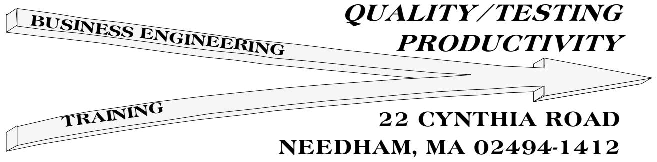
Testing the Untestable

Robin F. Goldsmith, JD
GO PRO MANAGEMENT, INC.
SYSTEM ACQUISITION & DEVELOPMENT
QUALITY/TESTING
PRODUCTIVITY
22 CYNTHIA ROAD
NEEDHAM, MA 02494-1412
INFO@GOPROMANAGEMENT.COM
WWW.GOPROMANAGEMENT.COM
(781) 444-5753 *VOICE/FAX*



Objectives

- Describe common familiar and not-so-familiar characteristics of requirements and designs which are considered “untestable”
- Suggest seldom-recognized reasons why others often resist testers’ concerns about (lack of) testability
- Describe ways to create test cases for seemingly untestable requirements and designs

Inadequate Requirements Frustrate Developers Too

- *But they don’t seem to dwell on it or let it stop them from coding*
- They make assumptions and technology- and design-based decisions, often without being aware they are doing it
- They also may create code based on informal sources of requirements information

Such code often differs from what testers have reason to expect

Testers Need to Know What the Requirements Are to Confirm that Systems Meet the Requirements

Too often testers receive requirements which are too late and inadequate

Getting timely, **testable** requirements increasingly is becoming testers' major concern

How can/should testers contribute effectively to getting the clear and accurate requirements they need?

Conventional Answer:
Testers Should Review Requirements

Conference -Attendee's Paradox Pitfall

Testing Gurus: *Testers, go back and make "them" let you participate in reviews to make sure requirements are* ***testable.***

- Other folks, such as managers, users, analysts, and sometimes developers ***are*** involved defining requirements up-front
- When testers impose themselves on the requirements process:
  - "Testability" can seem trivial
  - Involvement can backfire if they are not prepared to contribute meaningfully—requires business domain subject area knowledge that testers may not have
  - They'll prove it was right to exclude them

Keys to Effective Testing

- ➢ **Define correctness independently of actual results**
- ➢ **You must know what the "right answer" is**
- ➢ **Follow independent guidelines to be more thorough**
- ➢ **Systematically compare actual to expected results**

Test Input	**Actual Results**	**Expected Results**
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

Testability Issues Usually Refer to Difficulty Identifying Expected Results

- Inability to do so usually means requirements/ design are not sufficiently clear
  - Developers are likely to get it wrong too
  - Regardless, without a reliable test, there's no way to tell whether it's been implemented correctly
- It usually is more possible than presumed to define expected results

Tests often represent the clearest (perhaps only) statement of requirements/design

IEEE Std 830-1998 4.3 Characteristics of a good SRS

a) Correct ← **Content**

Format → **b) Unambiguous**

c) Complete ← **Content**

Format → **d) Consistent**

e) Ranked for importance and/or stability ← **Mgmt**

Format → **f) Verifiable**

g) Modifiable ← **Mgmt**

h) Traceable ← **Mgmt**

Testability Is a Format Issue
Won't reveal wrong requirements
Testability is irrelevant for overlooked requirements

IEEE Std 830-1998
b) ***Unambiguous***

"An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term.
In cases where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific….
Representations that improve the requirements specification for the developer may be counterproductive in that they diminish understanding to the user and vice versa….
Natural language is inherently ambiguous."

1. Inherently ambiguous terms **2. Logical ambiguity**

Warning:

Total Elimination of Ambiguity Is Probably Not Possible and Certainly May Not Be Practical or Necessary

Is it reasonable to expect presumably intelligent people to be able to read and make reasonable interpretations?

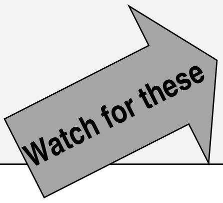
Note: The Internal Revenue Code has gone to great lengths to eliminate ambiguities, thereby making it virtually unintelligible

Options Give the Developer Latitude / *Weak Phrases Can Have Multiple Interpretations*

- Can
- May
- Optionally

- adequate
- as a minimum
- as applicable
- easy
- as appropriate
- be able to
- be capable
- but not limited to

- capability of
- capability to
- effective
- if practical
- normal
- provide for
- timely
- tbd



Seemingly easy to catch

William M. Wilson "Writing Effective Requirements Specifications"
http://satc.gsfc.nasa.gov/support/STC_APR97/write/writert.html

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

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.

Exercise: 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.

Add: Input Name, Address, Birth Date			Expected Result, Address	
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.

Add: Input Card#	Expiration MM-YYYY	Security Code	Expected Result
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

Luhn algorithm *or* ***Luhn formula****, also known as the "modulus 10" or "mod 10" algorithm*

- The formula verifies a number against its included check digit, which is usually appended to a partial account number to generate the full account number. This account number must pass the following test:
- Starting with the rightmost digit (which is the check digit) and moving left, double the value of every second digit. For any digits that thus become 10 or more, add their digits together as if casting out nines. For example, 1111 becomes 2121, while 8763 becomes 7733 (from 2×6=12 → 1+2=3 and 2×8=16 → 1+6=7).
- Add all these digits together. For example, if 1111 becomes 2121, then 2+1+2+1 is 6; and 8763 becomes 7733, so 7+7+3+3 is 20.
- If the total ends in 0 (put another way, if the total modulus 10 is congruent to 0), then the number is valid according to the Luhn formula; else it is not valid. So, 1111 is not valid (as shown above, it comes out to 6), while 8763 is valid (as shown above, it comes out to 20).

http://en.wikipedia.org/wiki/Luhn_algorithm

Exercise: 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.

Add: Input Card#, Expiration MM-YYYY, Address	Expected Result
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

Add: Input Card#, Expiration MM-YYYY	Expected Result
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"

IEEE Std 830-1998 d) **Consistent**

"An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict. The three types of likely conflicts in an SRS are as follows:

a) The specified characteristics of real-world objects may conflict. For example,
   1) The format of an output report may be described in one requirement as tabular but in another as textual.
   2) One requirement may state that all lights shall be green while another may state that all lights shall be blue.

b) There may be logical or temporal conflict between two specified actions. For example,
   1) One requirement may specify that the program will add two inputs and another may specify that the program will multiply them.
   2) One requirement may state that 'A' must always follow 'B,' while another may require that 'A' and 'B' occur simultaneously.

c) Two or more requirements may describe the same real-world object but use different terms for that object. For example, a program's request for a user input may be called a 'prompt' in one requirement and a 'cue' in another. The use of standard terminology and definitions promotes consistency."

Inconsistency

1. Expiration date MM-YY
2. Expiration date MM-YYYY
3. Expiration date month and year

Add Input Expiration Date	**Expected Result**
0907	Added 2007-09
09/07	Added 2007-09
09-07	Added 2007-09
09-2007	Added 2007-09
907	Error, "Invalid Date"

IEEE Std 830-1998
f) ***Verifiable (Testable)***

"A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable.
Nonverifiable requirements include statements such as 'works well,' 'good human interface,' and 'shall usually happen.' These requirements cannot be verified because it is impossible to define the terms 'good,' 'well,' or 'usually.' The statement that 'the program shall never enter an infinite loop' is nonverifiable because the testing of this quality is theoretically impossible."

Verification could be by examination or analysis, which is different from *Testable*—shown by writing a test case to demonstrate requirement is met.

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."

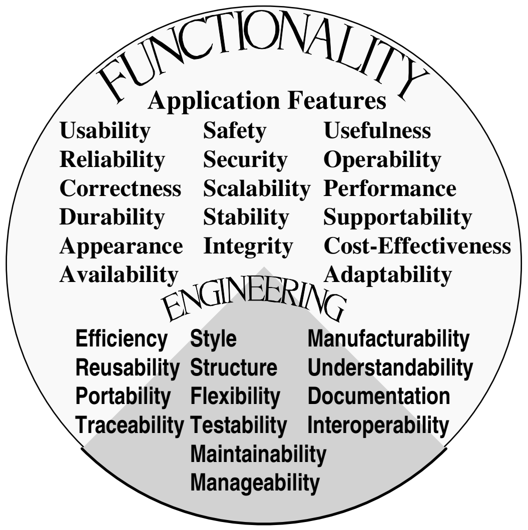
   Similarly, while "never" indeed is not testable, for one cannot be sure that a problem which has not occurred so far won't occur in the future, one could declare failure to occur in a specified number of instances and conditions gives sufficient confidence it won't occur in the future.

Where It's Impractical or Inappropriate to Create Actual Results, e.g.,

- A drug dosage or injury that will kill you
- Predicting future events, such as the economy or weather
- Performance tests involving huge volumes

Simulations, sampling, modeling, extrapolation from similar situations, and expert opinion reviews may suffice—look for reasonableness of results

✪ Addressing Quality Factors



Quality Factors, with Terms Such as "Easy," Can Be Defined Objectively, e.g.,

1. Intuitively.
   a) Understandable readily-observable instructions.
   b) Self-explanatory without extra training.
   c) Guided and prompted.
      1) Logical sequence of a small set of simple steps.
      2) Identify full set of choices when decisions are needed.
2. Validate data at time of entry.
   a) Immediately retrieve and reveal related data.
   b) Confirm context and business rules compliance.
   c) Provide clear explanations of errors.
3. Enable straightforward corrections and exit.
4. Any time of day from any place using devices including
   a) Computer
   b) Portable digital assistant
   c) Telephone

Tests of Terms Such as "Easy" Raise Issues of Creating Inputs/Conditions

- Test cases don't demonstrate "easy" directly
- Rather, the test cases must create conditions which would provoke errors if the function is not "easy," for example:
  – Rapid entry of large numbers of complex inputs
  – Use in dim light
  – Repeated entry and modification

An Often Overlooked Testability Issue: Inadequate Information to Test Fully

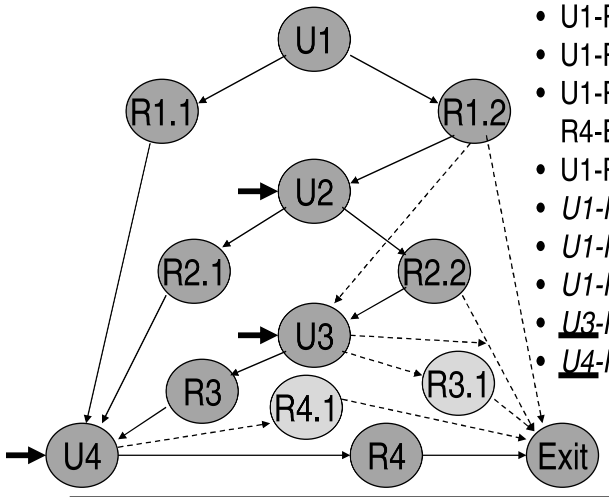
- Inadequacy of information often is not apparent on its face, so it's missed in reviews
- Spotted during rigorous, disciplined testing—which frankly isn't as common as presumed
- Especially likely with use cases and user stories which are seldom examined critically

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 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



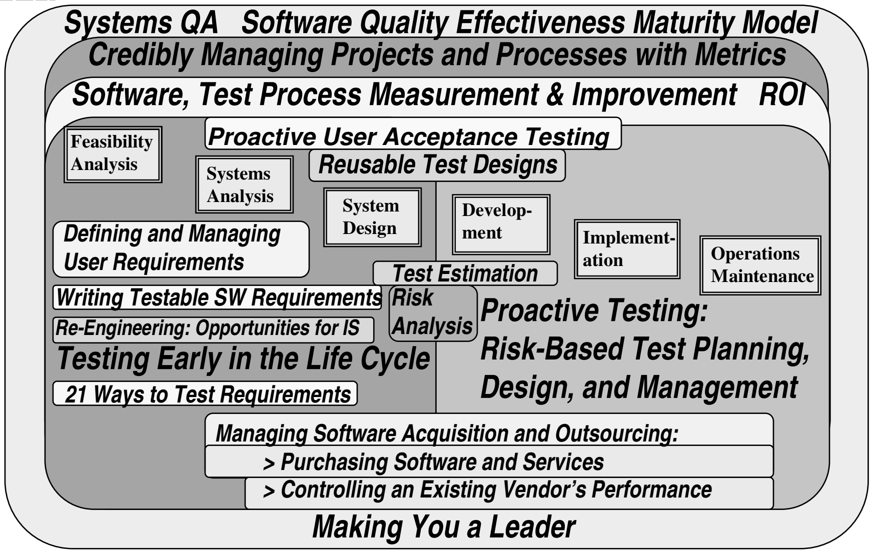
- U1-R1.1-U4-R4-Exit
- U1-R1.2-U2-R2.1-U4-R4-Exit
- U1-R1.2-U2-R2.2-U3-R3-U4-R4-Exit
- U1-R1.2-U3-R3-U4-R4-Exit
- *U1-R1.2-Exit*
- *U1-R1.2-U2-R2.2-Exit*
- *U1-R1.2-U2-R2.2-U3-Exit*
- *U3-R3.1-Exit*
- *U4-R4.1-Exit*

How many different inputs cause each path be executed?

Summary

- Ambiguous, inconsistent, and difficult-to-verify requirements and designs are most commonly what is considered “untestable”
- Testers’ concerns about (lack of) testability are often resisted as trivial nitpicking
- Test cases often can be created for seemingly untestable requirements/designs, may in fact serve as more useful definitions, and illustrate testability concerns in a manner more likely to be appreciated

GO PRO MANAGEMENT, INC.
Seminars and Consulting--Relation to Life Cycle


Systems QA Software Quality Effectiveness Maturity Model
Credibly Managing Projects and Processes with Metrics
Software, Test Process Measurement & Improvement ROI
Feasibility Analysis
Proactive User Acceptance Testing
Systems Analysis
Reusable Test Designs
System Design
Develop-ment
Implement-ation
Operations Maintenance
Defining and Managing User Requirements
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.
- 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.
- 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
- International Institute of Business Analysis (IIBA) Business Analysis Body of Knowledge (BABOK) subject expert.
- Admitted to the Massachusetts Bar and licensed to practice law in Massachusetts.
- Author of book: ***Discovering REAL Business Requirements for Software Project Success***

Software Quality Group of New England

SQGNE is made possible by the support of our sponsors:

Welcome to SQGNE's 15th season!

- An all-volunteer group with no membership dues!
- Supported entirely by our sponsors…
- Over 700+ members
- Monthly meetings - Sept to July on 2nd Wed of month
- E-mail list - contact John Pustaver **pustaver@ieee.org**
- SQGNE Web site: **www.swqual.com/sqgne/main.html**

SQGNE Volunteers / Mission

- John Pustaver - Founder and Director
- Steve Rakitin – Programs and web site
- Gene Freyberger – Annual Survey

Our gracious Hosts:

- Paul Ratty (Sun) - room, copies, cookies
- Tom Arakel (Sun) - room, copies, cookies
- Jack Guilderson (Sun) – A/V equipment

SQGNE Mission

- To promote use of engineering and management techniques that lead to delivery of high quality software
- To disseminate concepts and techniques related to software quality engineering and software engineering process
- To provide a forum for discussion of concepts and techniques related to software quality engineering and the software engineering process
- To provide networking opportunities for software quality professionals

ASQ Software Division

- Software Quality Live - for ASQ SW Div members…
- Software Quality Professional Journal www.asq.org/pub/sqp/
- CSQE Certification info at www.asq.org/software/getcertified
- SW Div info at www.asq.org/software

Software Quality Professional

SQGNE 2008-09 Schedule

Speaker	Affiliation	Date	Topic
1. Lou Cohen	**None**	**9/10/08**	**Introduction to using Quality Function Deployment on Software Projects**
2. Brian LeSuer	**Star Quality**	**10/8/08**	**A Survey of Test Automation Projects**
3. Howie Dow and Steve Rakitin	**None**	**11/12/08**	**Estimating using Wideband Delphi Method - An interactive exercise**
4. Russ Ohanian	**Tizor Systems**	**12/10/08**	**Integrating Agile into the Development Process**
5. Johanna Rothman	**Rothman & Assoc.**	**1/14/09**	**Schedule Games**
6. Carol Perletz	**None**	**2/11/09**	**The Nitty Gritty of QA Project Management**
7. Robin Goldsmith	**GoPro Management**	**3/11/09**	**Testing the Untestable**
8. Scott Matusmoto or Paco Hope	Cigital Networks	4/8/09	Automating security testing of web apps using cURL and Perl
9. Derek Kozikowski	None	5/13/09	Automated Functional Test Design
10. Stan Wrobel	CSC	6/10/09	Test Tool - Make or Buy?
11. Everyone		7/9/09	Annual Hot Topics Night…

Tonight's Speaker...

Testing the Untestable
Robin Goldsmith

Testability of requirements and design is a major concern for testers. When something is not testable, it's usually because it's not clear, which increases chances of development errors; and regardless, one cannot reliably do the job of testing whether implementation is correct. Defining testable Quality Factors (often called "nonfunctional requirements") is especially challenging. In this eye-opening interactive presentation, Robin Goldsmith shows how tests indeed can be created for seemingly untestable requirements/designs, with the side benefit of helping correct likely problem sources without engendering so much of the resistance testers typically tend to encounter.

* Common characteristics of untestable requirements and designs.
* Why others often resist testers' concerns about testability.
* Creating test cases for seemingly untestable requirements/designs.

Robin F. Goldsmith, JD has been President of Go Pro Management, Inc. consultancy since 1982. He works directly with and trains business and systems professionals in requirements analysis, quality and testing, software acquisition, project management and leadership, metrics, and process improvement. He partners with ProveIT.net in providing ROI Value Modeling™ tools, training, and advisory services.

Previously he was a developer, systems programmer/DBA/QA, and project leader with the City of Cleveland, leading financial institutions, and a "Big 4" consulting firm.

Author of the Proactive Testing™ methodology, numerous articles, and the recent Artech House book *Discovering REAL Business Requirements for Software Project Success*, and a frequent featured speaker at leading professional conferences, he was formerly International Vice President of the Association for Systems Management and Executive Editor of the *Journal of Systems Management*. He was Founding Chairman of the New England Center for Organizational Effectiveness. He belongs to the Boston SPIN and served on the SEPG'95 Planning and Program Committees.

Mr. Goldsmith held virtually all ASQ Boston Section leadership positions, and Chaired record attendance BOSCON 2000 and 2001 Annual Quality Conferences. He was a member of the ASQ Software Division Methods Committee and the IEEE Software Test Documentation Std. 829-2008 revision Committee. He is a member of the International Institute of Software Testing and International Institute (IIST) for Software Process (IISP) Body of Knowledge Advisory Boards. He is a subject expert on requirements and testing for TechTarget and a subject expert and reviewer for the International Institute of Business Analysis (IIBA) Business Analysis Body of Knowledge (BABOK).

He holds the following degrees: Kenyon College, A.B. with Honors in Psychology; Pennsylvania State University, M.S. in Psychology; Suffolk University, J.D.; Boston University, LL.M. in Tax Law. Mr. Goldsmith is a member of the Massachusetts Bar and licensed to practice law in MA.