Truly Transformational Shift-Left Proactive Testing™

Robin F. Goldsmith, JD
GO PRO MANAGEMENT, INC.
SYSTEM ACQUISITION & DEVELOPMENT
QUALITY/TESTING
PRODUCTIVITY
BUSINESS ENGINEERING
TRAINING
22 CYNTHIA ROAD
NEEDHAM, MA 02494-1461
INFO@GOPROMANAGEMENT.COM
WWW.GOPROMANAGEMENT.COM
(781) 444-5753

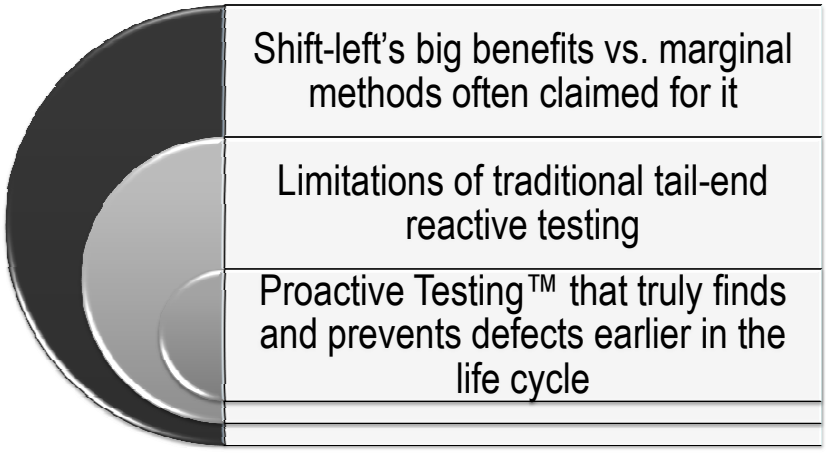
What Does "Shift Left" Mean to You?

- .
- .

"Shift Left" Efforts You Have Seen

What Was Done	How Well It Worked
• .	• .

Objectives



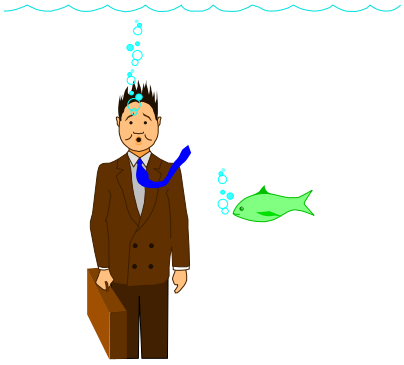
Typical Development Life Cycle

Tail-End Testing Issues

Maximum number of defects

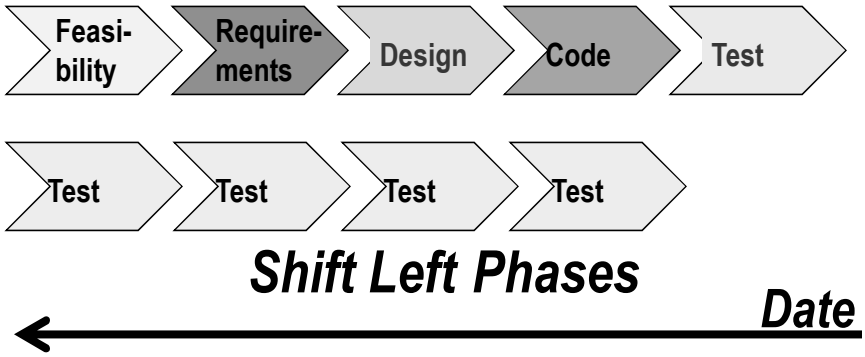
Minimal time to fix them

Maximal cost to fix each defect

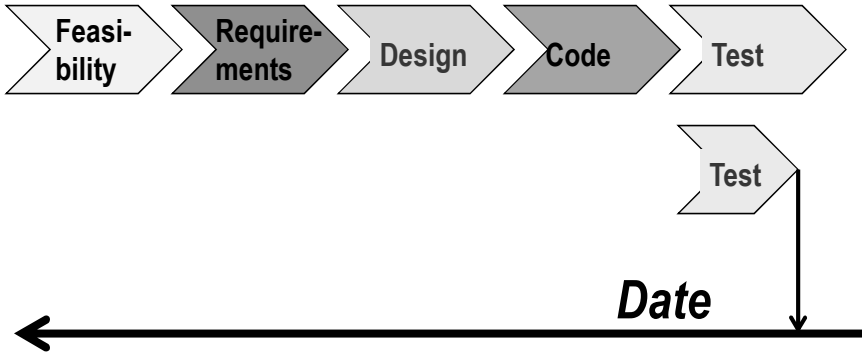


Too many missed and/or not fixed

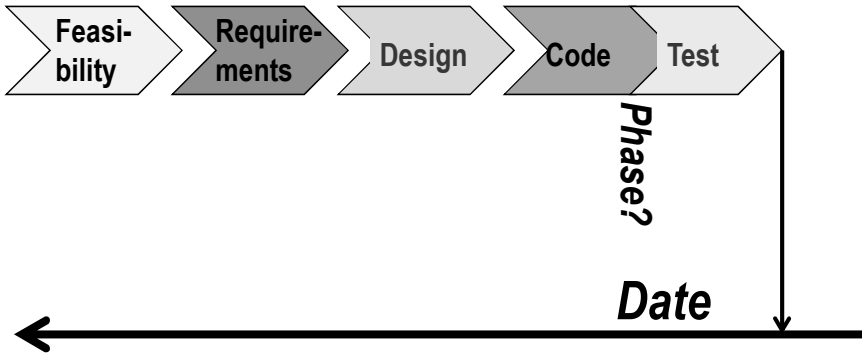
Solution—Shift Testing Left Two Ways



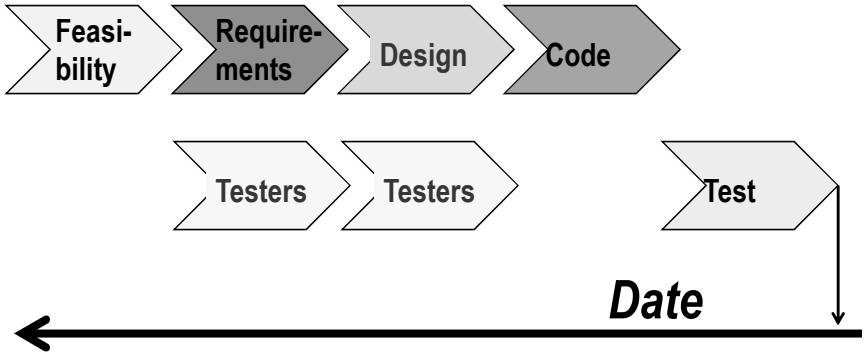
Claimed Shift-Left Approaches: Automated Testing



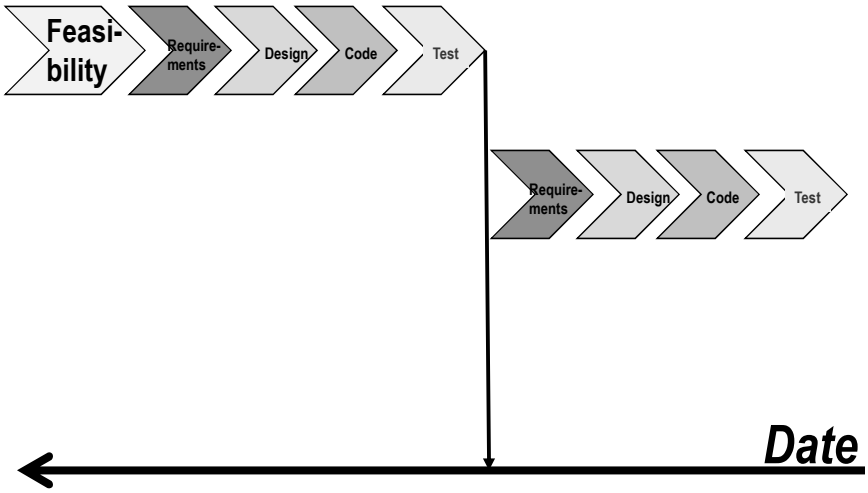
Rely on Developer to Test or Integrate QA/Testers with Developers



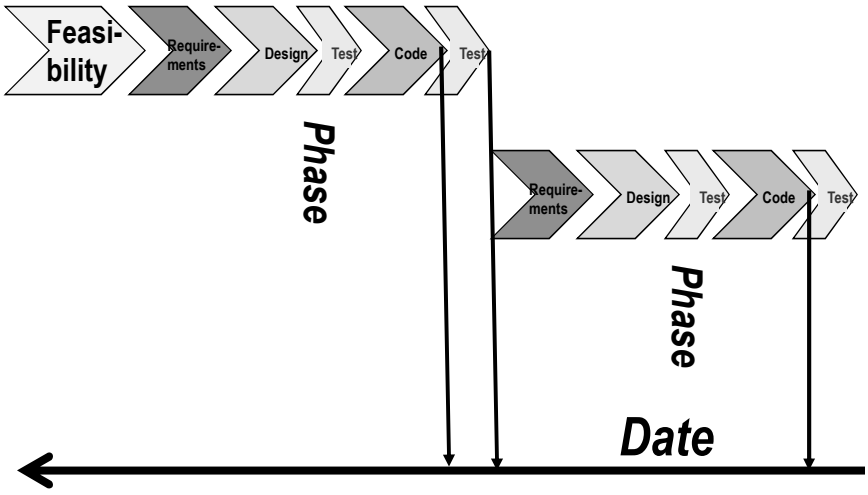
Involve Testers Observing Requirements and Design Definition



Small Iterations

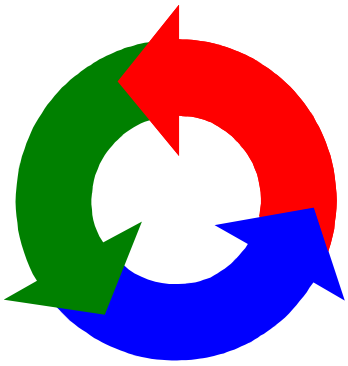


Small Iterations Plus Test-First



Test-First Development Surely Beats Traditional Test-Last (or Never) Coding

- Developer(s) decide how to test that code works and write code in the program being developed (Software Under Test—SUT) to perform the tests
- Then write program's regular, functional code
- Code works when included tests are passed



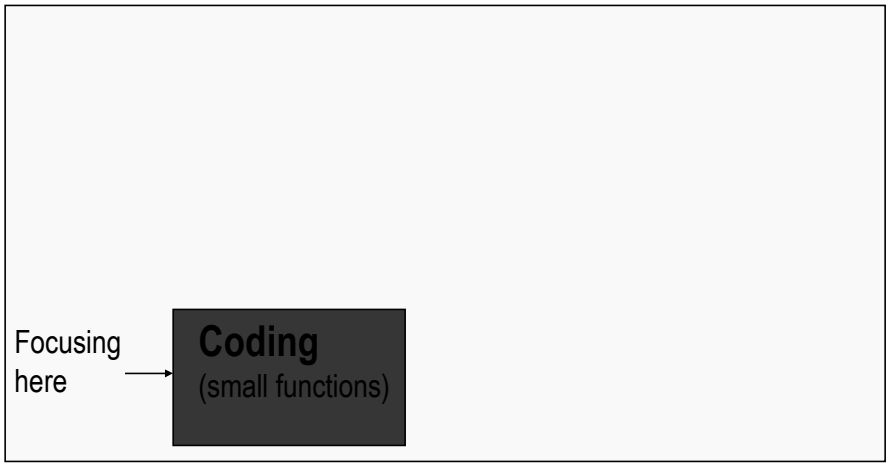
Included tests are re-executed for every change

Test-First Development Is Good; but Has Some Seldom-Recognized Limitations

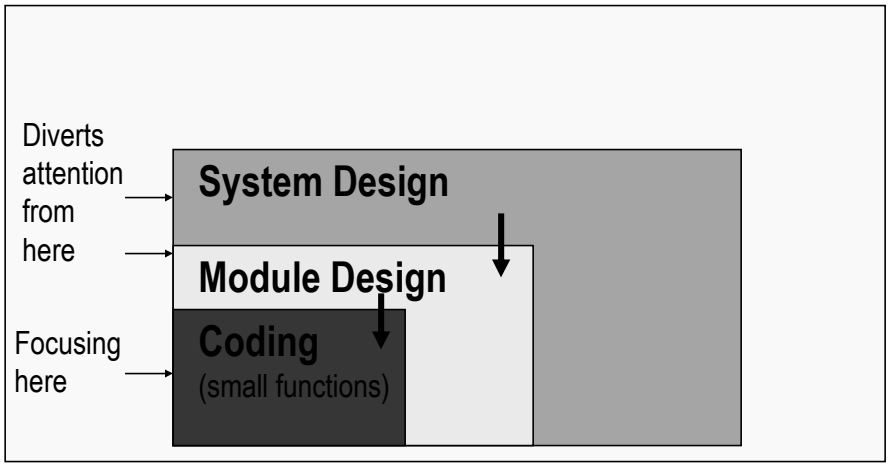
- Programmer/code-centric view can easily miss the bigger, more important issues to test
- Developer's (even the pair's) mindset defining tests is likely to be largely same as for the code
  - Mainly testing what is (going to be) written
  - Won't catch what developer doesn't understand adequately or overlooks
  - Developer still is unlikely to have a testing "break it" mindset or systematic test planning and design methods, so probably overlooks many conditions needing testing
- Agile's fanatical resistance to writing anything other than executable code, including tests, is high-effort with relatively low leverage payback

Plus the religious-like "How dare you question my Agile techniques?"

Coding Is Smallest Source of Errors



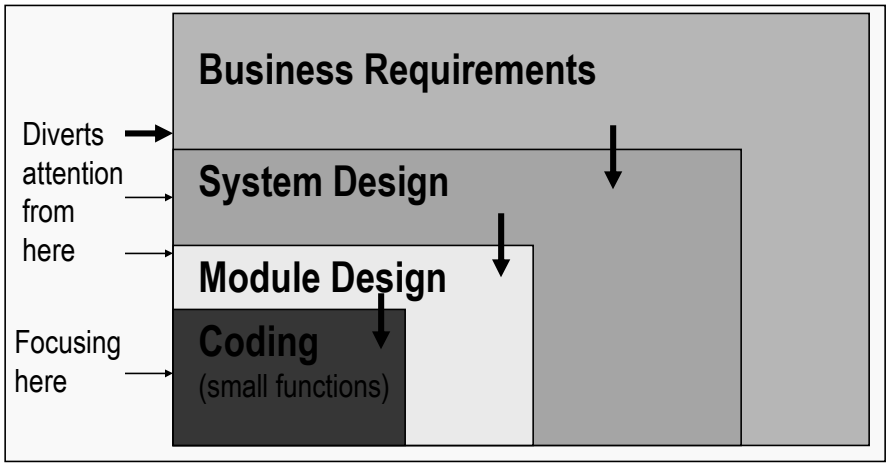
Coding Is Smallest Source of Errors



2/3 of errors in delivered code are in the design.

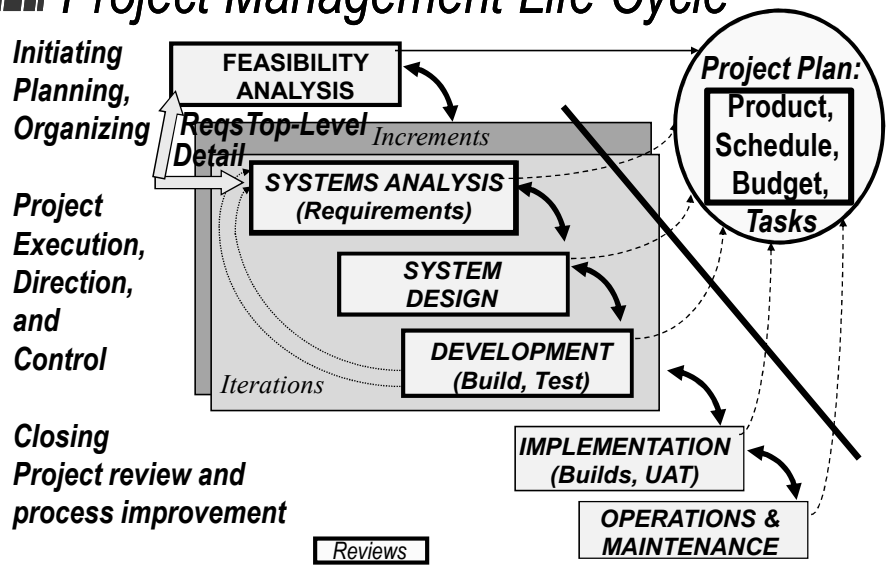
Does essentially having no design increase, decrease, or just mask that?

Coding Is Smallest Source of Errors



Missed/incorrect/unclear business requirements are biggest source of design problems

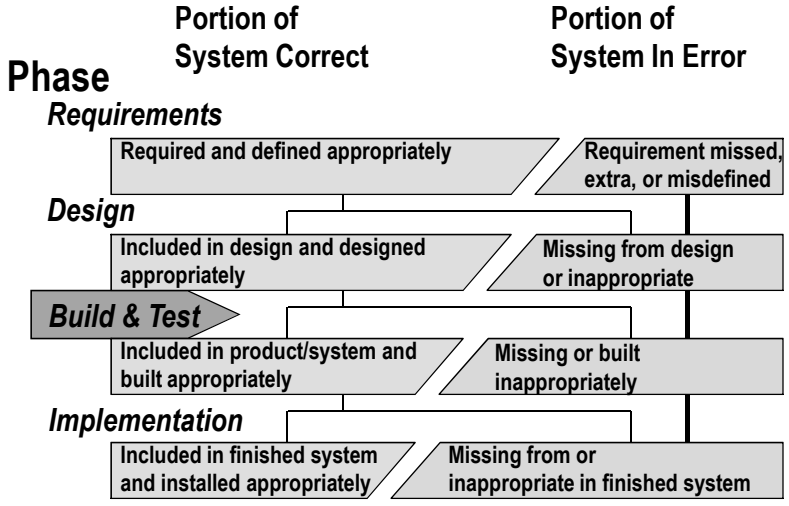
Project Management Life Cycle



Requirements Definition Usually Occurs in

Initiation/Feasibility Analysis	Systems Analysis (Requirements Phase)
• Top-level requirements • Compares two or more alternative approaches, including no change – Can it be done? – What are the estimated costs, benefits, and Return on Investment (ROI)? • Sets project, budget, and schedule—often in concrete and doomed to failure	• Drives requirements to detail • Translates to high-level conceptual design of solution consistent with chosen major approach • Should precede product/ vendor selection • ROI should be redone
Often occurs implicitly	***Often done iteratively***

Error Sources by Phase



IT (and Other?) Project Economics

- Maintenance is 66-90% of system cost
- Maintenance is mainly completion/ correction of development (wrong/missed requirements)
- 2/3 of finished system errors are requirements and design errors
- Fixing a requirements error will cost
  - 10X+ during development/construction
  - 75-1000X+ after installation (maintenance)

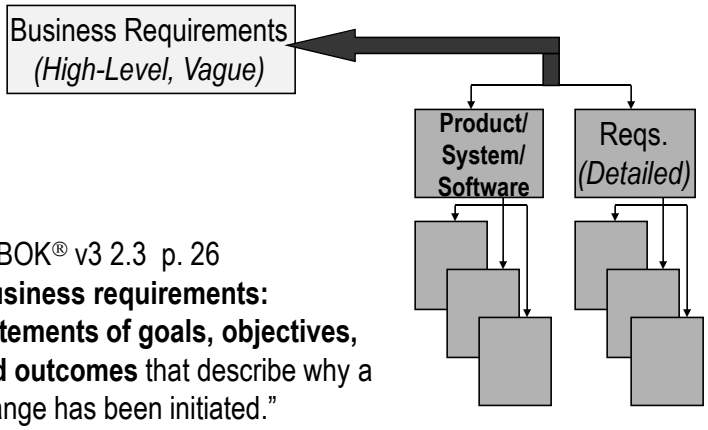
Do your organization's routine measures show these effects?

Big effects of true Shift-Left

➲Two Types of Requirements:

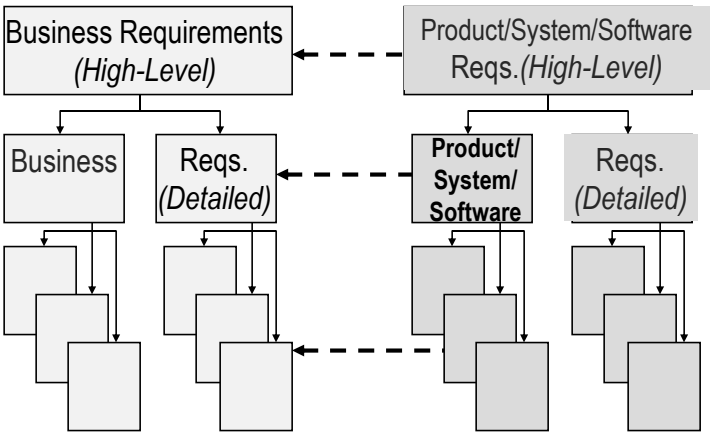
Business/User	Product/System/Software
• Business/user language & view, conceptual; *exists* within the business environment • Serves business objectives • ***What*** business results must be delivered to solve a business need (problem, opportunity, or challenge) and provide value when delivered/satisfied/met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

Even Requirements "Experts" Think the Difference Is Just Level of Detail

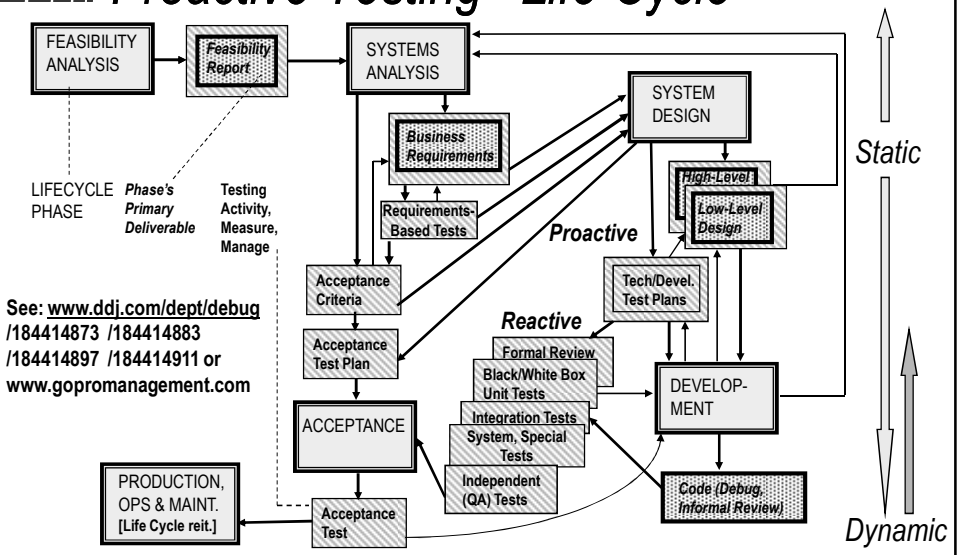


BABOK® v3 2.3 p. 26
"**Business requirements: statements of goals, objectives, and outcomes** that describe why a change has been initiated."

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



Proactive Testing™ Life Cycle



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

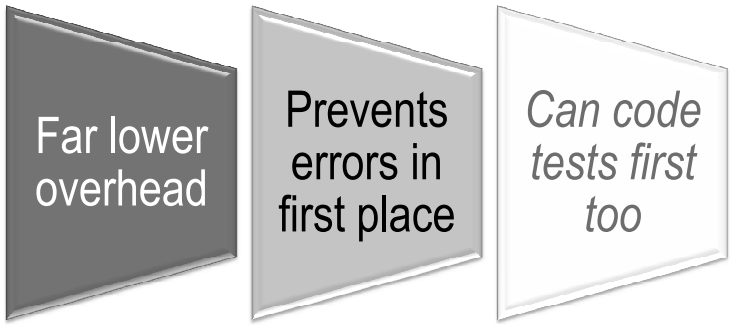
Key Proactive Testing™ Concepts *1 of 2*

- Develop iteratively—whatever size piece is coded should be designed and responsive to adequately defined REAL, *business* requirements, which in turn both should be tested
- Define more complete true user acceptance tests proactively at start, *keep independent of design*
- Use Proactive Testing™, in conjunction with more effective discovery and specification, techniques to improve the accuracy, completeness, and clarity/testability of the requirements and design
  - Write enough to help—but no more, and no less
  - Catch big-picture issues, keep refocusing based on risk

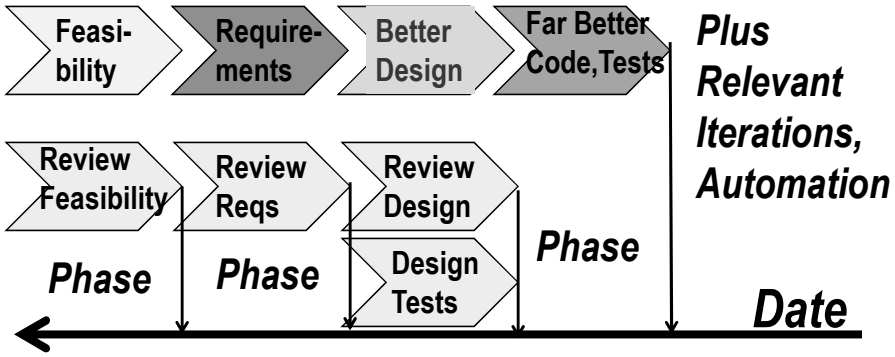
Key Proactive Testing™ Concepts 2 of 2

- **Let** higher-level (than just code) **testing** planning/design thought processes **drive development** to
  - Economically anticipate and avert larger consequences of design issues that ordinarily cause rework
  - Plan for coding/testing early to avoid biggest rework risks as well as implementing immediately useful functionality
  - Increase awareness of more of the frequently-overlooked conditions that code/tests must address
- Plan/design tests early, prioritize, promote reuse
  - Concisely define, detect issues top-down at varying levels
  - Create and apply reusable test designs and test cases
  - Implement selectively based on risk

Test Planning/Design vs. Test-First



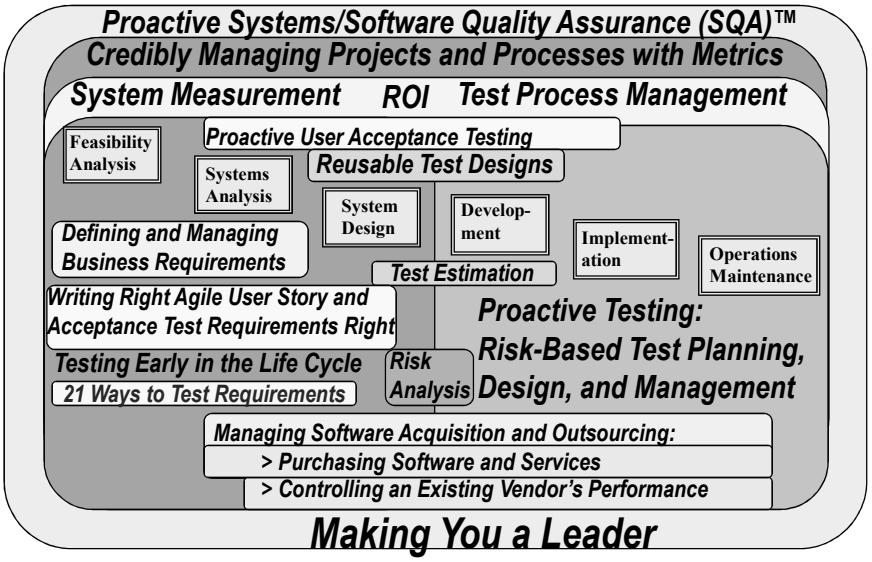
Proactive Testing Shifts Testing Left Both Ways



Summary

- Typical Agile test-driven development has advantages compared to traditional test-last (or never) development and reactive testing but also has (often unrecognized) limitations due to its narrow programmer-based focus
- Proactive Testing™ enables truly Agile quicker, cheaper, *and* better software development by feeding low-overhead high-leverage test planning and design information into development throughout the life cycle
- A variety of Proactive Testing™ techniques efficiently reveal numerous otherwise overlooked test conditions *at varying levels*, starting with detecting requirements and design defect early, which then can be addressed selectively based on risk and often can be reused

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



Robin F. Goldsmith, JD

robin@gopromanagement.com www.gopromanagment.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.
- 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 Working Group.
- Member IEEE P730-2014 standard for Software Quality Assurance Revision Working Group.
- International Institute of Business Analysis (IIBA) Business Analysis Body of Knowledge (BABOKv2) subject expert.
- TechTarget 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— Write Right Agile User Story and Acceptance Test Requirements Right***