COMP 5700/6700/6706
Software Process

Spring 2016

David Umphress

Process Redux

- Lesson: Process Redux
- Strategic Outcomes:
  - To gain exposure to common processes
- Tactical Outcomes:
  - To understand the complexities of process selection
  - To know the fundamental principles of light-weight processes in use today
- Instant take-aways:
  - Terminology: Scrum, LSD, XP, Crystal, TSP, FDD, DSDM, RUP

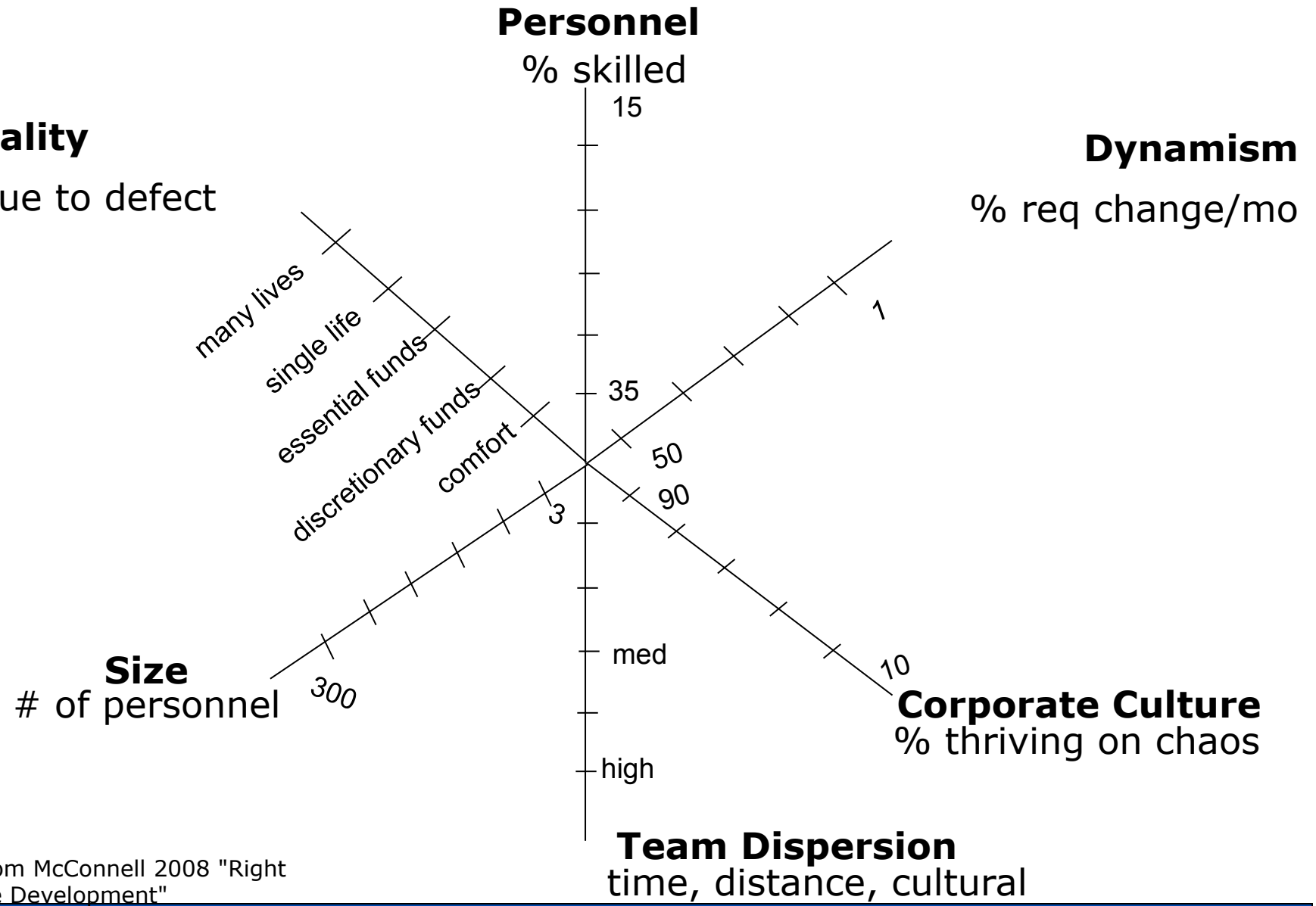
- Bookshelf items
  - Beck, K. 1999. *Extreme Programming Explained : Embrace Change.* Addison Wesley
  - Crystal. http://alistair.cockburn.us/Crystal+light+methods
  - Humphrey, W. 1995. A Discipline for Software Engineering. Addison Wesley
  - Humphrey, W. 2000. Introduction to the Team Software Process. Addison Wesley
  - Scrum. http://www.controlchaos.com
  - Unified Process. http://www.rational.com
  - XP. http://www.extremeprogramming.org
  - Wikipedia (of course) en.wikipedia.org

Syllabus

- Software engineering raison d'être
- Process foundations
- Common process elements
- Construction
- Reviews
- Refactoring
- Analysis
- Architecture
- Estimation
- Scheduling
- Integration
- Repatterning
- Measurements
- Process redux
- Process descriptions*
- Infrastructure*
- Retrospective

- **Process selection**
- **Common processes**
  - **Scrum**
  - **ASD**
  - **LSD**
  - **XP**
  - **Cleanroom**
  - **FDD**
  - **TSP**
  - **DSDM**
  - **RUP**

Process Selection Factors

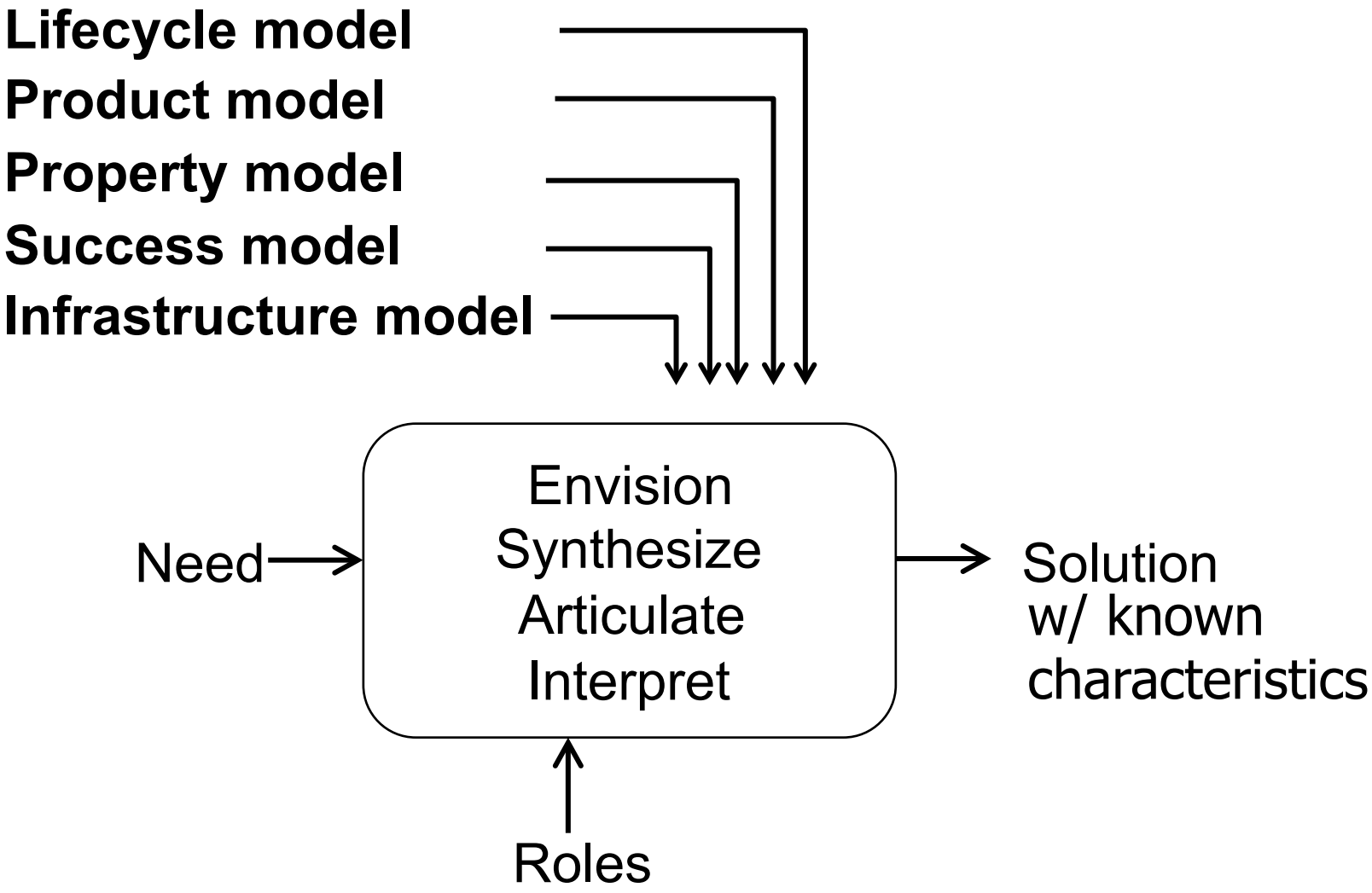


Adapted from McConnell 2008 "Right Sizing Agile Development"

Process Selection Factors

- Looking for the One-Size-Fits-All process?
  - Typical approaches
    - Process abstractions
      - Actions abstracted at the organizational level
      - Goes from descriptive to prescriptive in relation to organizational depth
        - » Process described in terms of general activities at top level
        - » Specific practices prescribed at bottom level
    - Process asset library
      - Multiple practices per process activity
        - » each practice has been characterized by advantages, limitations, usage, lessons learned, etc.
      - Projects assemble a process from process parts
  - Process selection constraints: Lessons from Industry
    - ... apply over small product size ranges, similar domains
    - ... does not apply to unprecedented systems
    - ... does not work for poorly managed projects
    - ... is unlikely to work where the engineering work is undisciplined

The Big Picture



Process Darwinism

Plan-driven

~~Heavy-weight processes~~

Agile

~~Light-weight processes~~

IEEE 1074, CMM, ISO 9001

XP, Scrum, etc etc etc

requirements containment vs adaptation

document-oriented vs artifact-oriented

design-oriented vs construction-oriented

predictive vs adaptive

process-oriented vs people-oriented

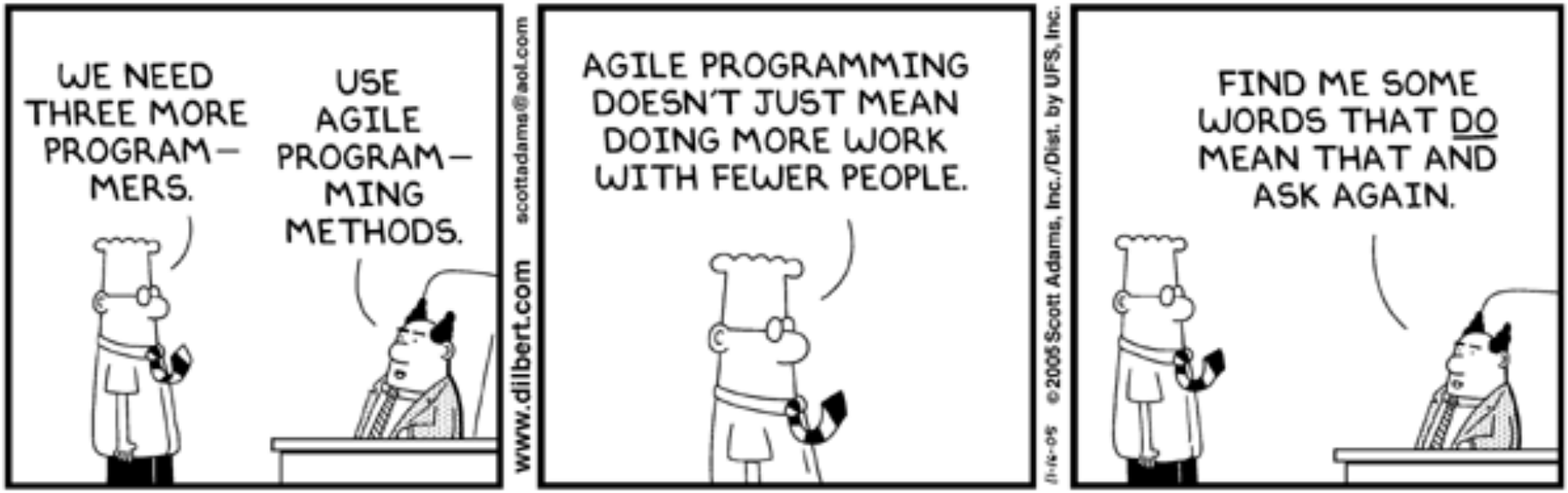
PCSE?

Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



Agile Process Success Factors

- The culture of the organization must be supportive of negotiation
- People must be trusted
- People must have generally higher levels of competency
- Organizations must live with the decisions developers make
- Organizations need to have an environment that facilitates rapid communication between team members

Agile Process Commonalities

- Small teams
- Adaptive to changing requirements
- Iterative development
- Documentation light
- Technical excellence

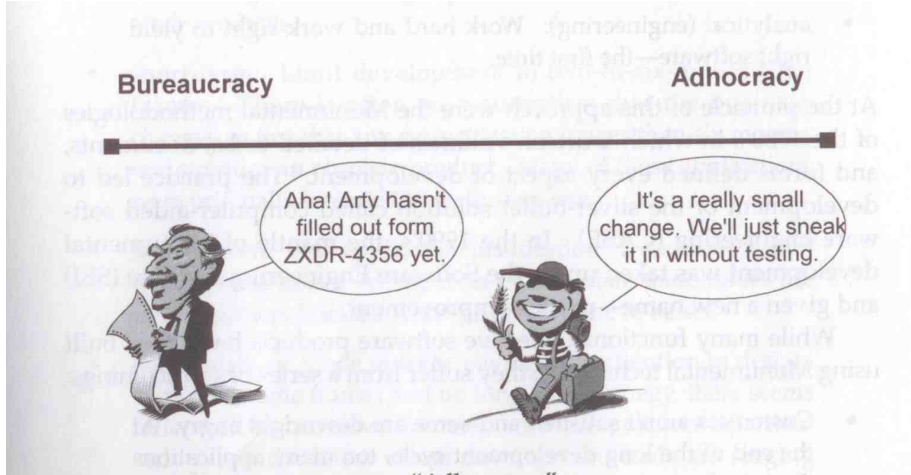


Figure 1.1: Bureaucracy versus "Adhocracy."

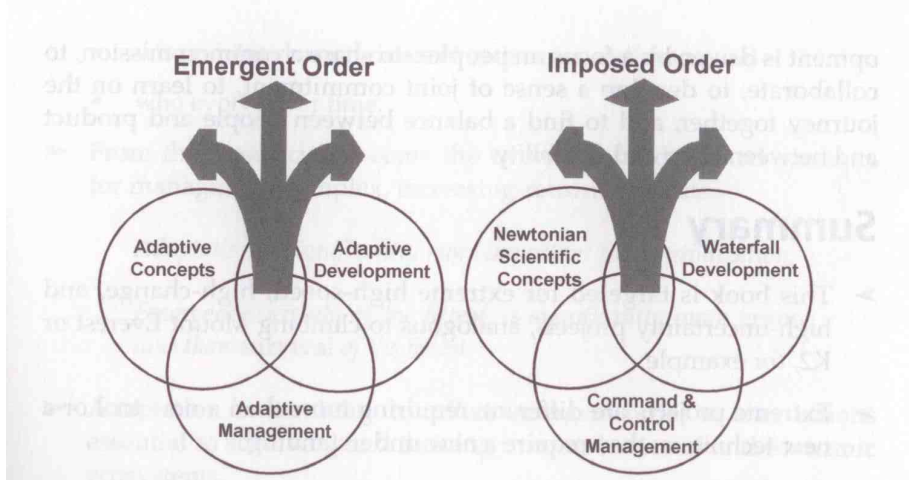


Figure 1.4: The Core Difference Between Adaptive and Traditional Models.

From Highsmith, J. 1998. *Adaptive Software Development*. Dorset House.

Process Alternatives

Weight*

- Scrum
- Adaptive Software Development (ASD)
- Lean Development (LD)
- Crystal
- Extreme Programming (XP)
- Feature-Driven Development (FDD)
- Team Software Process (TSP)
- Cleanroom
- Dynamic Systems Development Method (DSDM)
- Rational Unified Process (RUP)

* in my opinion

Scrum

- General
  - A managerial framework, not a technical one
  - Frequently used to introduce agility
- Principle
  - Software is not a defined process, but an empirical process the varies with the circumstances
- Tenets
  - Emergent behavior
  - Outcomes emerge with high dependence on relationship and context

Scrum

- Core techniques
  - Roles
    - Product owner
      - responsible for project business value
    - ScrumMaster
      - ensures that team is functional and productive
    - Team
      - self-organizes to get the work done
  - Ceremonies
    - Sprint
      - time-boxed work period (heuristic: max 30 days)
    - Sprint planning
      - team-product owner meeting to select sprint work
    - Daily scrum
      - team meeting on progress and obstacles
    - Sprint reviews
      - team end-of-sprint product demonstration
    - Sprint retrospectives
      - team examination of ways to improve the product and process.

Scrum

- Core techniques (con't)
  - Three artifacts
    - Product backlog
      - ordered list of product features
    - Sprint backlog
      - product backlog backlog to complete in a sprint, broken into tasks
    - Product Increment
      - shippable version of the product.
      - required of every sprint

Adaptive System Development

- General
  - Few specifics, mostly an outline of opportunities
  - More of philosophical molding of existing practices
- Principle
  - believe in continuous adaptation
- Tenets
  - Outcomes are naturally unpredictable
  - Three components:
    - Adaptive conceptual model … ecosystem
    - Adaptive development model … RAD
    - Adaptive management model … leadership/collaboration

Adaptive System Development

- Core techniques
  - Mission-driven
    - objective
    - profile
    - specification outline
  - Component-based
  - Iteration
    - speculate
    - collaborate
    - learn
  - Time-boxing
  - Risk-driven
  - Cross-functional teams
  - Distributed governance
  - Embracing change
  - Customer focus groups
  - Software inspections
  - Project postmortems
  - Process emergence

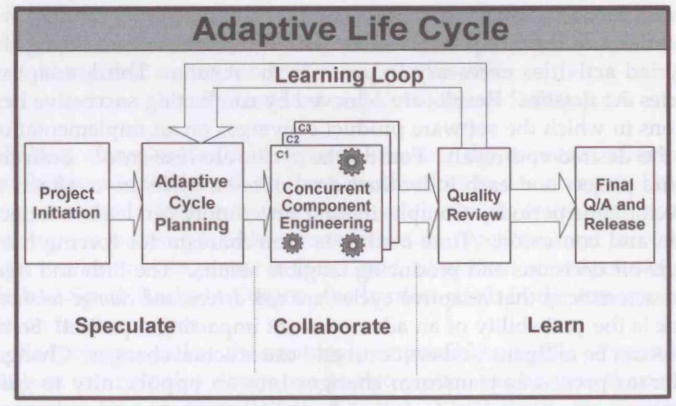


Figure 4.1: The Detailed Adaptive Life Cycle.

Lean Development

- General
  - Addresses entire product chain
  - Commonly adopted by organizations seeking an evolutionary approach to process insertion
- Principle
  - Perform only essential tasks
  - Focus on the people who add value
  - Flow value from demand
  - Optimize across the organization
- Tenets
  - eliminate waste
  - seek quick feedback
  - delay non-reversible decisions as long as possible
  - deliver as fast as possible
  - empower the team
  - build integrity into product
  - see the big picture

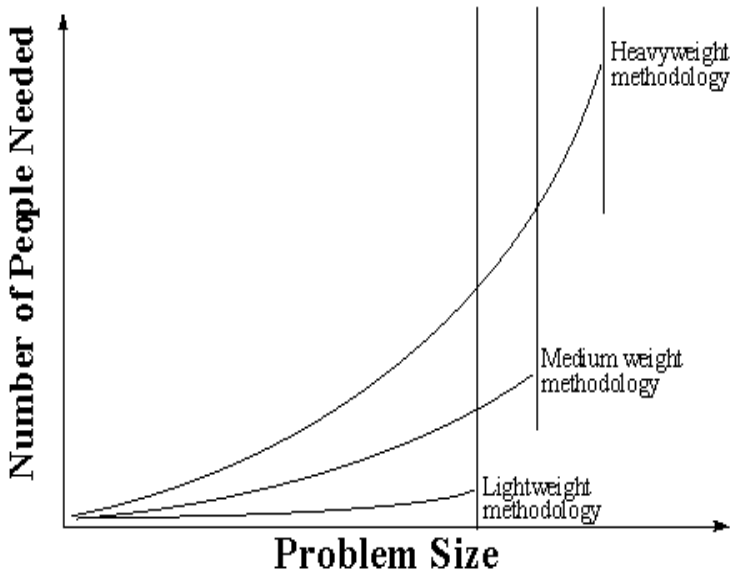
Lean Development

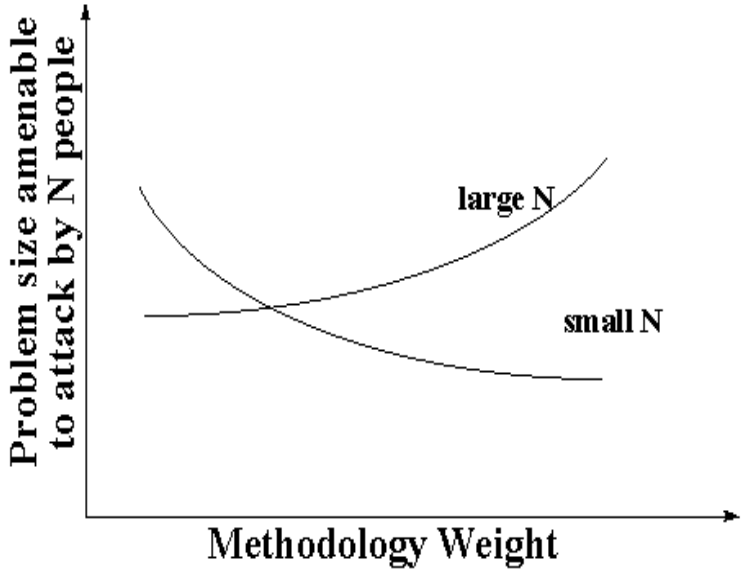
- Core techniques
  - Value stream mapping
  - Feedback loops
  - Solution convergence
  - Negotiable scope
  - Iterations
  - Synch and stabilize
  - Set-based development
  - Breadth-first decisions
  - Simplicity
  - Pull schedules
  - Prioritization of features
  - Communication
  - Developer empowerment
  - Communities of expertise
  - Model-driven design
  - Test-driven development
  - Refactoring
  - Appropriate measurements

Crystal

- General
  - Not a single method, but family of increasingly complex collection of practices
  - Commonly adopted by organizations seeking an evolutionary approach to process insertion
- Principle
  - Process depends on “ecosystem”
- Tenets
  - interactive, face-to-face communication
  - more people = heavier process
  - small increase in process greatly affects cost
  - process weight depends on project criticality

Crystal


Number of People Needed
Heavyweight methodology
Medium weight methodology
Lightweight methodology
Problem Size


Problem size amenable to attack by N people
large N
small N
Methodology Weight

Crystal

Criticality	1-4	6-20		...			500+
Life	L3	L10	L30	L80	L150	L300	L600
Essential money	E3	E10	E30	E80	E150	E300	E600
Discretionary money	D3	D10	D30	D80	D150	D300	D600
Comfort	C3	C10	C30	C80	C150	C300	C600

Number of people involved

Crystal

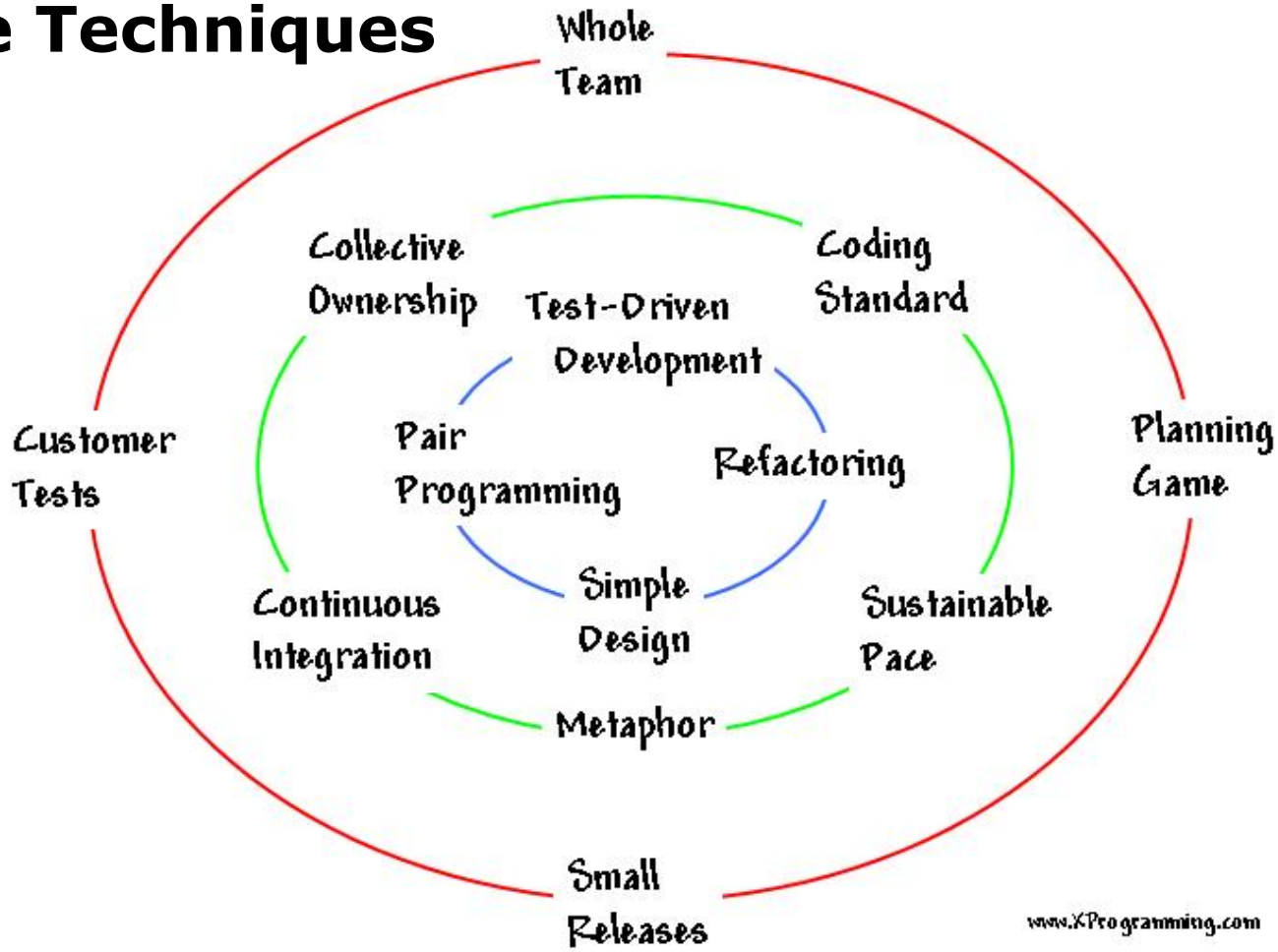
- Core techniques
  - Frequent delivery of usable code
  - Osmotic communication
  - Personal safety
  - Uninterruptable work time
  - Access to expert users
  - Automated tests, configuration management, and frequent integration
  - Regular process review, improvement

Extreme Programming

- General
  - Most widely recognized … and misapplied … agile method
  - Commonly adopted by organizations seeking a revolutionary approach to process insertion
- Principle
  - Takes common sense practices to the extreme
- Tenets
  - values:
    - communication … among customers and developers
    - simplicity … by minimizing feature creep
    - feedback … from customers and developers
    - courage … in making difficult decisions in support of other values

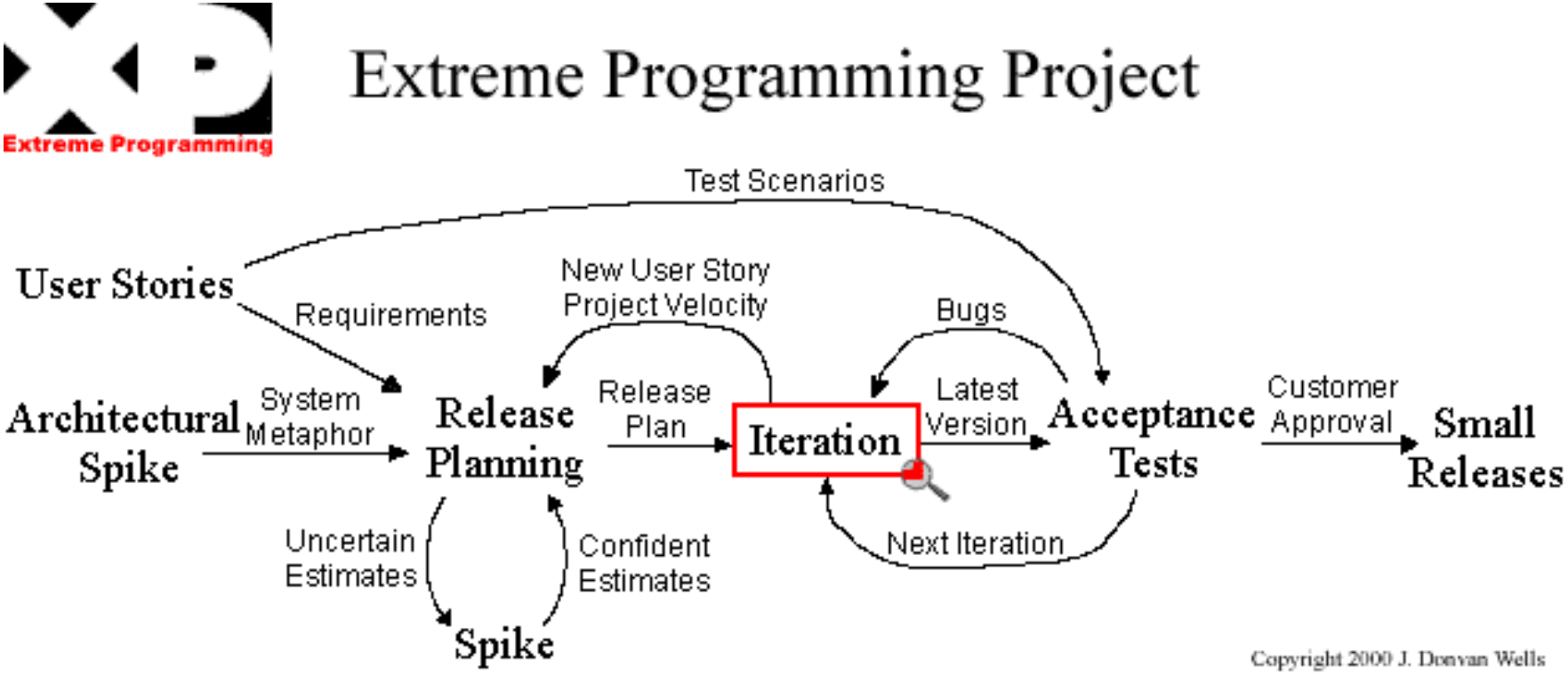
Extreme Programming

Core Techniques

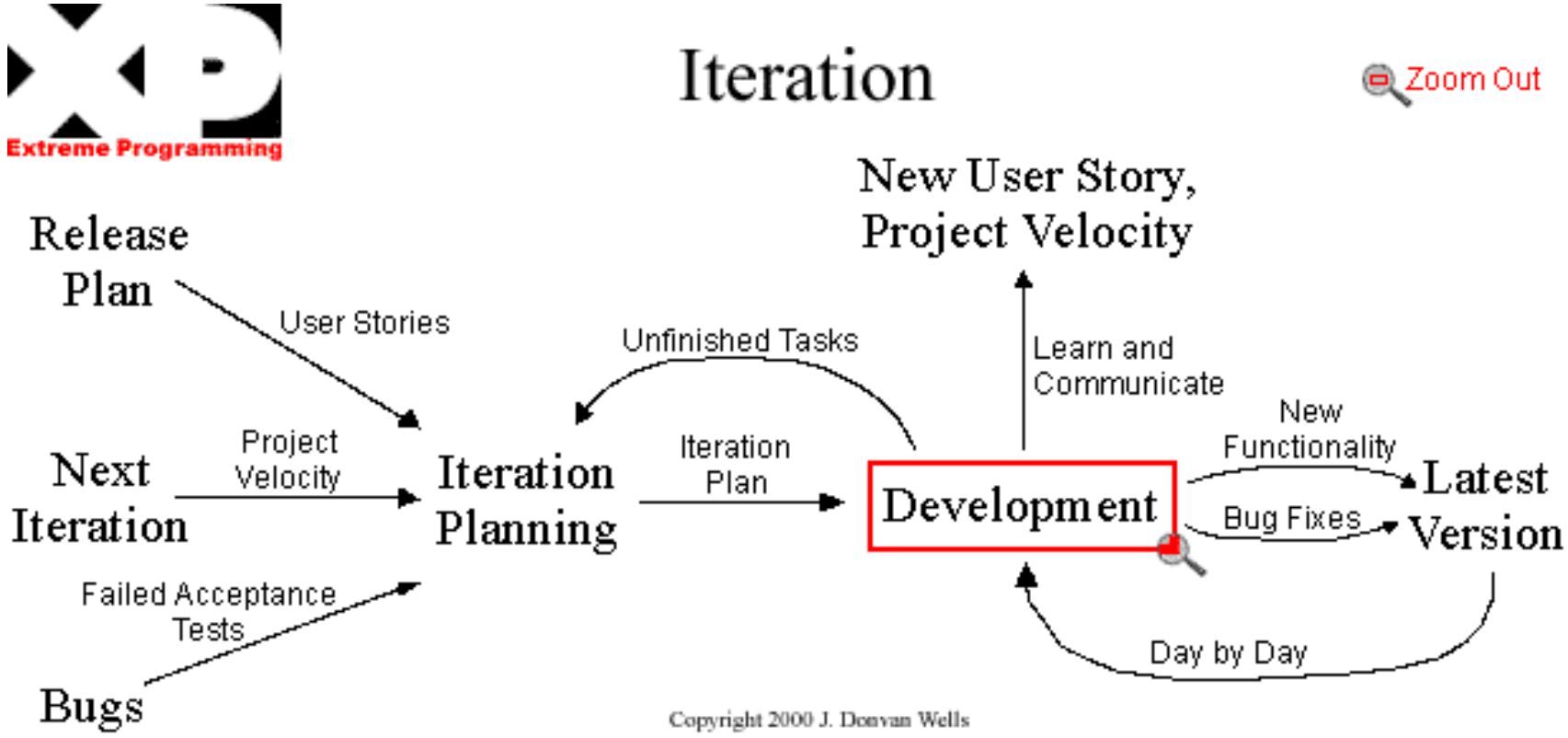


XP

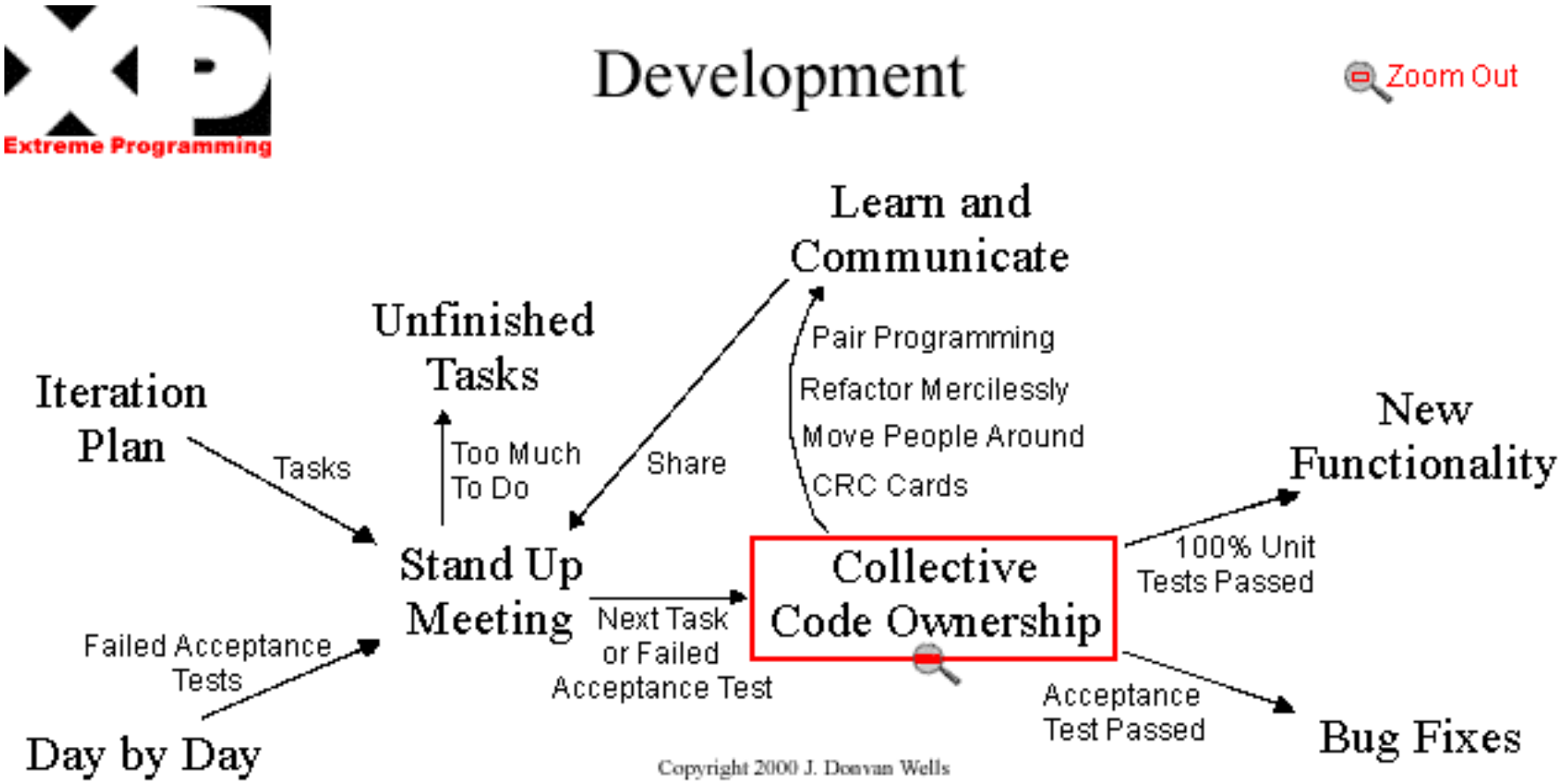
- Extreme Programming (XP)
  - values: communication, feedback, simplicity, courage



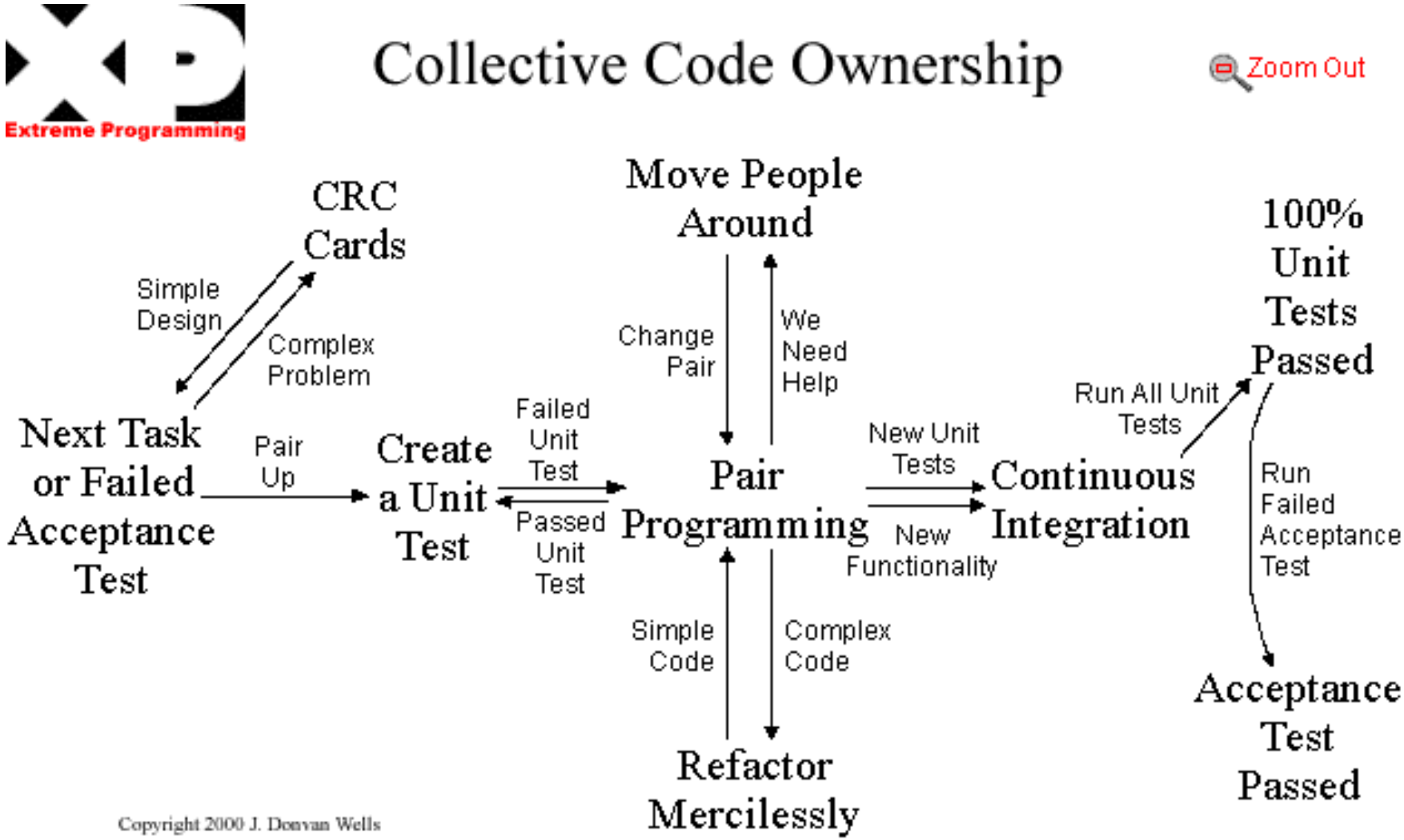
XP


Extreme Programming
Iteration
Zoom Out
New User Story,
Project Velocity
Release
Plan
User Stories
Unfinished Tasks
Learn and
Communicate
New
Functionality
Next
Iteration
Project
Velocity
Iteration
Planning
Iteration
Plan
Development
Latest
Version
Bug Fixes
Failed Acceptance
Tests
Day by Day
Bugs
Copyright 2000 J. Donvan Wells

XP


Extreme Programming
Development
Zoom Out
Learn and Communicate
Unfinished Tasks
Iteration Plan
Tasks
Too Much To Do
Share
Pair Programming
Refactor Mercilessly
Move People Around
CRC Cards
New Functionality
Stand Up Meeting
Next Task or Failed Acceptance Test
Collective Code Ownership
100% Unit Tests Passed
Failed Acceptance Tests
Acceptance Test Passed
Bug Fixes
Day by Day
Copyright 2000 J. Donvan Wells

XP


Extreme Programming
Collective Code Ownership
Zoom Out
CRC Cards
Simple Design
Complex Problem
Next Task or Failed Acceptance Test
Pair Up
Create a Unit Test
Failed Unit Test
Passed Unit Test
Move People Around
Change Pair
We Need Help
Pair Programming
Simple Code
Complex Code
Refactor Mercilessly
New Unit Tests
New Functionality
Continuous Integration
Run All Unit Tests
100% Unit Tests Passed
Run Failed Acceptance Test
Acceptance Test Passed
Copyright 2000 J. Donvan Wells

Extreme Programming

Planning

User stories are written.
Release planning creates the schedule.
Make frequent small releases.
The Project Velocity is measured.
The project is divided into iterations.
Iteration planning starts each iteration.
Move people around.
A stand-up meeting starts each day.
Fix XP when it breaks.

Coding

The customer is always available.
Code must be written to agreed standards.
Code the unit test first.
All code is pair programmed.
Only one pair integrates code at a time.
Integrate often.
Use collective code ownership.
Leave optimization till last.
No overtime.

Designing

Simplicity.
Choose a system metaphor.
Use CRC cards for design sessions.
Create spike solutions to reduce risk.
No functionality is added early.
Refactor whenever possible.

Testing

All code must have unit tests.
All code must pass all unit tests before it can be released.
When a bug is found tests are created.
Acceptance tests are run often and the score is published.

Feature-Driven Development

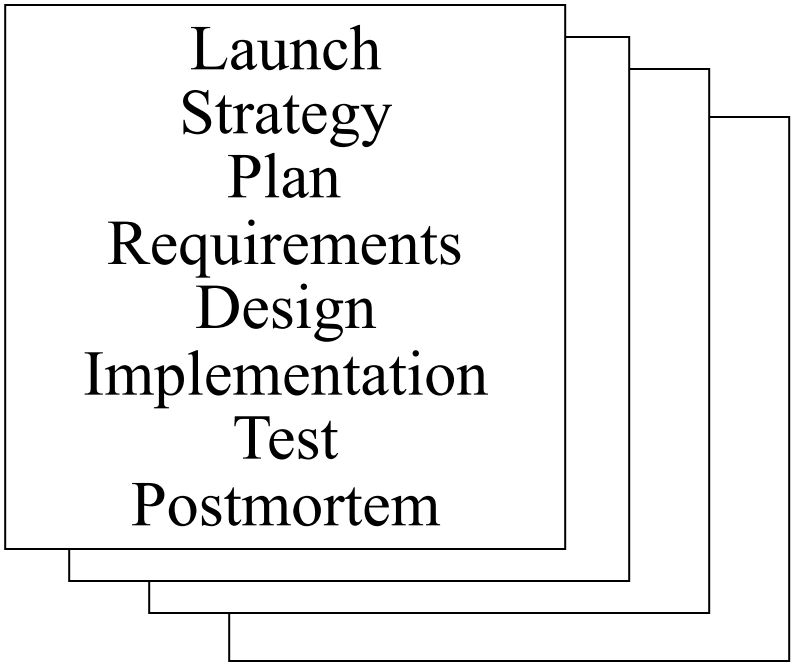
- General
  - Focus on good people with domain knowledge
- Principle
  - Put the process in the background to support, rather than drive, development
- Tenets
  - Develop Overall Model
  - Build Feature List
  - Plan By Feature
  - Design By Feature
  - Build By Feature

Feature-Driven Development

- Core techniques
  - User involvement
  - Project team empowerment
  - Frequent deliveries
  - Emphasis on critical functionality
  - Iterative development
  - Reversible changes
  - Baseline of requirements before project start
  - Ongoing testing
  - Communication and cooperation among stakeholders

TSP

- Team Software Process
  - team extension of PSP
  - principles
    - team roles
    - iterative development
    - empirical measurement
    - process scripts



TSP

Cleanroom

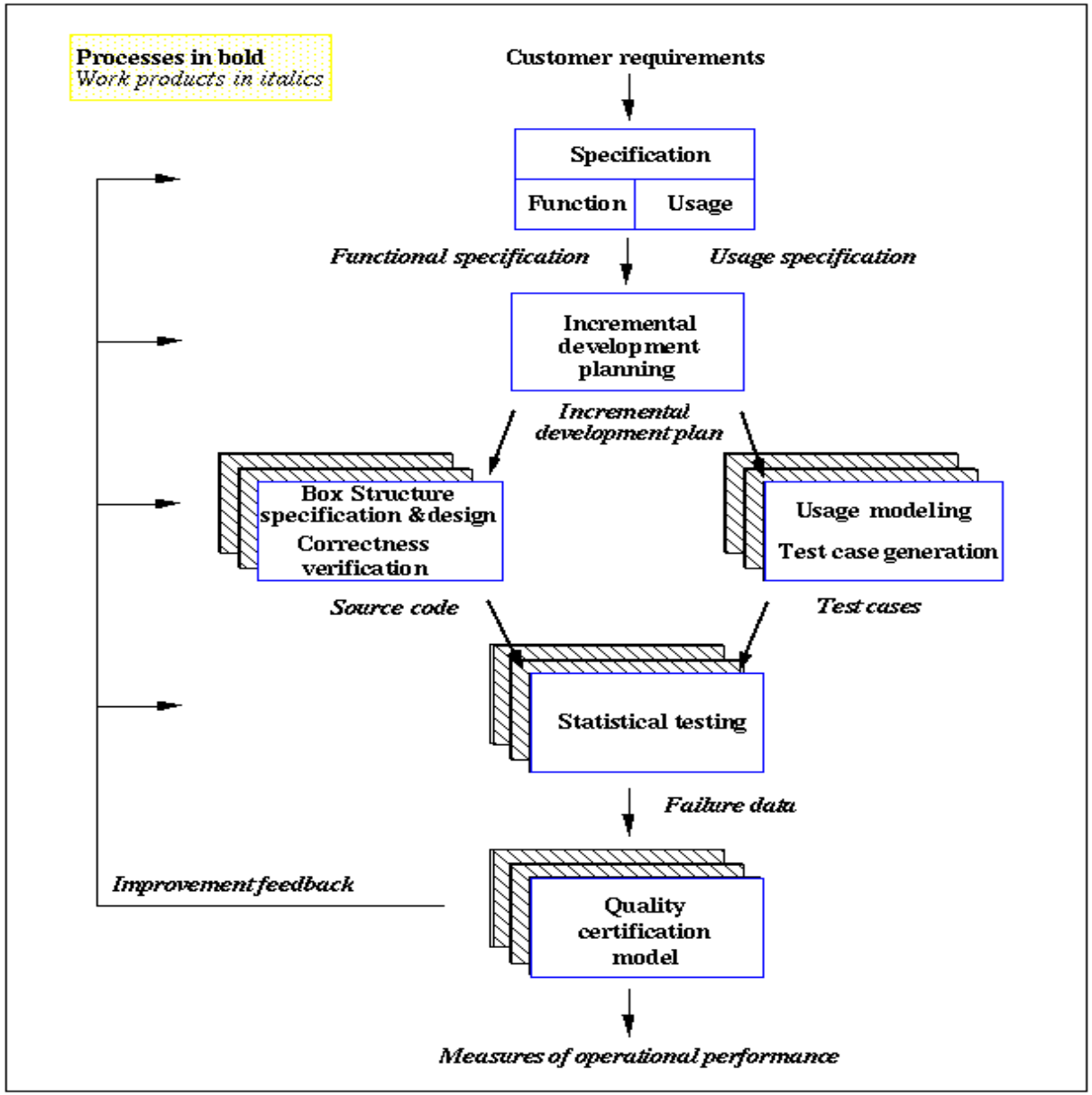
- Overview
  - “Cleanroom software engineering is a managerial and engineering process for the development of high quality software with certified reliability. Cleanroom focuses on defect prevention instead of defect correction, and certification of reliability for the intended environment of use.”
  - “Cleanroom represents a paradigm shift from traditional, craft-based practices to rigorous, engineering-based practices. Mathematical function theory is the basis for development practices, and applied statistics is the basis for testing practices. Cleanroom software engineering yields software that is correct by mathematically sound design, and software that is certified by statistically valid testing.”

http://source.asset.com/stars/loral/cleanroom/tutorial/overview.html

Cleanroom

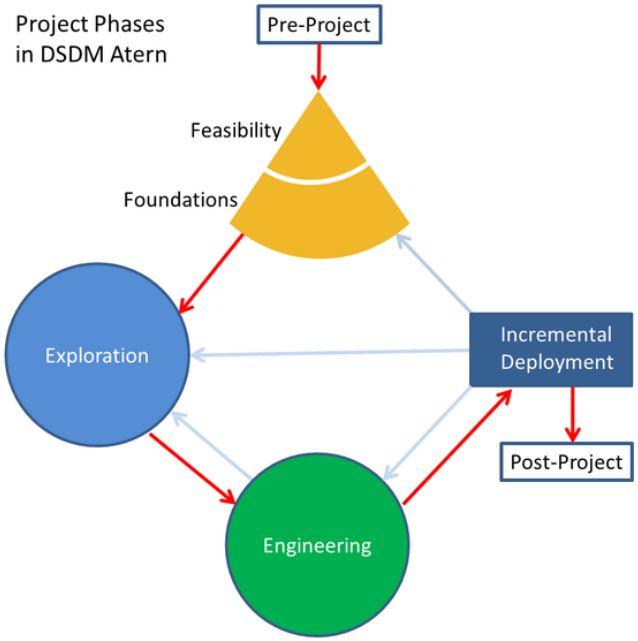
- Principles
  - Small teams
    - 6-8 people
  - Incremental development under statistical quality control
    - solutions are built in increments
    - progress and products are measured, actuals compared to planned
    - In-control process means OK to progress to next increment; out-of-control process means return to design
  - Software development based on mathematics
    - box structure model used for specification, design, and verification
    - output = f(input history)
  - Software tested on statistical principles
    - expected operational use is represented as usage model
    - test cases randomly generated from usage model
    - failure data interpreted according to statistical models

The Cleanroom Software Engineering Process



Dynamic Systems Development Method

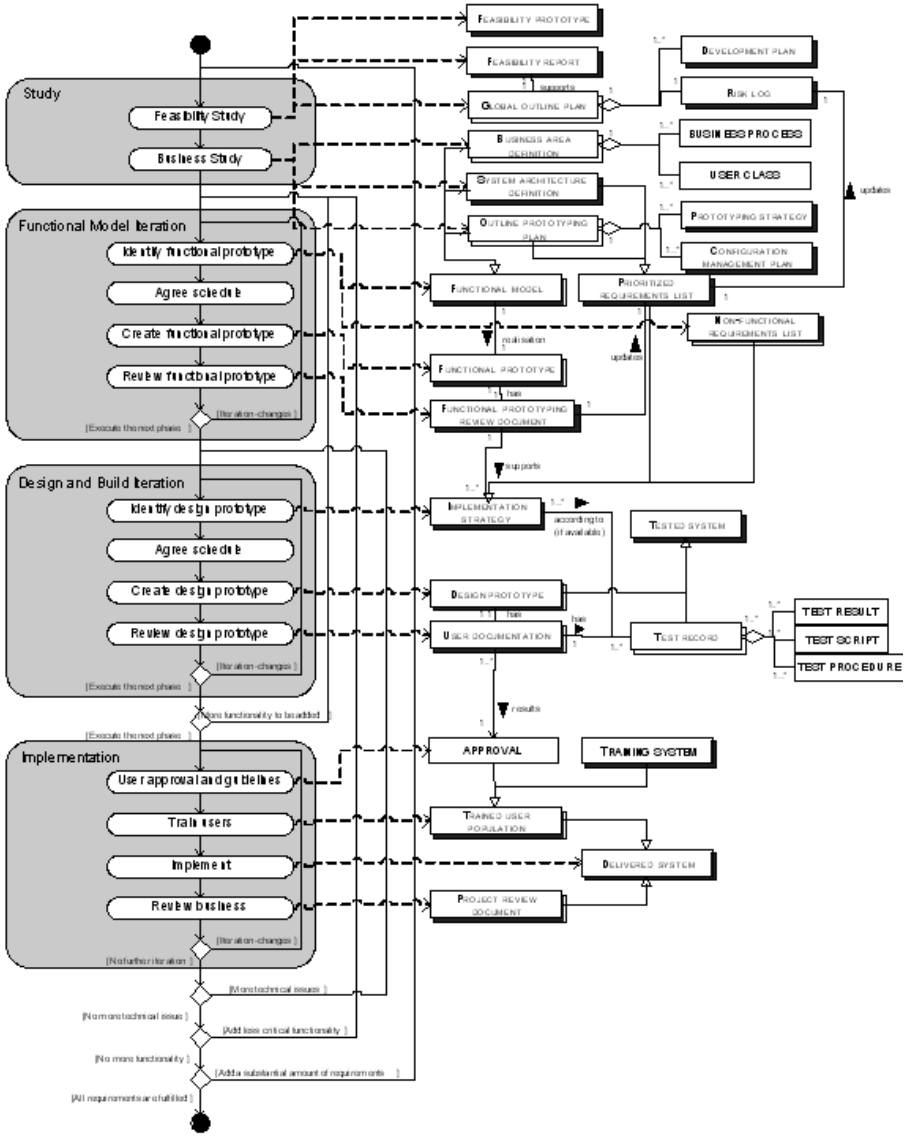
- General
  - Based on RAD (Rapid Application Development)
  - Primary user base is EU
  - Intellectual property protections inhibit wide-spread adoption
- Principle
  - Focus on information systems with tight schedules and budgets
- Tenets
  - Feasibility Study
  - Business Study
  - Functional Model (Prototype) Iteration
  - Design and Build Iteration
  - Implementation



Dynamic Systems Development Method

- Core Techniques
  - Project team empowerment
  - Frequent deliveries
  - Emphasis on critical functionality
  - Iterative development
  - Reversible changes
  - Baseline of requirements before project start
  - Ongoing testing
  - Timeboxing
  - MoSCoW
    - Must/Should/Could/Would prioritization
  - Prototyping
  - Workshop
    - bringing stakeholders together
  - Modeling

DSDM

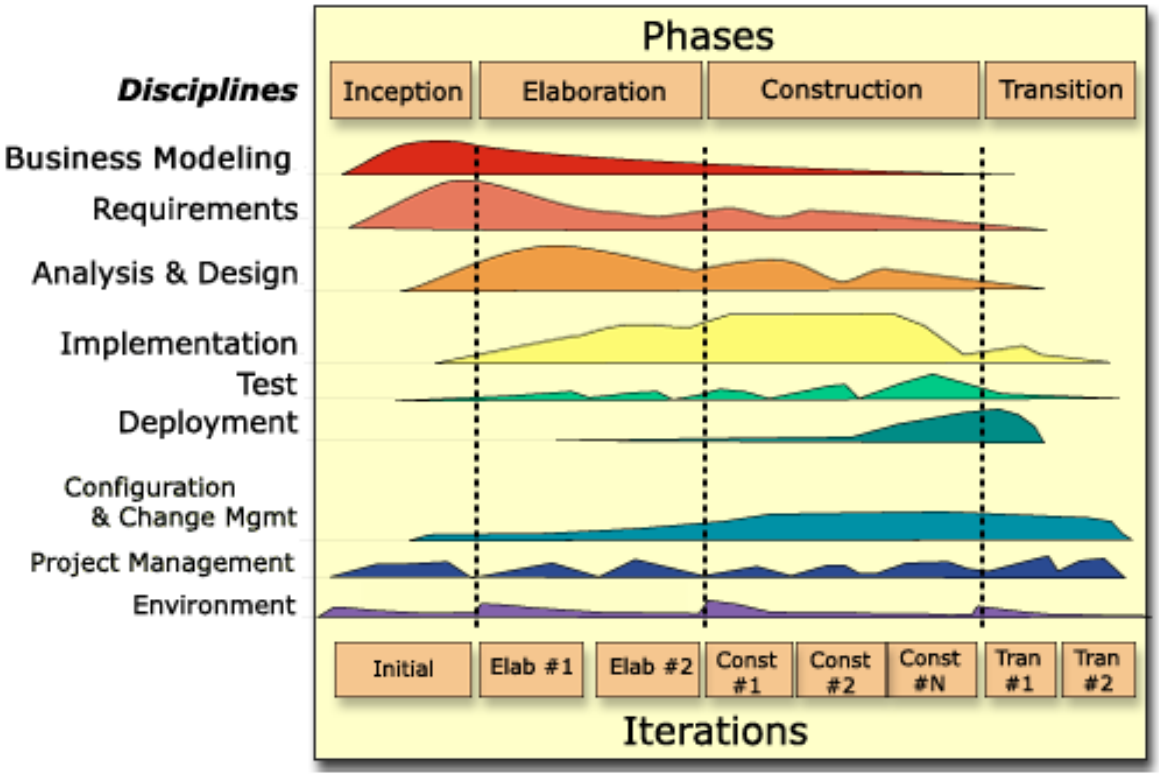

Study
Feasibility Study
Business Study
Functional Model Iteration
Identify functional prototype
Agree schedule
Create functional prototype
Review functional prototype
[Iteration changes]
[Execute the next phase]
Design and Build Iteration
Identify design prototype
Agree schedule
Create design prototype
Review design prototype
[Iteration changes]
[Execute the next phase]
[More functionality to be added]
[Execute the next phase]
Implementation
User approval and guidelines
Train users
Implement
Review business
[Iteration changes]
[No further iteration]
[More technical issues]
[No more technical issues]
[Add less critical functionality]
[No more functionality]
[Add a substantial amount of requirements]
[All requirements are fulfilled]
FEASIBILITY PROTOTYPE
FEASIBILITY REPORT
GLOBAL OUTLINE PLAN
BUSINESS AREA DEFINITION
SYSTEM ARCHITECTURE DEFINITION
OUTLINE PROTOTYPING PLAN
FUNCTIONAL MODEL
PRIORITIZED REQUIREMENTS LIST
NON-FUNCTIONAL REQUIREMENTS LIST
FUNCTIONAL PROTOTYPE
FUNCTIONAL PROTOTYPING REVIEW DOCUMENT
IMPLEMENTATION STRATEGY
DESIGN PROTOTYPE
USER DOCUMENTATION
APPROVAL
TRAINED USER POPULATION
PROJECT REVIEW DOCUMENT
DEVELOPMENT PLAN
RISK LOG
BUSINESS PROCESS
USER CLASS
PROTOTYPING STRATEGY
CONFIGURATION MANAGEMENT PLAN
TESTED SYSTEM
TEST RECORD
TEST RESULT
TEST SCRIPT
TEST PROCEDURE
TRAINING SYSTEM
DELIVERED SYSTEM
supports
updates
realisation
has
according to (if available)
results

Rational Unified Process

- General
  - Used in conjunction with Rational tools
  - Not out-of-the-box ready: must be tailored to the appropriate weight
- Principles
  - Emphasis on risk-driven spiral
- Tenets
  - Adapt the process
  - Balance stakeholder priorities
  - Collaborate across teams
  - Demonstrate value iteratively
  - Elevate the level of abstraction
  - Focus continuously on quality

Rational Unified Process

- Core Techniques



- Develop iteratively
- Manage requirements
- Use components
- Model visually
- Verify quality
- Control changes

+ Roles, Tasks, Work products

How Are Processes Adopted?

- Evolutionary
  - Process insertion is orchestrated by leader
  - Those involved with process are empowered to define it
  - Process formation is typically guided by specific process model
- Revolutionary
  - A specific process is selected
  - Those involved implement the process
  - Process implementation is typically phased

COMP5700/6700/6706 Goal Process

Minimal Guiding Indicators

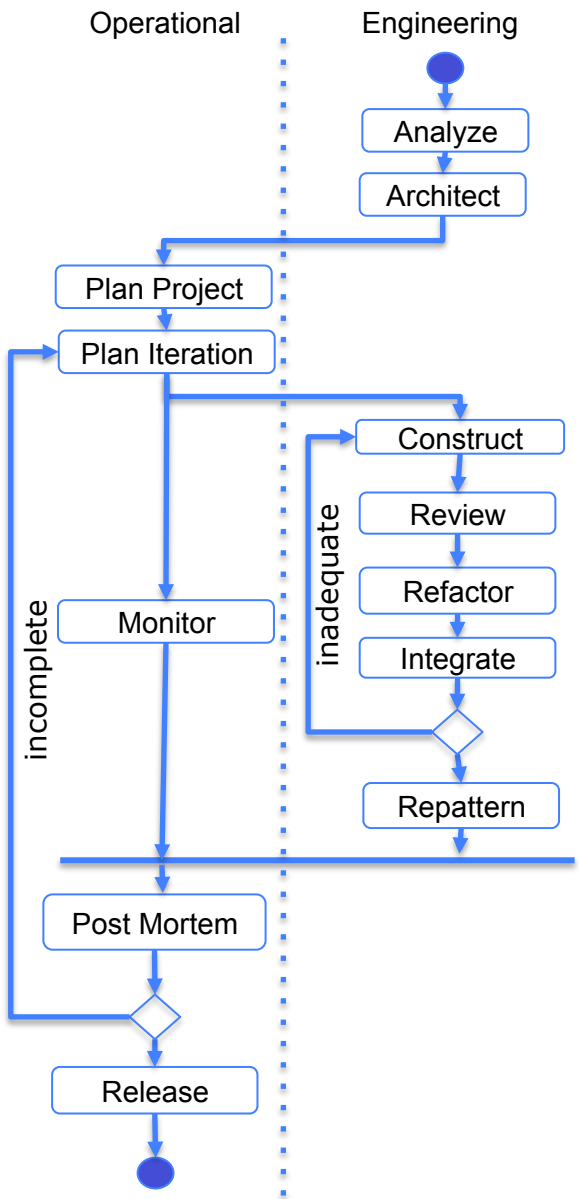
Goal	Indicator
Cost:	Don't care
Schedule:	PV/EV >.75
Performance:	
Product:	
NFR:	none
FR:	100% BVA
Process:	pain < value

Minimal Sufficient Activities

Engineering Activities
- Envision
  - Analyze
- Synthesize
  - Architect
- Articulate
  - Construct
  - Refactor
- Interpret
  - Review
  - Integrate
  - Repattern

Operational Activities
- Plan
  - Plan project
  - Plan iteration
- Monitor
- Release

Minimal Viable Process



Minimal Effective Practice

MSA	MEP
Analyze	Scenarios
Architect	CRC
Plan Project	Component-based estimation
Plan Iteration	Component-iteration map
Construct	TDD
Review	Review checklist Test code coverage
Refactor	Ad hoc sniffing
Integrate	Ad hoc
Repattern	Ad hoc
Monitor	Time log Change log Burndown
Post Mortem	PV/EV
Release	Eclipse zip spreadsheets

Final note

- the key to processes is in “accepting the process rather than the imposition of the process.”

  - Martin Fowler

Summary

Topics

- **Process selection**
- **Multi-person processes**

Key Points

- Scalability = fitting the project at hand
- Scalability requires a defined and managed process
- Software engineers are faced with projects of multiple levels of complexity
- Historical evolution of process strengths: heavy-weight vs light-weight
- Current trends lean toward small teams, adaptive processes
- Adaptive processes include Scrum, LSD, XP, Crystal, TSP, FDD, DSDM, UP