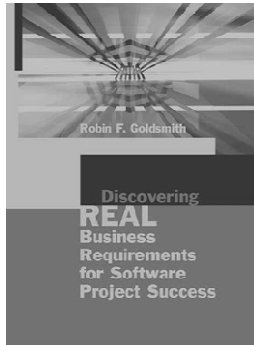
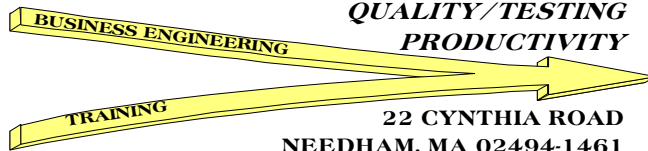


# Tips and Traps of Defining Requirements as Tests



*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

## Objectives

- Types of tests that clarify or even constitute the requirements.
- Ways to reduce requirements impacts when tests are defined inadequately.
- How defining requirements as tests can both help and diminish development and test effectiveness.

## Requirements in Agile Generally Are Considered to Be User Stories

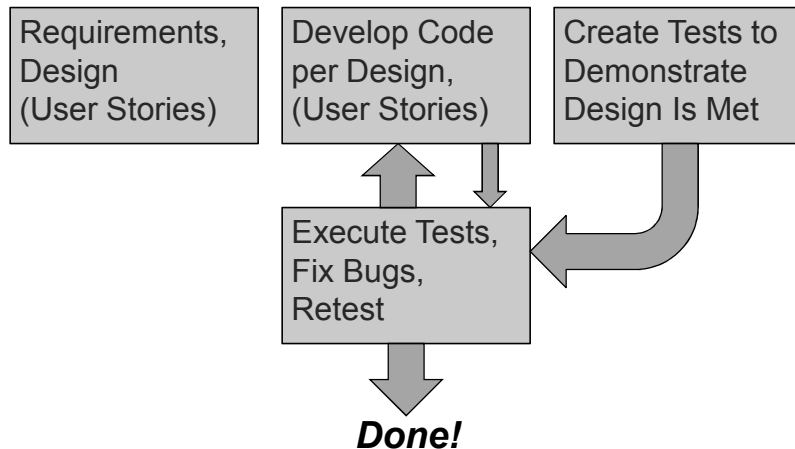
As a <type of user>  
I <want/can/am able to/need to/etc.>  
so that <some reason>

Mike Cohn

“User Stories, Epics and Themes”

<http://www.mountaingoatsoftware.com/blog/stories-epics-and-themes>

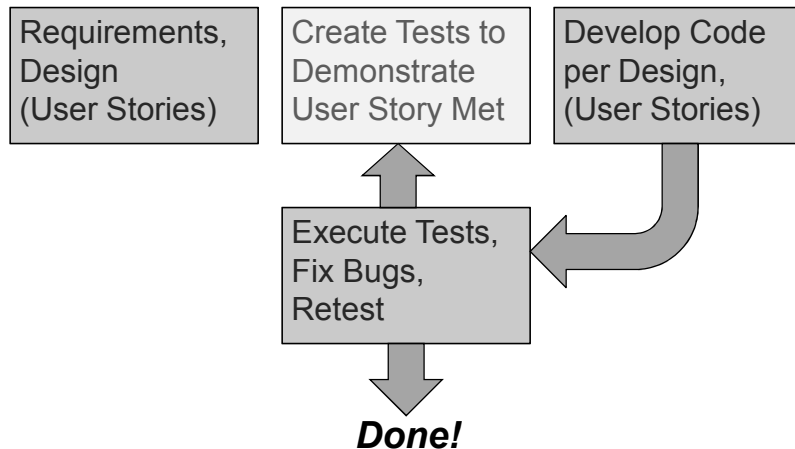
## Typical Development/Testing



## Typical Development/Testing Issues

- Testing isn't actually done, or at least
  - Not enough
  - Not well enough, still misses a lot
- Many defects trace back to requirements and design errors/misunderstandings, so lots of wasted work must be thrown out and redone
- Fixing defects long after their creation is much harder and more expensive

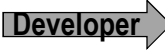

## Test-First Development/Testing



## ■■■■ eXtreme Programming (XP) Made “Test Driven” Development Popular

- Product Owner provides user story requirements
- Developer codes small executable automated unit tests based on how the unit should function
- Developer “pair programming” codes the program unit and runs it against the unit tests
- If unit test fails, modify/refactor the program
- When all unit tests pass, the unit is done
- Then ...

## ■■■■ User Stories Actually Are a Bit More

- Card (Product Owner)
  - As a <role>
  - I want <something>
  - So that <benefit>
- Conversation  **Working code**
- Confirmation (Product Owner)  **Unit Tests**
  - User story acceptance criteria, tests

**“Placeholder, reminder for a conversation”**

## ■■■■ *“Test Driven” Development Issues, Including Not Recognized by Agilists*

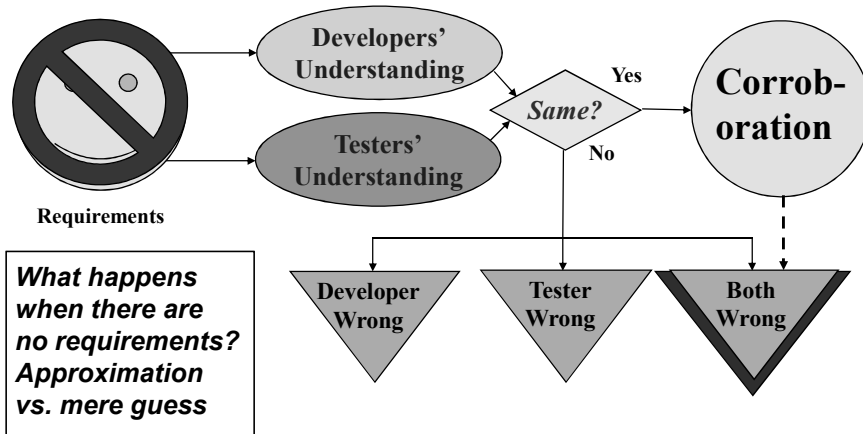
- Tests often actually reacting to developer’s mental picture of how code will be written
- Unit tests address structure of code units but not necessarily integrations or business value
- Developer’s interests and skills are development, not testing, so tests turn out to
  - Be perfunctory and weak
  - Miss a lot, without awareness they are missed

## ■■■■ *Acceptance Test Driven Development (ATDD)*

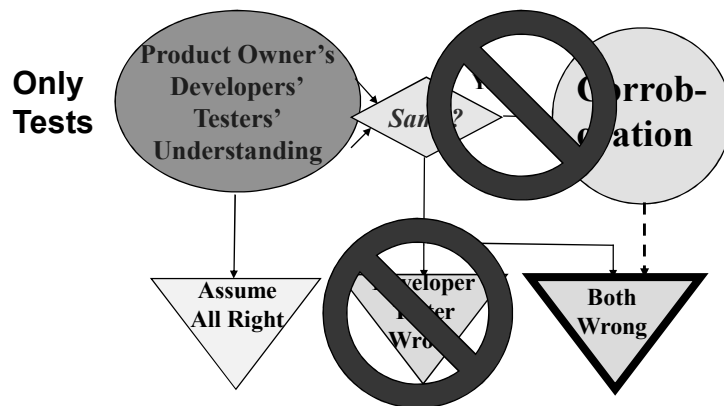
- Emphasizes Requestor/Developer/Tester *triad*
  - Defines thorough “single version of truth” set of User Story Acceptance Tests of functionality prior to coding
  - For every business rule and use case scenario to implement each user story as it is worked on
  - Drives test-first unit tests too
  - *The tests constitute a precise requirements definition*

***Passing all User Story Acceptance Tests necessary but  
not sufficient to ensure system meets customer needs***

# Requirements Drive Both Development and Testing



# When Requirements Are Left Out: What's the Impact?



## Testing vs. Experimentation

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

<u>Test Input</u>	<u>Actual Results</u>	<u>Expected Results</u>
Cust. #123	John P. Jones	Jones, John P.
New Cust’s name,address	Redisplays screen with fields cleared	“Added”
10 Widgets	\$14.99	\$14.99 \$ .75 tax

## It is Impossible to Test All Inputs, So Testing Involves Sampling

- Enter State Abbreviation
- Program looks up and displays state’s name

256	256
65	536

*How many possible inputs?*

True context is essential for picking the tests that find the most with the least time and cost

***Might writing them down be a good idea?***

## Swimming at J.P. McCaskey High



- Built during Depression
  - Swimming pool!!!
- Serves entire city
  - Diverse hygiene
- Shower, inspection (acceptance tests)
  - Hands and inner wrist
  - Back of neck

**Tests = Requirements?**

## Two Types of Requirements:

### Business/User/Customer

- Business/user/stakeholder/customer language & view, conceptual; *exist* 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**

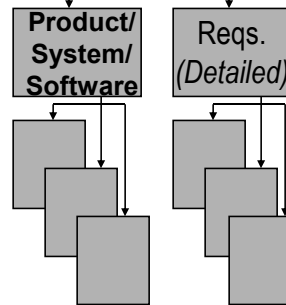
### Product/System/Software

- 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 features/external functions each piece of the product/system must perform to work as designed (Non/Functional Specifications)



## Even Requirements “Experts” Think the Difference Is Just Level of Detail

Business Requirements  
(High-Level, Vague)



BABOK® v3 2.3 p. 26  
**“Business requirements: statements of goals, objectives, and outcomes that describe why a change has been initiated.”**

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

Acceptance Tests

Business Requirements  
(High-Level)

Business

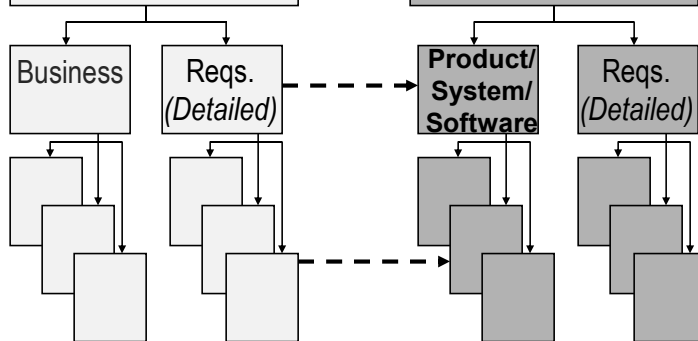
Reqs.  
(Detailed)

Technical/Development Tests

Product/System/Software  
Reqs. (High-Level)

Product/  
System/  
Software

Reqs.  
(Detailed)



## Tests Help Define Requirements, but ..

- Adequacy of User Story Acceptance Criteria
  - Too easily just “**See if it is correct**”
  - Overlooks important criteria not readily apparent from user story, especially
  - Integrations/interfaces and quality factors
- Adequacy of User Story Acceptance Tests
  - Missing scenarios, missing risk conditions
  - Not applying proven test design techniques
  - Redundant, non-value-adding insensitive tests
  - Poor/no risk analysis prioritization

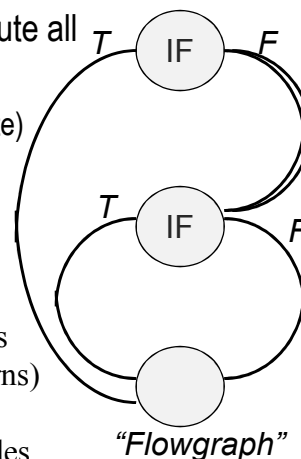
## Degrees of Structural Coverage

1. Execute module/routine from entry to exit

Within a module/routine, execute all

2. Lines of code, steps
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



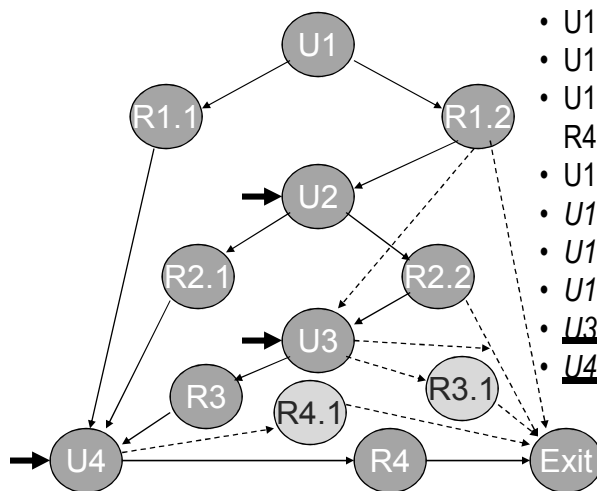
## Use Cases, Usually Are Defined as Requirements, also Are Test Cases

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 Vendor ID	R1.1. Vendor is found (U4)
	R1.2 Vendor is not found (U2)
U2. Enter vendor name	R2.1 Select vendor from list (U4)
	R2.2 Vendor is not in list (U3)
U3. Add vendor	R3 Vendor is added (U4)
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

## Inherent Ambiguity, Multiple Possible Interpretations

1. Minimize the number of errors when adding a customer.

**But, you can create test cases without yammering about “testability.”**

**Input** = 10 new customers not already in the customer database

**Expected Results** = No more than 1 of the 10 database entries contains different data from that which was supposed to be entered for primary customer fields: name, street address, city, state, and zip code

**= or just aid understanding Reqs?**

2. Provide sufficient customer identification information to determine adequately whether the customer is in the search list of customer names.

**Input** = First and last name and birth date (month, day, and year)

**Expected Results** = Uniquely distinguishes among individuals with identical first, middle, and last names at both same and different addresses, but with each having a unique birth date [database must include instances of each]

**The typical testability approach is to request that the requirement be reworded so it can be interpreted in only one way. Will it happen?**

## Logical Ambiguity, Alternative Consequences

1. If the customer has a credit card, enter the credit card number when adding the customer.

**Add: Input Credit Card#, Name      Expected Results in Database Name, Credit Card#**

3456789012345678	Smith, John J.	Smith, John J.	3456789012345678
Omitted	Smith, Mary B	Smith, Mary B.	blank
321098765432109	Smith, Jim X.	Error, “Invalid credit card number”,	not added

2. If the customer’s credit card number has been entered, the customer’s record can be accessed by credit card number.

**Inquiry: Input Credit Card#      Expected Results Displayed**

3456789012345678	Smith, John J.	3456789012345678
Blank	Error, “Customer not found”	
321098765432109	Error, “Customer not found”	
3210987654321098 [not on D/B]	Error, “Customer not found”	

## Logical Ambiguity, Unclear Reference

1. If the customer has the same name as another customer, add them to the customer database.

<u>Add: Input Name, Address, Birth Date</u>				<u>Expected Result, Address</u>	
Smith, John	123 Main St	1980-01-02	Added	123 Main St	
Smith, John J	123 Main St	1980-01-02	Error, "Already on file"		
Smith, John J Jr	123 Main St	2002-01-02	Added	123 Main St	
Smith, Mary B	123 Main St	1982-03-04	Added	123 Main St	
Smith, John J	524 Main St	1980-01-02	Added	524 Main St	

2. If a customer has both a street number and a PO Box, use it for the address.

Smith, Jim X	727 Main St	Box 10	1966-12-15	Added	Box 10	727 Main St
Smith, Jim Y	Box 10		1941-08-09	Added	Box 10	

## Logical Ambiguity, Implied Actions

1. If the customer has a credit card, verify the check digit equals a modulus 10 and confirm the expiration date has not yet passed; however, if it is MasterCard or Visa, enter the security code.

<u>Add: Input Card#, Expiration MM-YYYY, Security Code</u>				<u>Expected Result</u>	
545678901234567	12-2012	321		Error, "Card no. too short"	
5456789012345678	12-2012	321		Error, "Check Digit Wrong"	
5456789012345670	12-2012			Error, "No Security Code"	
5456789012345670	12-2006	321		Error, "Expired"	
5456789012345670	12-2012	321		Added	
5456789012345670	12-2011	321		Error, "Already on file"	
3456789012345678	12-2012			Error, "Card no. too long"	
345678901234567	12-2012			Error, "Check digit wrong"	
345678901234564	12-2012			Added	

## Or and And

1. If the customer's credit card is expired or not MasterCard and the address is only a PO Box, don't add the customer to the customer database.

If (the customer's credit card is expired or not MasterCard) and the address is only a PO Box, don't add the customer to the customer database.

If the customer's credit card is expired or (not MasterCard and the address is only a PO Box), don't add the customer to the customer database.

<u>Add:</u>	<u>Input Card#</u>	<u>Expiration MM-YYYY</u>	<u>Address</u>	<u>Expected Result</u>
	5456789012345670	12-2006	123 Main St	Added MC [first]
	5456789012345670	12-2006	123 Main St	Error, "Expired MC" [second]
	5456789012345670	12-2012	Box 10	Added MC
	345678901234564	12-2006	123 Main St	Error, "Expired Amex"
	345678901234564	12-2012	Box 34	Error, "Not MC, Only PO Box"
	345678901234564	12-2012	Box 34 542 Main St	Added Amex

## Logical Ambiguity, Unclear Boundaries

1. Add a customer only if the credit card expiration date is between 08/2007 and 12/2012

<u>Add:</u>	<u>Input Card#</u>	<u>Expiration MM-YYYY</u>	<u>Expected Result</u>
	5456789012345670	07-2007	Error, "Invalid Expiration Date"
	5456789012345670	01-2013	Error, "Invalid Expiration Date"
	5456789012345670	08-2007	Error, "Invalid Expiration Date"
	5456789012345670	12-2012	Error, "Invalid Expiration Date"
	5456789012345670	09-2007	Added
	5456789012345670	11-2012	Added
	5456789012345670	10-2007	Error, "Already on file"

## Verifiability

1. The credit card add function works well with a good human interface and usually can be completed within 20 seconds.

Two approaches:

1. Define “well,” “good,” and “usually” in objective operational terms. For example, “well” and “good” could mean that the add function can be performed in no more than 30 seconds with no more than one error which is caught and corrected during the add. “Usually” could mean that at least half of all adds are completed in 20 seconds.
2. Survey users to get their judgments.
3. Same as 2 but with specification of the qualitative characteristics constituting “well,” “good,” and “usually.”

Similarly, while “never” indeed is not testable, for one cannot be sure that a problem which has not occurred so far won’t occur in the future, one could declare failure to occur in a specified number of instances and conditions gives sufficient confidence it won’t occur in the future.

## Objectives

- Types of tests that clarify or even constitute the requirements.
- Ways to reduce requirements impacts when tests are defined inadequately.
- How defining requirements as tests can both help and diminish development and test effectiveness.

## Robin F. Goldsmith, JD

[robin@gopromanagement.com](mailto:robin@gopromanagement.com) [www.gopromanagement.com](http://www.gopromanagement.com)

- President of Go Pro Management, Inc. consultancy since 1982, working directly with and training professionals in requirements analysis, REAL ROI™, software acquisition, project management, Proactive Testing™ and SQA™.
- Featured speaker at leading conferences; frequent author, including "Unconventional Wisdom" TestHuddle.com blog.
- 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.
- President of the Software Quality Group of New England (SQGNE).
- 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.
- Attendee Networking Coordinator for STAR, Better Software, and Test Automation Conferences.
- 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 730-2014 standard for Software Quality Assurance total revision working group.
- International Institute of Business Analysis (IIBA) Business Analysis Body of Knowledge (BABOK v2) subject expert.
- TechTarget.com 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*

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

### Systems QA Software Quality Effectiveness Maturity Model Credibly Managing Projects and Processes with Metrics

#### Software, Test Process Measurement & Improvement

