



**PITFALLS OF RAD, AGILE/XP  
AND OTHER FORMS OF  
ITERATIVE DEVELOPMENT**

**AND  
HOW THEY LEAD TO  
PROJECT FAILURE**



***The following material is a copyrighted product of Lois Zells. Its' use and reproduction are governed by United States and International copyright laws. In the spirit of spreading these ideas, recipients of this document may reproduce this material as a complete unit (no extracts), provided the copyright notifications on the material are not in any way modified or obliterated. This material may not be used to write books or articles or to teach seminars. Except for internal use as just stated, this material may not be sold, reproduced, distributed, or modified without the writer permission of the author.***

**© Lois Zells, 2007, 2008 2009; All Rights Reserved.**

**Pages 46, 47, 49, 50: Adapted From  
"Boehm/Turner, BALANCING AGILITY & DISCIPLINE,  
Figure 2.2, © 2004 Pearson Education Inc.,  
Reproduced by permission of Pearson Education,  
Inc. All rights Reserved**

**Lois Zells  
1800 S PCH #54  
Redondo Beach, CA. 90277  
(310) 316-2810  
[lzells@aol.com](mailto:lzells@aol.com)**

# Lois Zells

- **Author**

- Managing Software Projects
- Total Quality Management For Software
- The AMA Project and Program Handbook
- Many Publications

- **Expert Witness Testimony**

- **Support Services**

- Project Mentoring
- Project Assessments
- Audits and Inspections
- Software Maturity Assessments
- Change Readiness Assessments

- **Consultant and Educator**

- Project Management
- Strategic Planning
- Software Engineering Techniques
- Methodologies
- Software Asset Management
- Product Development
- Total Quality Management
- Quality Function Deployment

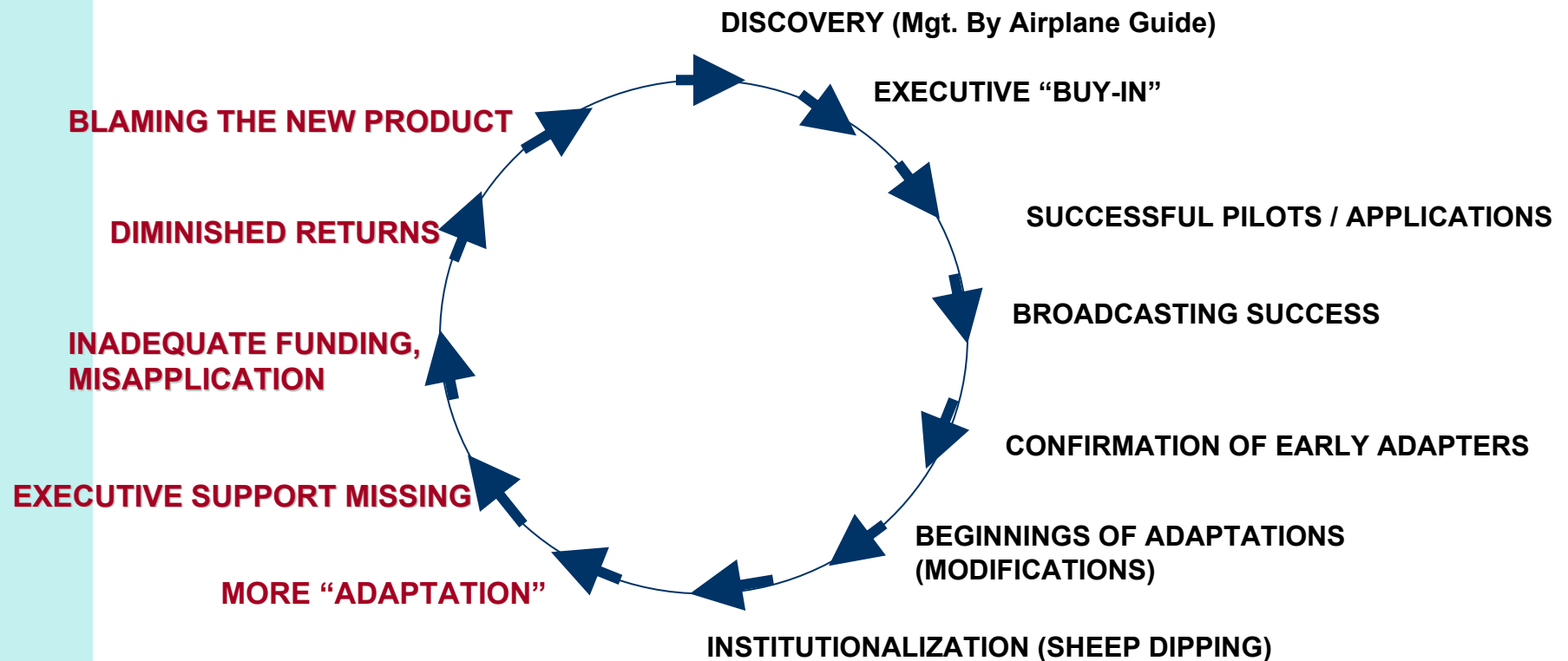
- **Yourdon/DeMarco Disciple**

- **Project Management Institute**

- Founder, Past Co-chair, & Past Executive Advisor I/S SIG
- 1993 Woman Of The Year
- Zells/Wilson Scholarship

© Lois Zells, 2007, 2008, 2009; All Rights Reserved

# Implementing New Technologies



Adapted from "Life Cycle of a Silver Bullet," Sarah Sheard, Software Productivity Consortium

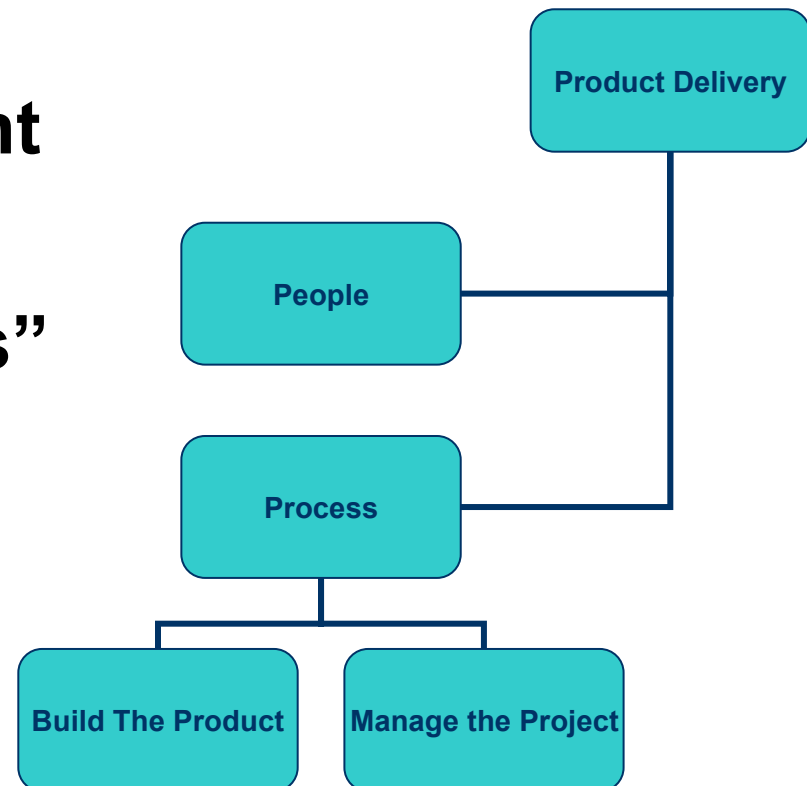
© Lois Zells, 2007, 2008, 2009; All Rights Reserved

# **We're Here Today To Talk About**

- **Best Practices In Iterative Development**
- **Major Contributors To Iterative Project Failures—And How to Avoid Them**
- **Is Pure Iterative The Way To Go?**

# Best Practices In Iterative Development

- People
- Product Development
- Project Management
- Process “Awareness”



# Best Practices In Iterative Development

- **People**
  - **Strong management support**
  - **Committed stakeholders**
  - **Sustained, ongoing customer involvement**
    - **Project participants**
    - **End-user demos**
  - **Shared, communal team work space**

# Best Practices In Iterative Development

- **People**

- **Teams**

- **Self-directed, self-organizing**
    - **Collaborative**
    - **Small team size:  $7 \pm 2$  practitioners**
    - **Include user participants**
    - **Not for beginners: **must have technical expertise****

# Best Practices In Iterative Development

## ● Product Development

- Incremental Iterative Development (IID)
- Quick delivery of **working features** paramount.
- Accommodates empirical process concept rather than prescriptive processes.
- User requirements captured in small low tech specifications.
- **Sustained end-user participation** in iteration demos.
- **Adding to/changing scope** of an in-progress iteration is **not allowed**.

# Best Practices In Iterative Development

- **Product Development**
  - Sustainable pace of work
  - Partitioning larger projects into smaller releases
  - **Highly-disciplined coding standards**
    - **Driven by the user interface—Designing for simplicity**
      - Highly cohesive
      - Loosely coupled
    - **Pair programming**
    - **Team ownership of code**
  - Test-driven development
  - **On-going and frequent refactoring**

# Best Practices In Iterative Development

- **Project Management**
  - **Project manager given real authority**
    - (e.g. defray/remove bottlenecks and roadblocks)
  - **Total, completed project  $\leq$  1 year**
  - **Adaptive rather than predicative planning**
  - **Time-boxed iterations (3-8 weeks)**
  - **Front-end planning for each iteration acceptable**

# Best Practices In Iterative Development

- **Project Management**
  - **Short planning sessions**
    - **Estimating**
      - **Participative**
      - **Often with Delphi Method**
      - **Drilling down planning exercises**
    - **For the next iteration(s)**
      - **At the inch-pebble level**
      - **Date-driven; or**
      - **Specification driven**
    - **High-level for rest of project**

# Best Practices In Iterative Development

- **Project Management**
  - **Frequent face-to-face status meetings**
    - Short intervals
    - Preparation
    - “No” excused absences
  - **Date and size changes**
    - OK from refinement
    - OK to re-plan
    - “Late” not in vocabulary

# Best Practices In Iterative Development

- **Process “Awareness”**
  - Frequent reflections
    - **Inspections**
    - **End user demos**
    - Adaptive planning
    - Refactoring
  - Continuous improvement: **frequent retrospectives** & strong emphasis on using experiences as input to process improvement in next “rounds.”

# Iterative Approaches To Development

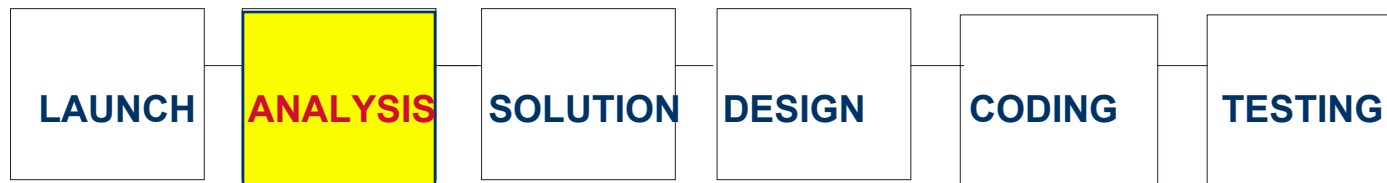
- Major Contributors To Iterative Project Failures
  - Misunderstanding of requirements and design still perpetuated.
    - Users are often confused by co-mingling of business and technical requirements and the associated inability to verify and validate.
    - Still hard to validate the internal formulas and the logic of the business rules.
  - Participants don't understand their roles and time commitments.
  - Unrealistic time, people, and dollar schedules to completion.
    - Inability to predict the number of iterations.
    - Belief that scope is hard to control; and it is hard to differentiate between normal development refinement and change of scope.
    - Schedule crunches may force poor designs and an architecturally unsound system.
    - All of the “ilities” may be affected: performability, reliability, maintainability, testability, usability... ..
  - Interfaces don't work together. Systems can't talk to each other.
  - Inspections (peer reviews) get shortchanged.
  - Testing Best Practices forgotten.

# Misunderstanding of Requirements and Design

- **Incorrect and incomplete requirements and design come from:**
  - missing or incorrect business algorithms
  - missing data
  - unstable databases and tables
  - constantly shifting technical architecture
- **Incorrect and incomplete requirements and design have the worst DEFECT impacts on:**
  - Coding
  - Testing
  - Project management schedules

# Analysis Co-Mingling

- **The Age-old Requirements Definition Dilemma**
  - **It's always been a challenge:**
    - **Requirements Overkill**
    - **Analysis Paralysis**
    - **They change their minds anyway**



# Know Thy Deliverables: Business Requirements Analysis

Defines ***WHAT*** business problem  
is to be solved

- Done from the user's perspective
- Some limited technical aspects

90% business-driven and 10% technically-driven

# Know Thy Deliverables: Business Requirements Analysis

- ***You will encounter noise:***
  - *Old man/machine interfaces*
  - *Old Departmental Boundaries*
  - *Old Geographic Boundaries*
  - *Out-of-date job descriptions*
  - *Out-of-date technologies*
- ***You will also find requirements for:***
  - *Processes that must be in place at the organizational level*
  - *Manual processes*
  - *Actual software development requirements*
- ***You are still looking for the core of the business requirements.***

# Exit Criteria: Business Requirements Analysis

- To define the business, must do:
    - The ***business*** architecture: all the “building blocks<sup>\*</sup>” and their ***interfaces***
    - Then, ***for each iteration you’ re delivering:***
      - Fully-defined business-related data definitions
      - All ***business-related rules*** for using data specified
        - ***business-related formulas***
        - ***business-related logic***
      - Test data
        1. All related user test data (including the census)
        2. Interface test data
      - End-users have certified inspection demo
  - Also good to do:
    - Performance criteria
    - User satisfaction traceability
    - Requirements traceability
    - Security requirements
- <sup>\*</sup> Business data and process components

# Iterative Approaches To Development

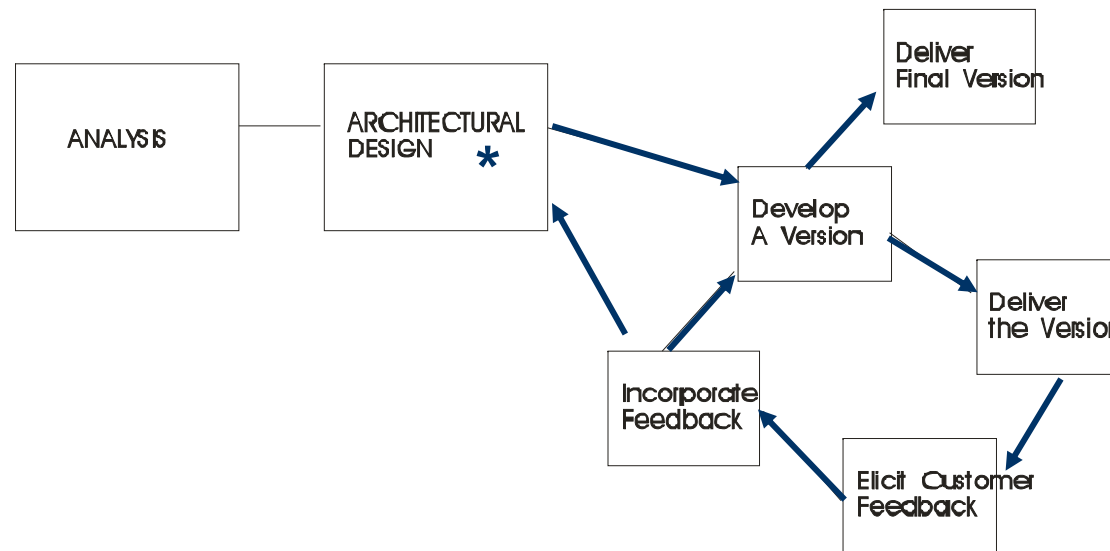
## Important critical success factors

- **Unrealistic Participant Commitments:**
  - Each iteration requires at least 1 LOB (line of business) expert
  - The user and the developer must work together to develop the iterations.
    - It is the role of the *user* to be in the project and *communicate the business requirements*.
    - It is the role of the *developer* to *translate* requirements in a manner that can be *understood and verified by the user*.

# Iterative Approaches To Development

- **There's no free lunch!**
  - Not necessarily less work and/or shorter schedules!!!
- **Unrealistic Schedule Commitments**
  - Iterations continue until a final product is developed that satisfies the customers' needs. (??How Many??)
    - Iterative Development means iterative planning!!!!
      - We are committed to the detailed plans for what we do know.
      - We do the best we can with what we don't know.
      - We are not accountable for what we don't know.
    - Time-boxing is date driven; and dates are met by reducing scope and not sacrificing quality.
    - Scope gets reduced using formal requirements prioritization.

# Iterative Approaches To Development Architectural Planning & Management



**\*  
Identify and manage the major interfaces!!!**

# Iterative Approaches To Development

- **Architecture: early and often**
  - **Must have** *good project managers and good architects.*
    - **Project managers cannot build realistic plans.**
    - **Architects cannot manage the interfaces.**
      - *No one can manage the integration of releases.*
  - *Deliverables reviewed, modified, and approved **over and over** again—with every new iteration.*

# Iterative Development Challenge: Shortchanging Inspections

- Team productivity = velocity = actual total ***working software*** completed and certified
- Iterations' programmer productivity:
  - Develop function: create unit tests, create build tests, code, execute unit tests, execute build tests
  - Each step is scheduled to take 1 day—or 5 days total
  - ***With no bugs***, one iterative programmer can finish 4 functions per month
  - With 5 programmers per team: then **20 functions per month**

# Iterative Development Challenge : Shortchanging Inspections

- **If unit testing and/or build testing is a glitch:**
  - Average one bug per unit test: .25 days to fix plus 1 bug per build test: .25 days to fix, times 2 function sets = 1 lost day per month/programmer
  - At 5 programmers per team = 1 lost week per team per month (1 lost function per month)
  - Unsatisfactory end-user demonstration (costs --??--)
  - Inventory of function deliverables goes down; project slips behind
- **If integration testing is a glitch:**
  - At a minimum back to the coders (maybe designers)
  - Average 6 builds per I/T testing group (2 coding units per build)
  - Average one bug per build integration interface, .5 days to study/fix times 6 builds = 3 lost days per month per programmer
  - At 5 programmers per team = 30 integration bugs per month = 3 lost weeks per team per month (3 lost functions per month)
  - Unsatisfactory end-user demonstration (costs --??--)
  - Inventory of function deliverables goes down; project slips behind
- **And more of the same for the rest of testing**
  - System Testing:
    - fix may require more design, code and repeat of S/T, potentially 3 days to fix
    - 3 bugs/programmer = 9 days lost/programmer, 45 days per team, 8 lost functions per month
  - Acceptance Testing:
    - fix may require more analysis, design, code and repeat of S/T and A/T, potentially 4 days to fix
    - 3 bugs/programmer = 12 days lost/programmer, 60 days per team, 12 lost functions per month

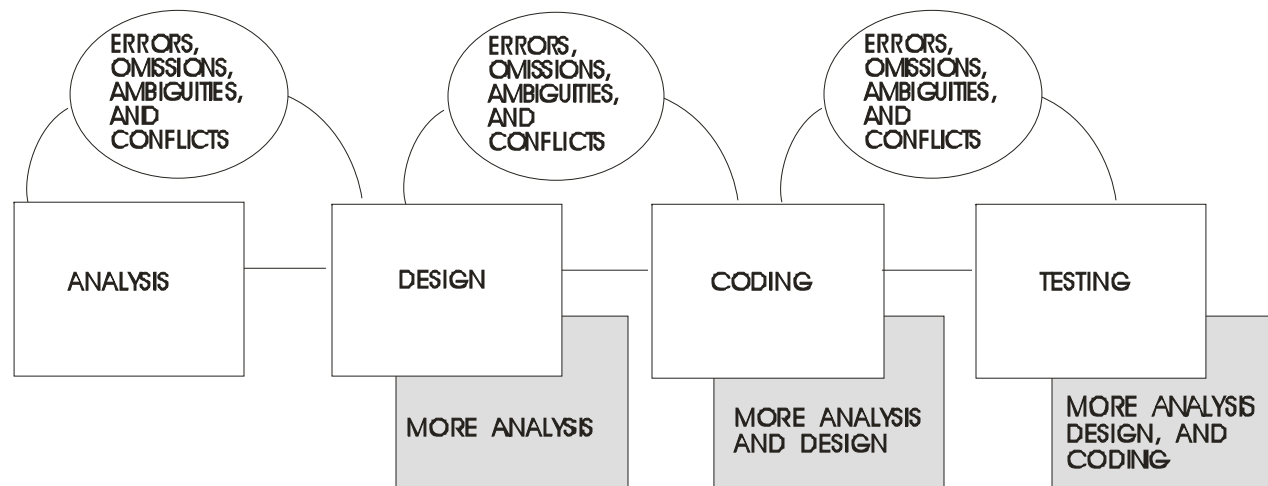
# Inspections/Peer Reviews: An Iterative Development Opportunity

- Reduce down-line bugs: increase productivity, shorten schedules and costs
    - Weigers: Adding peer reviews to unit coding cuts defects per unit found at S/T by 4/5 down to 1/5 as many defects per unit
      - Cost of peer reviews  $\approx$  10% of coding capacity
        - 1 day coding = (9 hours X 60 minutes) = 540 minutes
          - 540 minutes \* .10 = 54 minutes  $\approx$  1 hour cost per coding day per programmer per week X 4 weeks = 4hours per month X 5 team members = 20 hours per month per team or 2.5 days per month (1/2 of 1 lost function per month) X 5 programmers = 2.5 lost functions per month
      - Cost of 1/5 as many defects per unit per programmer at S/T
        - 3 bugs/programmer @ S/T X 1/5 = .6 bugs/programmer, 3 days per team, 3/5 of a function lost per month
      - Total cost of peer reviews = 3.1 lost functions per month
- D.J. Anderson

# Inspections—AND Continuous Improvement

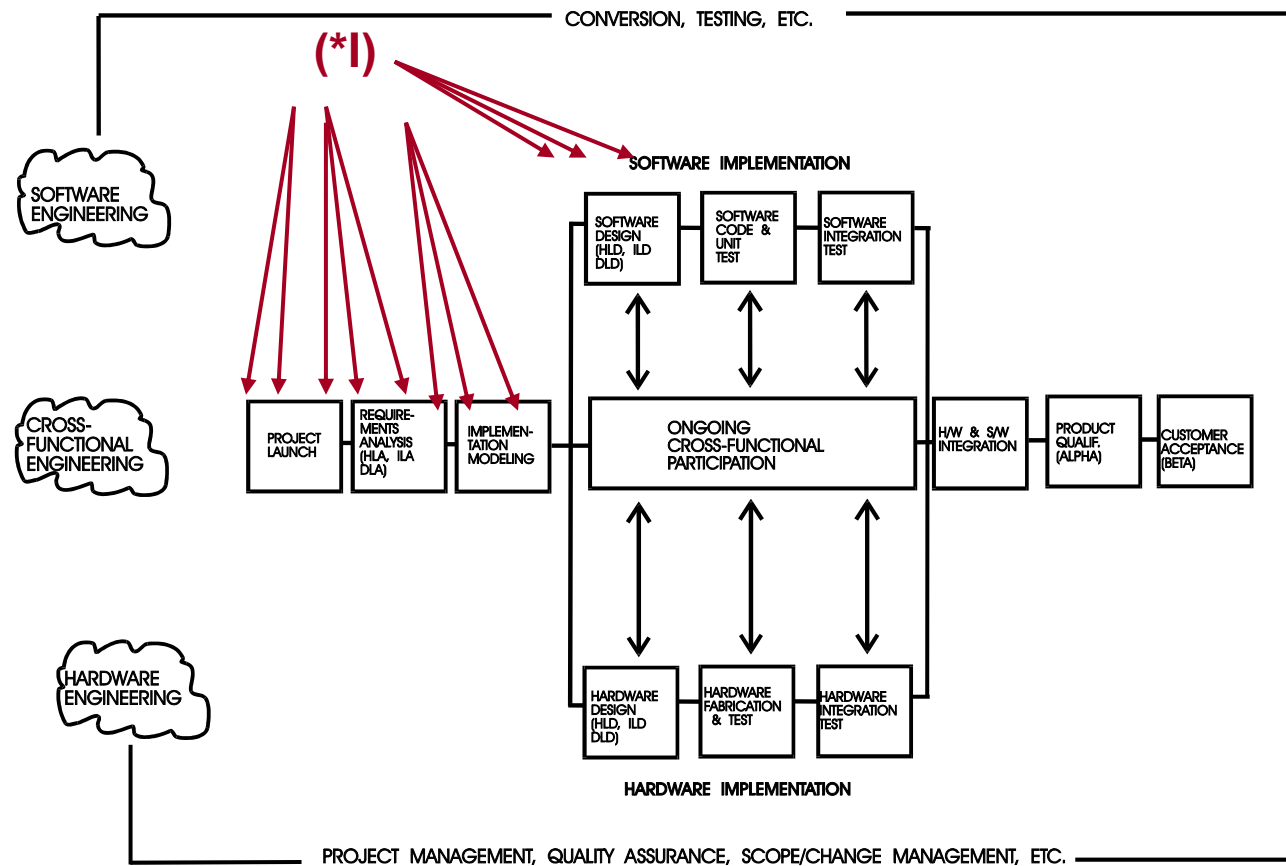
- **Product**
  - Inspect the deliverable
  - Evaluate exit criteria
  - **Test the specification (\*TS)**
  - Judge the outcome
    - Unconditional pass
    - Conditional pass
    - Fail
  - Capture the findings
- **Do Continuous Process Improvement**
  - Review the process used to produce the deliverable
  - Review the process used to produce the test data
  - Review the review process
  - Capture the findings
- **Summarize and communicate all findings**

# Inspections: Overcoming The Mistakes We Make



- Projects can be “on-time” until testing—and then get very, very late during testing
  - This poor quality development approach is one of the most significant causes of schedule overruns
- Cheaper, Faster, Better:
  - Do analysis during analysis, design during design... ..
  - **Add 30% For Inspections!**

# Inspections (\*I)



# Chronic Waste

- **What is the real cost of testing?**
  - List the events.
  - List the conditions.
  - List the scenarios.
  - Identify the inputs and outputs.
  - Describe the test.
  - Describe the output.
  - Create the test database.
  - Define the maintenance for the database.
  - **Execute the test one time.**

# Test-driven Development (TDD): Front-loading The Testing Effort

- **Front-load analysis and design**
- **Do analysis and design correctly**
- **Inspect the specifications with the user**
- **Test the specifications (\*TS)**
- **Do requirements traceability**
- **Implement change management**
- **Do regression testing**

# Testing In Iterative Development

- Testing
  - The goal of each increment is transform a requirement (that has been prioritized from the backlog) into a properly working functional software product.
  - Properly working equates to customer satisfaction (number 1)
    - Verbal requirements
    - Unspoken, take-for-granted requirements
    - Sizzle
  - It also equates to “clean” code\*
    - Free from all bugs before the increment can be certified
    - Adhere to coding standards
    - Been refactored
    - Contains no clever coding
    - Is easy to read and understand
    - Is easy to sustain and maintain

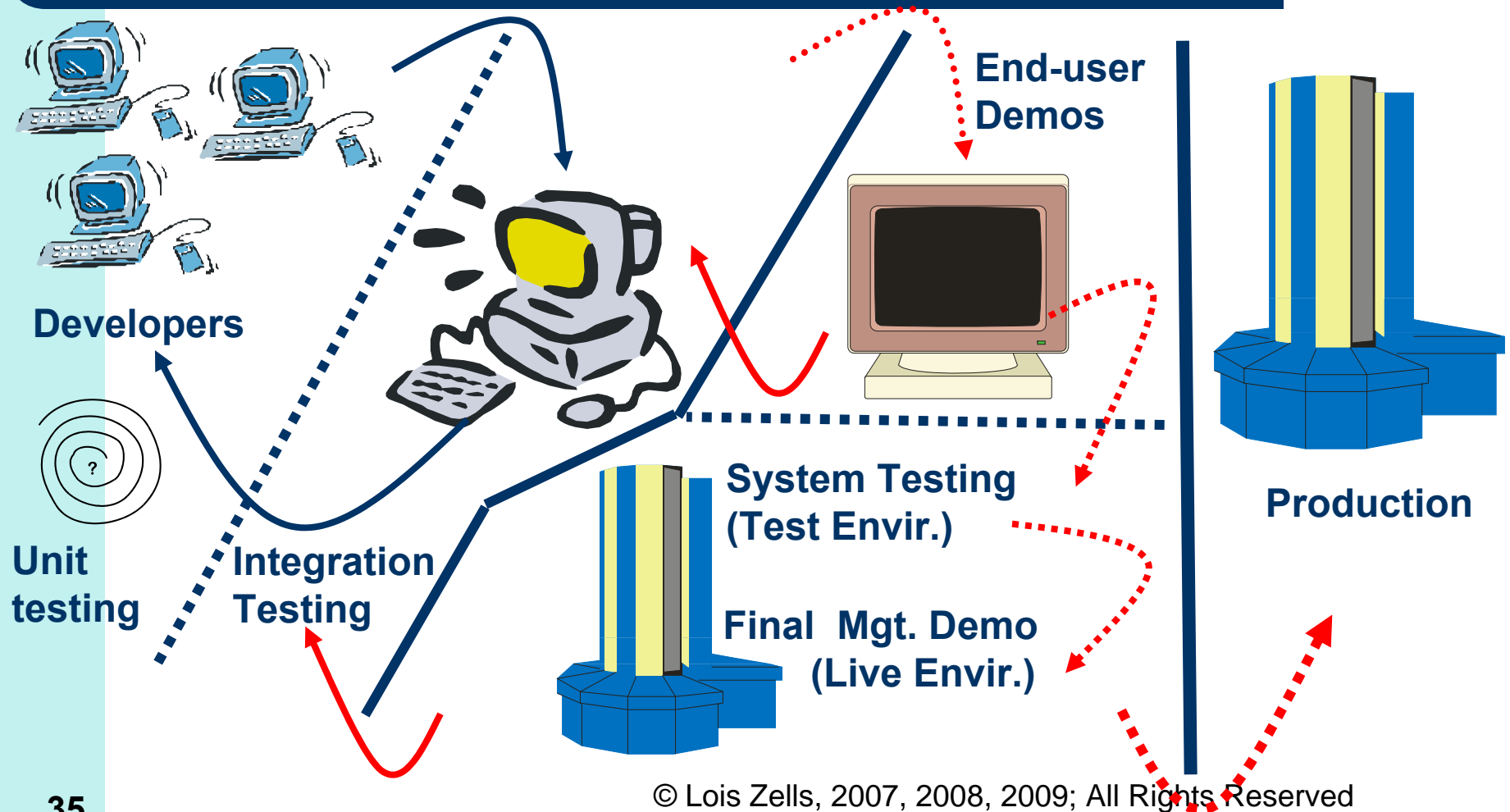
\* “Agile P/M With Scrum” Ken Schwaber

© Lois Zells, 2007, 2008, 2009; All Rights Reserved

# Testing in Iterative Development

- **Test-driven Requirements (TD<sub>R</sub>)**
  - Test stories done **before** requirements stories are actually elicited
  - Done by user project participant(s)—with help from Business QA and/or Process QA
  - User black box (functional) test cases—often now called acceptance tests
    - May be initiated with low-tech post-it workflow diagrams

# Iterative Testing: Separation of Powers



# Testing In Iterative Development

- **Estimating Test Effort**

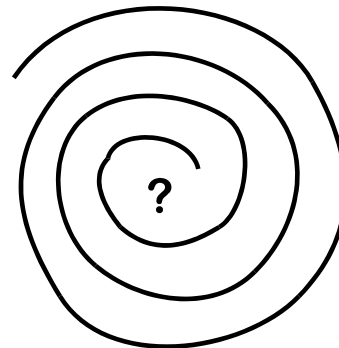
- **Some schools have one task called “Code” that includes estimated time to**

- **Code**
- **Test**
  - **Development**

- **Execution**
- **Evaluation**
- **Reporting**
- **Correction**

- **Repeat until “clean”**

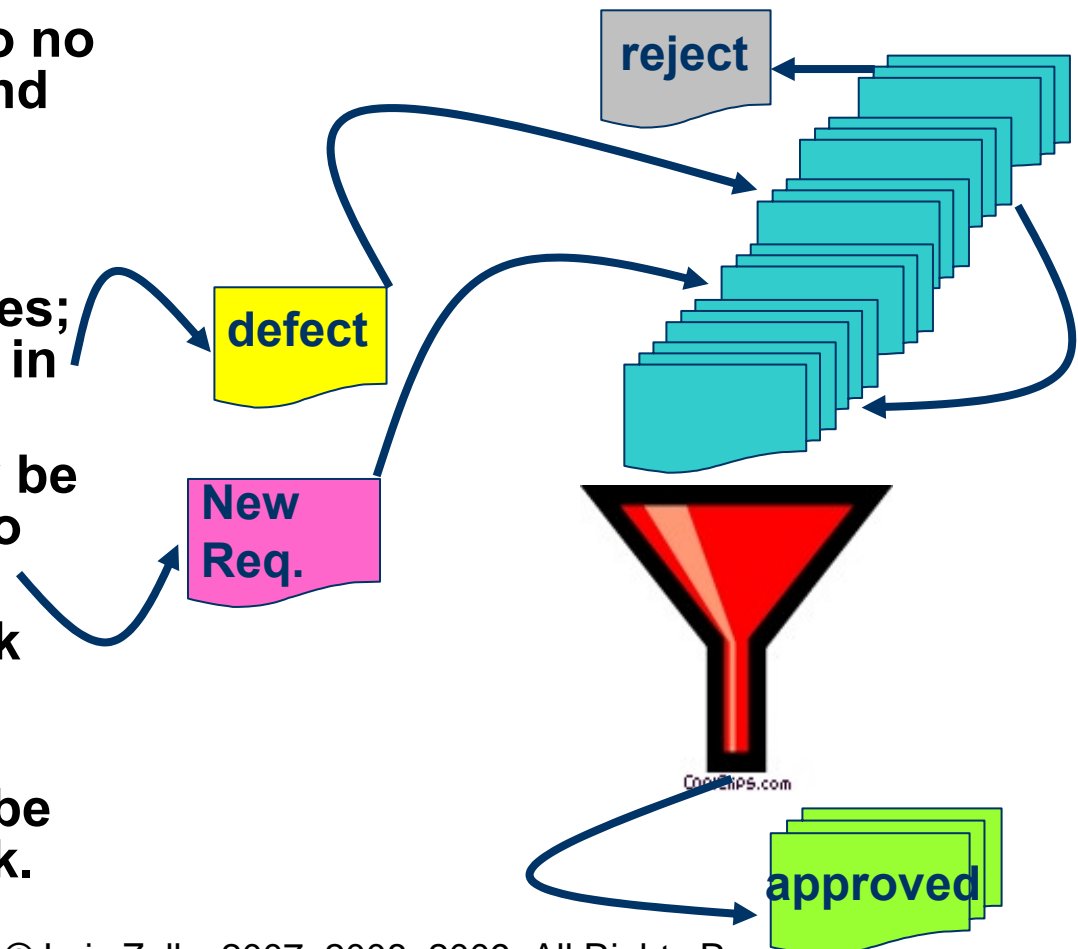
- **Some schools have separate tasks for each**



Trivial  
Simple  
Average  
Hard  
Mod. Hard  
Complex  
V. Complex

# Big Bad Bugs Become New Stories

- Errors that have little to no impact on increment end dates can be fixed as discovered.
- All other defects are written up as new stories; and added to the stack in its prioritized position.
- New requirements may be prioritized and added to the stack at any time.
- Other items in the stack can always be re-prioritized at any time.
- Some items may even be removed from the stack.



# Iterative Approaches To Development

- **Full agreement on the completion criteria**
- **Firm plan in place for how and when the groups will finish**
- **Rigorous procedures for synchronization**
  - **There must be a strong configuration management:**
    - **Keeping track of the change**
    - **keeping the specifications current to reflect the change, and**
    - **keeping the name of (and notifying) every person, every requirement, and every program that is affected by the change.**

# Iterative Approaches To Development

- **NEVER:**
  - Excuse for Hacking
  - Excuse for no requirements or design
  - Excuse for no methodology

# Iterative Approaches To Development

- **Development Strategies**
  - **Full Iterative Approach**
    - When total project scope is small and relatively uncomplicated; **and**
    - When all parties can accept fuzzy end-dates and dollars for completed **project**
- **But: up front preliminary work is always required (Launch and Initial Architecture)**

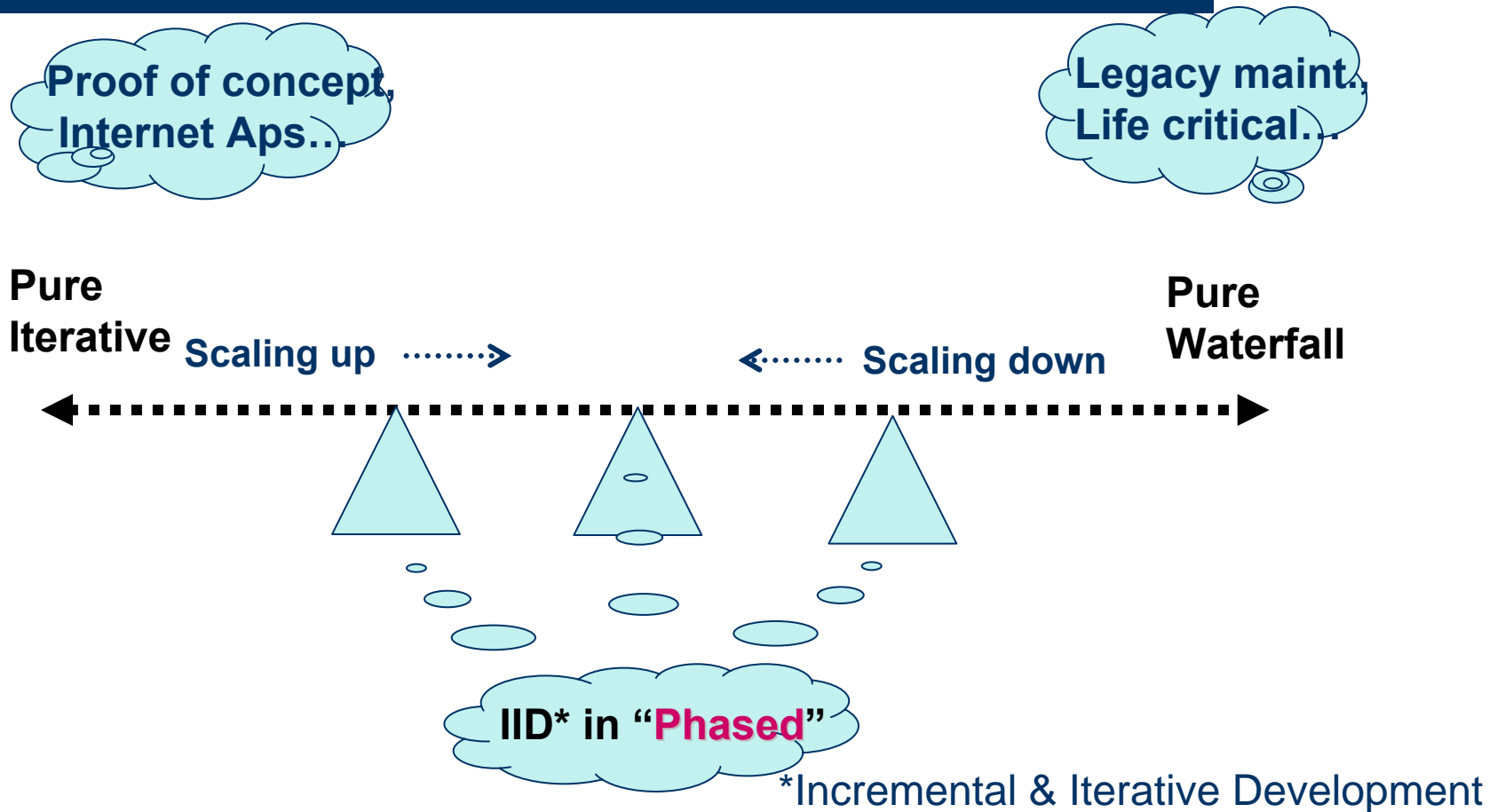
# Iterative Approaches To Development

- **Development Strategies**
  - **Modified Iterative Approach**
    - When scope is very large and/or relatively complicated; and/or
    - When completed project end-date and dollars are critical
      - Use iterations as a specification approach; but **build the “real” system using phase-driven development** —when
        - Requirements are not well-established or stable
        - Requirements are ambiguous
        - There are contradictions in the way different departments do the work
      - **Evolutionary Delivery of Individual Working Components--when**
        - Requirements are well-established and stable
        - Requirements are unambiguous
        - There are no contradictions in the way people work
  - **But: up front preliminary work is always required (Launch and Initial Architecture)**

# Zells' Iterative Development Premises

- **There is no “one-size fits all”**
  - **Iterative & Incremental Development**
    - **Iterative Development approach**
  - **Conventional “phase”-driven is not the same as waterfall**
    - **“Phase”-driven approach**
    - **Waterfall approach**
- **It is possible to mix the best of both worlds**
  - **And thereby work within the constraints of the organization; but**
  - **Harmonizing the “right” mix of the two is a multi-dimensional process**

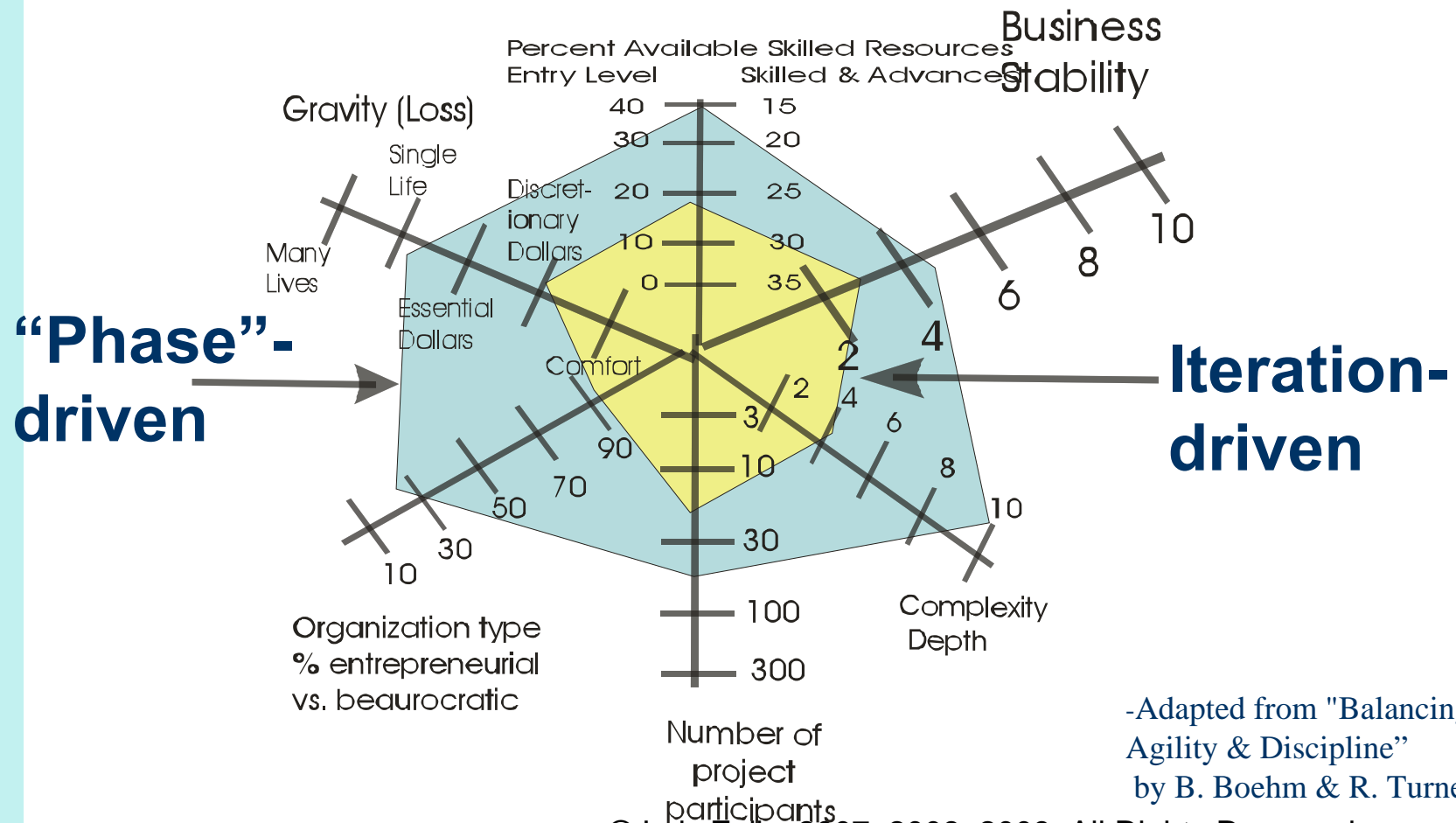
# Premise: A Continuum



# Making The Choice

- **Primarily user-interface systems**
  - Often small and straight-forward
  - Limited number of architectural interfaces
  - Not very deep processing
  - Users often widely-dispersed geographically
  - Good candidates for iteration-driven
- **Primarily internal transaction-driven systems**
  - Usually large and complex
  - Significant numbers of complex major architectural interfaces
  - Very deep processing
  - Users often widely-dispersed geographically
  - Good candidates for conventional “phase”-driven

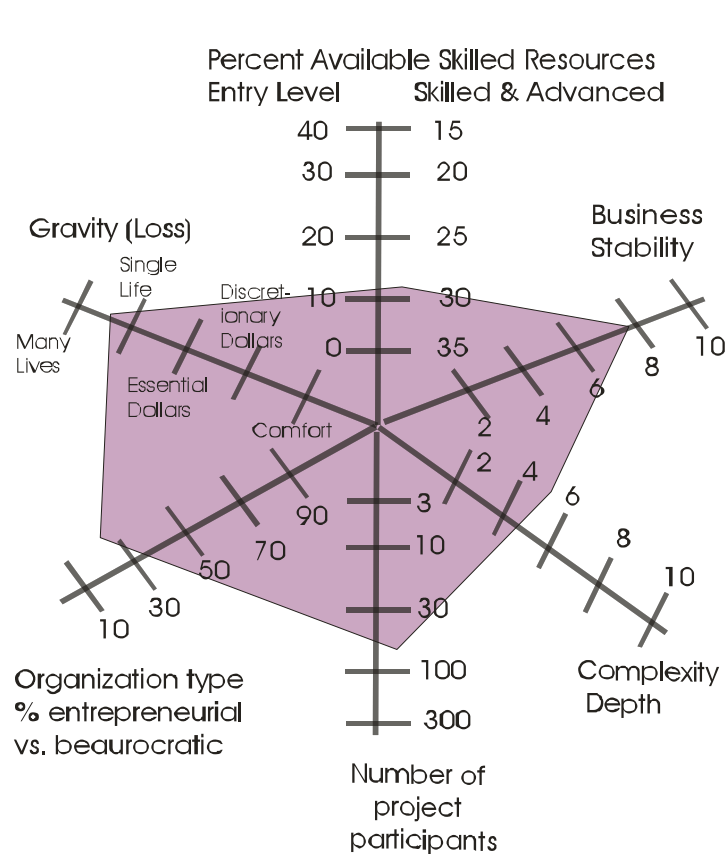
# Making The Choice: Easy Choices



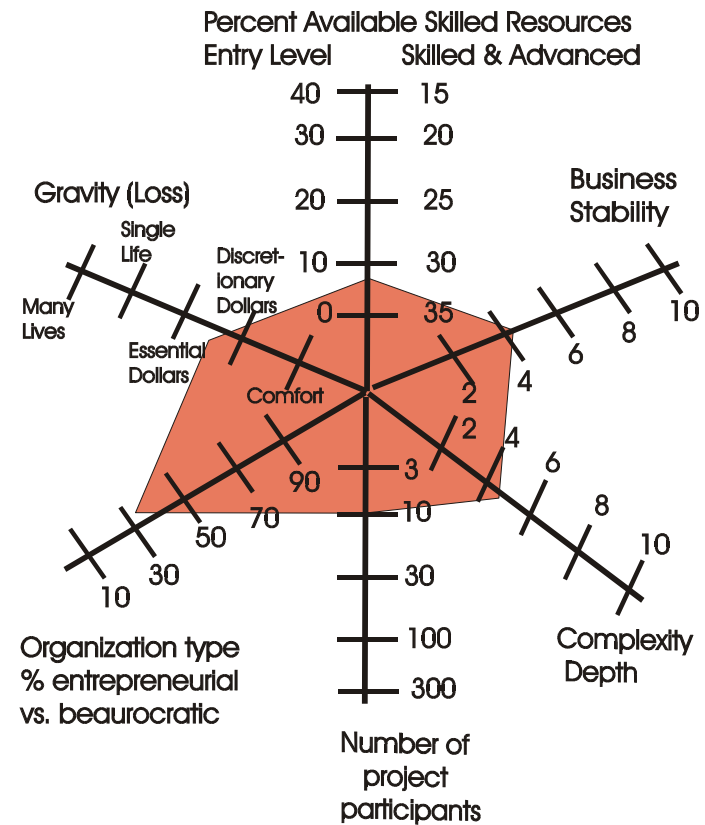
-Adapted from "Balancing Agility & Discipline" by B. Boehm & R. Turner

© Lois Zells, 2007, 2008, 2009; All Rights Reserved

# Making The Choice: Harder Choices



**“Phase” heavy**



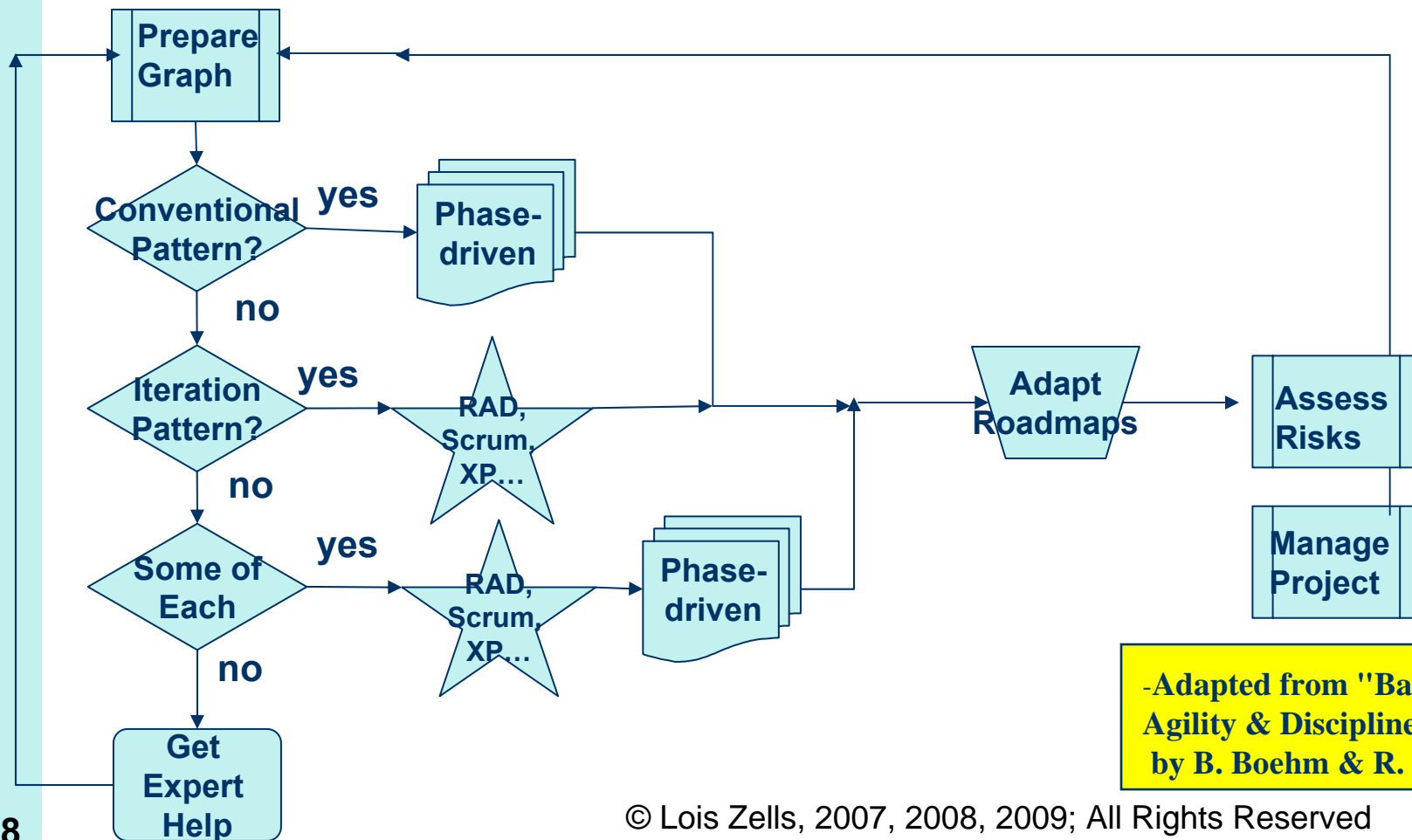
**Iteration heavy**

# Mixing It Up

- **Gather the facts.**
  - **Evaluate your own environment**
    - Do proof-of-concept testing if necessary
    - Assess Stakeholder influence
    - Develop the high-level architecture
    - Depth-sound for estimates
  - **Review industry information**
  - **Assess the pros and cons of each of factors**
    - Iteration-driven
    - Phase-driven

Trivial  
Simple  
Average  
Hard  
Mod. Hard  
Complex  
V. Complex

# Mixing It Up



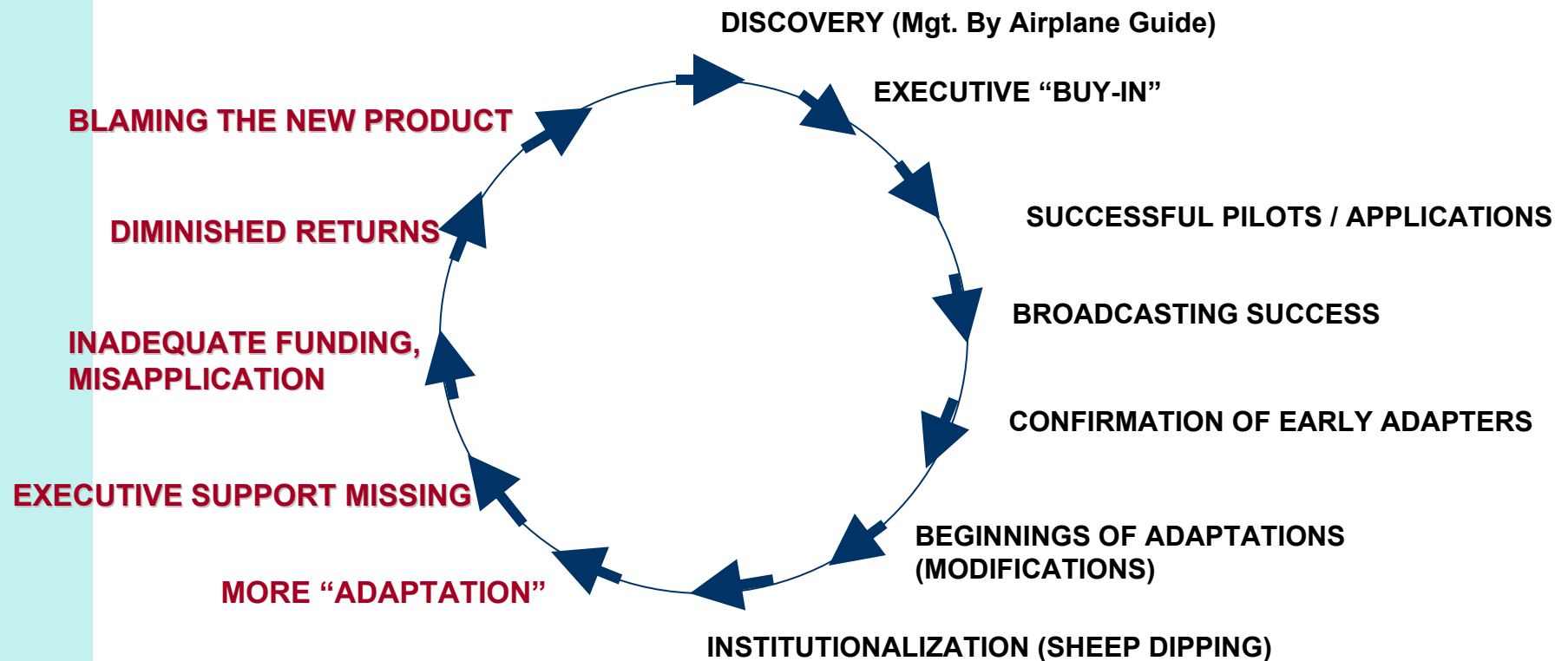
-Adapted from "Balancing Agility & Discipline" by B. Boehm & R. Turner

# Mixing It Up

- **Architecture always important**
  - Design for simplicity (to the extent possible)
  - Package iteration components separately
  - Always, always, always manage the architectural interfaces
    - The business process interfaces; and
    - The technical interfaces
- **“CRACK” \* users always necessary**
  - Focus on separating co-mingled business/technical requirements
  - Eliminate errors before end-user demos: inspect, inspect, inspect
  - Eliminate defects before delivery: test, test, test
- **Project management still important**
  - Manage “breadth” scope changes and “depth” knowledge increases
  - Stay focused on completion criteria
  - Use Adaptive planning

\*collaborative, representative, authorized, committed, knowledgeable  
Boehm & Turner "Balancing  
Agility & Discipline"

# Implementing New Technologies



Adapted from "Life Cycle of a Silver Bullet," Sarah Sheard, Software Productivity Consortium

© Lois Zells, 2007, 2008, 2009; All Rights Reserved

# Some Iterative Approaches

- ***“Agile & Iterative Development” by Craig Larman***
- **“Adaptive Software Development” by Jim Highsmith**
- **“SCRUM” by Ken Schwaber, Mike Beedle, Jeff Sutherland**
- **“Extreme Programming (XP)” by Kent Beck, Ward Cunningham, Ron Jeffries**
- **“Dynamic Solutions Delivery Model” (aka “Dynamic Systems Development Method”)**
  - DSDM
  - International DSDM Consortium [www.dsdm.org](http://www.dsdm.org)
- **“Crystal” by Allistair Cockburn**
- **“Feature-driven development” by Jeff DeLuca & Peter Coad**
- **“Lean Software Development” by Mary and Tom Poppendieck**
- **“Pragmatic Programming” by Andy Hunt & Dave Thomas**
- ***“Balancing Agility & Discipline” by B. Boehm & R. Turner***
- ***“Agile Management for Software Engineering” by David J. Anderson***

# Today We Talked About

- **Best Practices In Iterative Development**
- **Major Contributors To Iterative Project Failures—And How to Avoid Them**
- **Is Pure Iterative The Way To Go?**



# THANK YOU

More Questions?

[lzells@aol.com](mailto:lzells@aol.com)

(310) 316-2810

