

Software Project Management

Lecture 11

Today

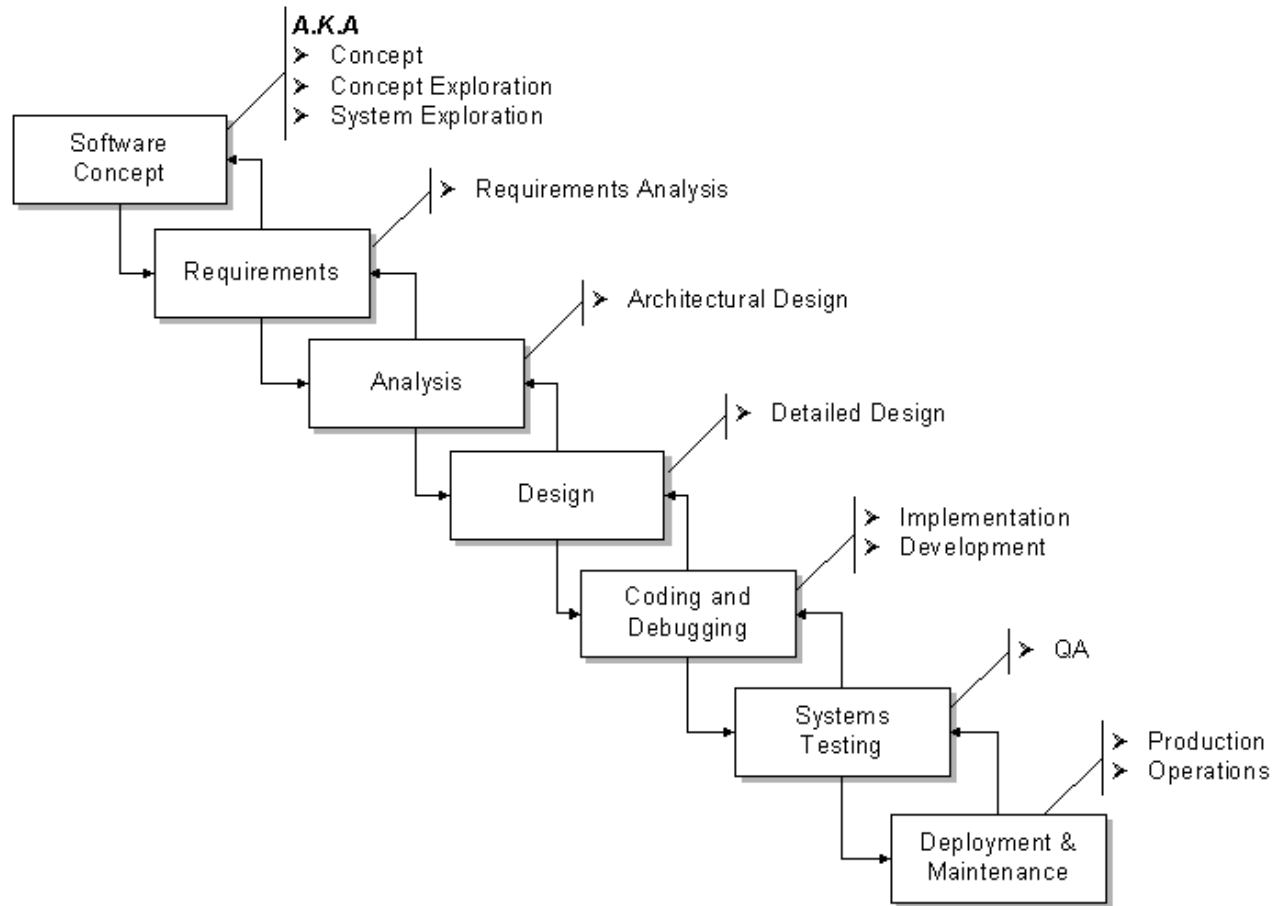
- 1. Phases in Detail
 - Step-by-step of typical software project
- 2. Lifecycle Planning
- 3. Project plans

- Next Week: Lots of Project-ish Details: WBS, PERT, CPM, Scheduling & Estimation

Lecture 10 & 11 Review

- PMI Fundamentals
- PMI Processes
- Project Organization
 - Functional, Project, Matrix Orgs.
- Initial documents
 - Statement of Work (SOW)
 - Project Charter
- Readings

Project Phases



Time Allocation by Phase

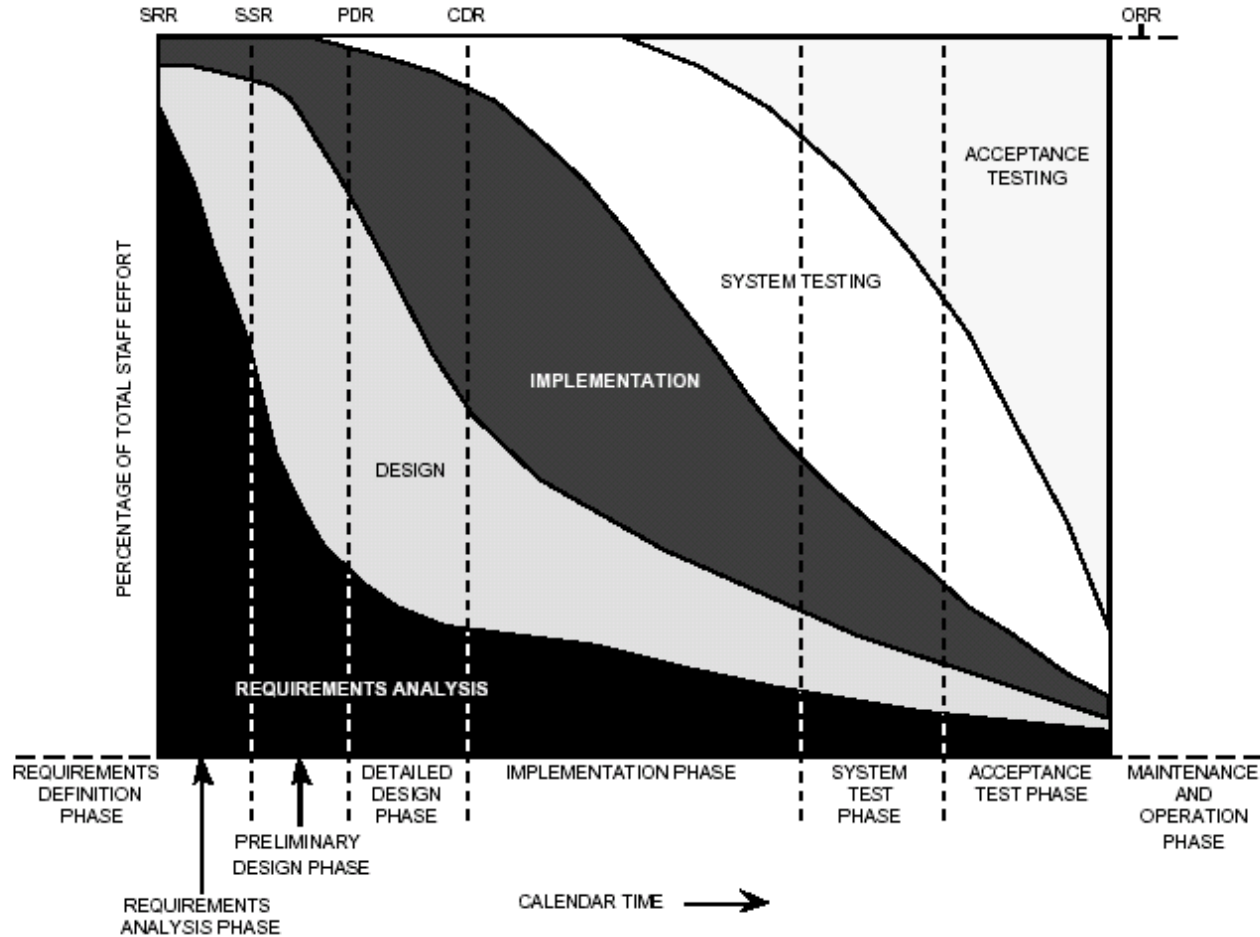
- Remember the 40-20-40 Rule
 - Specification-Implementation-Test

	Planning	Code & Unit Test	Integration & Test
Commercial DP	25%	40%	35%
Internet Systems	55%	15%	30%
Real-time Systems	35%	25%	40%
Defense Systems	40%	20%	40%

Time Allocation by Phase

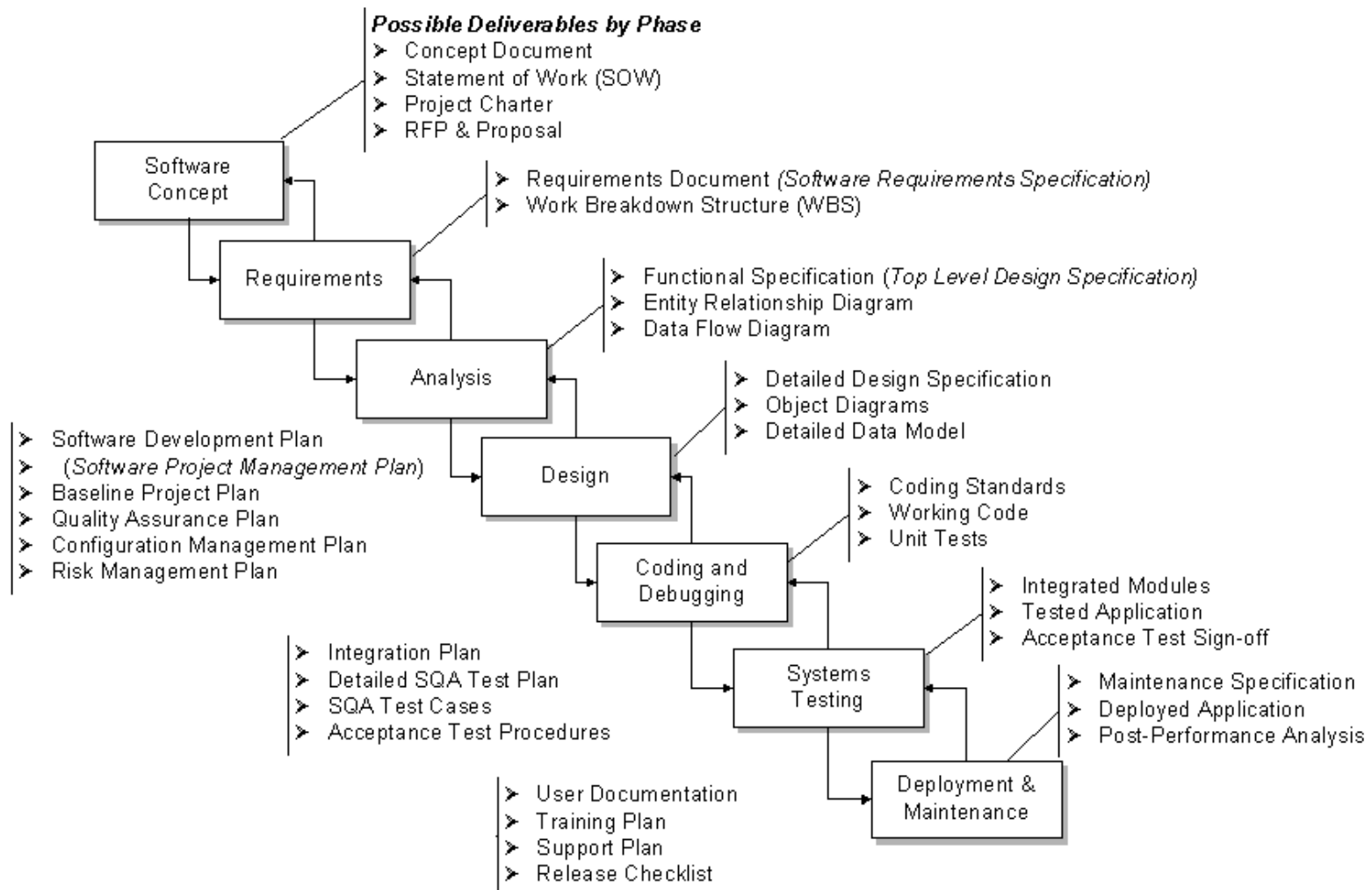
Activity	Small Project (2.5K LOC)	Large Project (500K LOC)
Analysis	10%	30%
Design	20%	20%
Code	25%	10%
Unit Test	20%	5%
Integration	15%	20%
System test	10%	15%

Activities by % of Total Effort



NASA's "Manager's Handbook for Software Development"

Potential Deliverables by Phase



Concept Exploration

- The “Why” phase
- Not a “mandatory formal” phase
 - Sometimes called the “pre-project” phase
- Collecting project ideas
 - Then the “funneling” process
- Project Justification
 - ROI
 - Cost-benefit analysis
 - Project Portfolio Matrix
- Initial planning and estimates

Concept Exploration

- Possibly includes Procurement Management:
 - RFP Process
 - Vendor selection
 - Contract management
- Gathering the initial team
 - Including PM if not already on-board
- Identify the project sponsor
 - Primary contact for approval and decision making
- Potential Phase Outputs:
 - Concept Document, Product Description, Proposal, SOW, Project Charter

Concept Exploration

- Characteristics & Issues
 - Lack of full commitment and leadership
 - Some frustrations:
 - Management only getting rough estimates from development
 - Development not getting enough specifics from customer
 - Finding a balanced team
 - Budget sign-off may be your 1st major task
 - Achieved via:
 - Good concept document or equivalent
 - Demonstration of clear need (justification)
 - Initial estimates

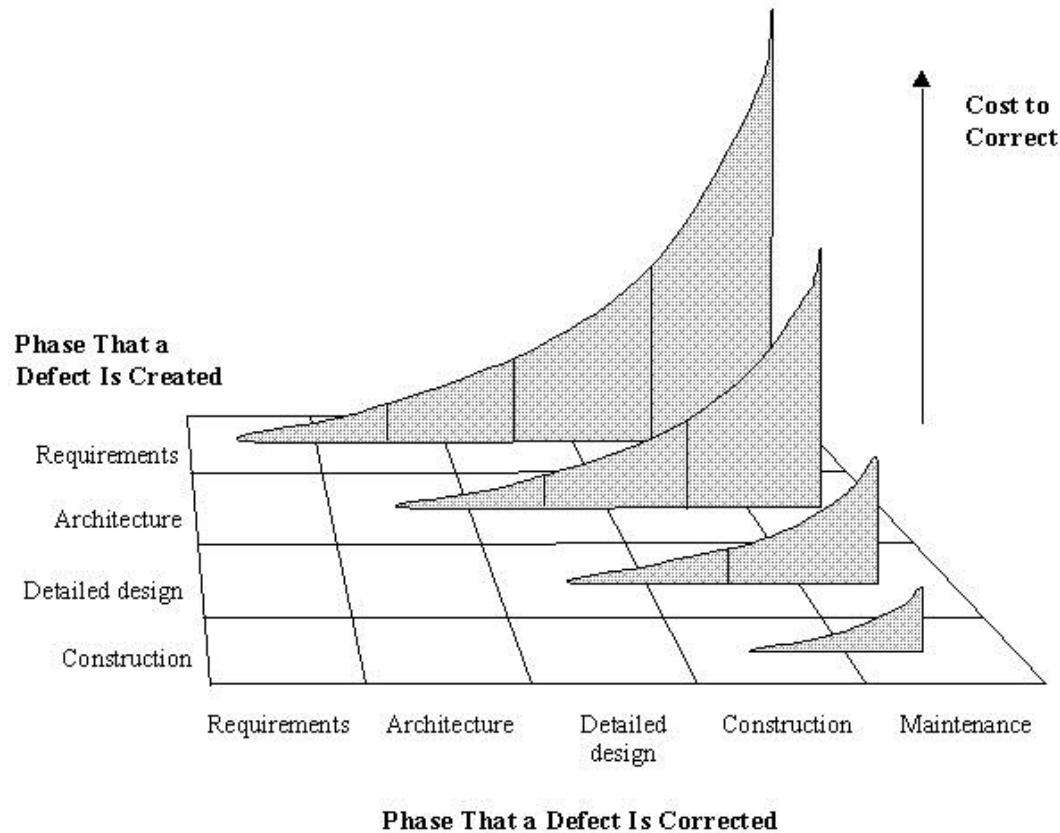
Requirements

- The “What” phase
- Inputs: SOW, Proposal
- Outputs:
 - Requirements Document (RD)
 - a.k.a. Requirements Specification Document (RSD)
 - Software Requirements Specification (SRS)
 - 1st Project Baseline
 - Software Project Management Plan (SPMP)
 - Requirements Approval & Sign-Off
 - Your most difficult task in this phase

Requirements

- Perhaps most important & difficult phase
- Shortchanging it is a ‘classic mistake’
- Can begin with a Project Kickoff Meeting
- Can end with a Software Requirements Review (SRR)
 - For Sponsor and/or customer(s) approval

Why are Requirements so Important?



Copyright 1998 Steven C. McConnell. Reprinted with permission from *Software Project Survival Guide* (Microsoft Press, 1998).

Requirements

- Characteristics & Issues
 - Conflict of interest: developer vs. customer
 - Potential tug-of-war:
 - Disagreement on Features & Estimates
 - Especially in fixed-price contracts
 - Frequent requirements changes
 - Achieving sign-off
- Project planning occurs in parallel

Requirements

- Requirements are capabilities and condition to which the system – more broadly, the project – must conform

2 Types of Requirements

- **Functional** (behavioral)
 - Features and capabilities
- **Non-functional** (a.k.a. “technical”) (everything else)
 - Usability
 - » Human factors, help, documentation
 - Reliability
 - » Failure rates, recoverability, availability
 - Performance
 - » Response times, throughput, resource usage
 - Supportability
 - » Maintainability, internationalization
 - Operations: systems management, installation
 - Interface: integration with other systems
 - Other: legal, packaging, hardware

Requirements

- Other ways of categorizing
 - Go-Ahead vs. Catch-up
 - Relative to competition
 - Backward-looking vs. Forward-looking
 - Backward: address issues with previous version
 - Forward: Anticipating future needs of customers
- Must be prioritized
 - Must-have
 - Should-have
 - Could-have (Nice-to-have: NTH)
- Must be approved

Early Phase Meetings

- Project Kickoff Meeting
- Project Brainstorming Meeting
 - Clarify goals, scope, assumptions
 - Refine estimates
- WBS Meeting

Analysis & Design

- The “How” Phases
- Inputs: Requirements Document
- Outputs:
 - Functional Specification
 - Detailed Design Document
 - User Interface Specification
 - Data Model
 - Prototype (can also be done with requirements)
 - Updated Plan (improved estimates; new baseline)

Analysis & Design

- a.k.a. Top-level design & detailed design
- Continues process from RD
- Ends with Critical Design Review (CDR)
 - Formal sign-off
 - Can also include earlier Preliminary Design Review (PDR) for high level design

Analysis & Design

- Characteristics & Issues
 - Enthusiasm via momentum
 - Team structure and assignments finalized
 - Delays due to requirements changes, new information or late ideas
 - Issues around personnel responsibilities
 - Unfeasible requirements (technical complexity)
 - Resource Issues
 - Including inter-project contention

Development

- The “Do It” phase
- Coding & Unit testing
- Often overlaps Design & Integration phases
 - To shorten the overall schedule
 - PM needs to coordinate this

Development

- Other concurrent activities
 - Design completion
 - Integration begins
 - Unit testing of individual components
 - Test bed setup (environment and tools)
 - Project plans updated
 - Scope and Risk Management conducted

Development

- Characteristics
 - Pressure increases
 - Staffing at highest levels
 - Often a “heads-down” operation
- Issues
 - Last-minute changes
 - Team coordination (esp. in large projects)
 - Communication overhead
 - Management of sub-contractors

Integration & Test

- Evolves from Dev. Phase
- Often done as 2 parallel phases
 - Partial integration & initial test
- Starts with integration of modules
- An initial, incomplete version constructed
- Progressively add more components

Integration & Test

- Integration primarily a programmer task
- Test primarily a QA team task
- Integration:
 - Top-down: Core functionality first, empty shells for incomplete routines (stubs)
 - Bottom up: gradually bind low-level modules
 - Prefer top-down generally

Integration & Test

- Tests
 - Integration testing
 - Black & White-box testing
 - Load & Stress testing
 - Alpha & Beta testing
 - Acceptance testing
- Other activities
 - Final budgeting; risk mgmt.; training; installation preparation; team reduced

Integration & Test

- Characteristics & Issues
 - Increased pressure
 - Overtime
 - Customer conflicts over features
 - Frustration over last-minute failures
 - Budget overruns
 - Motivation problems (such as burnout)
 - Difficulty in customer acceptance
 - Esp. true for fixed-price contracts

Deployment & Maintenance

- Installation depends on system type
 - Web-based, CD-ROM, in-house, etc.
- Migration strategy
- How to get customers up on the system
 - Parallel operation
- Deployment typically in your project plan, maintenance not

Deployment & Maintenance

- Maintenance
 - Fix defects
 - Add new features
 - Improve performance
- Configuration control is very important here
- Documents need to be maintained also
- Sometimes a single team maintains multiple products

Deployment & Maintenance

- Characteristics & Issues
 - Lack of enthusiasm
 - Pressure for quick fixes
 - Insufficient budget
 - Too many patches
 - Personnel turnover
 - Regression testing is critical
 - Preferably through automated tools

Lifecycle Planning

- a.k.a. Lifecycle Management or SDLC
- Greatly influences your chance of success
- Not choosing a lifecycle is a bad option
- Three primary lifecycle model components
 - Phases and their order
 - Intermediate products of each phase
 - Reviews used in each phase

Lifecycle Planning

- Different projects require different approaches
- You do not need to know all models by name
- You should know how that if given a certain scenario what sort of SDLC would be appropriate
- There are more than covered here
- A lifecycle is not a design, modeling or diagramming technique
 - The same technique (UML, DFD, etc) can be used with multiple lifecycles

Pure Waterfall

- The “granddaddy” of models
- Linear sequence of phases
 - “Pure” model: no phases overlap
- Document driven
- All planning done up-front

Waterfall Risk

- Why does the waterfall model “invite risk”?
- Integration and testing occur at the end
 - Often anyone’s 1st chance to “see” the program

Pure Waterfall

- Works well for projects with
 - Stable product definition
 - Well-understood technologies
 - Quality constraints stronger than cost & schedule
 - Technically weak staff
 - Provides structure
 - Good for overseas projects

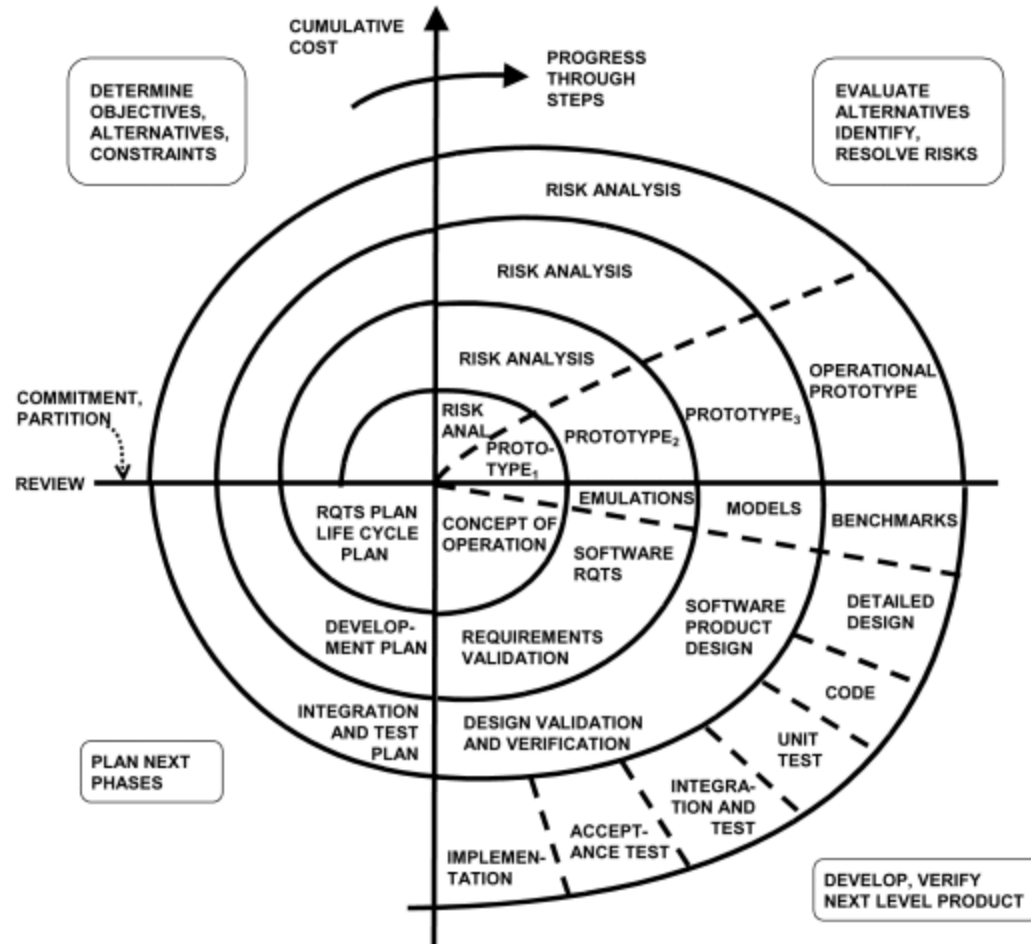
Pure Waterfall

- Disadvantages
 - Not flexible
 - Rigid march from start->finish
 - Difficult to fully define requirements up front
 - Can produce excessive documentation
 - Few visible signs of progress until the end

Code-and-Fix

- “Code-like-Hell”
- Specification (maybe), Code (yes), Release (maybe)
- Advantages
 - No overhead
 - Requires little expertise
- Disadvantages
 - No process, quality control, etc.
 - Highly risky
- Suitable for prototypes or throwaways

Spiral



Spiral

- Emphasizes risk analysis & mgmt. in each phase
- A Series of Mini-projects
- Each addresses a set of “risks”
 - Start small, explore risks, prototype, plan, repeat
- Early iterations are “cheapest”
- Number of spirals is variable
 - Last set of steps are waterfall-like

Spiral

- Advantages
 - Can be combined with other models
 - As costs increase, risks decrease
 - Risk orientation provides early warning
- Disadvantages
 - More complex
 - Requires more management

Modified Waterfall – Sashimi

- Overlapping phases
- Advantages
 - Reduces overall schedule
 - Reduces documentation
 - Works well if personnel continuity
- Disadvantages
 - Milestones more ambiguous
 - Progress tracking more difficult
 - Communication can be more difficult

Questions?
