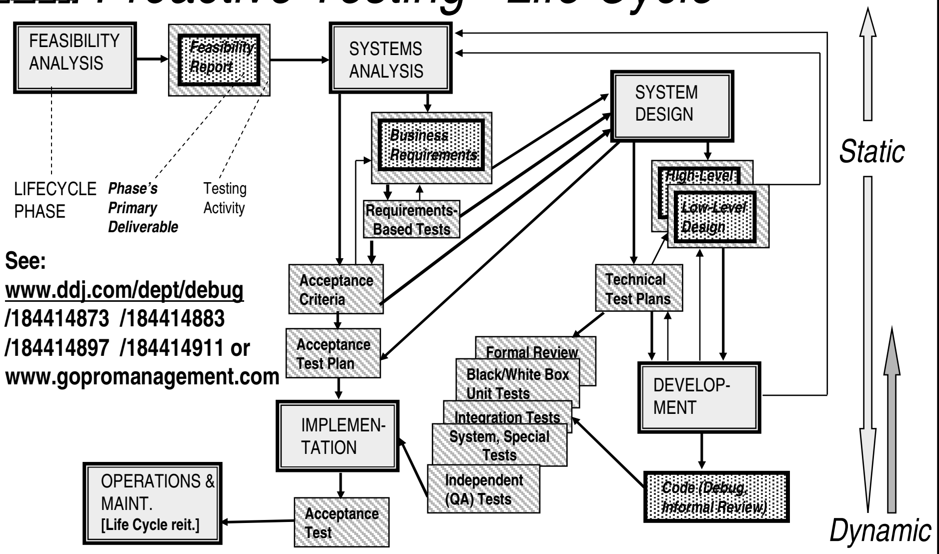
- Proactive User Acceptance Testing™
  - *Requirements phase* creation of requirements-based tests
  - Proactive User Acceptance Criteria set framework for later test execution, and spot incorrect and overlooked requirements
- Proactive technical test planning/design—*design phase*
  - Master Test Planning spots showstopper missed requirements
  - Detailed Test Planning spots incomplete, incorrect capabilities, including for current buzzword *continuous integration*
  - Black and white box design of test cases spots overlooked, unclear, erroneous conditions

Large

Medium

Small

Proactive Testing™ Life Cycle



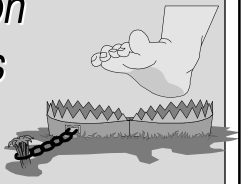
See:
www.ddj.com/dept/debug
/184414873 /184414883
/184414897 /184414911 or
www.gopromanagement.com

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

Pitfall: Superficial Over-Reliance on Use Cases Misses Test Conditions



- Use Cases can be user requirements (but usually are ***usage*** requirements of the anticipated product/system, i.e., high level design)
- Exercising a Use Case is a requirements-based test
- ❑ 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, often only, exercised path
  - ❑ Alternative steps and paths are often missed

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

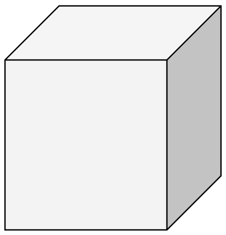
Technical View

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***		

Proactive Testing™ Detects Typical Missed Use Case Steps, Conditions

- ➢ Structuring a Use Case step-by-step helps reveal requirements to test that often are missed in a written spec or in a Use Case written from scratch
- ➢ Technical View requirements also need to be tested within the Use Case execution, but almost always are overlooked in traditional Use Cases
- ➢ Structural flow requirements also should be tested

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

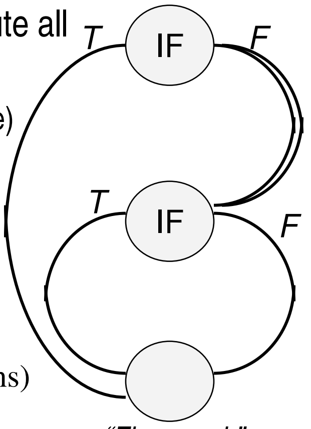
Degrees of Structural Coverage

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



"Flowgraph"

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)

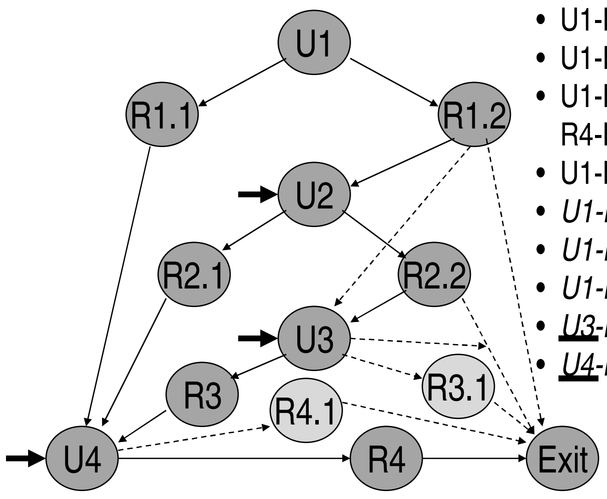
Developers need to test from requirements too

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*

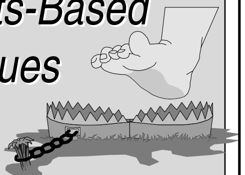
U1. Enter customer number	R1.1.	Customer is found (U4)
	R1.2	Customer is not found (U2)
U2. Enter customer name	R2.1	Select customer from list (U4)
	R2.2	Customer is not in list (U3)
U3. Add customer	R3	Customer is added
U4. Enter order	R4	Order is entered (Exit)

Flowgraph of Use Case



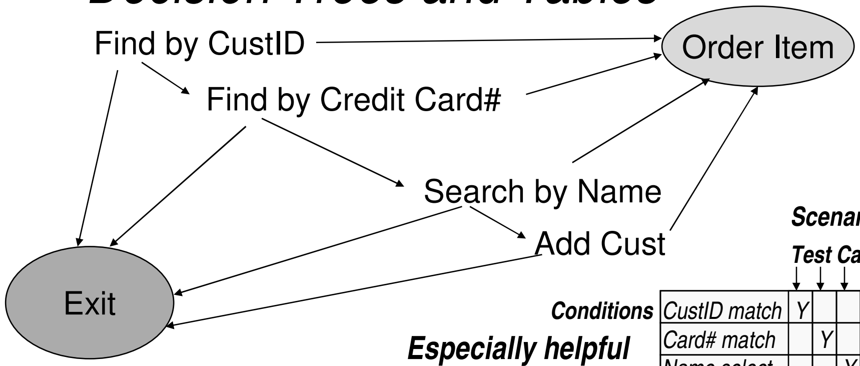
- U1-R1.1-U4-R4-Exit
- U1-R1.2-U2-R2.1-U4-R4-Exit
- U1-R1.2-U2-R2.2-U3-R3-U4-R4-Exit
- U1-R1.2-U3-R3-U4-R4-Exit
- *U1-R1.2-Exit*
- *U1-R1.2-U2-R2.2-Exit*
- *U1-R1.2-U2-R2.2-U3-Exit*
- *U3-R3.1-Exit*
- *U4-R4.1-Exit*

Pitfall: Gurus Equate Requirements-Based with Low-Level Test Case Techniques



- Typically emphasize one or a few specific structured methods to design test cases by systematically elaborating each business rule, e.g.,
  - Decision trees and tables
  - Business logic mapping (including Cause and Effect Graphing), like white box but of functionality
- ❑ Implies one test per scenario covers all requirements
- ❑ Focusing on detail can be forest-and-trees
  - ❑ Masks other, especially bigger, requirements
  - ❑ Large number of tests identified can be overwhelming

✓Designing Tests: Decision Trees and Tables



Scenarios

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

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?

Proactive Testing™ Planning/Design Reveals Missed Tests/Requirements

- Low-level systematic test case design techniques are necessary but not sufficient; one must be cautious of illusion that such techniques suffice to identify all requirements completely
- Focus first on the bigger, higher-level and prioritize by risk before drilling selectively into detail
- Test cases can also be used to capture some of requirements detail without duplicating writing

Exercise: Find Customer by Credit Card

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

Valid

- VISA, 16 digits, “4…”
- Mastercard, 16 digits, “5…”
- Amex, 15 digits, “3…”
- Entered as only digits
- Entered with dashes
- Card held by multiple custs
- Edit and re-enter

Invalid

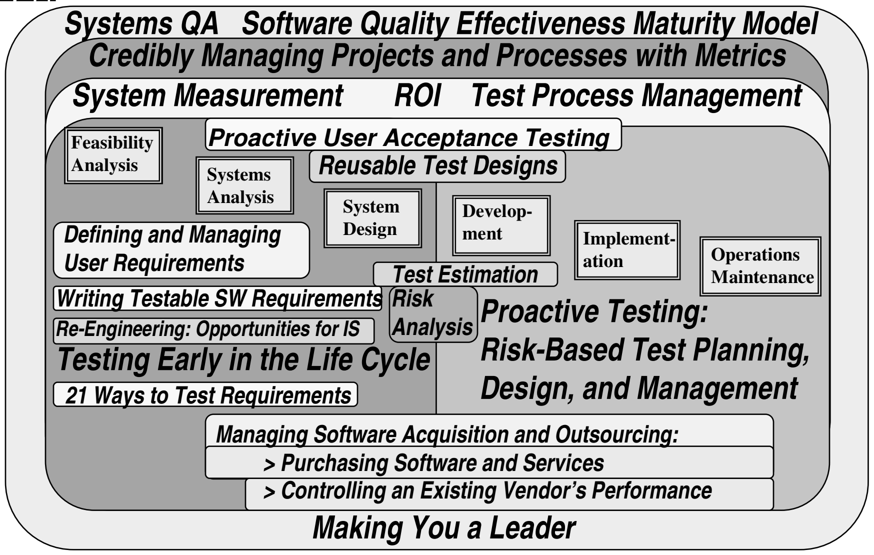
- Leading digit wrong
- 16 digits for Amex
- 15 digits for VISA, MC
- Wrong check digit
- Expired card
- Card cancelled, stolen
- Customer not on file
- Card not on file for cust

What does this tell us about requirements and their tests?

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


Systems QA Software Quality Effectiveness Maturity Model
Credibly Managing Projects and Processes with Metrics
System Measurement
ROI
Test Process Management
Feasibility Analysis
Proactive User Acceptance Testing
Systems Analysis
Reusable Test Designs
System Design
Develop-ment
Defining and Managing User Requirements
Implement-ation
Operations Maintenance
Test Estimation
Writing Testable SW Requirements
Risk Analysis
Re-Engineering: Opportunities for IS
Proactive Testing: Risk-Based Test Planning, Design, and Management
Testing Early in the Life Cycle
21 Ways to Test Requirements
Managing Software Acquisition and Outsourcing:
> Purchasing Software and Services
> Controlling an Existing Vendor's Performance
Making You a Leader

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 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.
- 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.
- Member IEEE Std. 829 for Software Test Documentation Standard Revision Committee
- Admitted to the Massachusetts Bar and licensed to practice law in Massachusetts.
- Author of book: ***Discovering REAL Business Requirements for Software Project Success***