

Introduction to Software Life Cycles

CSCI 5828: Foundations of Software Engineering
Lecture 06 — 09/10/2015

Goals

- Present an introduction to the topic of software life cycles
 - concepts and terminology
 - benefits and limitations
 - examples

Background (I)

- In software engineering, “process is king”
 - That is, the process by which we do things is of utmost importance
- We want our activities to be coordinated and planned
 - that is, “engineered”
- Why?
 - **A high quality process increases our ability to create a high quality product**

Background (II)

- **process**
 - a series of steps that **people** follow involving **activities** and **resources** that produce an intended **output** of some kind
- Activities are arranged into a **workflow** with
 - **sequences of steps** (supports basic work practice)
 - **branches** (supports conditional behavior)
 - **loops** (supports iteration)
- Each **activity**
 - has a **set of inputs** and/or entry criteria
 - and may produce an output that is used in a subsequent step

Background (III)

- A process typically has a set of guiding principles about why you should follow its particular approach
 - it should be able to articulate the goals of each of its activities
- A process uses resources, subject to a set of constraints
 - two primary constraints: **schedule** (time) & **budget** (money)
- Designers of software life cycles created their particular life cycle to help software engineers achieve their goals while meeting their constraints
 - Unfortunately, few life cycles offer guidance on what to do when a limit has been reached
 - i.e. you've run out of time or you've run out of money
 - Agile is different, as we shall see

Background (IV)

- Why bother with defining and following a life cycle for software development?
 - Impose **consistency** and **structure** on the work practice of an organization
 - especially across project teams in a single organization
 - or across two or more projects performed by the same team
 - provide a vehicle for **capturing/measuring performance** to
 - improve future performance by a particular team
 - to provide evidence needed to change/improve the process
- To answer the question: **What do I do today?** 😊

Background (V)

- Similarities and differences with manufacturing processes
 - Software life cycles are similar to manufacturing processes
 - You need to **design the process** to produce a **high quality product**
 - You need to **monitor** the process and look for ways to **improve** it
 - The process organizes the steps to ensure the product can be produced within budgetary and scheduling constraints
- BUT
 - in manufacturing, design is “short”, production is “long” and most of your costs are tied up in production; use varies from instant to long lived
 - in software, design is “long” (and difficult), production is instantaneous (it’s trivial to create a new copy of the final system) and use can be “forever”

Typical Steps in a Software Life Cycle

- Feasibility; Development of a Business Plan
- Requirements Analysis and Specification
- Design
- Implementation and Integration
- Operation and Maintenance
- **Pervasive Concerns**
 - Testing
 - Change Management
 - Configuration Management
 - Build Management and Continuous Integration

Heads-Up

- In the following slides (10-29), I adopt a traditional perspective of SE
 - one that is consistent with the “waterfall” model of development
 - one that assumes a development context with many large stakeholders
 - one that assumes “requirements and design up front”
- We will revisit and unpack this material as we present/investigate agile life cycles more deeply
 - A lot of this material is “musty” from a modern software engineering perspective but it is important to understand the changes that Agile life cycles made to the more traditional perspective of SE

Feasibility and Business Plan

- In some (most?) development contexts
 - an idea for a new software system does NOT lead straight to requirements
 - instead, just enough of the proposed system is defined/discussed to assess
 - whether it is technically feasible to develop
 - whether there are enough resources to develop it
 - whether it will produce enough revenue to justify the costs of development
- Many proposed systems fail to get past this stage

Requirements Analysis and Specification

- **Problem Definition** ⇒

- **Requirements Specification**

- determine exactly what client wants and identify constraints
 - develop a contract with client
 - Specify the product's task explicitly

- **Difficulties**

- client asks for wrong product
 - client is computer/software illiterate

- specifications may be ambiguous, inconsistent, incomplete

- **Validation**

- extensive reviews to check that requirements satisfy client needs
 - look for ambiguity, consistency, incompleteness
 - develop system/acceptance test plan

Design

- **Requirements Specification \Rightarrow**

- **Design**

- develop architectural design (system structure)
 - decompose software into modules with module interfaces
 - develop detailed design (module specifications)
 - select algorithms and data structures
 - maintain record of design decisions

- **Difficulties**

- miscommunication between module designers
 - design may be inconsistent, incomplete, ambiguous

- **Verification**

- extensive design reviews (inspections) to determine that design conforms to requirements
 - check module interactions
 - develop integration test plan

Implementation and Integration

- **Design \Rightarrow Implementation**

- implement modules and verify they meet their specifications
- combine modules according to architectural design

- **Difficulties**

- module interaction errors
- order of integration has a critical influence on product quality

- **Verification and Testing**

- code reviews to determine that implementation conforms to requirements and design
- develop unit/module test plan: focus on individual module functionality
- develop integration test plan: focus on module interfaces
- develop system test plan: focus on requirements and determine whether product as a whole functions correctly

Operation and Maintenance

- **Operation \Rightarrow Change**

- maintain software after (and during) user operation
- determine whether product as a whole still functions correctly

- **Difficulties**

- design not extensible
- lack of up-to-date documentation
- personnel turnover

- **Verification and Testing**

- review to determine that change is made correctly and all documentation updated
- test to determine that change is correctly implemented
- test to determine that no inadvertent changes were made to compromise system functionality

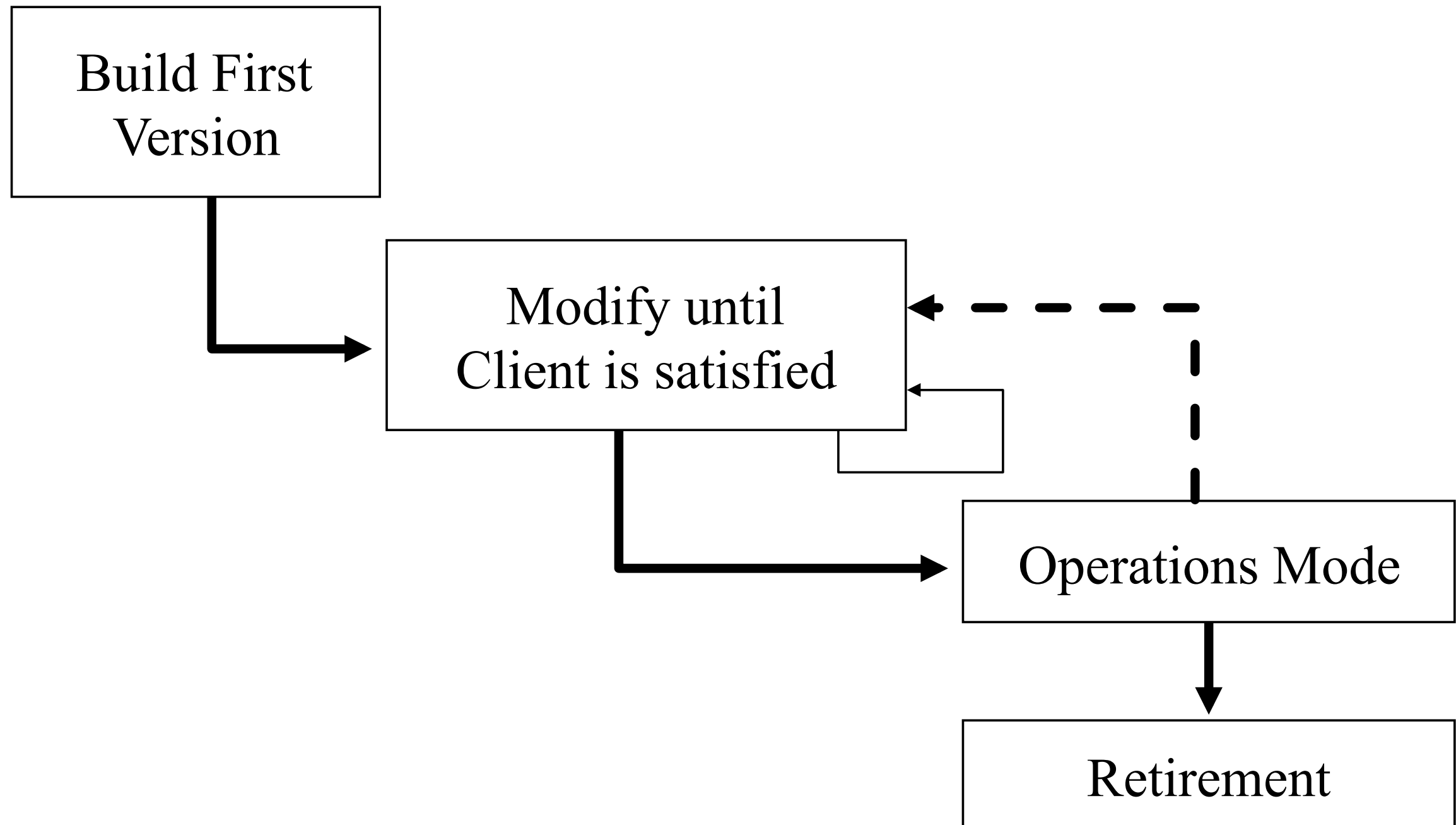
Discussion

- You will see the previous five activities appear in almost every software life cycle
- Within each of these major types of development activities, there will be
 - lots of different sub-activities
 - UI design, code reviews, refactoring, build management, configuration management, deployment, testing, profiling, debugging, etc.
 - meetings, e-mail, texting, IM, phone calls, etc. (i.e. **coordination**)
 - change requests, identification of problems, resolution of ambiguities, problem solving, etc.
 - “controlled chaos”

Example Life Cycles

- One Anti Life Cycle
 - “Code & Fix”
- Exemplars
 - Waterfall
 - Rapid Prototyping
 - Incremental
 - Spiral Model
 - Rational Unified Process

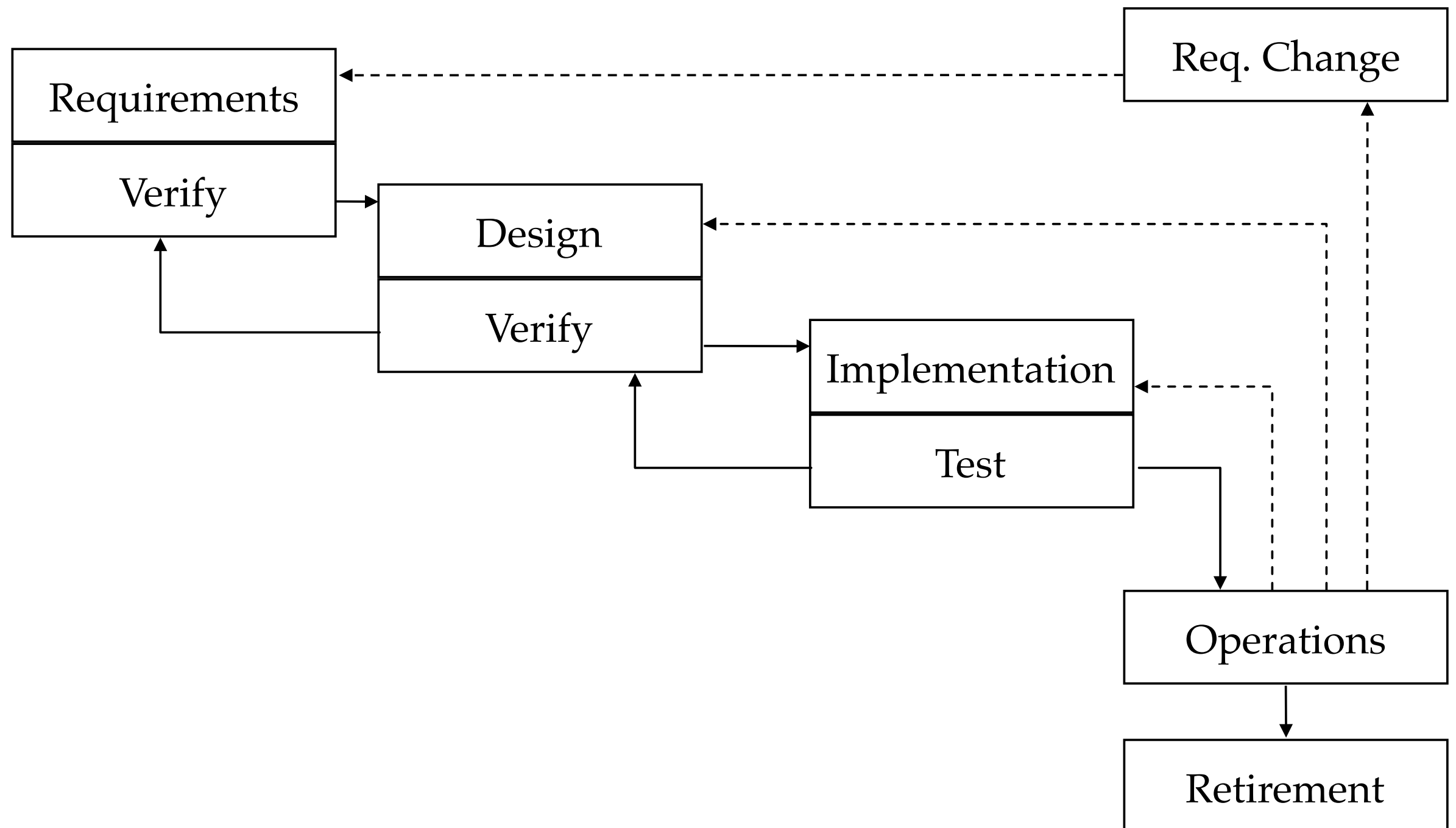
Code & Fix



Discussion

- Useful for small-scale, personal development
- Problems become apparent in any serious coding effort
 - No process for things like versioning, testing, change management, etc.
 - If you do any of these things, you are no longer doing “code and fix”
 - Difficult to coordinate activities of multiple programmers
 - Non-technical users cannot explain how the program should work
 - Programmers do not know or understand user needs

Waterfall



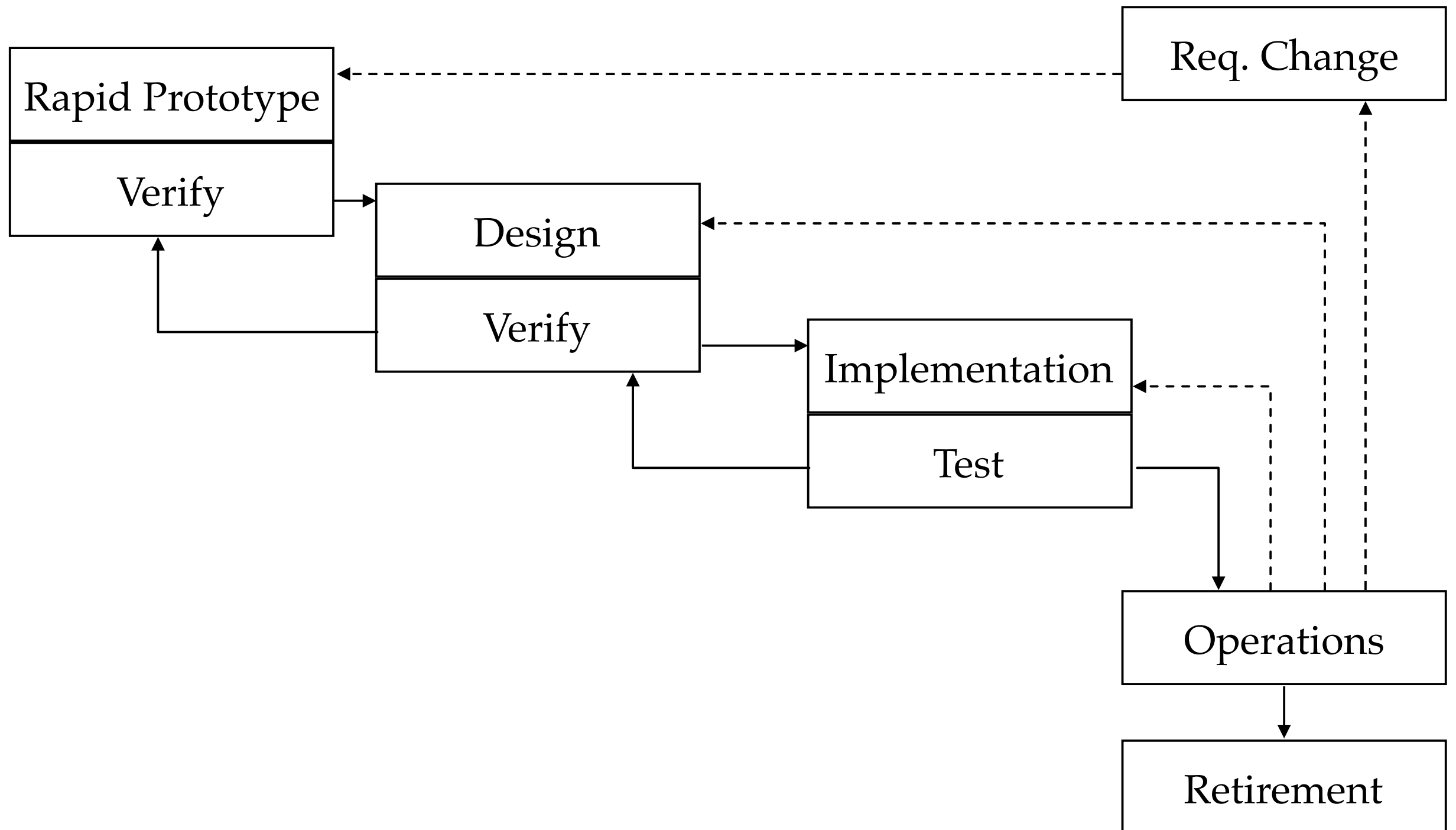
Discussion

- Proposed in early 70s by Winston Royce
 - as how **NOT** to run a software development project (!!!)
- Widely used (even today)
- Advantages
 - Straightforward to Measure
 - Possible to move between stages when the need occurs
 - Experience applying steps in past projects can be used in estimating duration of steps in future projects
 - Produces software artifacts that can be re-used in other projects

Discussion

- The original waterfall model had disadvantages because it disallowed iteration
 - This made the process inflexible and monolithic
 - Making estimates about how long the process would take was difficult
 - Did not deal well with changing requirements
 - Maintenance phase not handled well
- However, these are challenges that all life cycle models face
- The “waterfall with feedback” model was created in response
 - Slide 19 shows the “with feedback” version

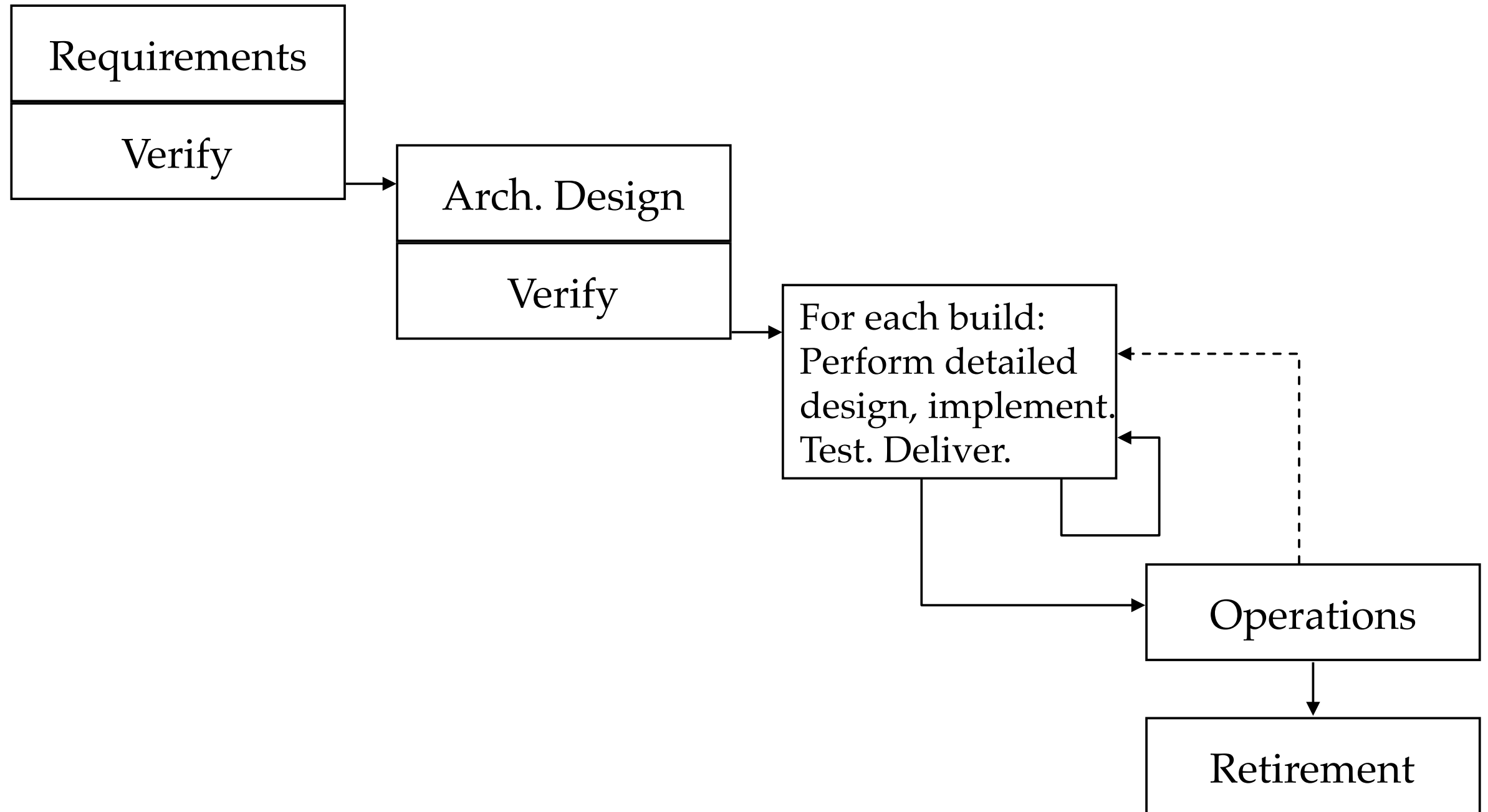
Rapid Prototyping



Discussion

- Prototypes are used to develop requirements specifications
 - Once reqs. are known, waterfall is used
- Prototypes are discarded once design begins
 - Prototypes should not be used as a basis for implementation. Prototyping tools do not create production quality code
- In addition, customer needs to be “educated” about prototypes
 - they need to know that prototypes are used just to answer requirements-related questions
 - otherwise, they get impatient!

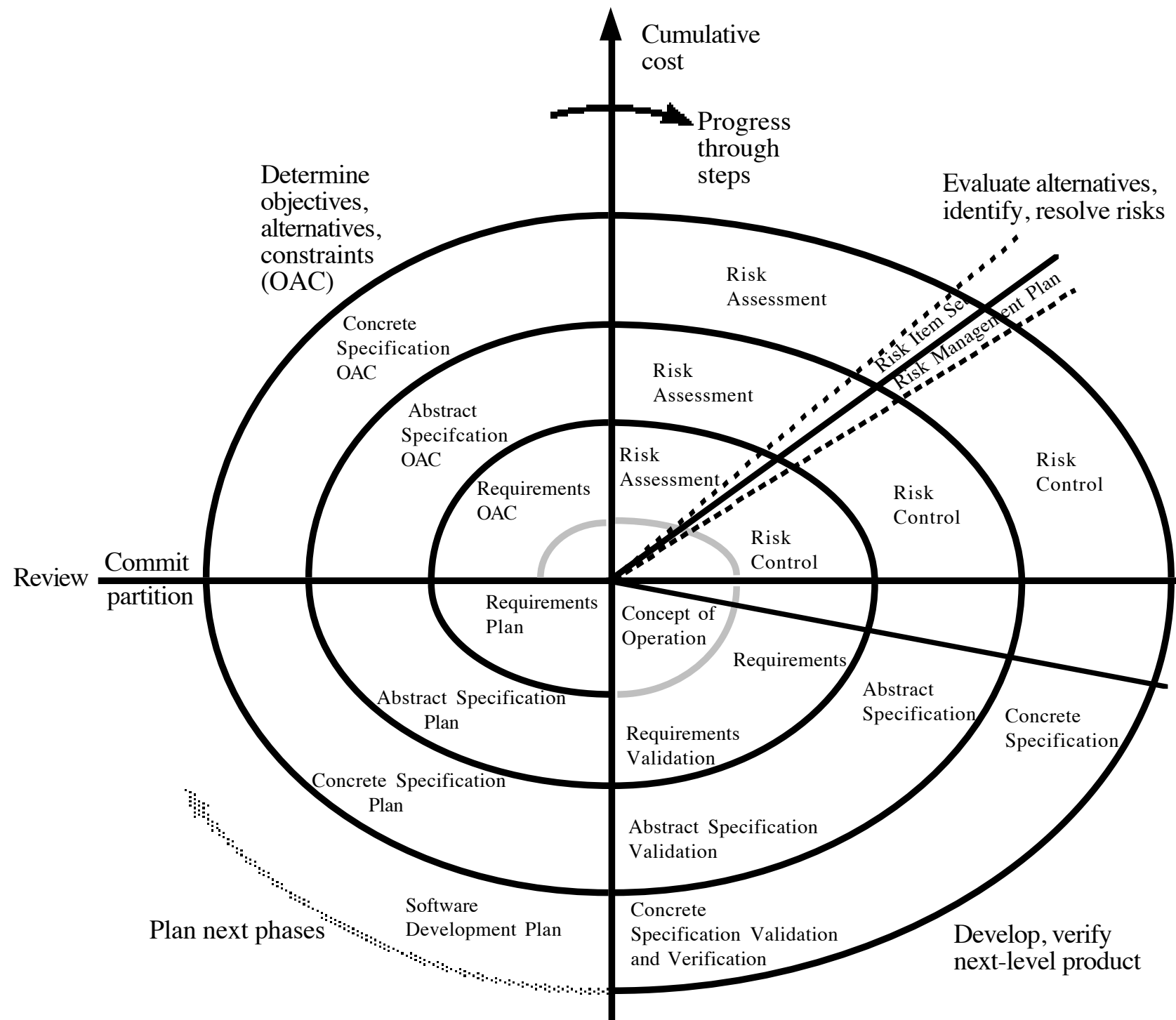
Incremental



Discussion

- Used by Microsoft (at least when building Windows XP)
 - Programs are built everyday by the build manager
 - If a programmer checks in code that “breaks the build” they become the new build manager!
- Iterations are planned according to features
 - e.g. features 1 and 2 are being worked on in iteration 1
 - features 3 and 4 are in iteration 2, etc.
- This life cycle also specifies two critical roles
 - product manager and program manager
 - Note: the original link is no longer active; fortunately I saved a copy

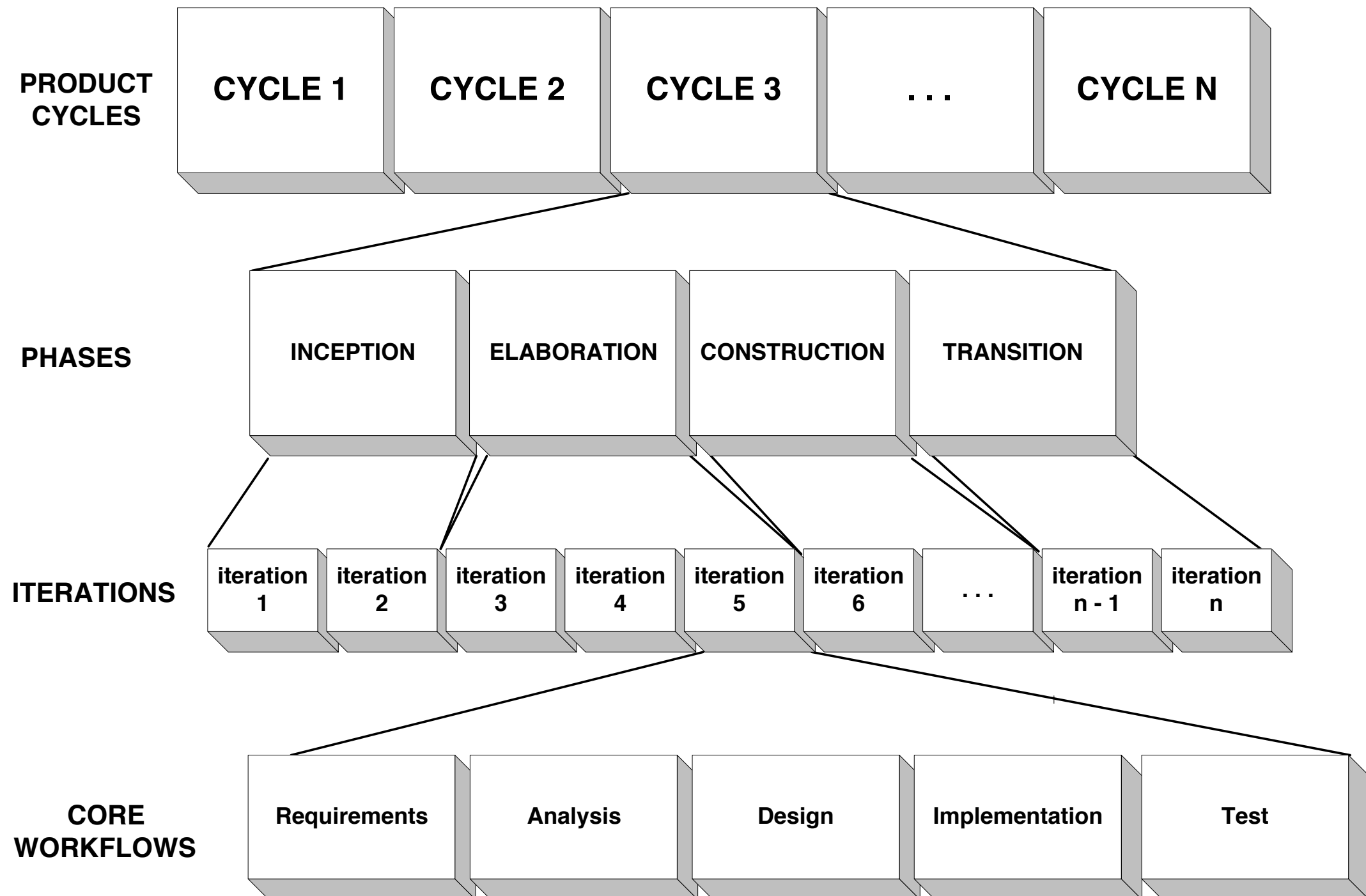
Spiral Model [Boehm, 1988]



Discussion

- Similar to Iterative Model, but:
 - each iteration is driven by “risk management”
 - Determine objectives and current status
 - Identify Risks
 - Develop plan to address highest risk items and proceed through iteration
- Repeat

Rational Unified Process



Discussion

- A variant of the waterfall model with all of the major steps
 - It advocates the use of object-oriented analysis and design techniques throughout
- Our “big three” concepts from Lecture 1 writ large
 - Specification: objects and classes used in all phases
 - Translation: objects and classes go from high level specs to extremely detailed specs that can be translated directly to code
 - some OO A&D tools will generate source code based on UML designs
 - Iteration: Product Cycles \Rightarrow Phase \Rightarrow Iterations \Rightarrow Major Life Cycle Steps
- A step towards agile in that the activities are “fractal”
 - You may find yourself performing implementation and testing during project inception

Summary

- Life cycles make software development
 - predictable, repeatable, measurable, and efficient
- High-quality processes should lead to high-quality products
 - at least it improves the odds of producing good software
- We've seen
 - Typical stages in software life cycles
 - Examples of traditional software life cycles

Coming Up Next

- Lecture 7: Introduction to Agile Life Cycles