ICS 52: Introduction to Software Engineering

Fall Quarter 2001

Professor Richard N. Taylor

Lecture Notes

http://www.ics.uci.edu/~taylor/ics52_fq01/syllabus.html

University of California, Irvine

Add/Drop Policy

- Second week of classes
  - Deadline to add
- Second week of classes
  - Deadline to drop

Course Web Site

http://www.ics.uci.edu/~taylor/ics52_fq01/syllabus.html

- Contents
  - Information on the instructors
  - Overview and prerequisite knowledge
  - Textbooks
  - Schedule
  - Assignments and assessment
  - Teaching assistants
  - Keeping in touch
  - Computing
  - Academic dishonesty

Be Involved…But Don't Be Too Involved

- You cheat, you fail!
  - Final grade is "F", irrespective of partial grades
  - Project, midterm, final
- To avoid being a cheater
  - Always do your work by yourself
  - Do not borrow work
  - Do not lend work
    - » Do not put your work on the Web
- "Your TA is your friend, but your friend is not your TA"
  - Your friend's help may be cheating

Discussion Section

- Assignments: questions and answers
- Details of tools and methods

Positioning in ICS Curriculum

- ICS 121: Software methods and tools
  - Rigor and formality
  - Additional software design strategies
  - Additional analysis and testing strategies
  - Configuration management
- ICS 122: Software Specification and Quality Engineering
- ICS 123: Software Architectures, Distributed Systems, and Interoperability
- ICS 125
  - Management issues
  - Working in a team
  - A scaled-up project

A note on class attendance and the book…

- What I say in class takes precedence over what's in the slides and what's in the book
- What's in the slides takes precedence over what's in the book

Levels of Mastery

(See course website for definitive list)

- Competency
  - Software lifecycle
  - Requirements specification
  - Architectural design
  - Module design
  - (Programming)
  - Testing and quality assurance
- Literacy
  - SE principles
  - Alternative software architectures
  - Requirements engineering issues
- Familiarity
  - Configuration management
  - Concurrency
  - Software process alternatives

–"Scratching the surface of software engineering"

–" Fitting you to become an amateur software engineer"

Introduction

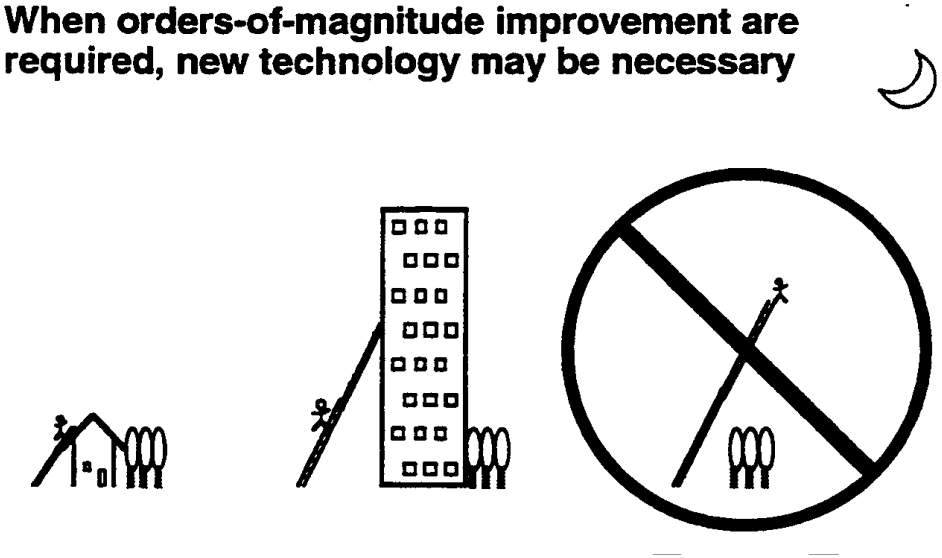
- Context
- Matters of scale
- Distribution of software costs
- Differences from programming
- Product and process
- Elements of Science, Engineering, Management, and Human Factors

Context

Small project	Huge project
You	Teams
Build what you want	Build what they want
One product	Family of products
Few sequential changes	Many parallel changes
Short-lived	Long-lived
Cheap	Costly
Small consequences	Large consequences

Programming Engineering

Matters of Scale



- High powered techniques not appropriate for all problems (Using an elephant gun to kill a fly)
- The ICS 52 pedagogical problem:
  - the problem must be small enough to complete in 10 weeks
  - you work on the project by yourself
  - you don't have to live with the consequences of your decisions
  - your customers are too reasonable

Distribution of Software Costs

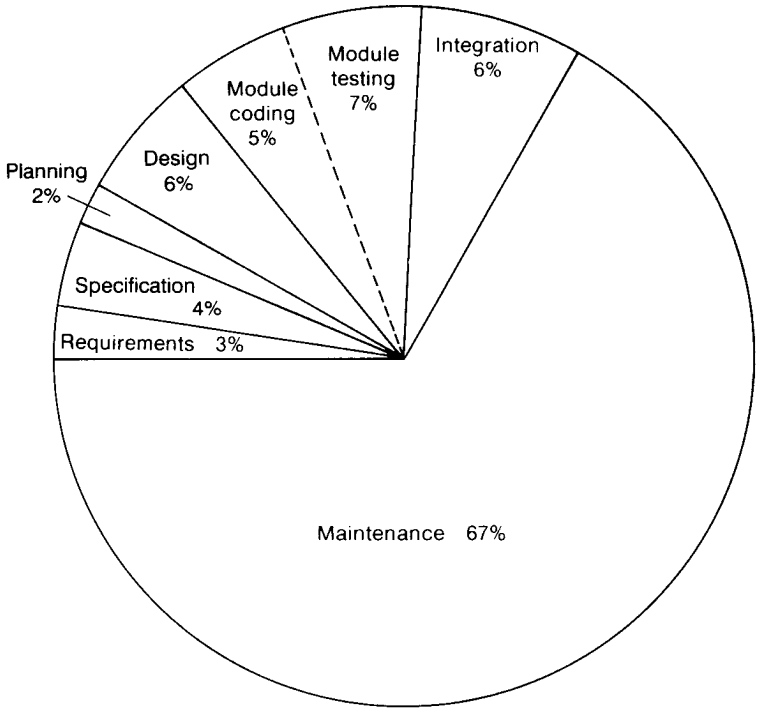


Figure 1.2 Approximate relative costs of the phases of the software process.

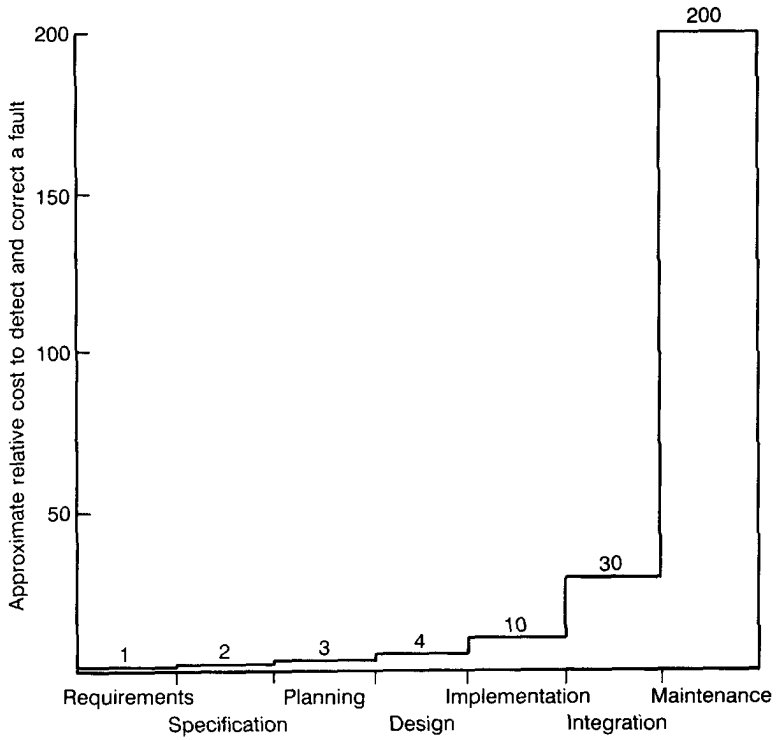


Figure 1.4 Points on solid line of Figure 1.3 plotted on a linear scale.

Differences from Programming

- Software engineering includes, e.g..:
  - determining **what** to build
  - organizing teams to cooperatively build systems;
  - analysis and testing
  - lifecycle system engineering
  - software architecture

Product and Process

- Which is the more important corporate asset: products or development processes?
  - Products: the only thing that brings in revenue
  - Process: the only thing you retain
    - The asset that distinguishes you from your competitor en route to a product
    - The asset that gets you to your next product
    - The asset that determines key properties of your products

Science, Engineering, Management, Human Factors

- Science: empirical studies; theories characterizing aggregate system behavior (e.g. reliability)
- Management: organizing teams, directing activities, correcting problems
- Human factors: user task understanding and modeling; ergonomics in user interface design
- Engineering: tradeoffs, canonical solutions to typical problems
  - Tradeoffs and representative qualities
    - Pick any two:
      - Good, fast, cheap
      - Scalability, functionality, performance

Software Engineering Principles

- Separation of concerns
  - Divide problem into parts that can be dealt with separately.
    - » example: automobile fuel flow systems from tires and drive train
    - » divide and conquer (horizontally)
  - Abstraction
    - » Divide problem into relevant parts and irrelevant details, and ignore the irrelevant parts (more important and less important, w.r.t. the current set of problem solving objectives)
    - » divide and conquer vertically
  - Modularity
    - » Separating a problem into parts that can be dealt with separately, using abstraction to determine "public" interfaces, dealing with the details as a private, internal matter.
- Compositionality
  - Allow two or more objects of a single kind to be composed such that the result is an object of that kind
  - "First-class citizens"

Software Development as a Problem Solving Activity

- Problem (application) characteristics
  - Ill-formed
  - Not completely specifiable?
  - Subject to constant change
- Learning from other disciplines
  - Architecture: Requirements, sketch, blueprints, construction
    - » Strengths:
      - Phasing of activities
      - User input and review
      - User looks at sketch, but only minimally involved in construction
    - » Weaknesses:
      - Lots of domain knowledge on the part of the consumer
      - We know what kind of change can be made at each stage
      - Progress easily measurable
  - Legislation: Commission, committee, congress, bureaucracy
    - » Strengths:
      - Intangible product
      - Unforeseen consequences
      - Difficult to measure progress
      - Laws get "patched"
      - Importance of careful reviews highlighted
    - » Weakness of analogy:
      - Difficult to test laws
      - Not a rigorous discipline

Software Processes

- Elements
  - Activities (“phases”)
  - Artifacts
    - » Can include process specifications
  - Resources
    - » People (their time and cost)
    - » Tools (their time and cost)
- Relationships between the elements
  - precedence, requires, produces, refines to
  - ...
- Constraints
  - Time
  - Cost
  - Qualities (repeatable process?)

Waterfall Approach

- Waterfall Model (Winston Royce)
  - Centered on defining documents that describe intermediate products
  - User feedback and changes accommodated as an afterthought

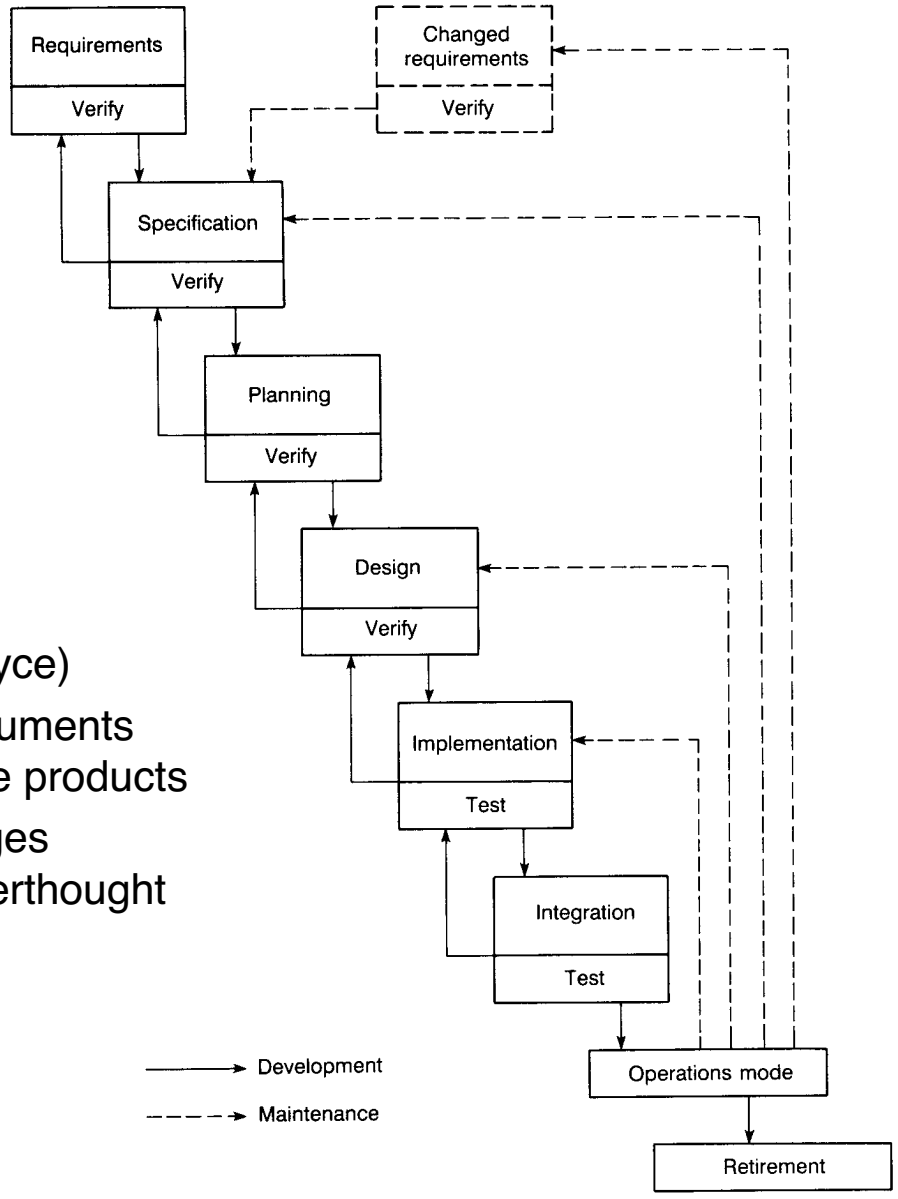
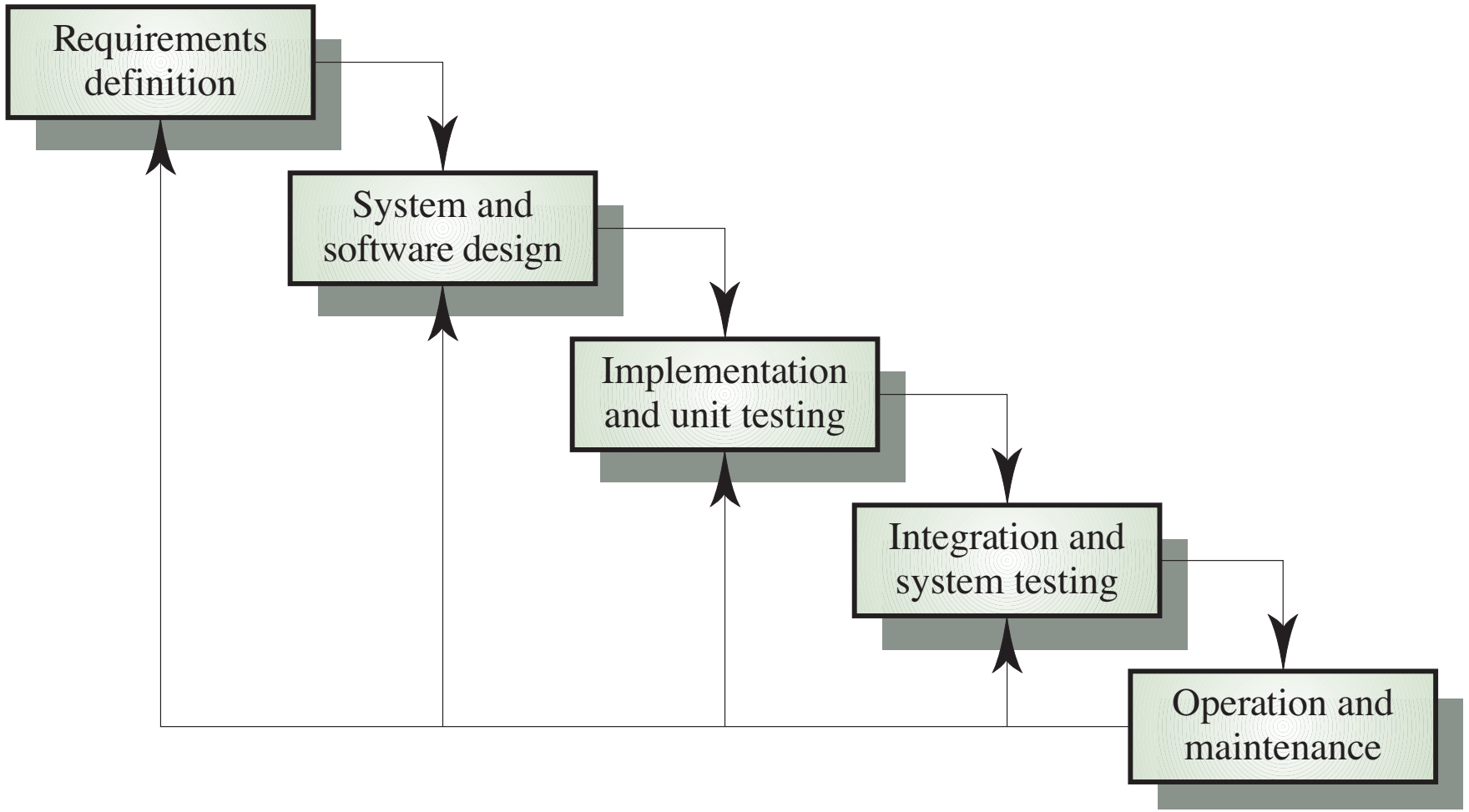


Figure 3.2 Waterfall model.

Source: Schach, ibid..

Waterfall model



Spiral Model

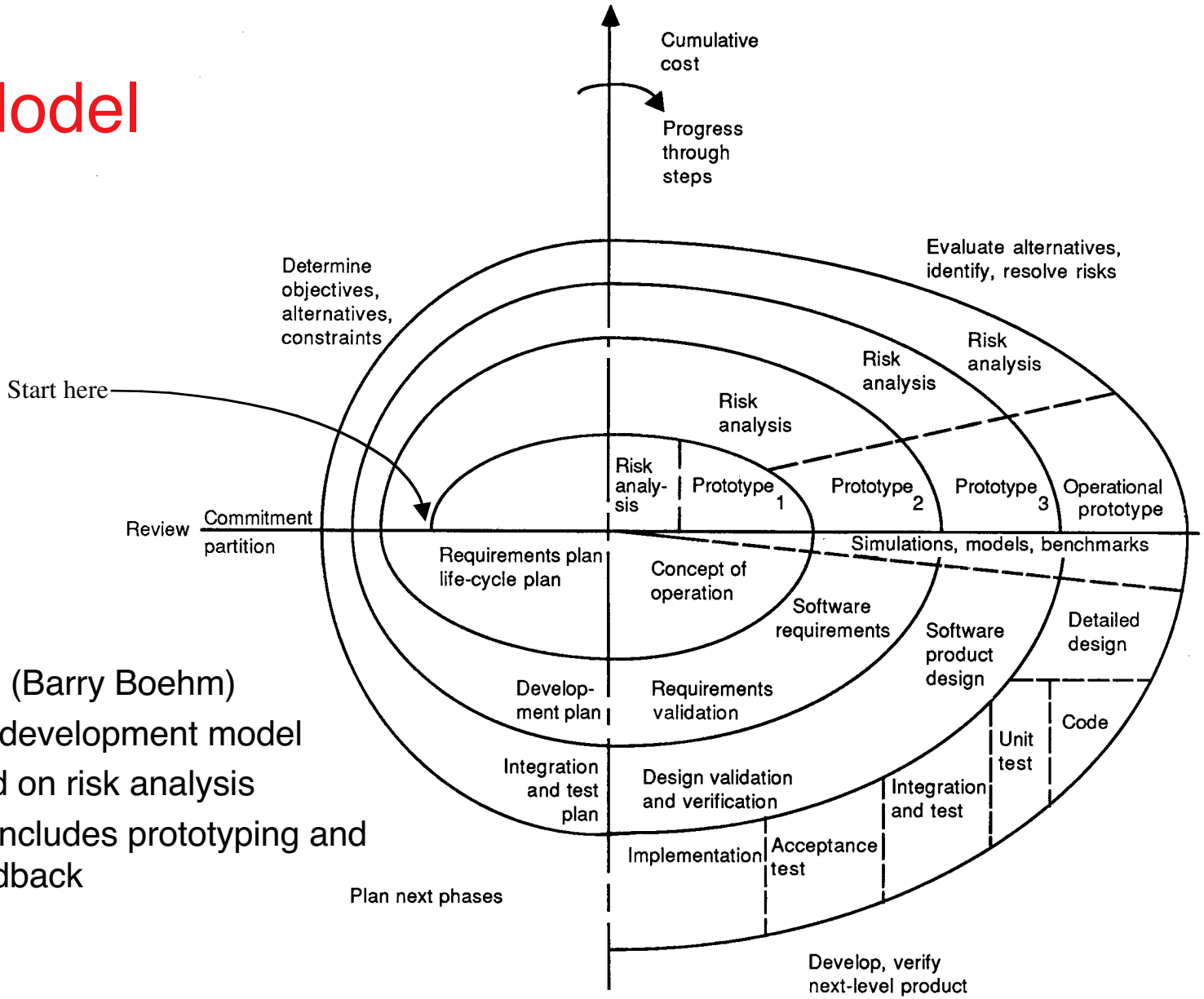


Figure 2. Spiral model of the software process.

- Spiral Model (Barry Boehm)
  - Iterative development model
  - Centered on risk analysis
  - Directly includes prototyping and user feedback

Source: Barry Boehm, "A Spiral Model of Software Development and Enhancement, IEEE Computer, May 1988

Software Risk Items

Table 4. A prioritized top-ten list of software risk items.

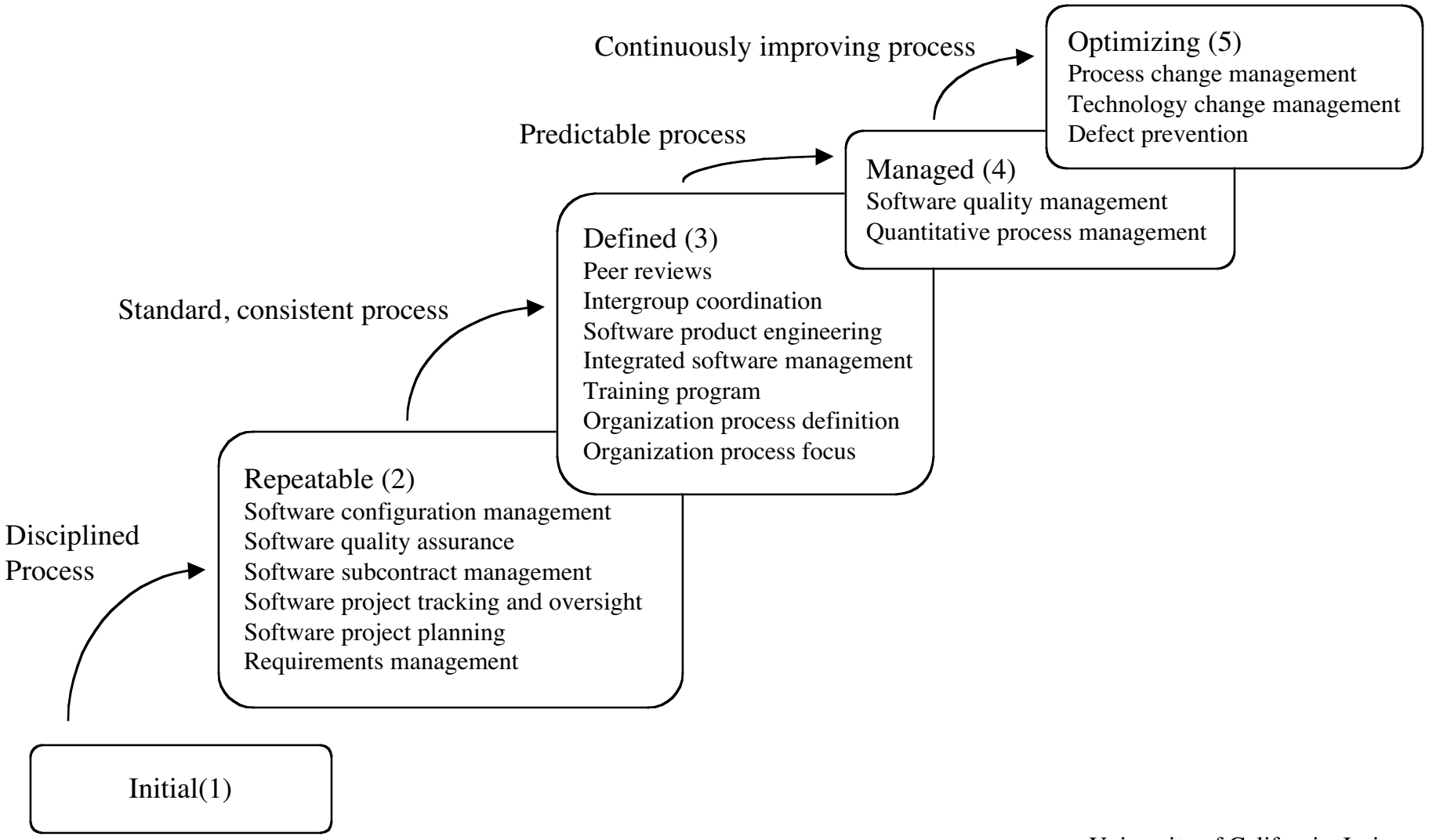
Risk item	Risk management techniques
1. Personnel shortfalls	Staffing with top talent, job matching; teambuilding; morale building; cross-training; pre-scheduling key people
2. Unrealistic schedules and budgets	Detailed, multisource cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing
3. Developing the wrong software functions	Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals
4. Developing the wrong user interface	Task analysis; prototyping; scenarios; user characterization (functionality, style, workload)
5. Gold plating	Requirements scrubbing; prototyping; cost-benefit analysis; design to cost
6. Continuing stream of requirement changes	High change threshold; information hiding; incremental development (defer changes to later increments)
7. Shortfalls in externally furnished components	Benchmarking; inspections; reference checking; compatibility analysis
8. Shortfalls in externally performed tasks	Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding
9. Real-time performance shortfalls	Simulation; benchmarking; modeling; prototyping; instrumentation; tuning
10. Straining computer-science capabilities	Technical analysis; cost-benefit analysis; prototyping; reference checking

Table 5. Software Risk Management Plan.

1.	Identify the project's top 10 risk items.
2.	Present a plan for resolving each risk item.
3.	Update list of top risk items, plan, and results monthly.
4.	Highlight risk-item status in monthly project reviews. • Compare with previous month's rankings, status.
5.	Initiate appropriate corrective actions.

Source: Barry Boehm, "A Spiral Model of Software Development and Enhancement, IEEE Computer, May 1988

SEI's Capability Maturity Model



A Comparison of Life Cycle Models

Model	Strengths	Weaknesses
Build-and-Fix	Fine for small programs that do not require much maintenance	Totally unsatisfactorily for nontrivial programs
Waterfall	Disciplined approach Document driven	Delivered product may not meet client's needs
Rapid Prototyping	Ensures that delivered product meets client's needs	A need to build twice Cannot always be used
Incremental	Maximizes early return on investment Promotes maintainability	Requires open architecture May degenerate into build-and-fix
Synchronize-and-stabilize	Future user's needs are met Ensures components can be successfully integrated	Has not been widely used other than in Microsoft
Spiral	Incorporates features of all the above models	Can be used only for large-scale products Developers have to be competent at risk-analysis

ICS 52 Software Life Cycle

- Requirements specification
  - Interview customer (TA)
  - Focus on “what”, not “how”
- Architectural and module design
  - Based on provided “official” requirements specification
  - Focus on interfaces
- Implementation
  - Based on provided “official” design
  - Focus on good implementation techniques
- Testing
  - Based on provided “official” implementation
  - Focus on fault coverage and discovery