Lecture 3: Software Life Cycles

Kenneth M. Anderson
Foundations of Software Engineering
CSCI 5828 - Spring Semester, 2000

Today's Lecture

- Briefly Review Software Life Cycles
- Discuss problems associated with them

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

Software Artifacts

- Intermediate Software Products
  - Demarcate end of phases
  - Enable effective reviews
  - Specify requirements for next phase
- Form
  - Rigorous
  - Machine processible (highly desirable)
- Content
  - Specifications, Tests, Documentation

Example Artifacts

- Options Document
  - Problem Definition
  - Potential Solutions
  - Proposed System
- Cost-Benefit Analysis
  - Benefits
    - Achievable Goals
  - Costs
    - Development & Maint.
  - Analysis
    - Net improvement
- Requirements
  - Boilerplate
  - Project scope
  - Project history
  - Current System
  - New System
  - Requirements
- Preliminary Plan
  - Statement of Work
    Mgmt, Docs, Testing Plans
  - Schedules

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

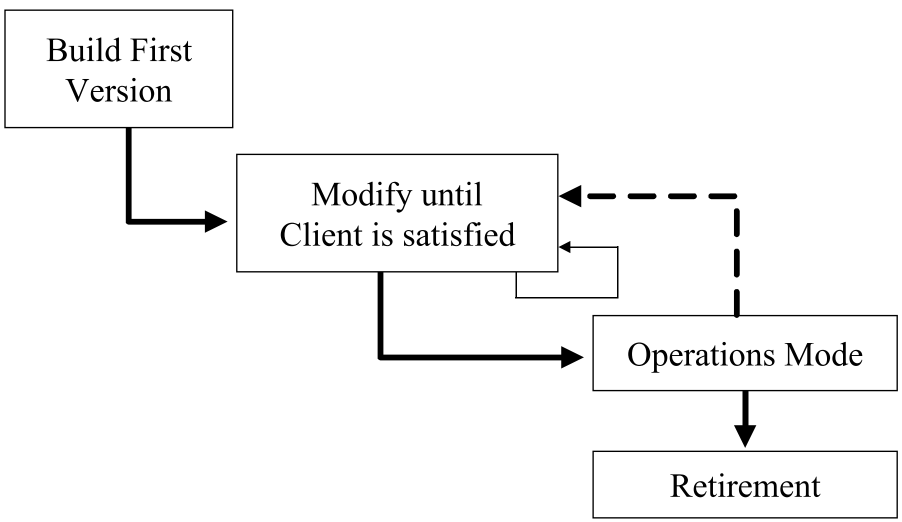
Implementation and Integration

- 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

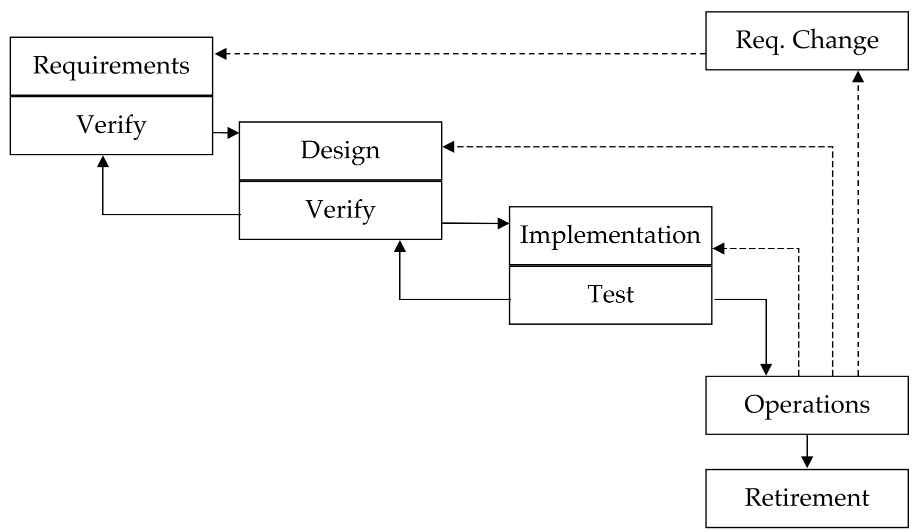
Operation and Maintenance

- 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 (check that no affected software has regressed)

Build-and-Fix



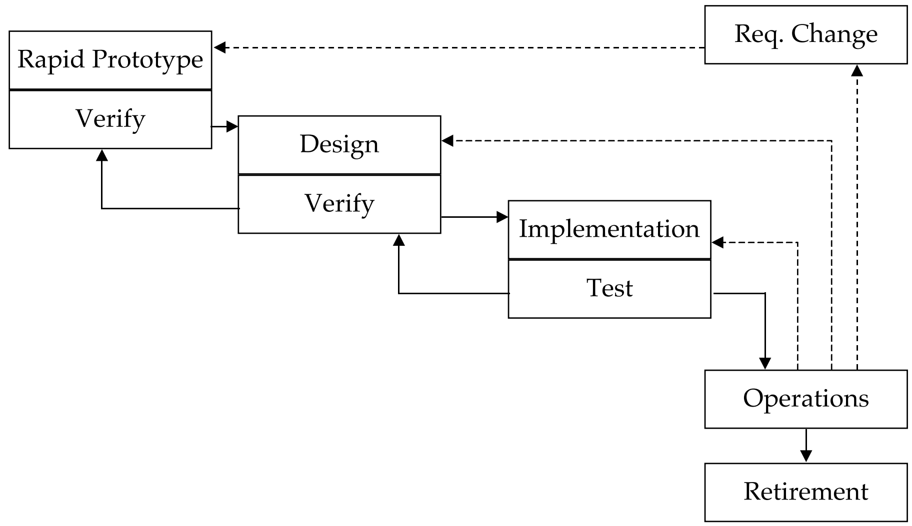
Waterfall Model



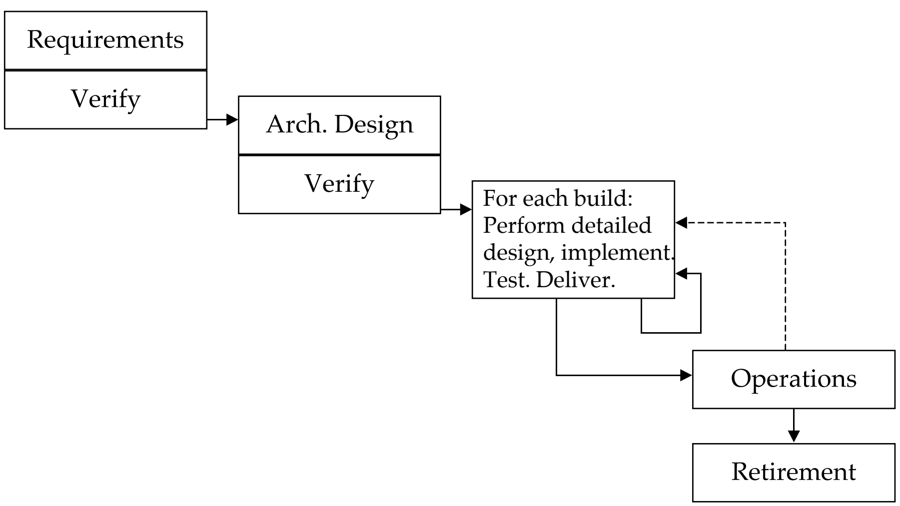
Two views on Waterfall

- Business Systems
  - Enterprise initiatives lead to feasibility studies
    - This starts the waterfall in motion
- Engineering Applications
  - Waterfall starts much later in the process
    - Software may not be considered until after concept exploration and experimental prototyping of global engineering system

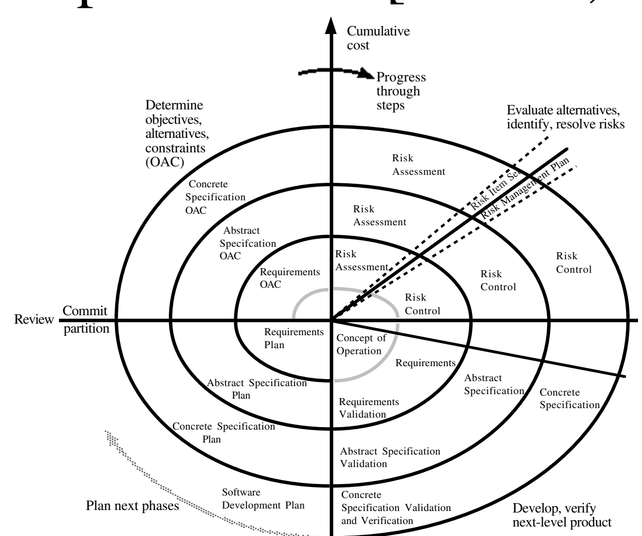
Rapid Prototyping



Incremental



The Spiral Model [Boehm,1988]



Object-Oriented Life Cycles

- Obtain customer requirements for the OO System
  - Identify scenarios or use cases
  - Build a requirements model
- Select classes and objects using basic requirements
- Identify attributes and operations for each object
- Define structures and hierarchies that organize classes
- Build an object-relationship model
- Build an object-behavior model
- Review the OO analysis model against use cases

Life Cycle Problems

- The user's view of software development
  - The waterfall is not "real" to them
- Consider Construction of a House
  - Decisions are visible
    - The lot
    - The position of the house on the lot
    - Landscaping
    - Pouring the Foundation

Constructing a House, continued

- As each decision is made, the "user" can see its effects
  - Its easy to see that making a change to the position of the house on the lot is expensive after the foundation is poured
- Its harder to determine what events in a software life cycle "casts things in concrete!"

Software-based Example

```
if (employee_age > 60) then
  …
end if;
```

Imagine the implications if the actual retirement age changed to 59.5

- how many instances of the "magic number" 60?
- floating point package?
- tax implications?

Consequences of the Change

- Integer to Rational
  - Or to stay with integers
    - change all values to months (round up or down?)
- Was “60” used for other purposes?
  - If so, you must ensure that the code isn’t intertwined
- Update all requirements documents, design documents, specifications, etc.

Life Cycle Problems

- Requirements are incomplete
- Waterfall is expensive
- It takes too long
- Too many variations
- Communications Gap
- Assumes “What” can be separated from “How”
- Error Management

Requirements are Incomplete

- Boehm reports that incomplete requirements cause downstream costs to increase exponentially!
- Issues
  - Computerization affects Environment
  - “Report Effect”
  - Lack of Visibility
  - People are not used to attaining completeness
    - Consider the construction of an airplane
      - Many details are covered by standards…

It costs too much!

- The waterfall was introduced when
  - computer time was more expensive than person time
    - forced extensive desk planning
    - use of time and space optimized
- Now, computer time is extremely cheap
  - but our methods haven’t changed (at least not much)!
- The management of artifacts as the life cycle progresses requires more and more resources
  - New methods must focus on this information management task

It takes too long!

- Example Waterfall (> 400 important entities)
  - 114 major tasks over 87 different organizations
  - 39 deliverables requiring 164 authorizations
- All of this allows people to "talk" about the project rather than "doing" the project!
- Inevitably, a project taking too long, gets cut short
  - results in incomplete or non-functional system

It takes too long! (continued)

- What to do?
  - Experience will help
  - CMM-like methods will increase the organization's ability to predict schedules
  - Rules needed when project is shortened
    - What requirements are removed?
    - How is the system's functionality scaled back?

Too many variations!

- Key problems
  - communication between practitioners
    - each builds large systems but use
      - different vocabulary
      - different steps
      - different deliverables
  - Difficult to assess life cycle critically
    - Problems are shared by all; but without common understanding how are root causes found?

End-User Communications Gap

"What we understand to be the conventional life cycle approach might be compared with a supermarket at which the customer is forced to provide a complete order to a stock clerk at the door of the store with no opportunity to roam the aisles–comparing prices, remembering items not on the shopping list, or getting a headache and deciding to go out for dinner…"

[McCracken and Jackson, 1982]

Communications Gap, continued

- User involvement throughout the life cycle
  - Participatory Design, HCI, and CSCW fields
- Watch out for communications gap within a development team!
  - Horizontal Team Integration considered bad
    - Tends to be little review; no chance for self-correction
  - Vertical Teams better; maintenance still a problem

"What vs. How"

- Life cycles assume: a problem description can be separated from a problem solution
- Humans do not typically behave this way!
  - People like to consider a range of solutions
    - What are the trade-offs?
    - A solution strategy may help clarify the problem
  - How do we integrate "normal" human behavior into modern life cycles?

Error Management

- It is impossible to predict all of the errors that a software system must handle
- Thus, a module's initial design is very likely to be incomplete!
  - Some errors may exist only because of a particular implementation strategy
  - if so, an implementation choice may then impact the interface of the module (which is typically set during design)