Software Life Cycles

Kenneth M. Anderson
Foundations of Software Engineering
CSCI 5828 - Spring Semester, 2001
(Guest Lecture)

Today's Lecture

- Discuss Software Life Cycles
  - Why do we need them?
  - What types exist?
    - Code and Fix (hacking)
    - Waterfall
    - Iterative
    - Rapid Prototype
    - Spiral
  - Advantages and Disadvantages

Background

- In Software Engineering:

  "Process is King"
  - We want our activities to be coordinated and planned, e.g. "engineered"
  - The reason? A high quality process should increase our ability to create a high quality product

Use of Process

- Car Assembly
  - An assembly line is a process for producing cars.
  - A significant amount of work goes into not just designing a car but into designing the process used to build that car
- Software Engineering
  - The same principles can be applied to developing a software system

Key Difference

- There is a key difference between software engineering and car assembly, however.
- In car assembly, design time for the car is “short”, the majority of the work lies in manufacturing
  - In software engineering, we face the reverse situation, creating new copies of a software system is trivial, it’s the design that is hard
  - Thus, there will be significant differences in the processes used to develop software

Software Life Cycle

- A series of steps that organizes the development of a software product
- Duration can be from days to years
- Consists of
  - people!
  - overall process
  - intermediate products
  - stages of the process

Phases of a Software Life Cycle

- Standard Phases
  - Requirements Analysis & Specification
  - Design
  - Implementation and Integration
  - Operation and Maintenance
  - Change in Requirements
  - Testing throughout!
- Phases promote manageability and provide organization

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
  - check for feasibility, testability
  - develop system/acceptance test plan

Design

- Requirements Specification —> 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

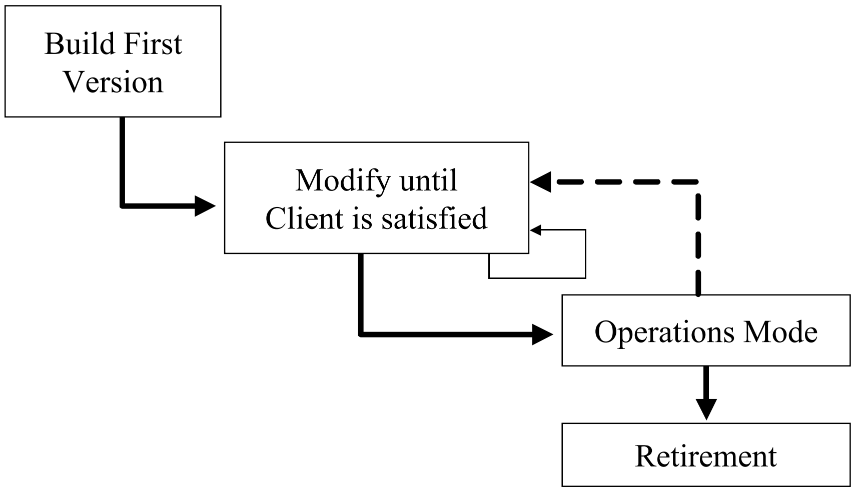
Implementation and Integration

- Design —> 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 —> 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

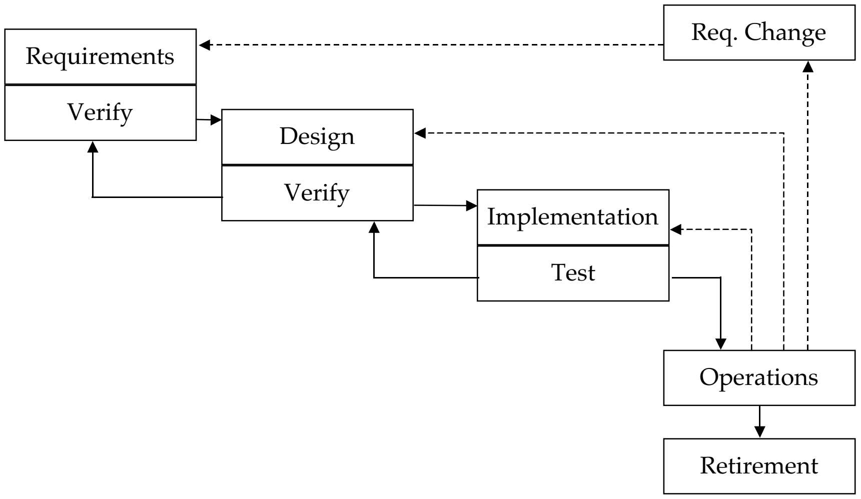
Code-and-Fix (Not a Life Cycle!)



Discussion of Code-and-Fix

- Useful for “hacking”
- Problems become apparent in any serious coding effort
  - No process for things like versioning, configuration management, testing, etc.
  - 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 Model



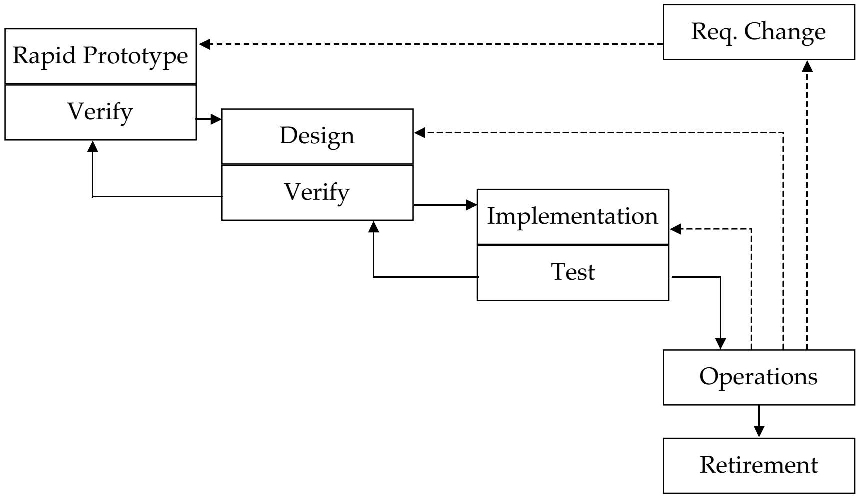
Discussion of Waterfall

- Proposed in early 70s
- Widely used (even today)
- Advantages
  - Measurable Progress
  - 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

Waterfall, continued

- The original waterfall model had disadvantages because it disallowed iteration
  - Inflexability
  - Monolithic
  - Estimation is difficult
  - Requirements change over time
  - Maintenance not handled well
- These are problems with other life cycle models as well
- The “waterfall with feedback” model was created in response
  - Our slides show this model

Rapid Prototyping



Req. Change

Rapid Prototype

Verify

Design

Verify

Implementation

Test

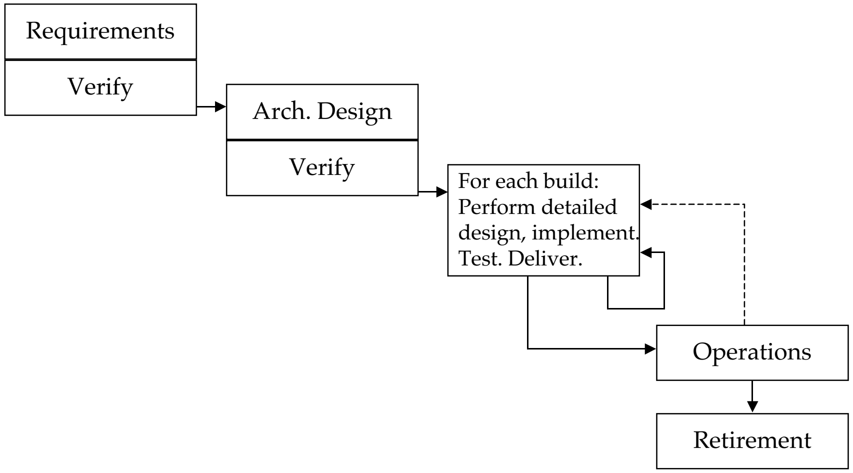
Operations

Retirement

Discussion of Rapid Prototyping

- Prototypes are used to develop reqs. spec.
- 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



Requirements

Verify

Arch. Design

Verify

For each build:
Perform detailed
design, implement.
Test. Deliver.

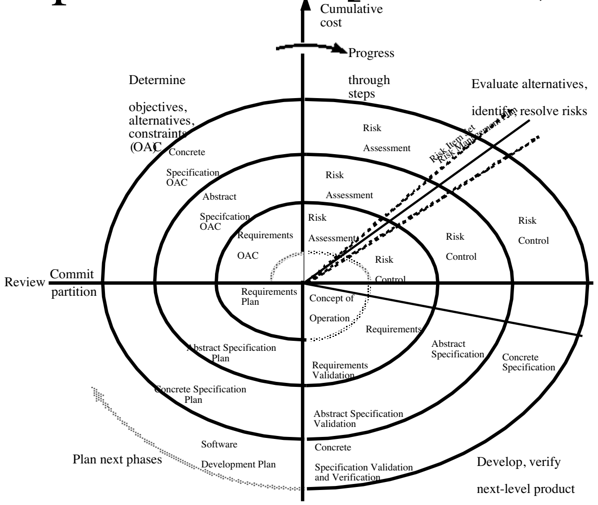
Operations

Retirement

Discussion of Incremental Model

- Used by Microsoft
  - 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 classified according to features
    - e.g. features 1 and 2 are being worked on in this iteration, features 3 and 4 are next

The Spiral Model [Boehm,1988]



Discussion of Spiral Model

- Similar to Iterative Model, but:
  - – each iteration is driven by “risk management”
    - Determine objectives and current status
    - Identify Risks
    - Next iteration addresses highest risk items
    - Repeat

Summary

- Life cycles make software development
  - – predictable
  - – repeatable
  - – measurable
  - – efficient
- High-quality processes should lead to high-quality products
  - – at least it improves the odds of producing good software