Add Steak to Exploratory Testing's Parlor-Trick Sizzle

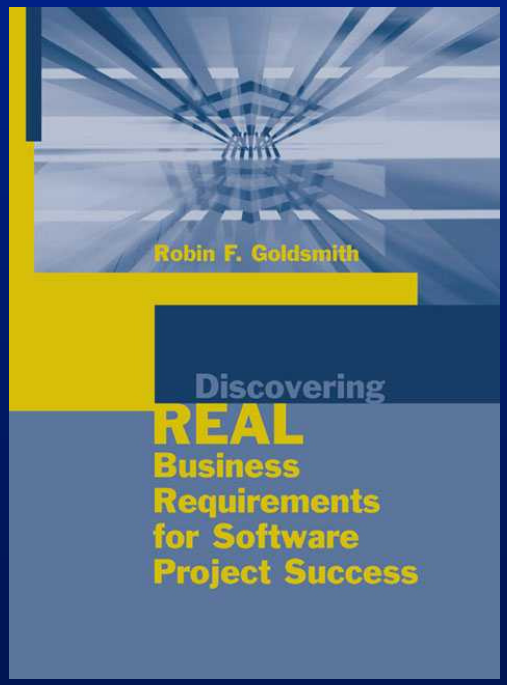
Robin F. Goldsmith, JD

GO PRO MANAGEMENT, INC.

SYSTEM ACQUISITION & DEVELOPMENT

QUALITY/TESTING

PRODUCTIVITY



22 CYNTHIA ROAD

NEEDHAM, MA 02494-1412

INFO@GOPROMANAGEMENT.COM

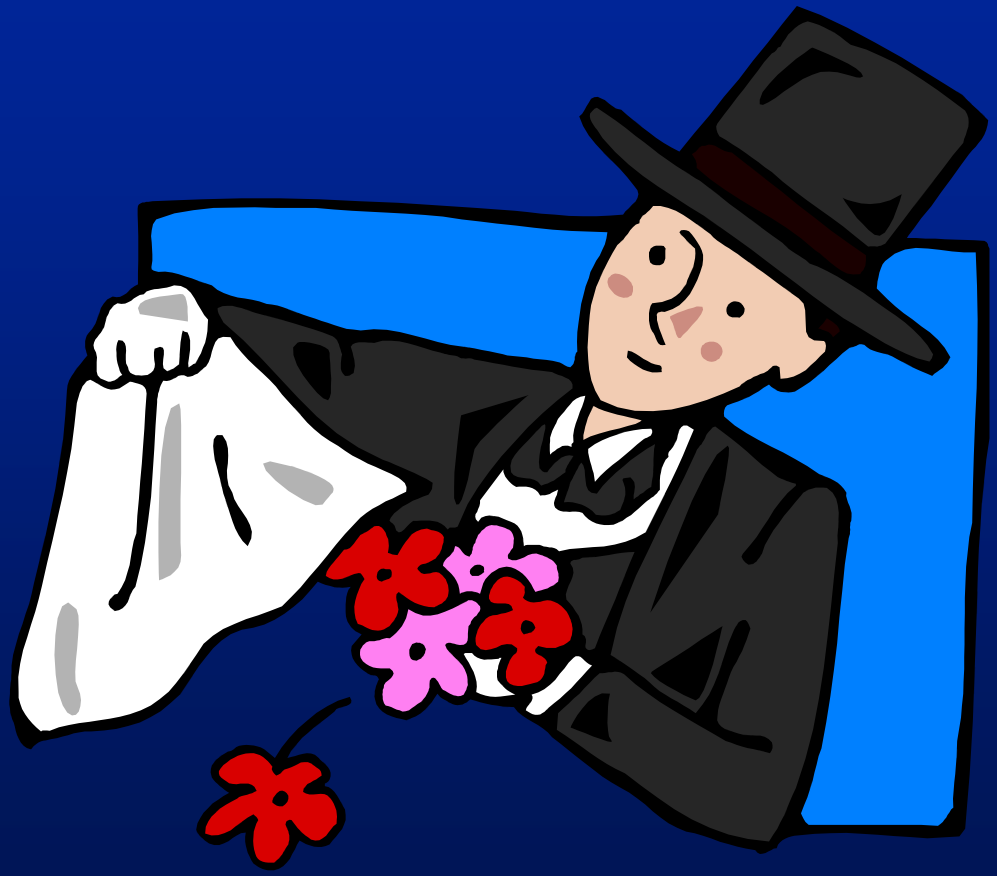
WWW.GOPROMANAGEMENT.COM

(781) 444-5753 *VOICE/FAX*

Despite Its Current Prevalence, Exploratory Testing Is NOT All There Is

A parlor trick is a simple magic trick which is generally easy to execute. Such tricks are used to amuse people at parties, and are sometimes called party tricks.

http://www.wisegeek.com/what-is-a-parlor-trick.htm



Objectives

- Identify key underlying rationale for and concepts of exploratory testing.
- Explain why such tests may
  - Divert test time and resources to superficial situations
  - Distract testers from catching more important errors that exploratory methods are likely to miss.
- Describe more reliable and useful approaches that
  - Put contextual testing into context
  - Leverage exploratory testing strengths in ways that provide more substantive value by detecting more important defects.

Key Exploratory Testing Concepts

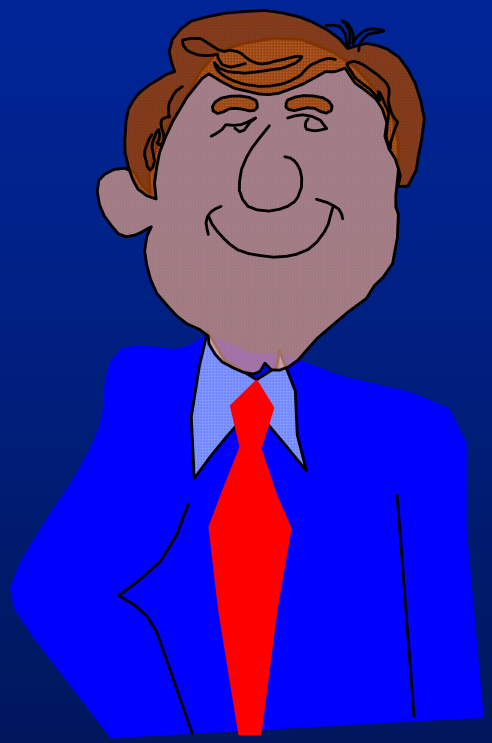
① I'm a smart guy and can find plenty of defects, so I just need to spend my time executing tests

② Written test planning and design is a waste of time and effort

③ Written requirements are not necessary for [and for some reason, shouldn't be basis of] testing

① *I'm a smart guy and can find plenty of defects, so I just need to spend my time executing tests*

- Look at all the cool bugs I identify in this software I've never even seen before
- Explore to find out how the code actually operates
- The more time I have to execute tests, the more bugs I find; so I should spend all my time running tests



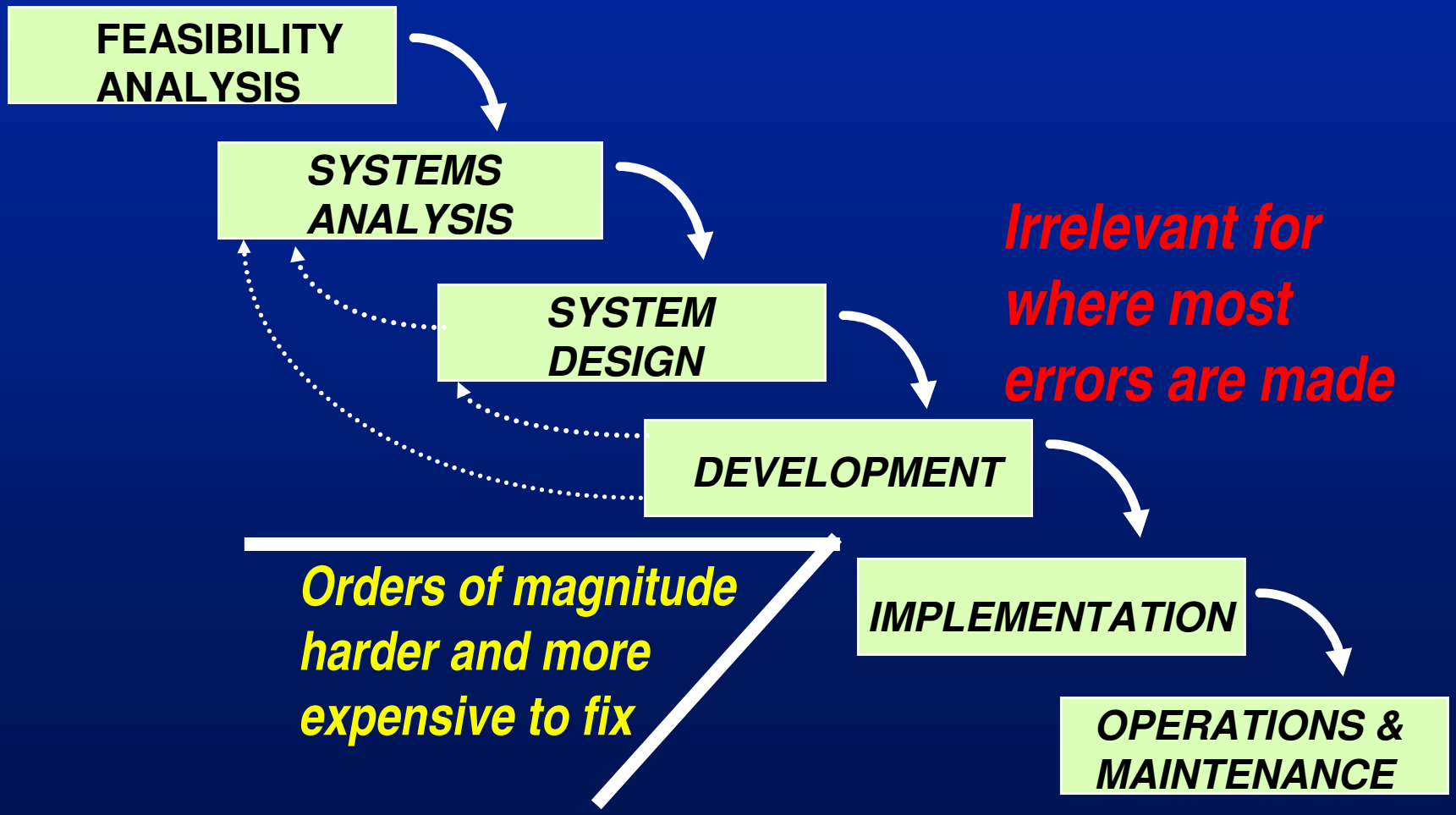
Is it inescapably evident to you how smart I am?

Click, Click, Kaboom!

Provoking strange software behavior often may be the main rationale for exploratory testing.

While certainly attention-grabbing and even entertaining, such feats may be testing's version of parlor tricks—all sizzle and no steak.

Exploratory Testing Begins After Virtually All Errors Already Have Been Made

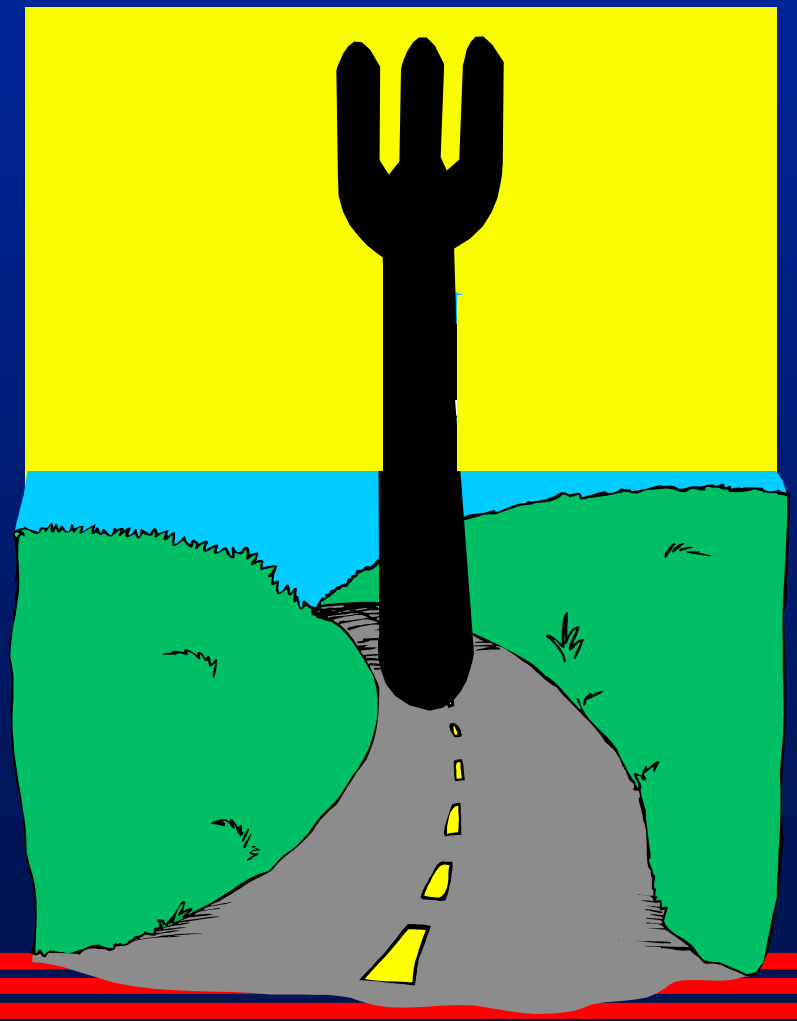


Testing vs. Experimentation

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

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

What Are the Chances Exploratory Testing Catches the Most Important Defects?



If you don't know where you're going, any road will do.

"When you come to a fork in the road, take it."

- Yogi Berra

Ref: Johnny Carson, "Tea Time Movies"

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?

②Written test planning and design is a waste of time and effort

- The more time spent writing tests, the less time to run tests→so don't write anything.
- Apparently equates written test planning and design with test scripts that contain extensive keystroke-level procedural detail
- Perhaps to emphasize that nothing is written, some prominent Exploratory gurus do advocate guiding testing with nonsensical mnemonics only they can remember (and thus show how smart they are)

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

An automated test execution tool can do both: faster, cheaper, and more reliably

- **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
  - Virtually assures finding the least amount of errors

It's not just Exploratory folks who recognize this!

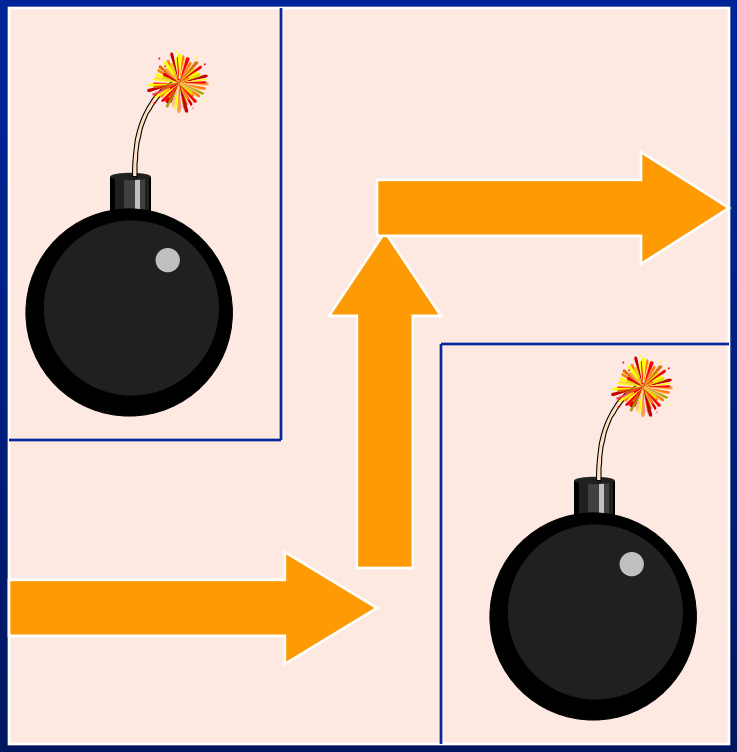
Apparent Shortcomings of Testing from Scripts

- Experienced testers find two-three times as many errors with same script (Cem Kaner)

  So Exploratory advocates often conclude:

  Don't write anything

- Many others realize a very procedurally-detailed script creates the minefield effect



Beware the minefield effect
Software "learns" to pass tests

News Flash!

http://go.techtarget.com/r/4190934/347388 **Software Quality News:**

Kaner: Exploratory testing better than scripted testing

By Jennette Mullaney, Associate Editor
05 Aug 2008 I SearchSoftwareQuality.com

TORONTO -- Exploratory testing is superior to scripted testing, resulting in better tests and better testers, noted tester Cem Kaner told attendees at the Conference of the Association for Software Testing (CAST). Kaner, a software engineering professor at the Florida Institute of Technology, advocated that testers use checklists rather than scripts in his keynote speech, "The Value of Checklists and the Danger of Scripts: What Legal Training Suggests for Testers."

So, What's the Value of Writing Something vs. Not Writing It?

- ✓ Don't forget it
- ✓ Can share it
- ✓ Can review it
- ✓ Can refine it
- ✓ Can repeat it
- ✓ Can use it as a guide
- ✓ Can check against it
- ✓

A checklist is written and has these strengths, but also has weaknesses:

- 👎 Generic rather than specific to application
- 👎 Can give illusion of coverage
- 👎 May actually prevent thoughtfulness

Effective testers update structured tests with Exploratory info

Importance of Writing Test Plans/Designs as Part of Design (Before Coding)

- Time to write is same, but pre-coding adds value
  - More thorough, what should be, not just what is
  - More efficient
    - Less delay
    - Fewer interruptions
  - Actually cuts the developer's time and effort
    - Test planning is one of 15 ways to test the design
    - Prevents rework

Test Planning/Design Can Reduce Development Time and Effort

Proactive

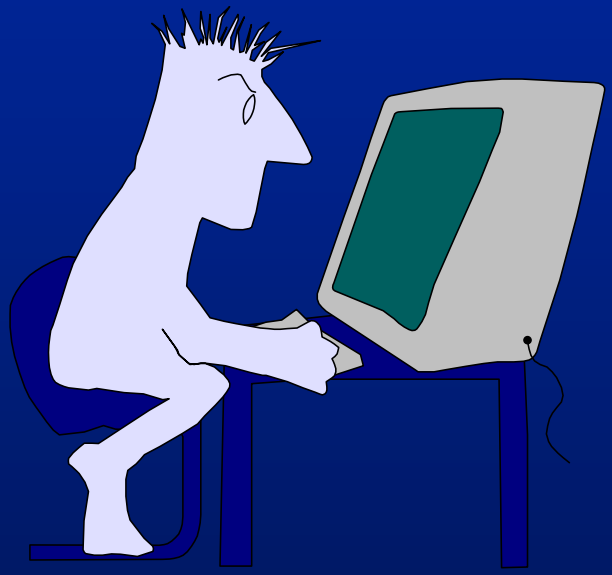
Simple Spec:

Operator enters customer number

C123

Computer displays customer name

Jones, John P.



Rework ≅ ~~40%~~ of Developer Time

WIIFM

Could it be coded wrong? What effect?

Test Case=Input/Condition, Expected Result

Test Case Specification

Input, Condition

Operator enters customer number at location *X*.

Expected Result

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

Test Case Values

Customer Number	Customer Name
C123	*Jones, John P.*
C124	*not found*

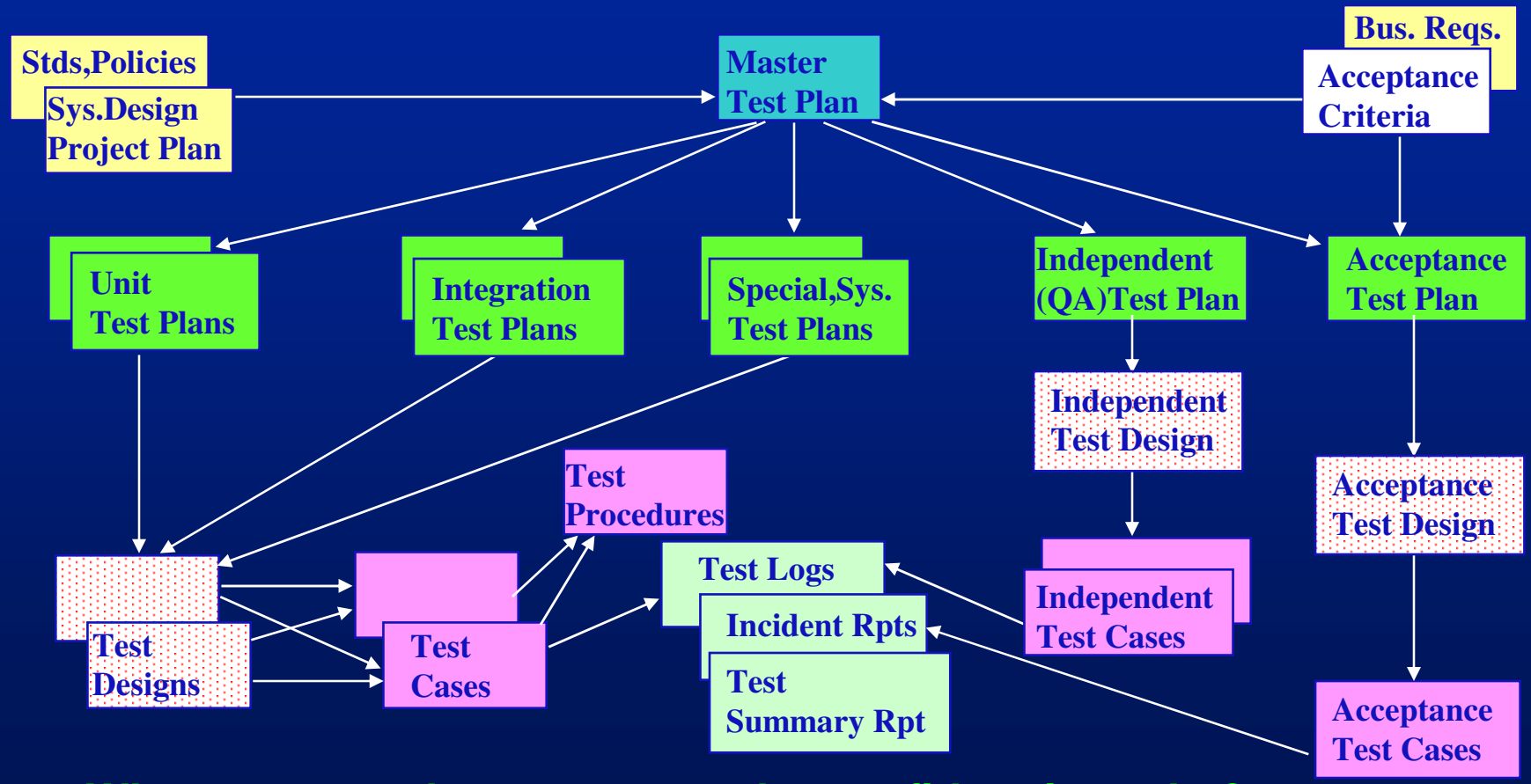
Test Script—Does Not Need Procedure

Input	Expected Result	Actual
Menu=*Find Customer*	Customer entry screen	
Cust. No. = *C123*	Cust. Name *Jones, John P.*	
Cancel button	Menu	
Menu=*Find Customer*	Customer entry screen	
Cust. No. = *C124*	Cust. Name *Not Found*	
Cancel button	Menu	

Test Matrix—All Apply to Same Procedure

Test No.	Input Cust. No.	Type	Active	Expected Results Cust. Name	Actual
1	*C123*	*10*	*A*	*Jones, John P.*	
2	*C124*	*10*	*A*	*not found*	

Testware--Test (Plan) Documentation per ANSI/IEEE Std. 829-2008



What must we demonstrate to be confident it works?

Proactive Testing™ Planning/Design Spots & Prevents Bigger Risks Due to Design Errors

- **Test Plans** are project plans for the Testing (sub) Project
- Objectives, strategies, guidelines, standards
- Identify testing tasks, resources, effort, duration →schedule, budget Based on:
  - The set of tests to demonstrate (detailed test plans in Master Test Plan, test design specifications in Detailed Test Plans)
  - Test support, environment, hardware, software, facilities
  - Ancillary and administrative activities

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

One → Many ← One

③ *Written requirements are not necessary for [and for some reason, shouldn't be basis of] testing*

- Requirements are defined so poorly, and requirements change so much, that it's not appropriate to rely [solely, or even at all] on written requirements
- "There is nothing … that suggests requirements must be made absolutely clear and precise. "
- "Skilled testers evaluate the product against their understanding of unstated requirements and use their observations to challenge or question the project team's shared understanding of quality."

"Risk and Requirements-Based Testing"
James Bach *IEEE Computer* June 1999

Just Being Smart, At Best…

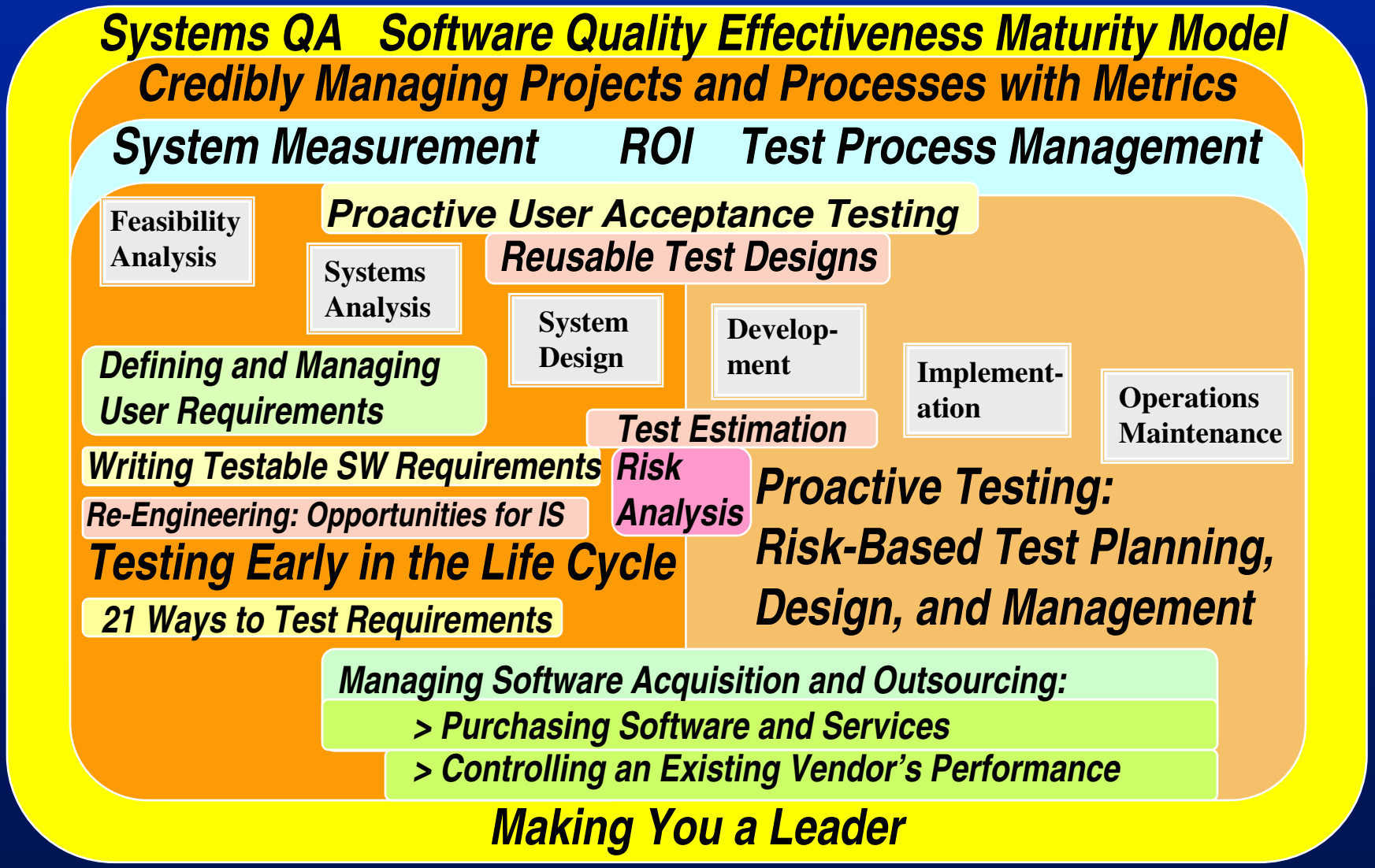
- May be relevant for judging superficialities evident in the user interface
- **Cannot possibly tell for sure whether the system meets REAL, business requirements, which are what provide value, especially with respect to**
  - Wrong and overlooked functionality
  - Business rules and most quality factors
  - Actual usage

Mainly will find "Parlor Trick" defects

Summary

- Exploratory Testing that relies largely on cleverness without adequate knowledge of REAL business requirements is likely (much more than recognized) to find mainly superficial parlor trick defects—and then only when they are hardest and most expensive to fix
- Meaningful Proactive Testing™ planning and design economically finds and prevents bigger risks than Exploratory Testing is likely even to become aware of
- Write no more than is useful—but no less—and update structured tests with Exploratory findings so they continually improve and rely less and less on guessing

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.
- 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 record-setting BOSCON 2000 and 2001, ASQ Boston Section's Annual Quality Conferences.
- TechTarget, SearchSoftwareQuality requirements and testing subject expert.
- Member IEEE Std. 829 for Software Test Documentation Standard Revision Committee.
- Member IEEE P730 Working Group rewriting IEEE Std. 730-2002 for Software Quality Assurance Plans.
- Member IEEE P1805 Working Group developing a Requirements Capture Language (RCL) standard.
- 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***