# Software Engineering
# Lecture 02

## Software Process

***Referenced documents may be accessed via the URLs located on the course Angel page. Off-campus access will require authentication.***

# Outline

- Software Process
- Software Processes for Large Systems
  - Waterfall Process
  - Spiral Development Process
  - Unified Process
- Agile Development Processes
  - Agile Manifesto
  - XP, Scrum
  - Criticisms of Agile Methods
- Process Improvement Initiatives

# Software Process - 1

"The process by which user needs are translated into a software product"

***IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990**

# Software Process - 2

- A ***software process*** includes
  - Includes life-cycle model
  - Use of specific tools or techniques
  - Modeling/Design/Coding or other standards
  - Metrics for measuring effectiveness
  - Etc.
- Wide variation of processes currently in use

# Ad-Hoc Development

(1) Build an initial version of the product
(2) Deliver to customer
(3) Modify until customer is satisfied

- Adequate for very small systems
- Does not scale to large software systems

# Software Processes for Large Systems

UAH
CPE 353

# Examples

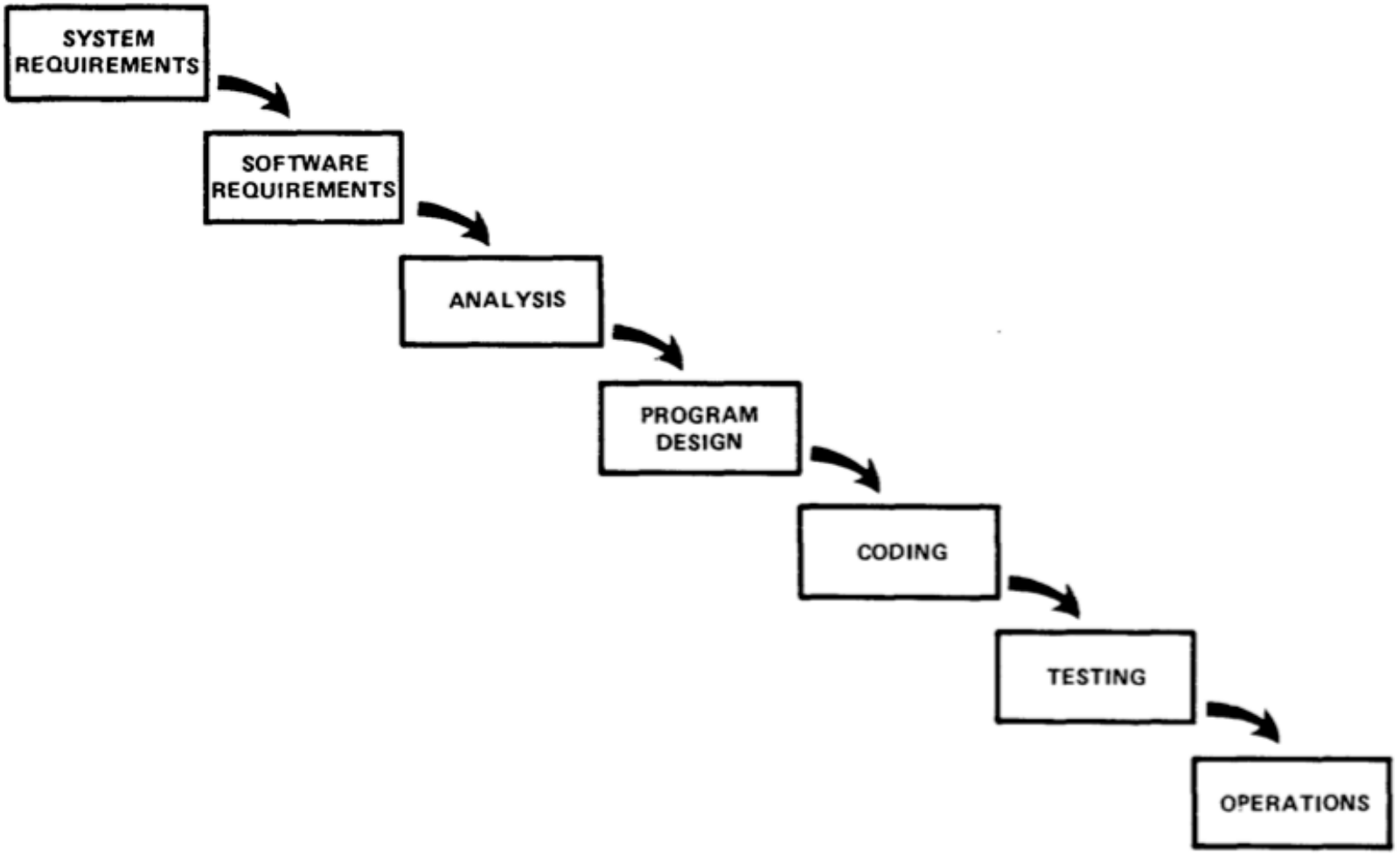
- Waterfall Process
- Spiral Development
- Unified Process

# Waterfall Process

- Process intended for the development of large systems
- Document driven
- Includes opportunities for customer involvement after requirements but before delivery

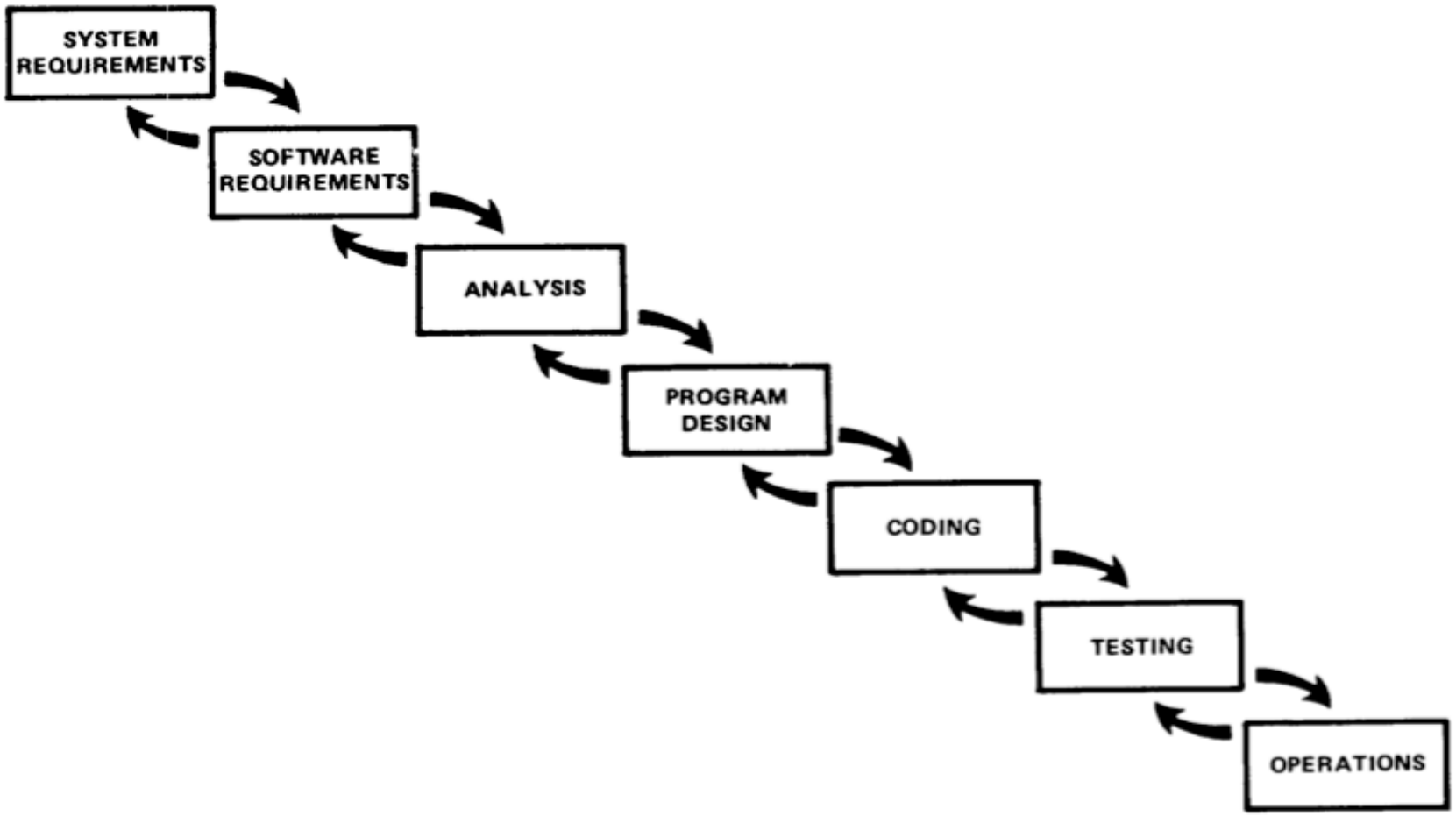
Winston Royce, "Managing the Development of Large Software Systems",
**Technical Papers of Western Electric Show and Convention (WesCon)**, August 25-28, 1970.

# Waterfall Process – Ideal



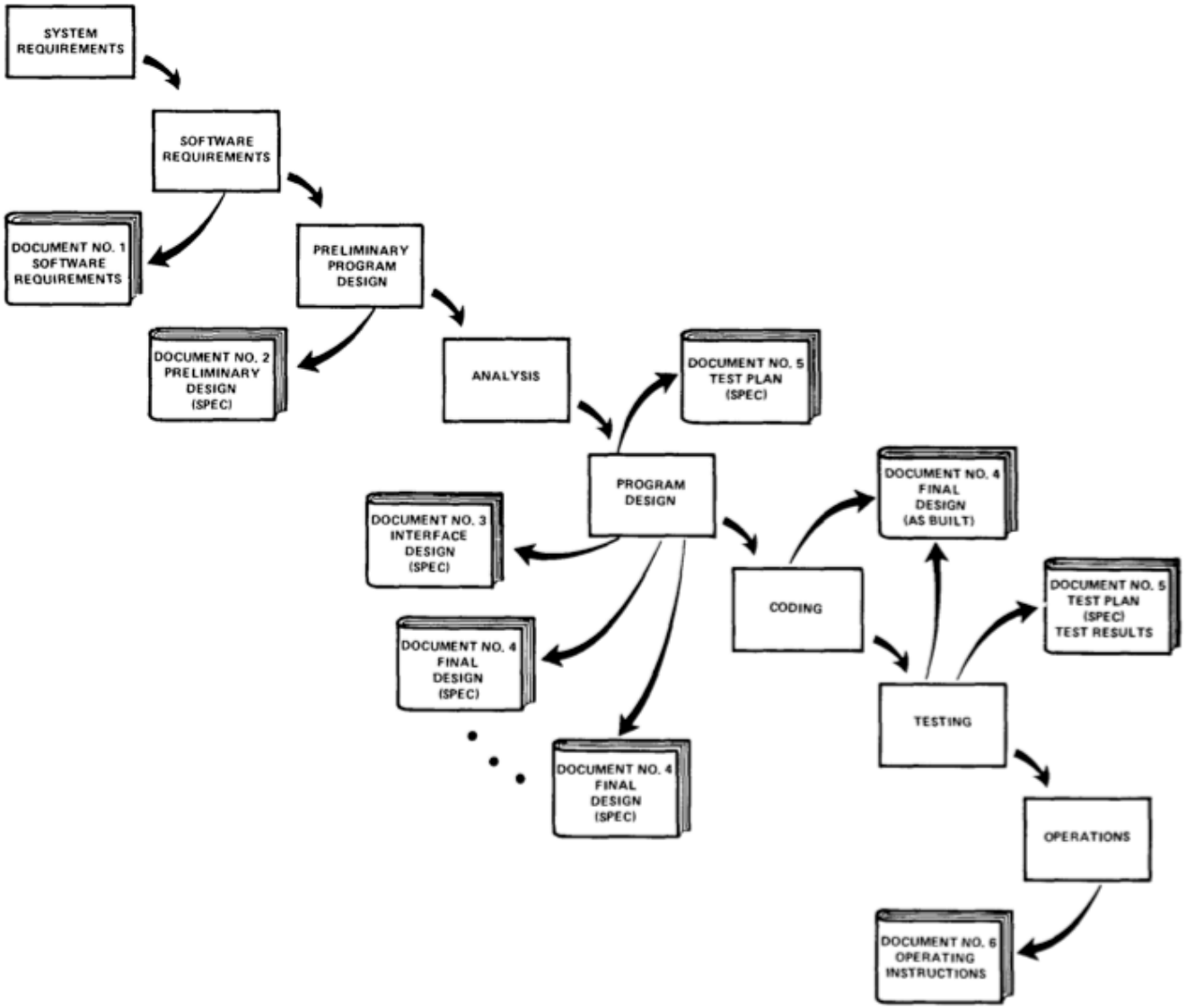

Winston Royce, "Managing the Development of Large Software Systems",
**Technical Papers of Western Electric Show and Convention (WesCon)**, August 25-28, 1970.

# Waterfall Process – Iteration



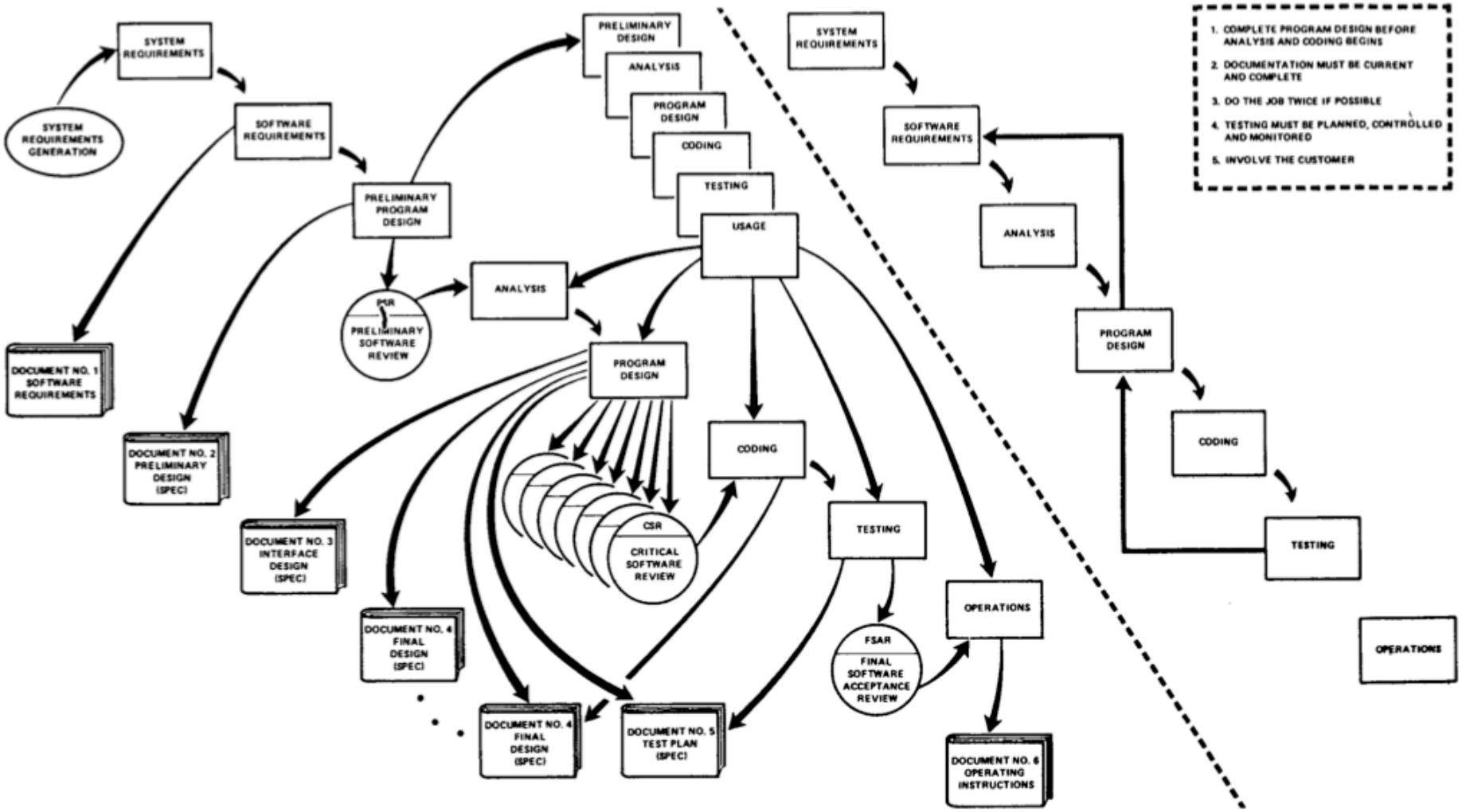

Winston Royce, "Managing the Development of Large Software Systems",
**Technical Papers of Western Electric Show and Convention (WesCon)**, August 25-28, 1970.

# Waterfall Process – Docs




Winston Royce, "Managing the Development of Large Software Systems",
**Technical Papers of Western Electric Show and Convention (WesCon)**, August 25-28, 1970.

# Waterfall Process – Twice?




Winston Royce, "Managing the Development of Large Software Systems",
**Technical Papers of Western Electric Show and Convention (WesCon)**, August 25-28, 1970.

# Waterfall Process – Issues

- Document-driven
  - Schedule and budget consumed in the production of documents
  - Documents requirement ongoing maintenance to remain consistent with each other and delivered product
  - Document paralysis possible
- Substantial budget and schedule may be consumed before customer sees any code product
  - Delayed customer feedback increases risk

# IEEE J-STD-016-1995

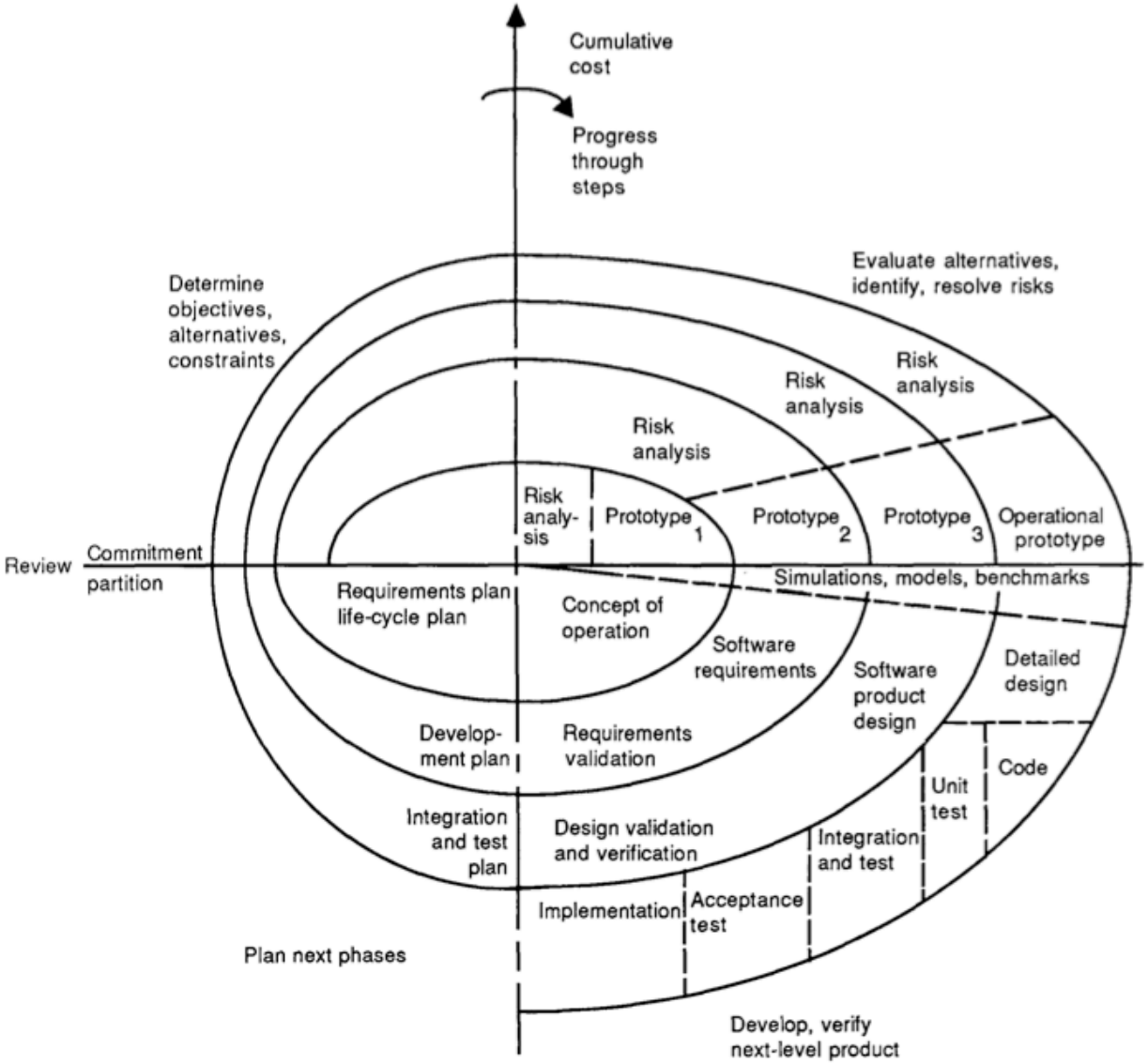
- Basis for many industrial software development processes currently in use today
- Link on Angel course page

# Spiral Development

- Risk-Driven Process
  - Uses risk analysis to identify and prioritize objectives for next iteration
  - Prototypes used to explore mitigation options

Barry Boehm, "A Spiral Model of Software Development and Enhancement", ***IEEE Computer***, vol. 21, no. 5, May 1988, pp. 61-72.

# Spiral Development




Barry Boehm, "A Spiral Model of Software Development and Enhancement", ***IEEE Computer***, vol. 21, no. 5, May 1988, pp. 61-72.

# Spiral Development – Issues

- While good results have been achieved at TRW on internal products, not as easy to apply when dealing with external customers
- Requires experience at risk analysis

Barry Boehm, "A Spiral Model of Software Development and Enhancement", ***IEEE Computer***, vol. 21, no. 5, May 1988, pp. 61-72.

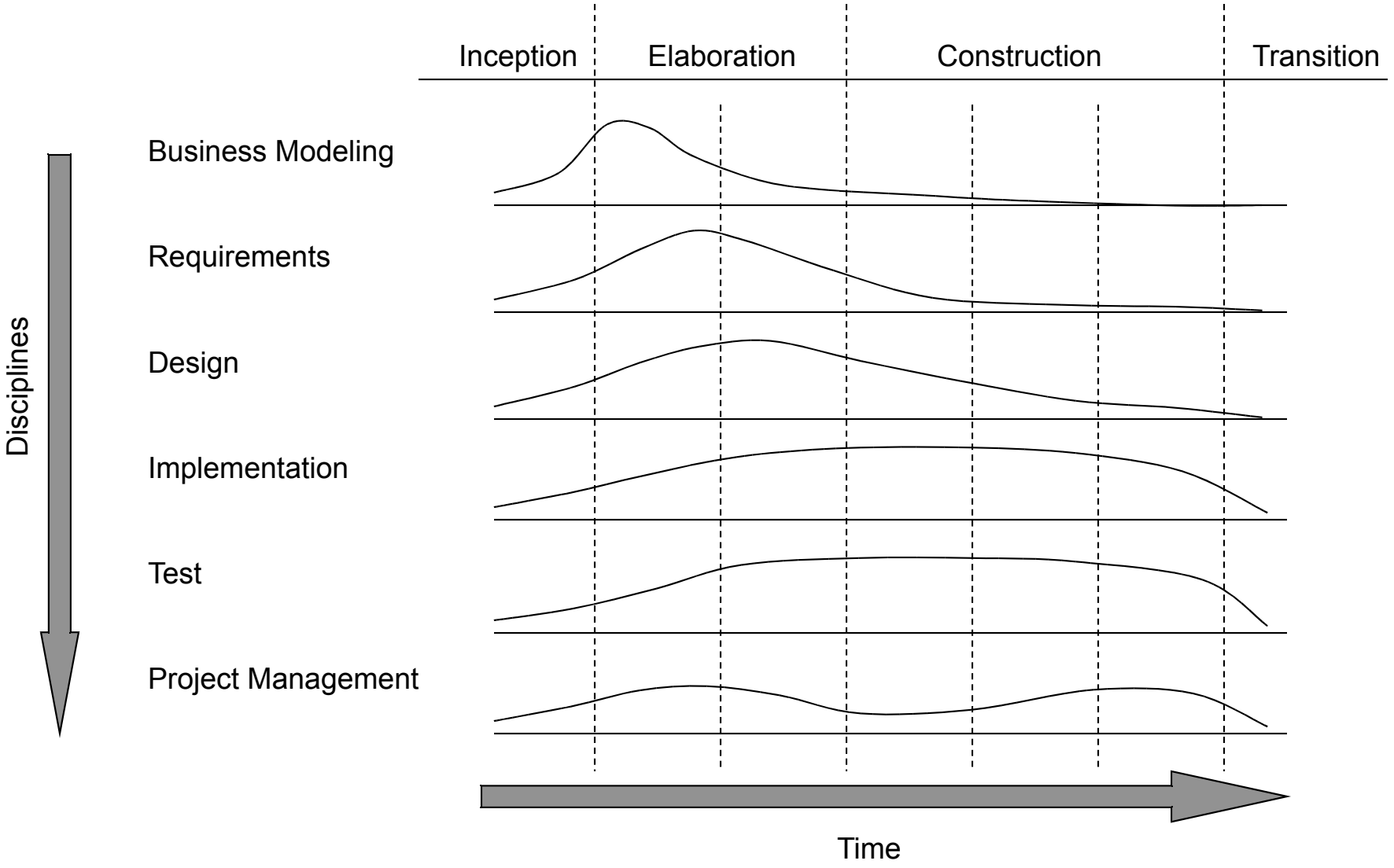
# Unified Process - 1

- Iterative and incremental process intended for large systems
- Developed by
  - Grady Booch, Jim Rumbaugh
    - Notation for capturing OOAD
  - Ivar Jacobson
    - Objectory OOAD methodology

# Unified Process - 2

- Combines
  - Object-Oriented Analysis and Design
  - Unified Modeling Language (UML)
  - Short, time-boxed iterations ending with delivery of a partial product
  - Allows early and frequent customer feedback
  - Risk-driven approach from Spiral development to attack high-risk, high-value items early

# Unified Process




**Craig Larman, *Applying UML and Patterns*, 2nd Ed., 2002**

# Unified Process

- "Best Practices" integrated into UP
  - Iterative software development
  - Requirements management
  - Visual software modeling
  - Test early, test often, test realistically
  - Configuration and change management

# Agile Development Processes

UAH
CPE 353

# Motivation for Agile Processes - 1

- Despite Waterfall, Spiral, Unified, and other processes, many project teams still struggle to satisfy cost, schedule, and quality goals
- ***Agile Processes*** are an attempt to address short-comings of traditional processes by focusing on delivery of value to the customer

# Motivation for Agile Processes - 2

- Perceived problems with other processes
  - Lengthy development efforts
    - Large systems are often multi-year efforts with traditional processes
    - Extensive documentation activities delay implementation
    - By the time implementation occurs, the customers needs may have changed substantially

# Motivation for Agile Processes - 3

- Perceived problems with other processes
  - Inability to adequately address changing requirements
    - Customers needs evolve over time
    - Lengthy development efforts consume substantial resources making it more difficult to adapt to substantial requirements changes late
  - Implicit assumption that the requirements are completely understood when project begins

# Motivation for Agile Processes - 4

- Perceived problems with other processes
  - Heroic Developer Efforts
    - May allow the team to satisfy an intermediate milestone, but they tend to degrade long term performance and may result in employee turnover
  - Complex methodologies
    - Numerous, interdependent artifacts generated
    - Developers may have to assume multiple roles

# Motivation for Agile Processes - 5

- Perceived problems with other processes
  - Wasted or Duplicated Effort
    - Requirements and design details are often captured multiple times – in dedicated documents and in the code itself
    - For the documents to be useful, they must be maintained so that they match the code

# Agile Processes - 1

- Family of processes that embrace:
  - ***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.

***Manifesto for Agile Software Development,***
http://agilemanifesto.org/

# Agile Processes - 2

- Agile processes emphasize
  - ***Short, Time-Boxed Iterations***
    - Each iteration should end with value delivered to the customer (i.e. functional partial product)
    - Why short iterations?
      - **Parkinson's Law**
        - » "Work expands so as to fill the time available for its completion"
        - » Long-term deadlines make this worse
        - » Short-term deadlines focus the team on completion of the next functional increment

# Agile Processes - 3

- Agile processes emphasize
  - Why short iterations? – continued
    - Forced Prioritization and Decisiveness
    - Team Satisfaction
      - » Tasks are checked off with the end of each iteration
    - Stakeholder Confidence
      - » As functionality is delivered on each iteration, all stakeholders can see that progress is being made

# Agile Processes - 4

- Agile processes emphasize
  - ***Incremental Design***
    - Design for what you know now
    - You may waste significant time devising a speculative design that may be discarded later as requirements change
    - As requirements change, your design may require improvement

***Essentials of Software Engineering,***
Frank Tsui and Orlando Karam, 2007

***Applying UML and Patterns,***
Craig Larman, 2002.

# Agile Processes - 5

- Agile processes emphasize
  - ***Early and Continuous User Feedback***
    - Mitigates risk
  - ***Face-to-face communication***
  - ***Minimal documentation***
    - Source code is primary document
  - ***Acceptance of change***

# Twelve Principles of Agile - 1

***Principle #1***

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

***Principle #2***

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

***Manifesto for Agile Software Development,***
http://agilemanifesto.org/principles.html

# Twelve Principles of Agile - 2

***Principle #3***

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

***Principle #4***

Business people and developers must work together daily throughout the project.

***Manifesto for Agile Software Development,***
http://agilemanifesto.org/principles.html

# Twelve Principles of Agile - 3

***Principle #5***

Build projects around motivated individuals.
Give them the environment and support they need,
and trust them to get the job done.

***Principle #6***

The most efficient and effective method of
conveying information to and within a development
team is face-to-face conversation.

***Manifesto for Agile Software Development,***
http://agilemanifesto.org/principles.html

# Twelve Principles of Agile - 4

***Principle #7***

Working software is the primary measure of progress.

***Principle #8***

Agile processes promote sustainable development.

The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

***Manifesto for Agile Software Development,***
http://agilemanifesto.org/principles.html

# Twelve Principles of Agile - 5

***Principle #9***

Continuous attention to technical excellence and good design enhances agility.

***Principle #10***

Simplicity--the art of maximizing the amount of work not done--is essential.

***Manifesto for Agile Software Development,***
http://agilemanifesto.org/principles.html

# Twelve Principles of Agile - 6

***Principle #11***

The best architectures, requirements, and designs emerge from self-organizing teams.

***Principle #12***

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

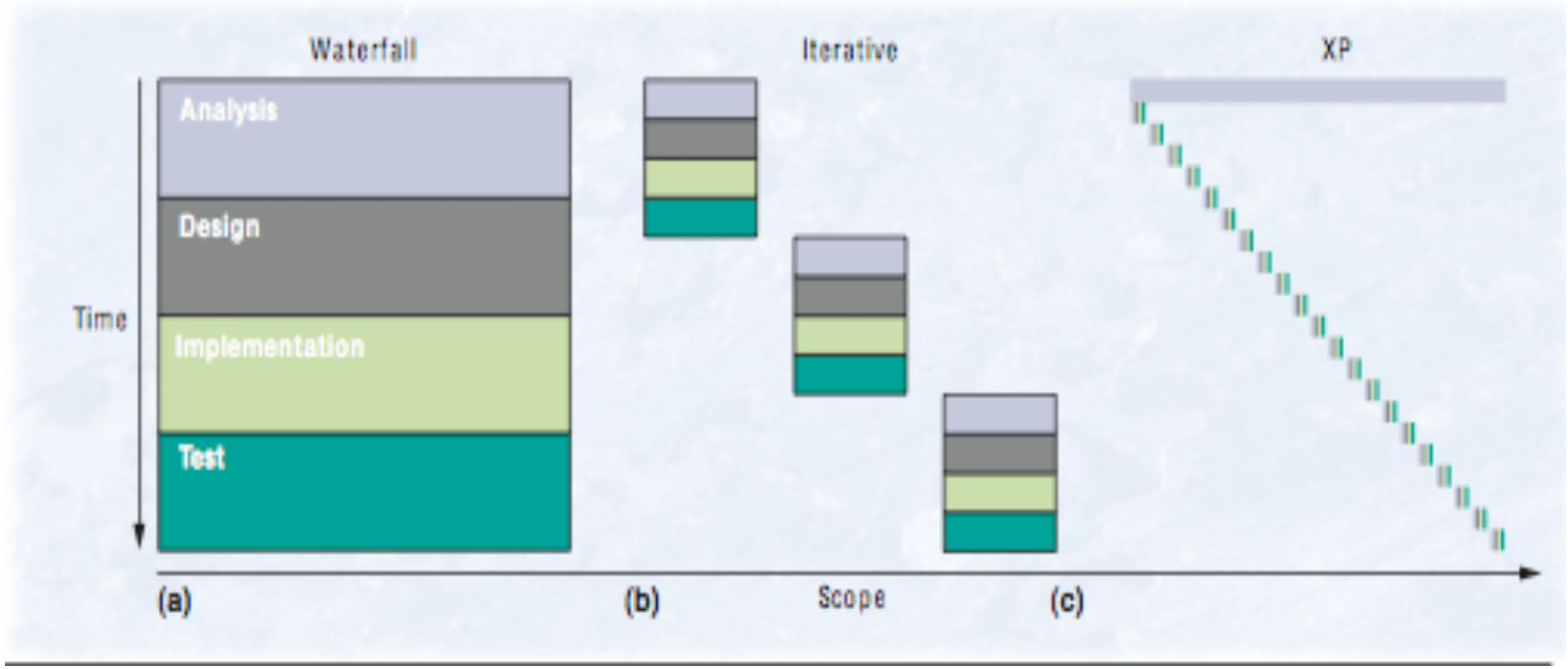
***Manifesto for Agile Software Development,***
http://agilemanifesto.org/principles.html

# Extreme Programming (XP) - 0

- Developed by Kent Beck in 1990s while he worked on C3, the Chrysler Comprehensive Compensation System
- XP is a light-weight process for
  - Small to medium-sized teams
  - Projects with vague or rapidly changing requirements
  - Iterative and incremental process

***Extreme Programming Explained,***
Kent Beck, 1999

# Extreme Programming (XP) - 1




Kent Beck, “Embracing Change with Extreme Programming”
***IEEE Computer,*** October 1999, pp.70-77.

# Extreme Programming (XP) - 2

- ***Risk*** is the basic problem in software development
  - Goal of XP is to mitigate risk

***Extreme Programming Explained,***
Kent Beck, 1999

# Extreme Programming (XP) - 3

- Four variables to control during software development
  - **Cost**
  - **Time**
  - **Quality**
  - **Scope**
- Focus on controlling ***Scope*** as a means to manage the other three variables

***Extreme Programming Explained,***
Kent Beck, 1999

# Extreme Programming (XP) - 4

- Four Values of XP
  - **Communication**
    - Verbal collaboration between developers and stakeholders
  - **Simplicity**
    - Design for today's requirements, not some future requirements that may be discarded anyways
    - The design may be ***refactored*** later if need be

***Extreme Programming Explained,***
Kent Beck, 1999

# Extreme Programming (XP) - 5

- Four Values of XP (continued)
  - **Feedback**
    - From the software itself via **testing**
    - Iterative planning provides customer feedback on cost and schedule impact of new requirements
  - **Courage**
    - Required when designing for now, not future
    - Required when major refactoring is needed

***Extreme Programming Explained,***
Kent Beck, 1999

# Twelve XP Practices - 1

- **Planning Game**
  - Business Issues
    - Scope, Priority, Release Composition and Dates
  - Technical Issues
    - Estimates, Consequences, Process

- **Small Releases**
  - Each release should be as small as possible
  - Each new feature should be complete

***Extreme Programming Explained,***
Kent Beck, 1999

# Twelve XP Practices - 2

- **Metaphor**
  - The common, overall view of the system

- **Simple Design**
  - The Right Design is the one that
    - System (code and tests) communicates everything that must be communicated
    - Runs all tests
    - No duplicate logic
    - Has fewest possible classes and methods

***Extreme Programming Explained,***
Kent Beck, 1999

# Twelve XP Practices - 3

- **Testing**
  - Programmers write their own unit tests and 100% of unit tests must be passed
  - Unit tests are written BEFORE code is written
  - Customers write functionality tests

- **Refactoring**
  - Restructuring of system without changing the overall behavior to remove duplication, simplify, or add flexibility

***Extreme Programming Explained,***
Kent Beck, 1999

# Twelve XP Practices - 4

- **Pair Programming**
  - All production code is written by two programmers at one machine
- **Collective Ownership**
  - Anyone can change any code in the system at any time
- **Continuous Integration**
  - Code integrated and system built frequently each day each time a task is completed

***Extreme Programming Explained,***
Kent Beck, 1999

# Twelve XP Practices - 5

- **40-hour Week**
  - Never work overtime two weeks in a row
- **On-Site Customer**
  - Real user is a part of the development team
- **Coding Standards**
  - All code written according to standards to ensure quality communication through the code

***Extreme Programming Explained,***
Kent Beck, 1999

# Closer Look at Key XP Practices

# Design in XP

- Design Strategy in XP
  - Start with a test
  - Design and implement just enough to get this test ***and the previous tests*** running successfully
  - Repeat
  - If there is an opportunity to simplify the design, then simplify it
- Few design artifacts generated

***Extreme Programming Explained,***
Kent Beck, 1999

# Coding in XP - 1

- Unit Tests are written **BEFORE** the code
- Automated Testing provides immediate feedback
- Pair Programming is a key element

***Extreme Programming Explained,***
Kent Beck, 1999

# Coding in XP - 2

- **Pair Programming**
  - All production code written by pairs of programmers sharing a single computer
  - Implicit peer review improves quality
  - Coding standards reduce squabbling over minor issues (indentation, curly brace placement, etc.)
  - Use of the simplest design that will work increases the likelihood that both people understand the task at hand

***Extreme Programming Explained,***
Kent Beck, 1999

# Testing in XP - 1

- **Unit Testing**
  - Unit testing is emphasized in XP
  - Unit tests are created **BEFORE** coding begins
  - Tests must be **Isolated** and **Automated**
    - An ***Isolated Test*** that fails does not result in the failure of other tests
    - **Automated** testing returns a Pass/Fail verdict and these tests may be rerun whenever **refactoring** has occurred (**regression testing**)
  - Code must pass 100% of its Unit Tests
  - Programmers write Unit Tests

***Extreme Programming Explained,***
Kent Beck, 1999

# Testing in XP - 2

- **Acceptance Tests**
  - Focus on testing functionality that is captured in the set of stories the software should satisfy
  - Customers write the Acceptance Tests in conjunction with the team's dedicated Tester
  - Acceptance tests are developed BEFORE coding begins

- ***There should be no feature without automated tests***

***Extreme Programming Explained,***
Kent Beck, 1999

# Scrum - 1

- An iterative and incremental Agile process developed by Jeff Sutherland, Ken Schwaber, and others in 1990s
- Each iteration is called a **sprint** which lasts from 2-4 weeks
- **product backlog** is a prioritized list of desired features
- **sprint backlog** is the subset features that will be attempted on the sprint

# Scrum - 2

- **Scrum** (15 minute daily meeting)
  - What did you do since last Scrum?
  - Do you have any obstacles?
  - What will you do before the next Scrum?
- Each sprint ends with a ***demonstration of new functionality***
- Items not completed are returned to the product backlog
- Relies on a self-directed team

# Software Process Improvement Initiatives

UAH
CPE 353

# Why Do Projects Fail? - 1

- ***Unrealistic Schedules***
  - Encourages a mad rush to get a product built, regardless of the product's quality or how well the product meets the customers needs

# Why Do Projects Fail? - 2

- ***Inappropriate Staffing***
  - For timely delivery, the development team must have enough people with the right skill sets
  - Moreover, management must protect the team from interruptions and distractions

***Winning with Software: An Executive Strategy***
Watts S. Humphrey, 2002

# Why Do Projects Fail? - 3

- ***Changing Requirements***
  - Project requirements may change as the product is being developed
  - After a certain point, requirements change may have a significant negative impact on the development effort

***Winning with Software: An Executive Strategy***
Watts S. Humphrey, 2002

# Why Do Projects Fail? - 4

- ***Poor Quality Work***
  - Rework is not free
  - "Quality is Free", Philip Crosby, 1979
    - It takes no more time to develop a high-quality product than a poor-quality product
    - It may even take **less time** due to rework
  - Ditching quality-control steps in an effort to avoid late delivery often results in late delivery

***Winning with Software: An Executive Strategy***
Watts S. Humphrey, 2002

# Why Do Projects Fail? - 5

- ***Believing in Magic***
  - New technologies that are not Silver Bullets
    - Object-Oriented Analysis and Design
    - Test automation
    - Commercial-Off-the-Shelf (COTS)

***Winning with Software: An Executive Strategy***
Watts S. Humphrey, 2002

# Capability Maturity Model (CMM)

- Capability Maturity Model (CMM)
  - Strategy for improving software processes
  - Relative quality of a development process is called its Maturity Level
  - Five Maturity Levels in the CMM
    - Each level has characteristic Key Process Areas (KPAs)
    - Allows companies to make incremental changes
    - Quantitative process quality measurements provide feedback on the effectiveness of the changes
    - DOD contracts require a Maturity Level of 3 or more
  - Ultimate goal of CMM
    - *Continuous Process Improvement*
  - CMMI is newest version of this assessment framework

# Capability Maturity Model (CMM)

| CMM Level | Description |
|---|---|
| 5 - Optimizing | Continuous process improvement via quantitative process feedback; piloting new technologies |
| 4 - Managed | Detailed measures of software process and products are quantitatively understood and controlled. |
| 3 - Defined | The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software. |
| 2 - Repeatable | Basic project management processes are established to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications. |
| 1 - Initial | The software process is characterized as *ad hoc* and occasionally even chaotic. Few processes are defined, and success depends on individual effort. |

**See Paulk et al., *The Capability Maturity Model: Guidelines for Improving the Software Process***

# Capability Maturity Model (CMM)

| CMM Level | Key Process Areas (KPAs) |
|---|---|
| 5 - Optimizing | Process change management<br>Technology change management<br>Defect prevention |
| 4 - Managed | Software quality management<br>Quantitative process management |
| 3 - Defined | Peer reviews<br>Intergroup coordination<br>Software product engineering<br>Integrated software management<br>Training program<br>Software process definition<br>Software process focus |
| 2 - Repeatable | Software configuration management<br>Software quality assurance<br>Software subcontract management<br>Software project tracking and oversight<br>Software project planning<br>Requirements management |
| 1 - Initial | |



**See Paulk et al., *The Capability Maturity Model: Guidelines for Improving the Software Process***

# Costs of Process Improvement

- Costs associated with process improvement initiatives
  - Initial assessment
  - Training
  - Tools
  - Reassessment
- Costs of not pursuing process improvement
  - Cost and schedule overruns
  - Poor quality products
  - Personnel turnover
  - Marketing disadvantage