ICS 52: Introduction to Software Engineering

Fall Quarter 2004

Professor Richard N. Taylor

Lecture Notes

http://www.ics.uci.edu/~taylor/ICS_52_FQ04/syllabus.html

University of California, Irvine

Add/Drop Policy

- All Adds done through WebReg (not add cards)
- Second week of classes
  - Deadline to add
- Second week of classes
  - Deadline to drop

Course Web Site

http://www.ics.uci.edu/~taylor/ICS_52_FQ04/syllabus.html

- Contents
  - Information on the instructor, TA, and readers
  - Overview and prerequisite knowledge
  - Textbook
    - » Hans van Vliet
    - » Backup: Ghezzi, Jazayeri, and Mandrioli & Sommerville
  - Schedule
  - Assignments and assessment
  - 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
- No discussion section meetings until September 29th

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

Software Engineering

- "*A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers*." [Ghezzi, Jazayeri, Mandrioli]

- "*Multi-person construction of multi-version software*." [Parnas]

Software Engineering

- "*A discipline whose aim is the production of fault-free software, delivered on-time and within budget, that satisfies the user's needs. Furthermore, the software must be easy to modify when the user's needs change.*" [Schach]

- "*Difficult.*" [van der Hoek]

Software Engineering

- "It's where you actually get to design big stuff and be creative." [Taylor]

Context

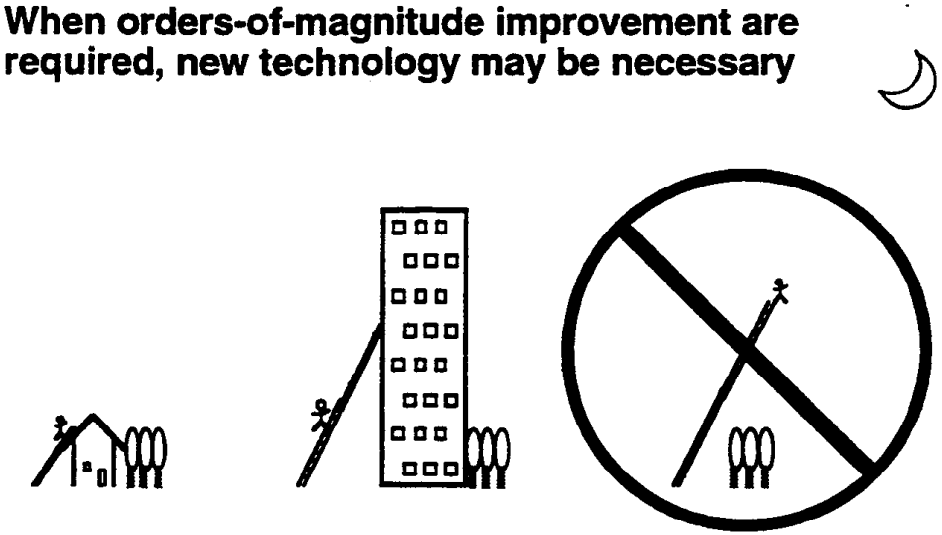
Small project	Huge project
You	Teams
Build what you want	Build what they want
One product	Family of products
Few sequential changes	Many parallel changes
Short-lived	Long-lived
Cheap	Costly
Small consequences	Large consequences

Programming Engineering

Differences from Programming

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

Matters of Scale



- High powered techniques not appropriate for all problems (Using a forklift to carry a paperback novel)
- 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

People

- The single most important factor in the success/failure of a product
- Scarce resource
  - Quality
  - Suitability
  - Cost
- Many different kinds of people
  - Managers
  - Programmers
  - Technical writers

Not the focus of ICS 52: see ICS 131

Processes

- Essential to achieve a quality product
- Scarce resource
  - Quality
  - Suitability
  - Cost
- Many different kinds of processes
  - Bug tracking
  - Change approval
  - Quality assurance

Focus of ICS 52

Tools

- Needed to support people and processes
- Scarce resource
  - Quality
  - Suitability
  - Cost
- Many different kinds of tools
  - Drawing
  - Analysis

  } people support

  - Project management
  - Source code management

  } process support

Not the focus of ICS 52: see ICS 121

Product

- Result of applying people, processes, and tools
- Consists of many deliverables
  - Software
  - Documentation
  - User manuals
  - Test cases
  - Design documents
- Intrinsic qualities
  - Safety
  - Reliability
  - User friendliness

People + Processes + Tools = Product

People, Processes, Tools, Products

- Products are always the eventual goal
  - Selling products creates revenue
  - Selling good products creates lots of revenue
  - Selling bad products creates little revenue
- People, processes, and tools are retained by organization
  - Build a reputation through the quality of products
  - Create organizational culture
  - Important to keep the team intact

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 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

Processes

- Institute processes through which software is engineered
  - Cover all steps from initial idea and requirements to delivery, maintenance, and final retirement
  - Make sure we do the right things/things right
  - Make sure we do not forget to do anything
  - Different processes for different kinds of software
- Not a silver bullet [Brooks “No Silver Bullet”]
  - Software is still intrinsically difficult to deal with
  - Processes help, but cannot guarantee anything

Remember: People + Processes + Tools => Product

Software Processes

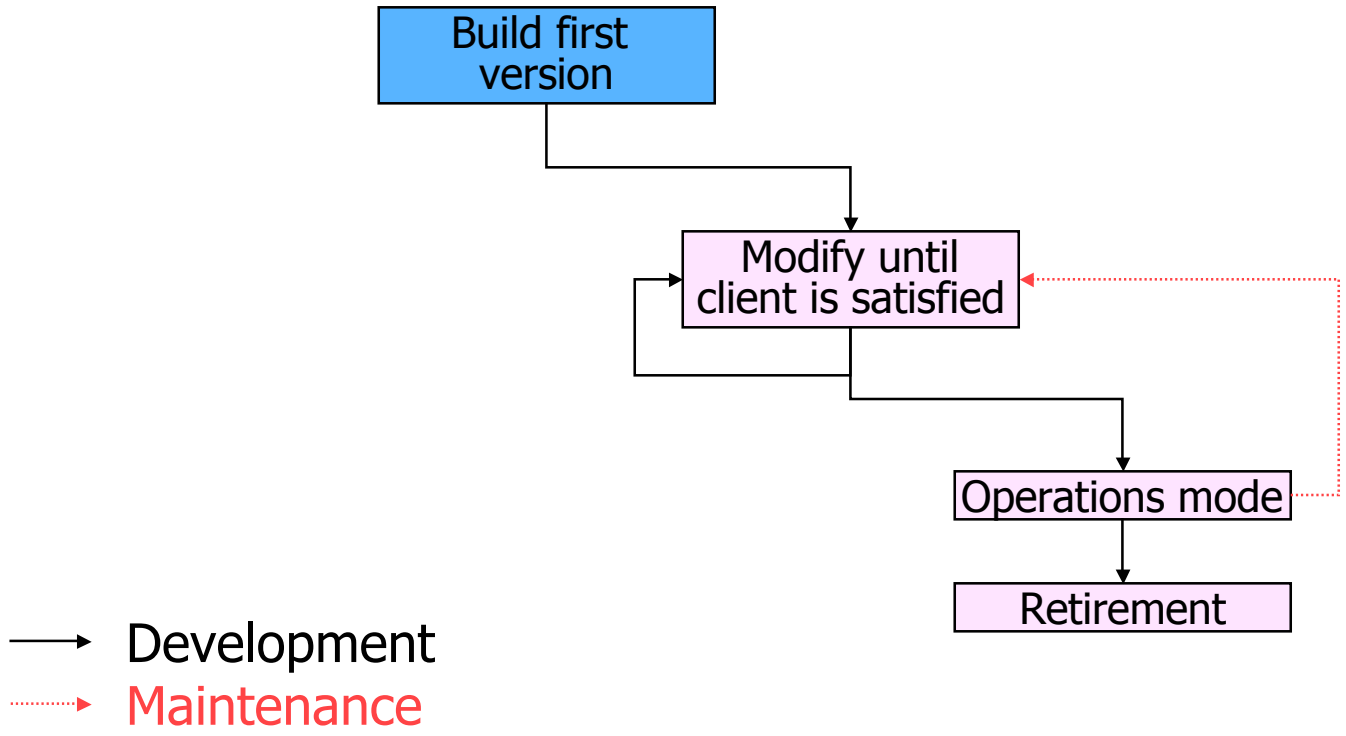
- Elements
  - Activities (“phases”)
  - Artifacts
    - » Can include process specifications
  - Resources
    - » People (their time and cost)
    - » Tools (their time and cost)
- Relationships between the elements
  - precedence, requires, produces, refines to
  - ...
- Constraints
  - Time
  - Cost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

Software Life Cycle Models

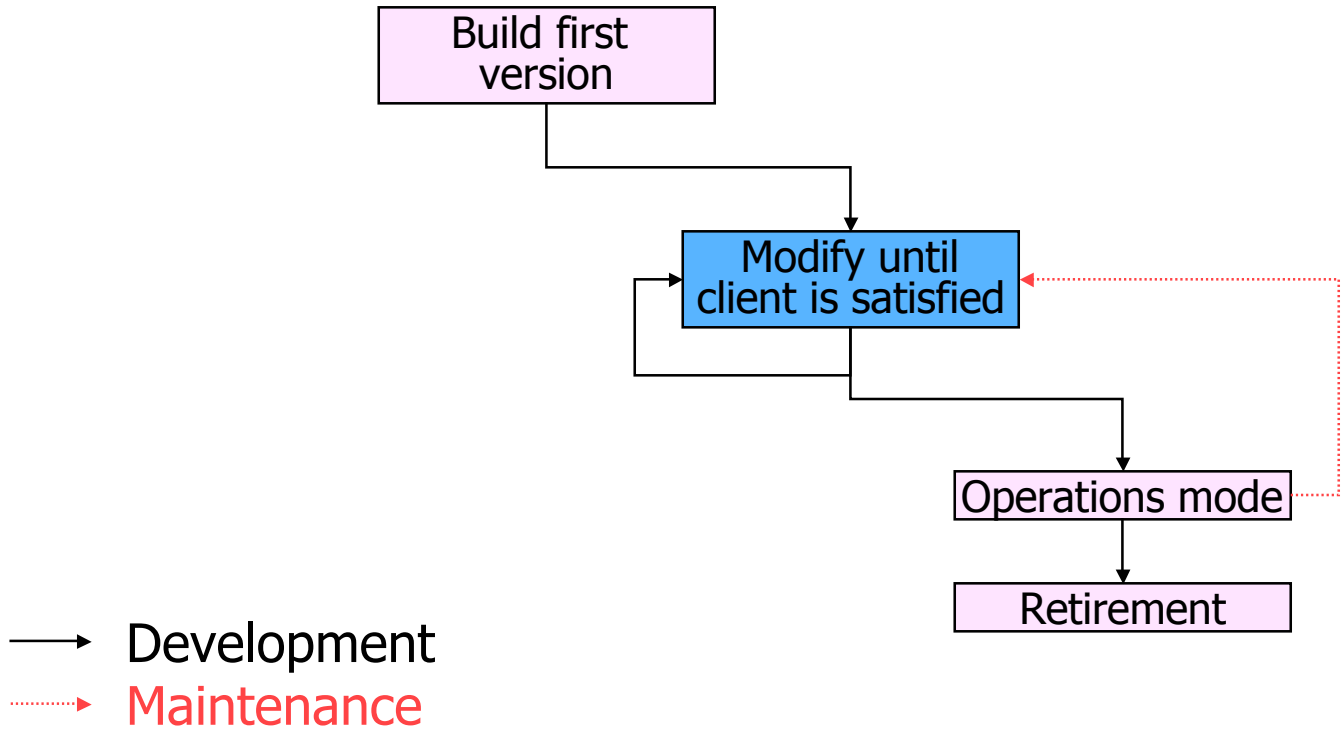
- Build-and-fix
- Waterfall
- Rapid prototyping
- Incremental
- Synchronize-and-stabilize
- Spiral

A software life cycle model is a high-level process

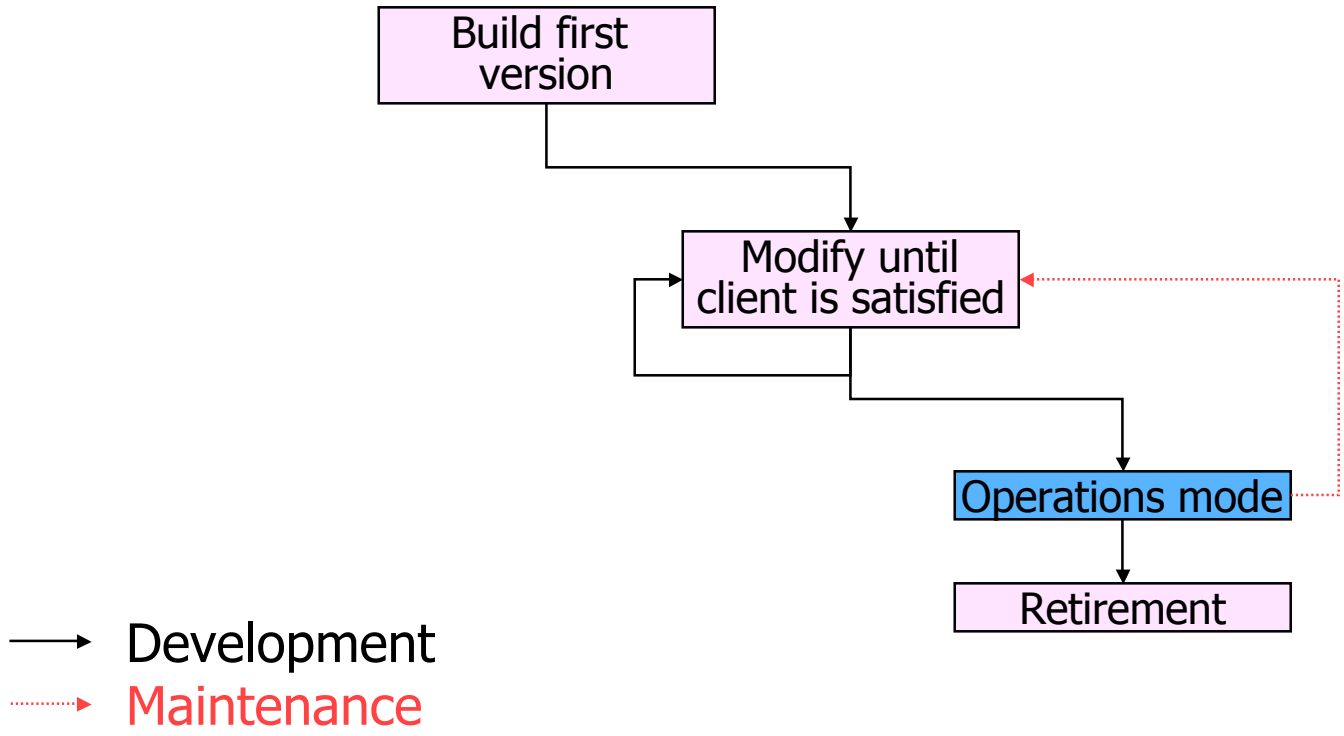
Build-and-Fix



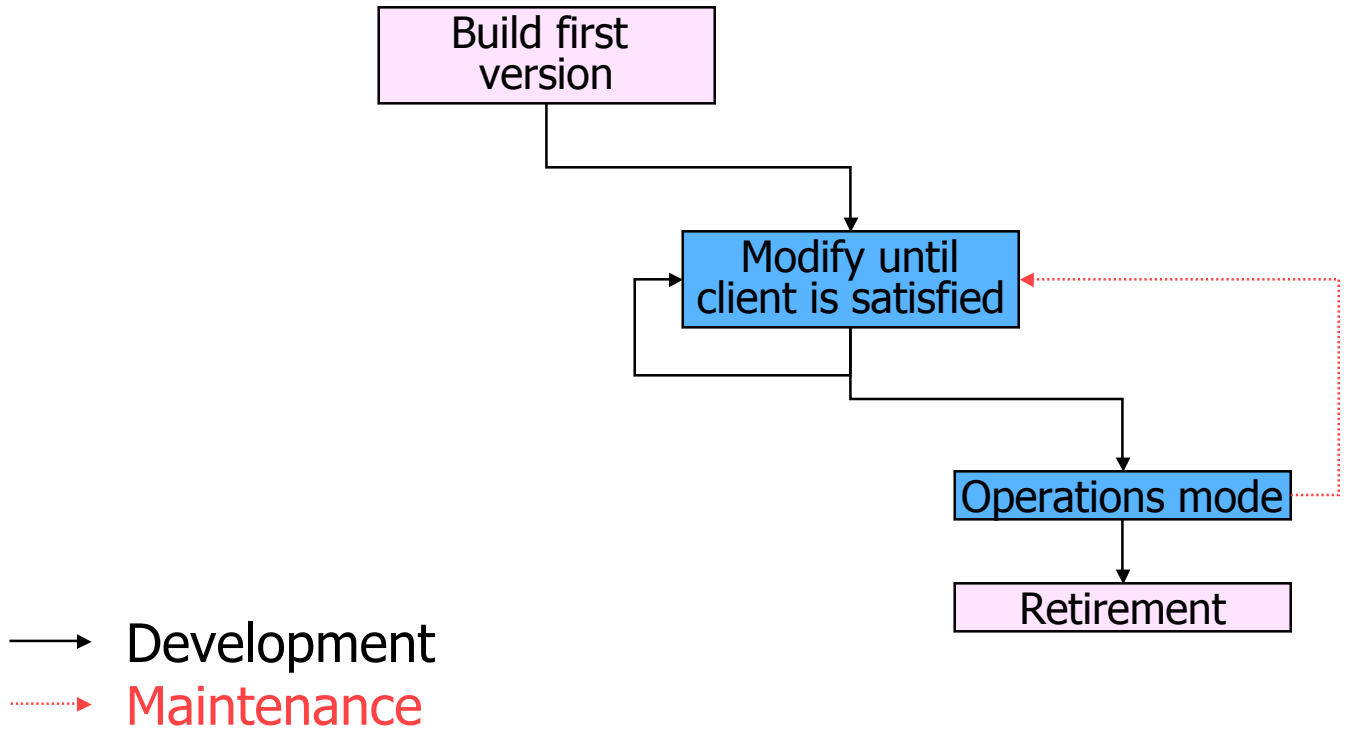
Build-and-Fix



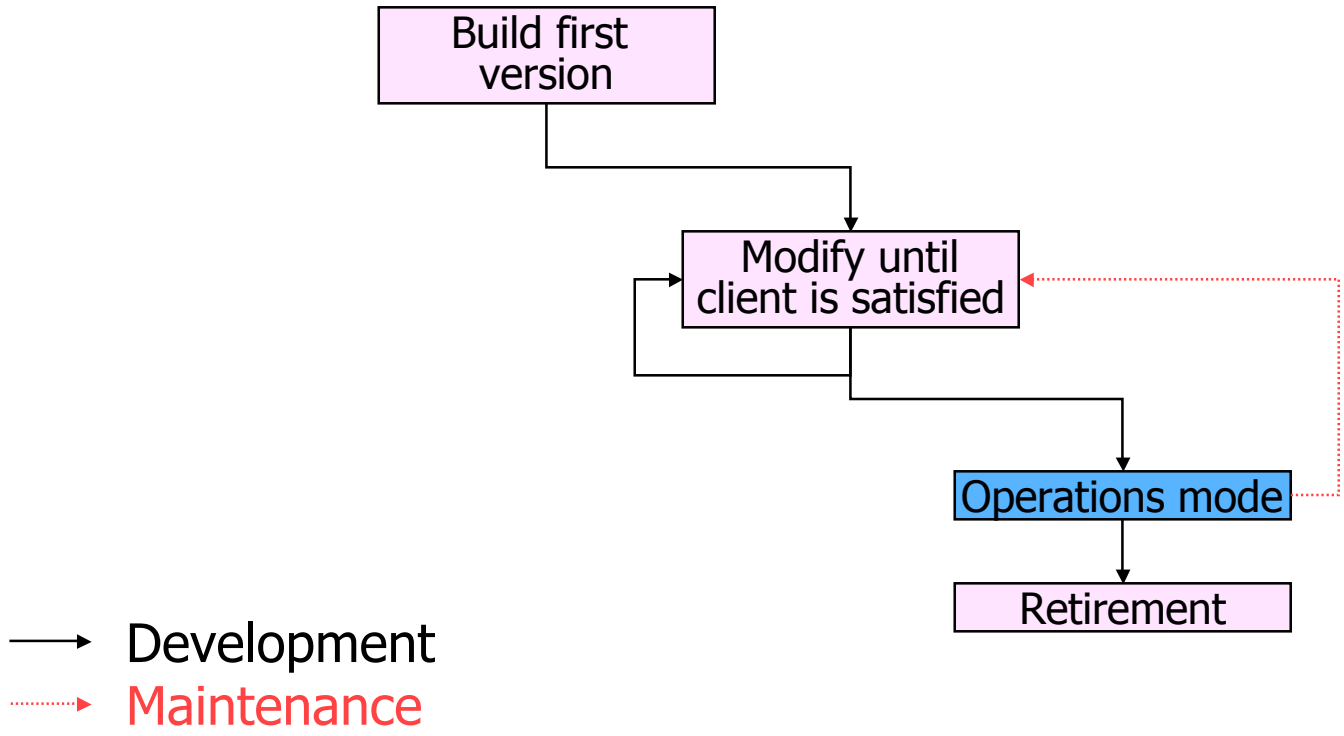
Build-and-Fix



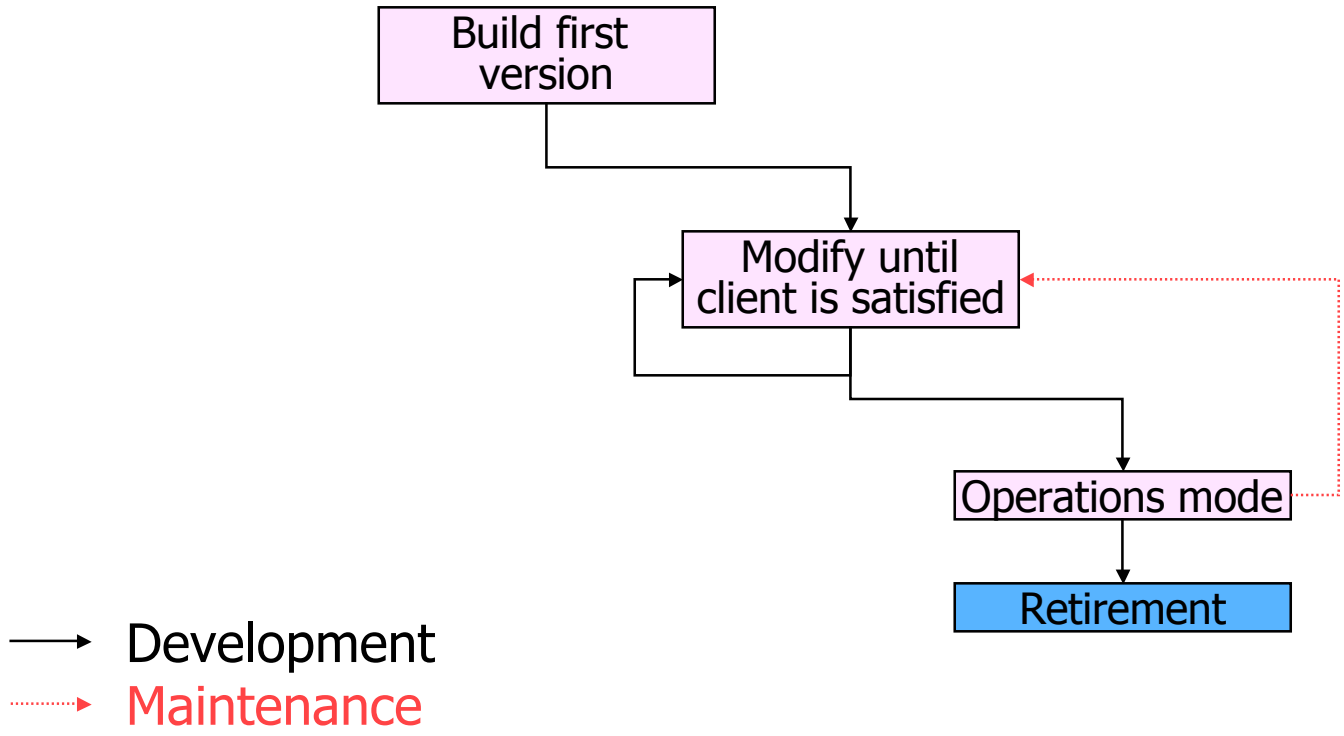
Build-and-Fix



Build-and-Fix



Build-and-Fix



Waterfall Approach

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

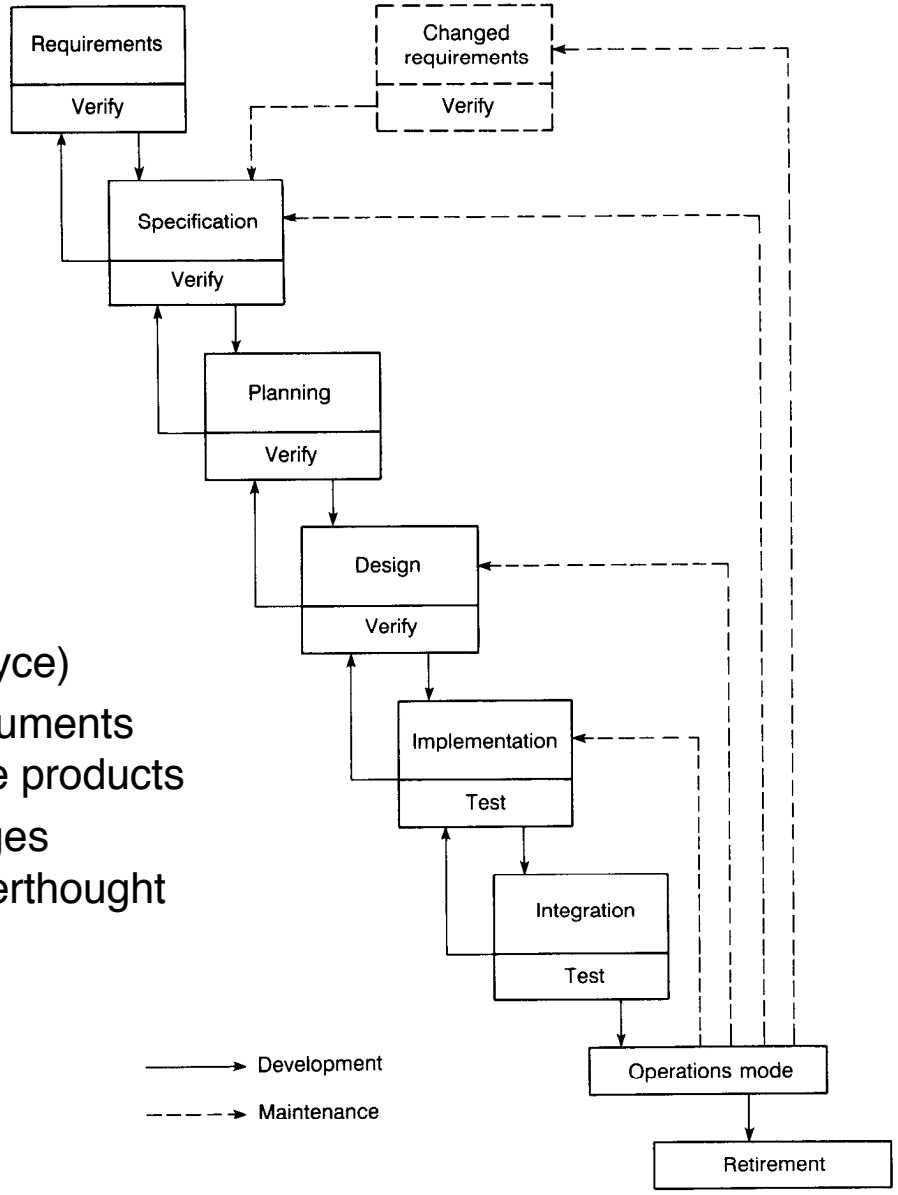
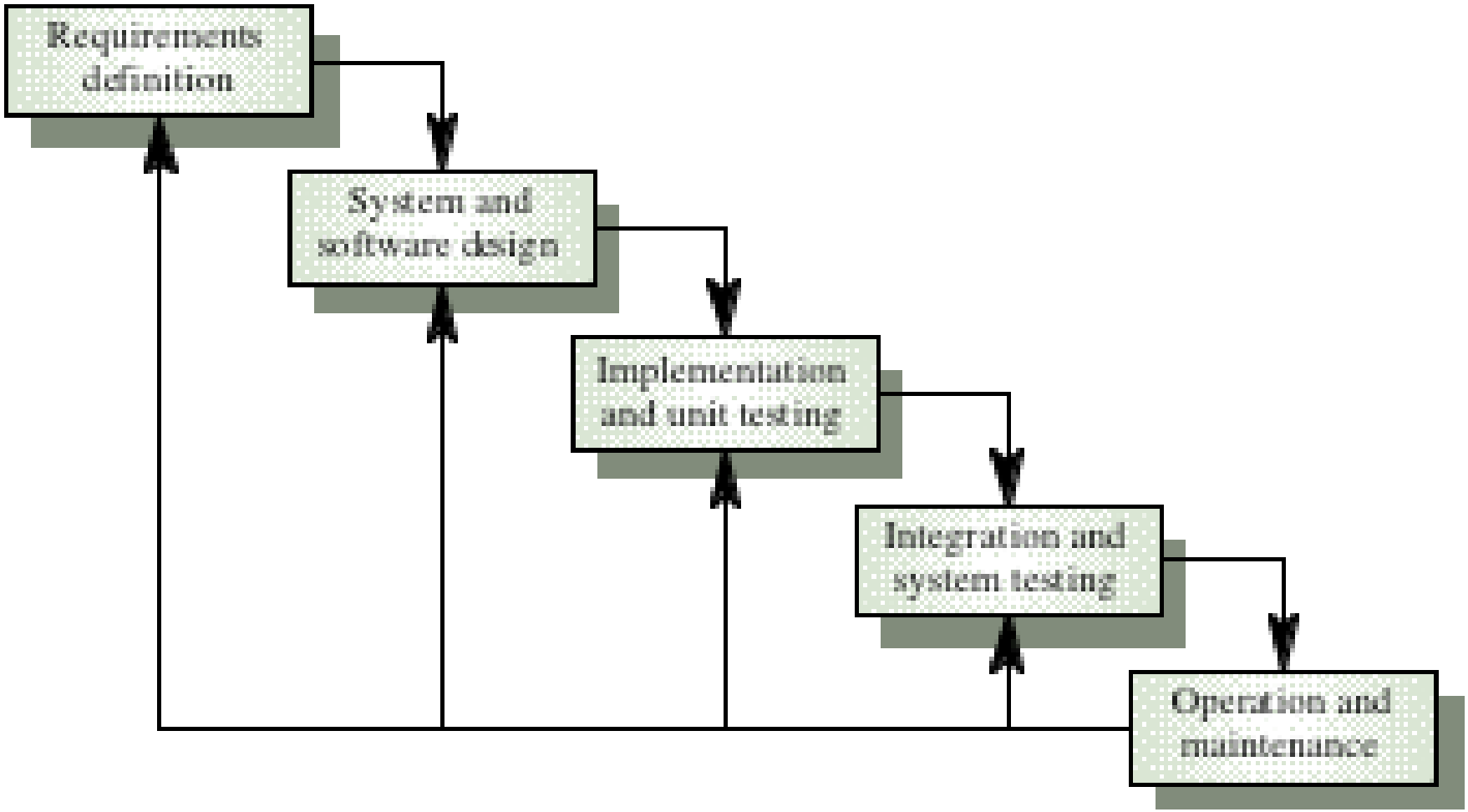


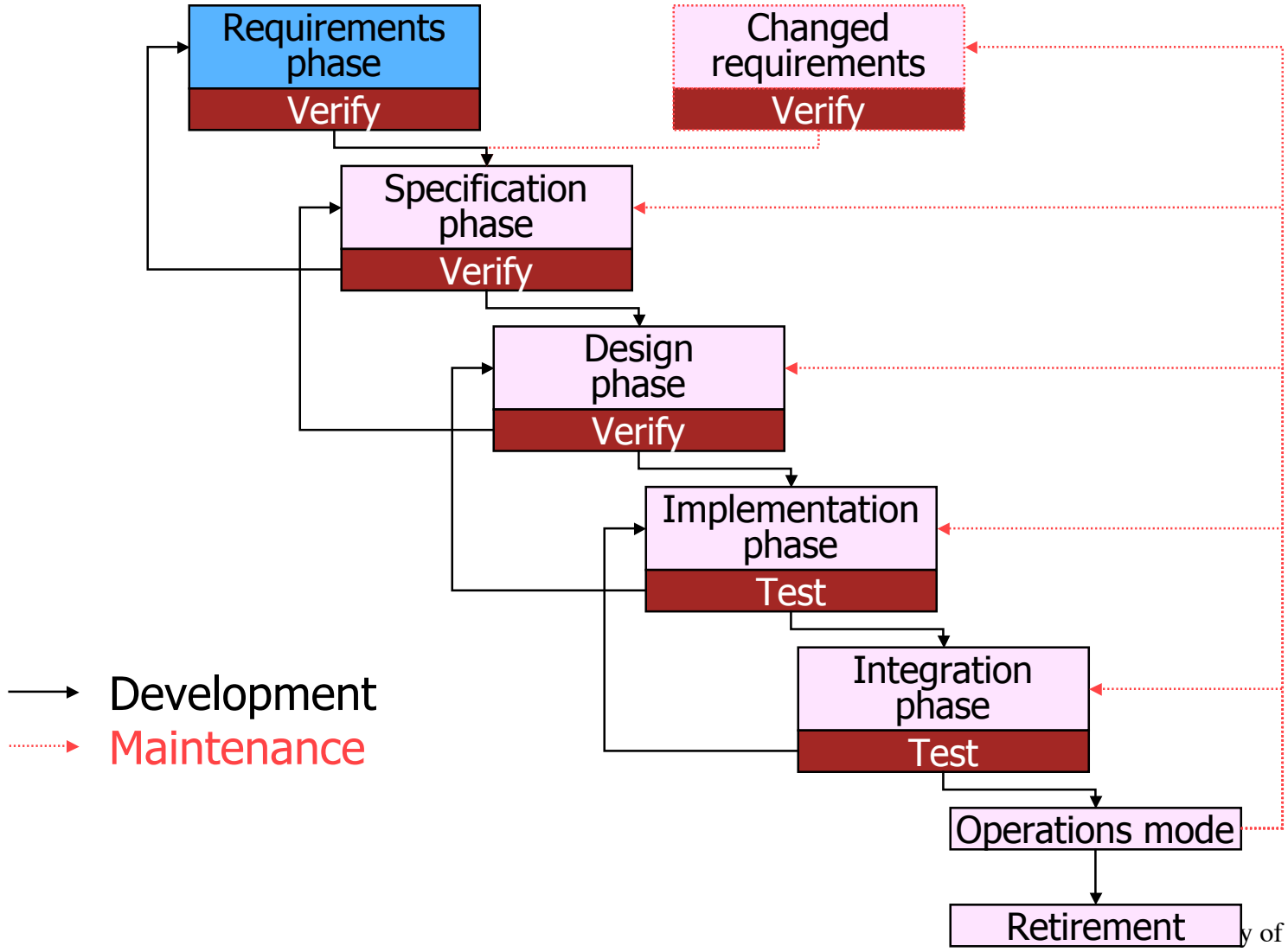
Figure 3.2 Waterfall model.

Source: Schach, ibid..

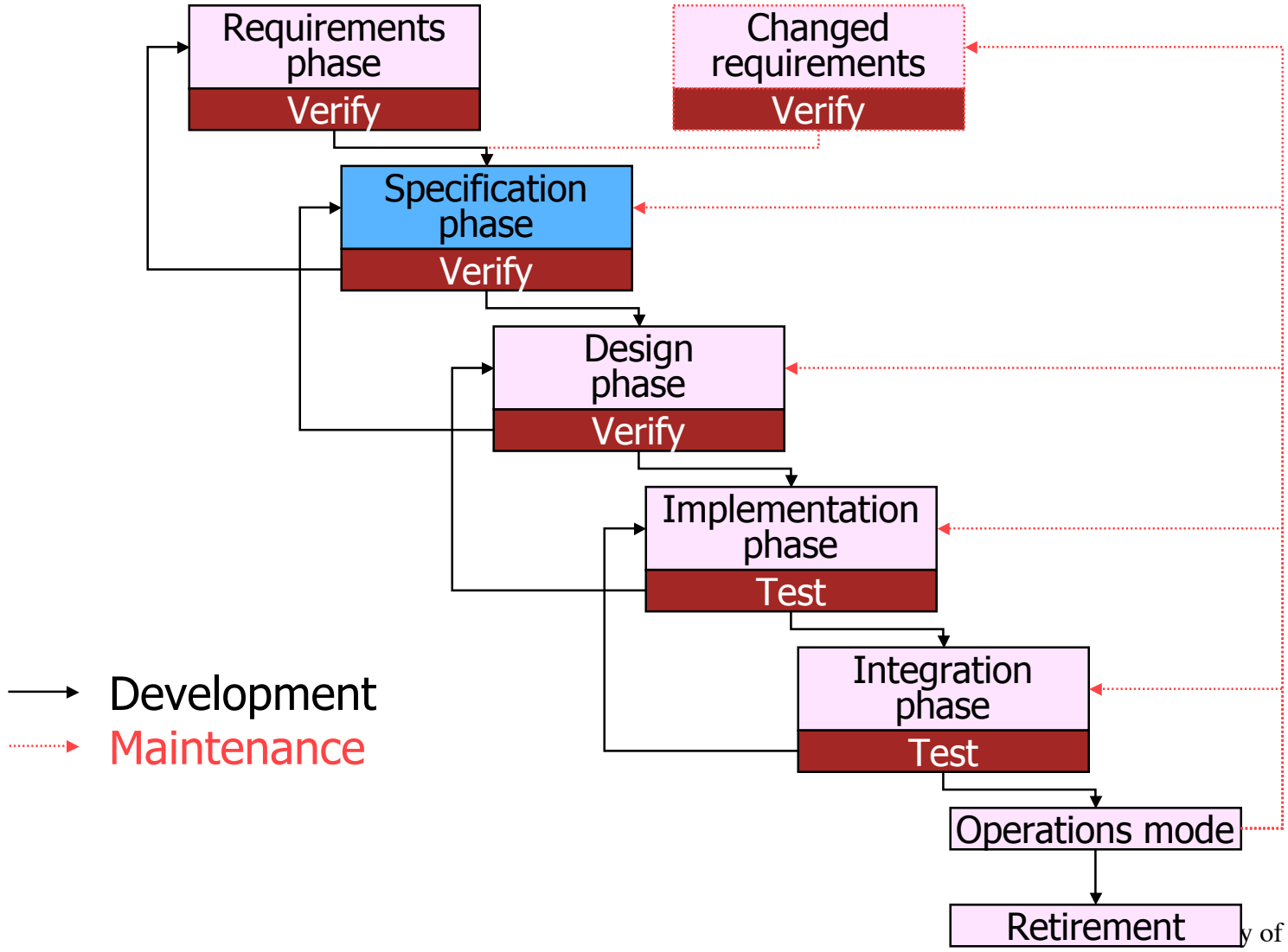
Waterfall model



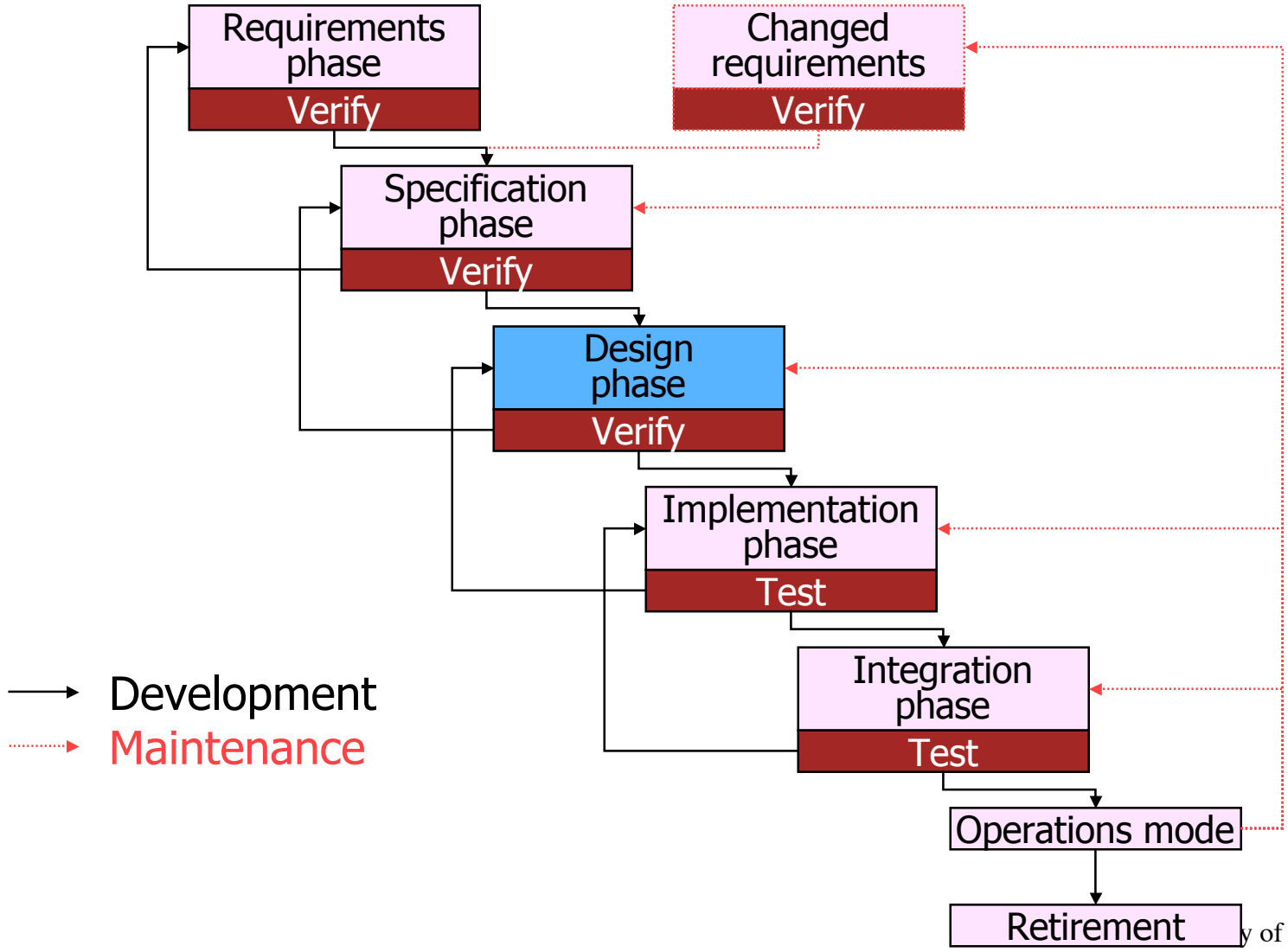
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

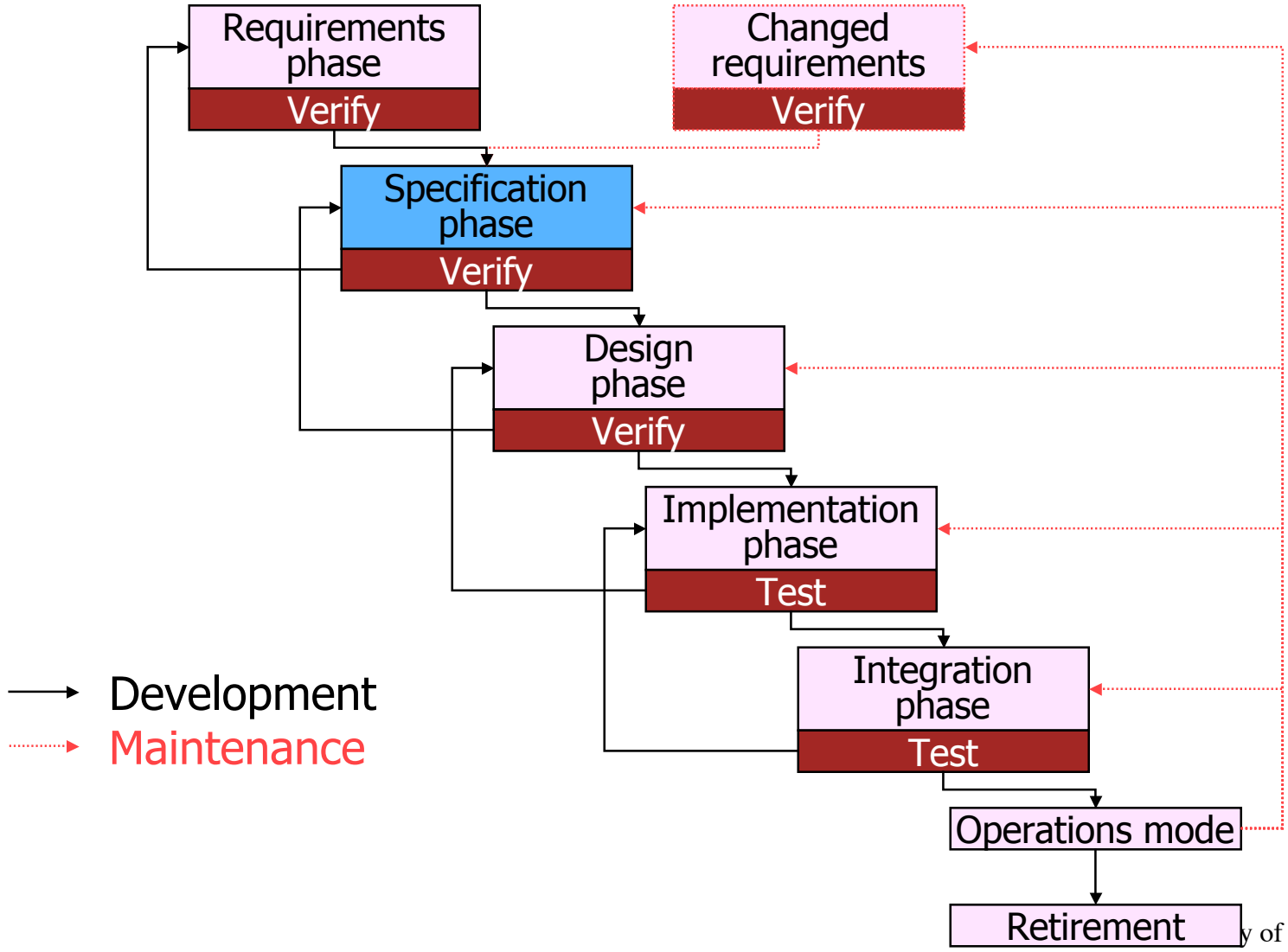
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

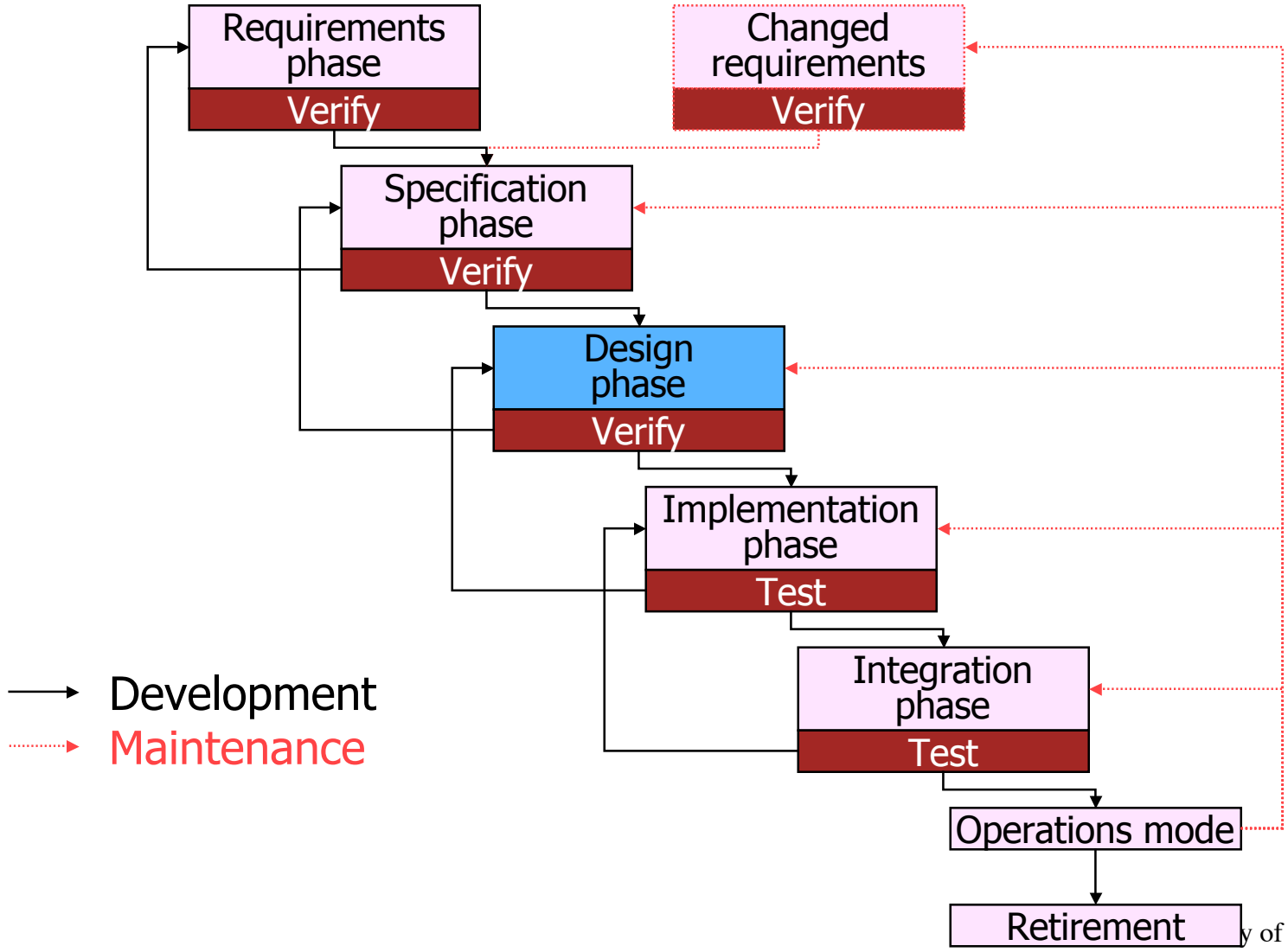
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

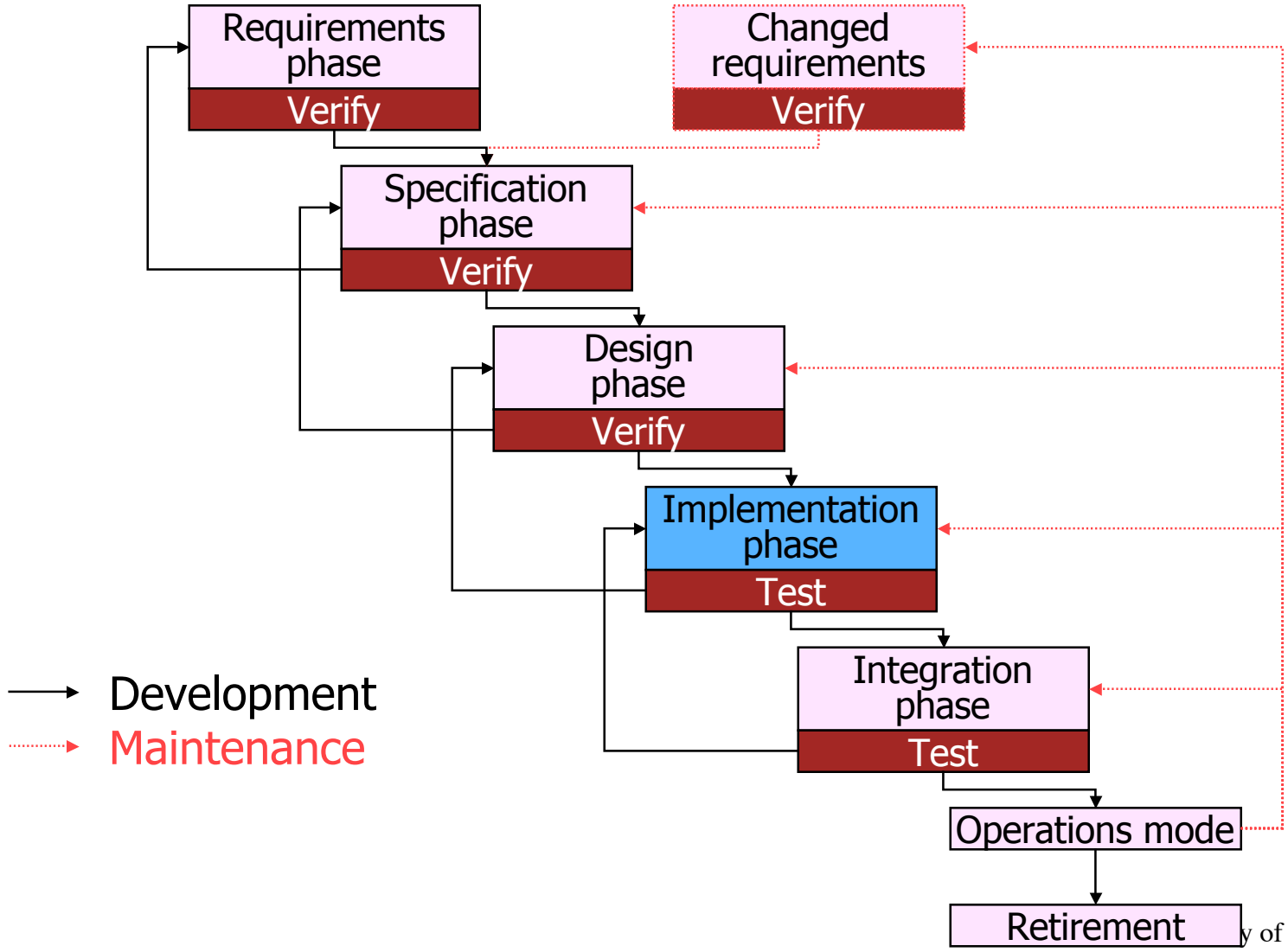
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

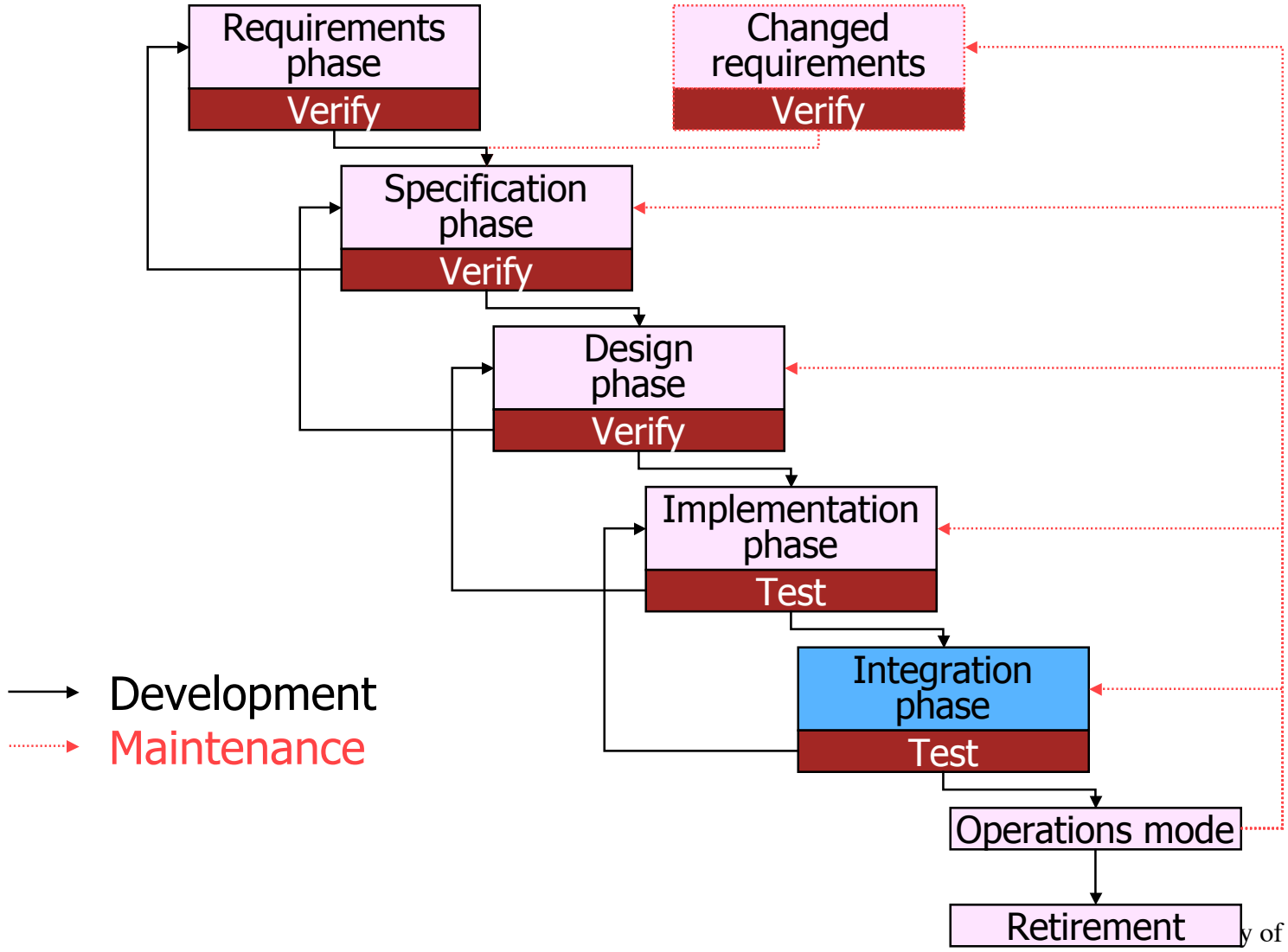
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

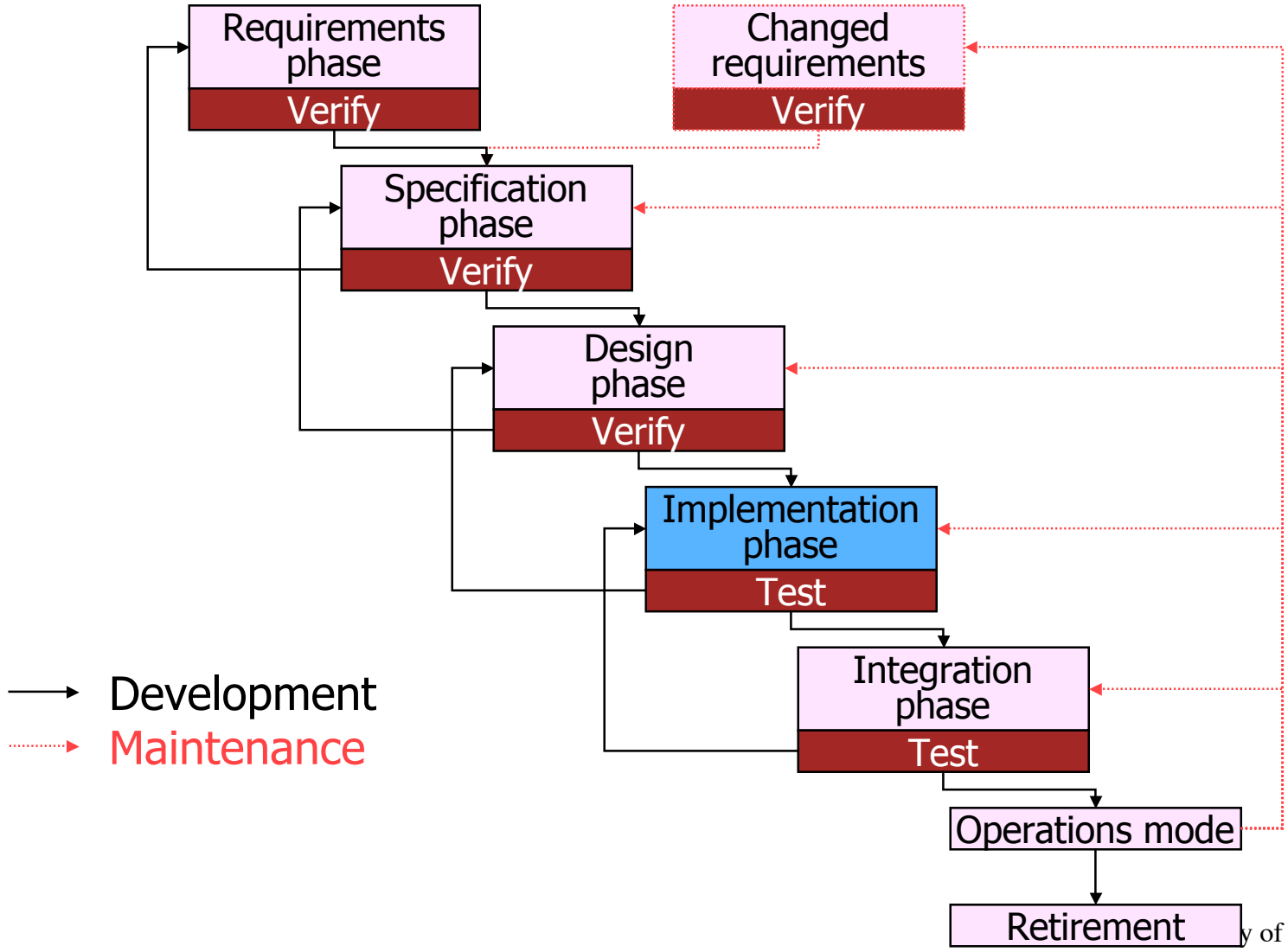
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

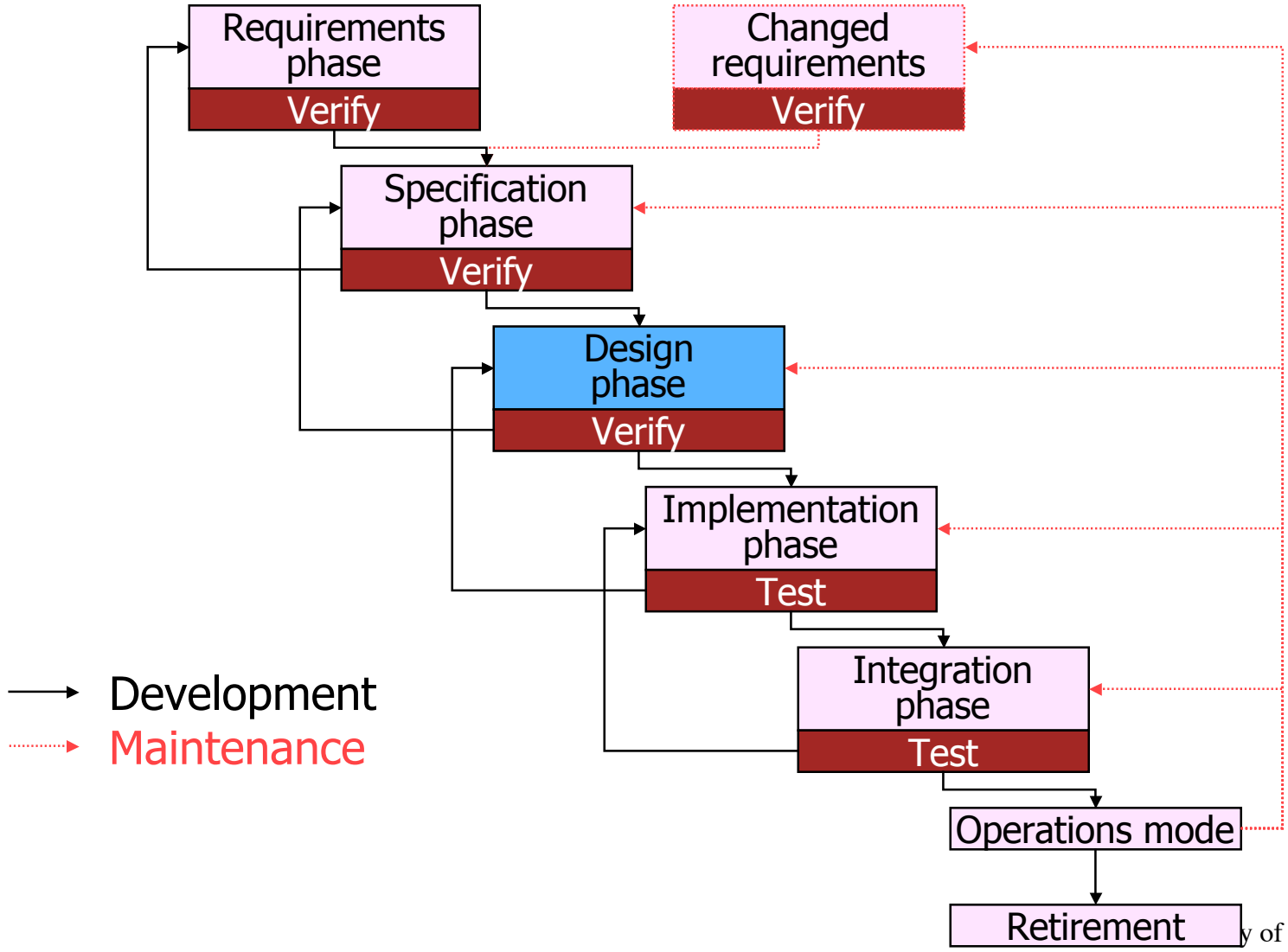
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

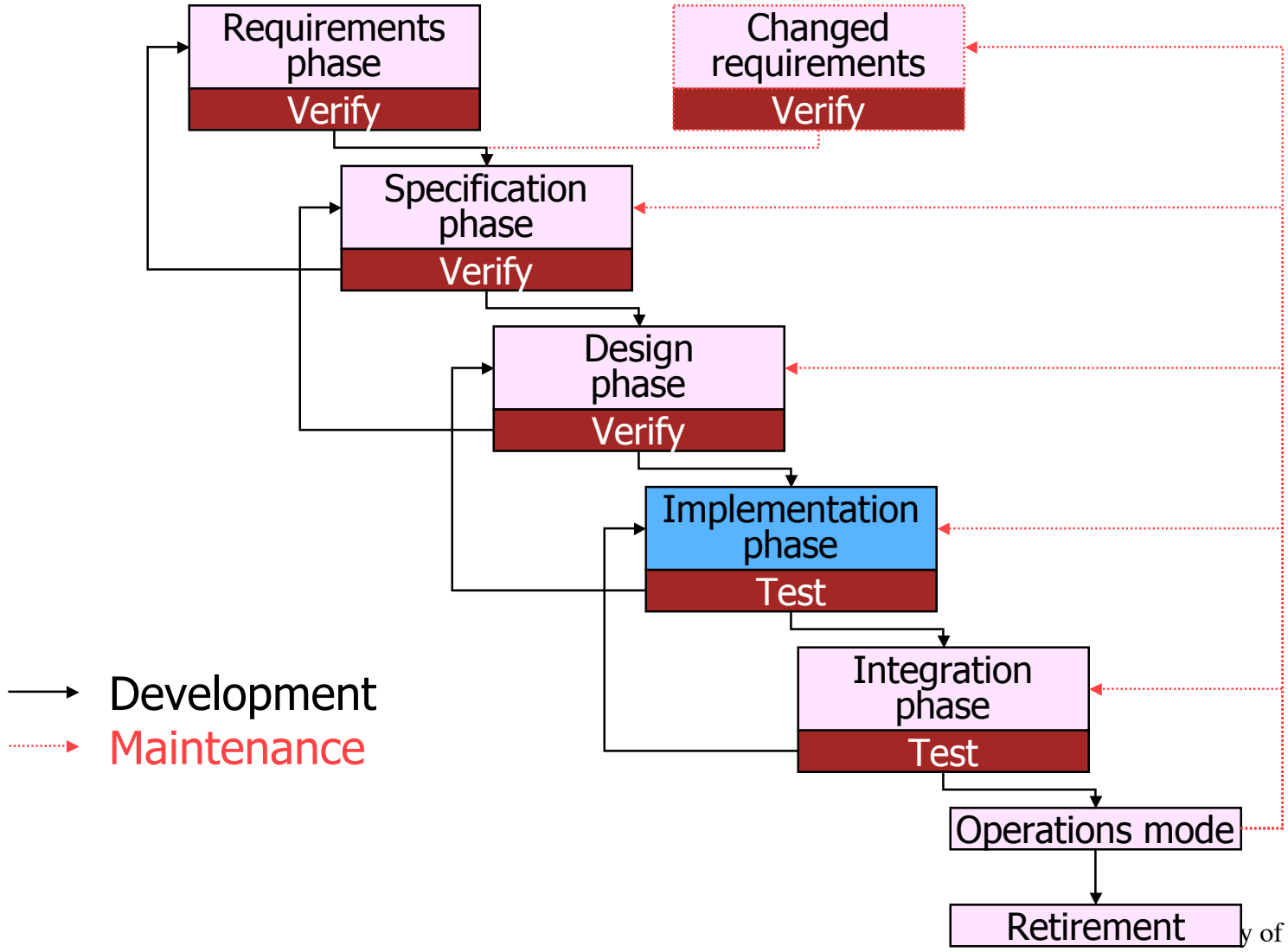
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

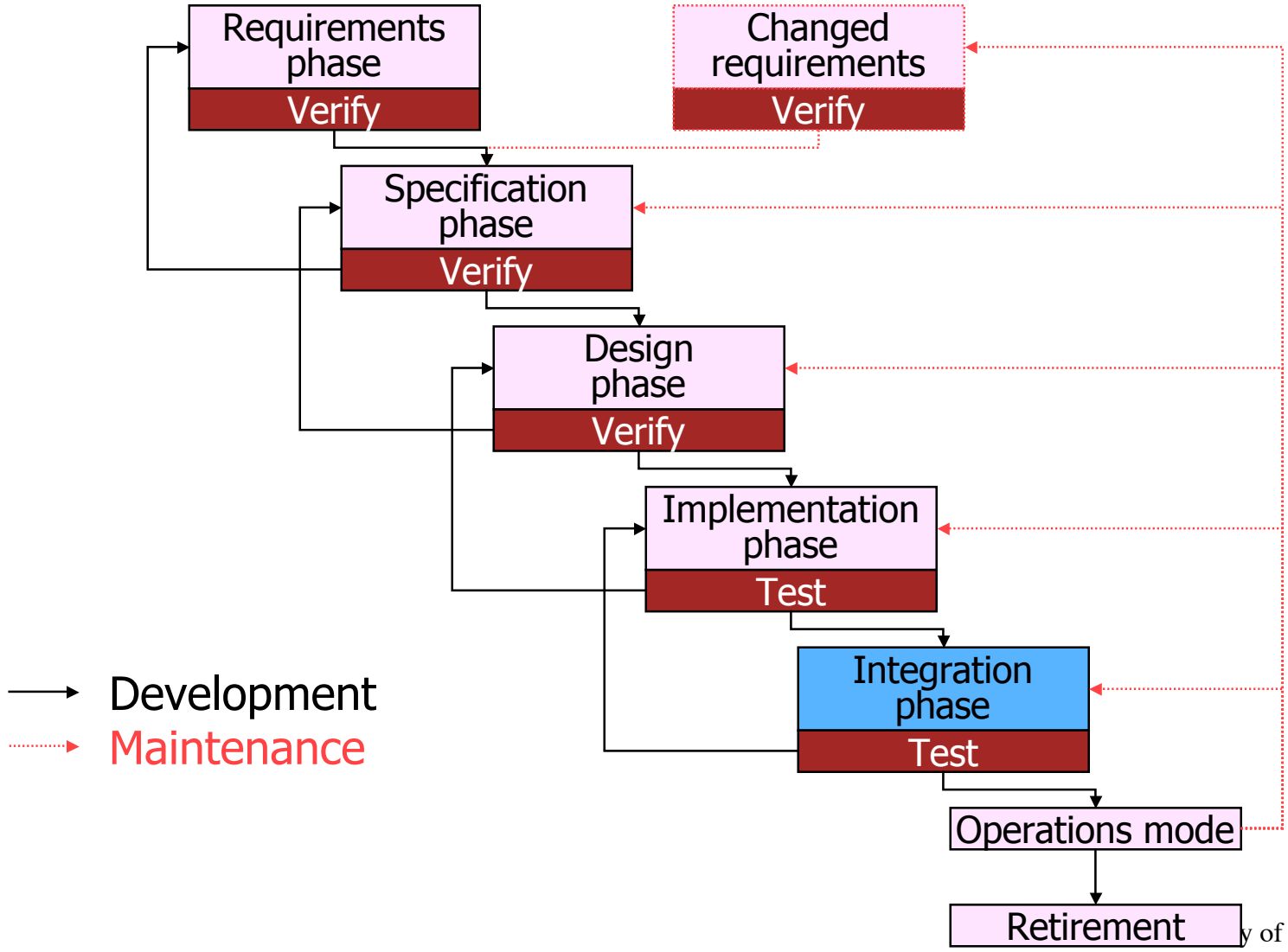
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

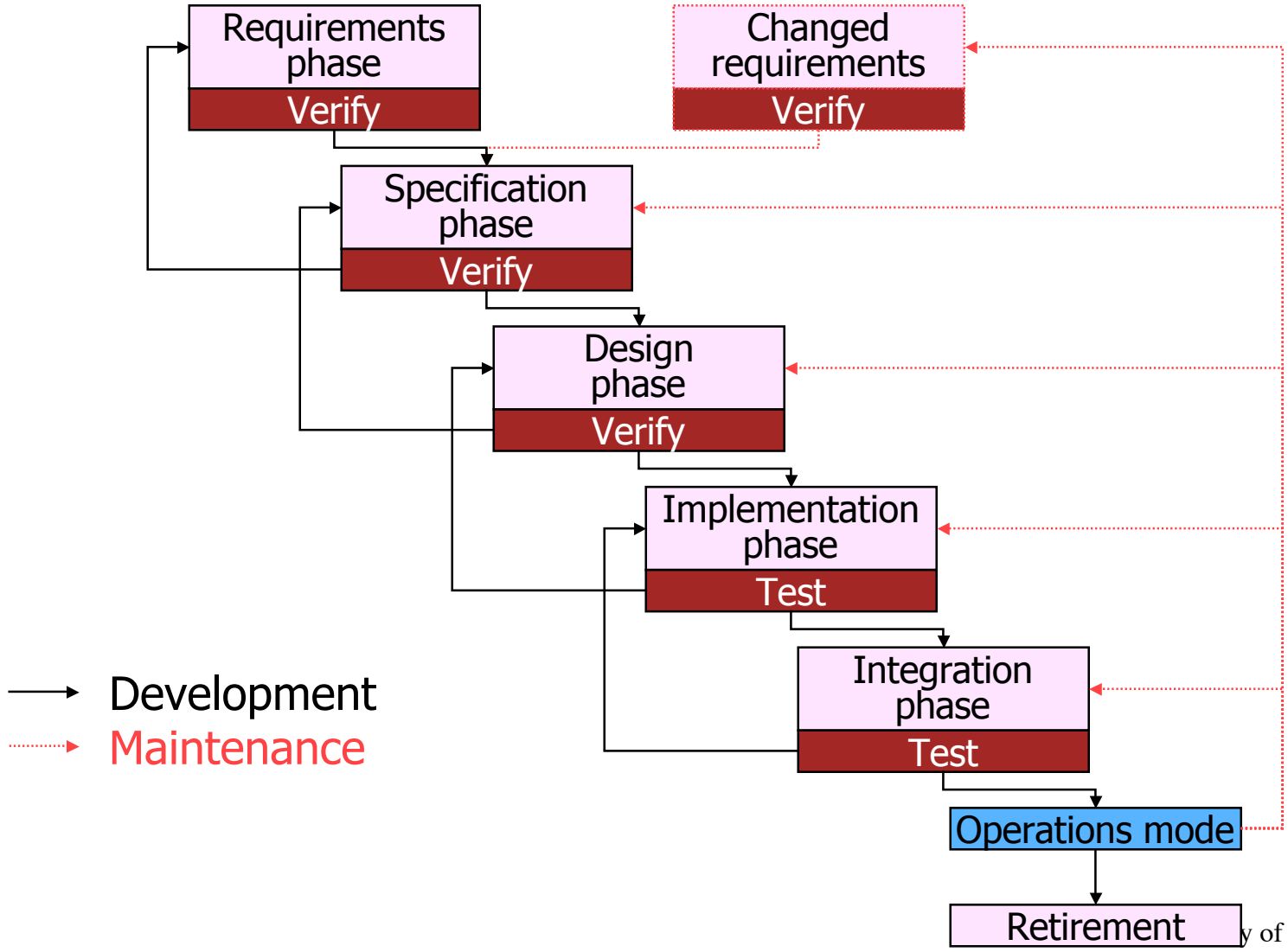
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

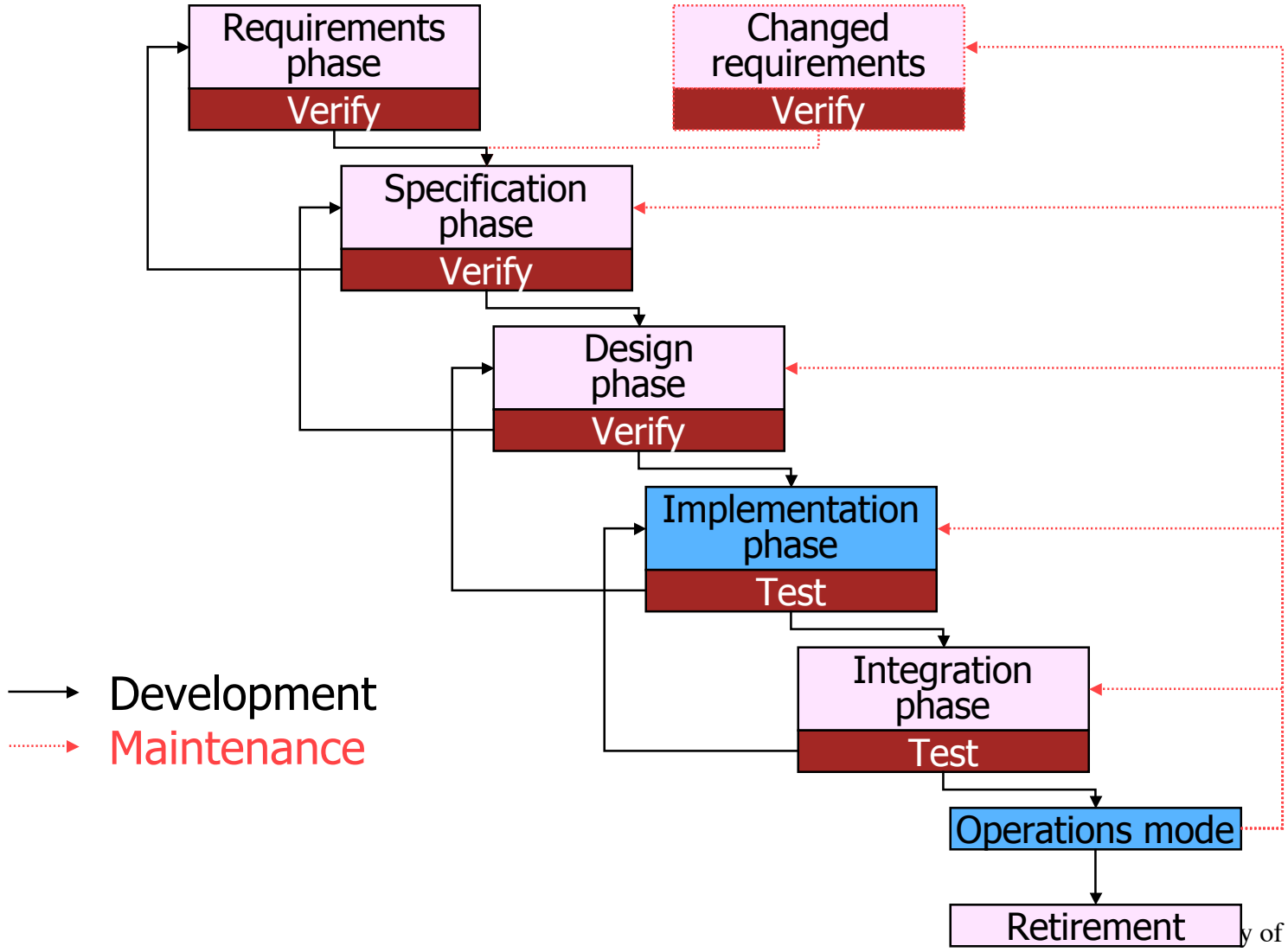
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

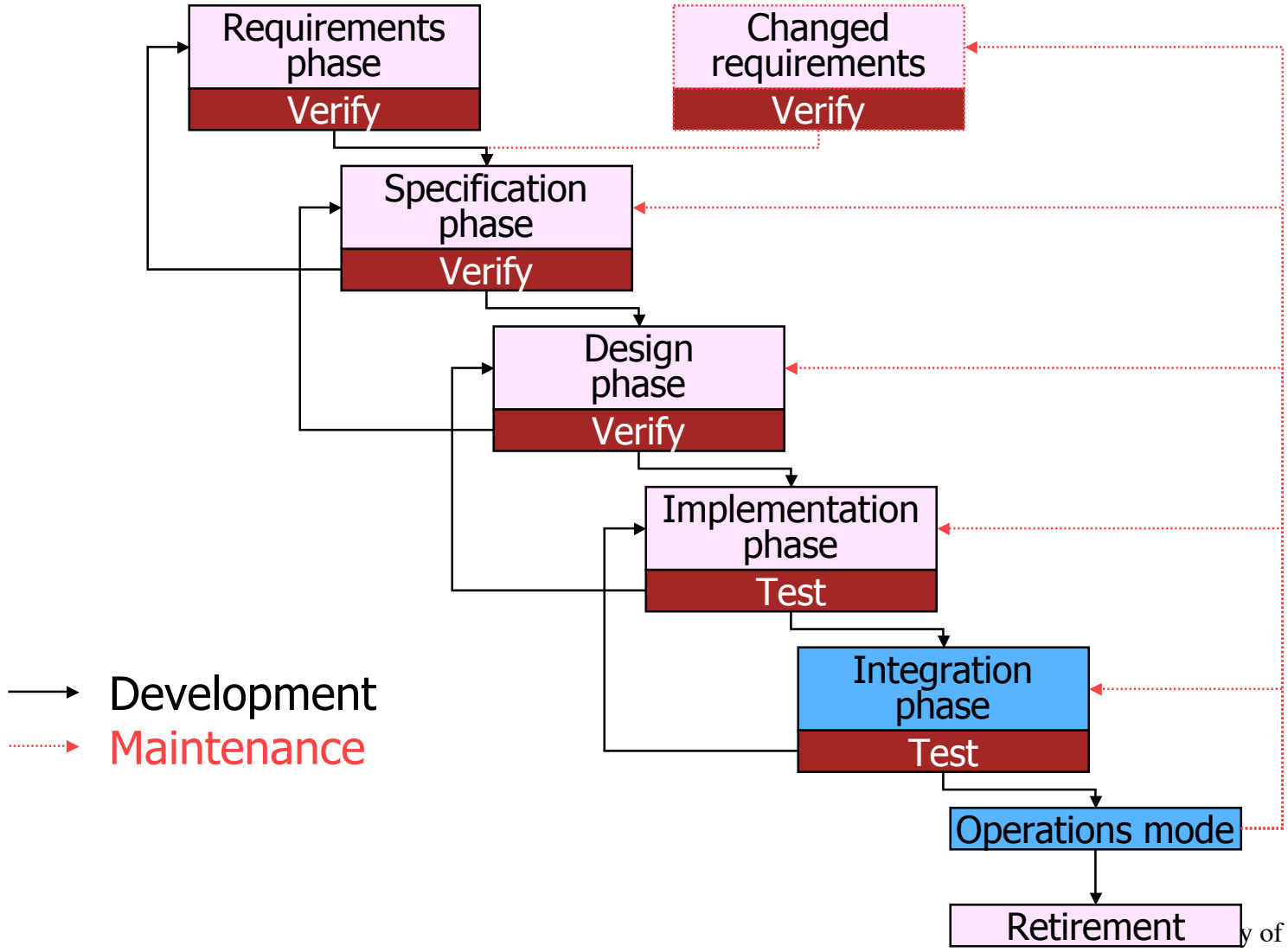
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

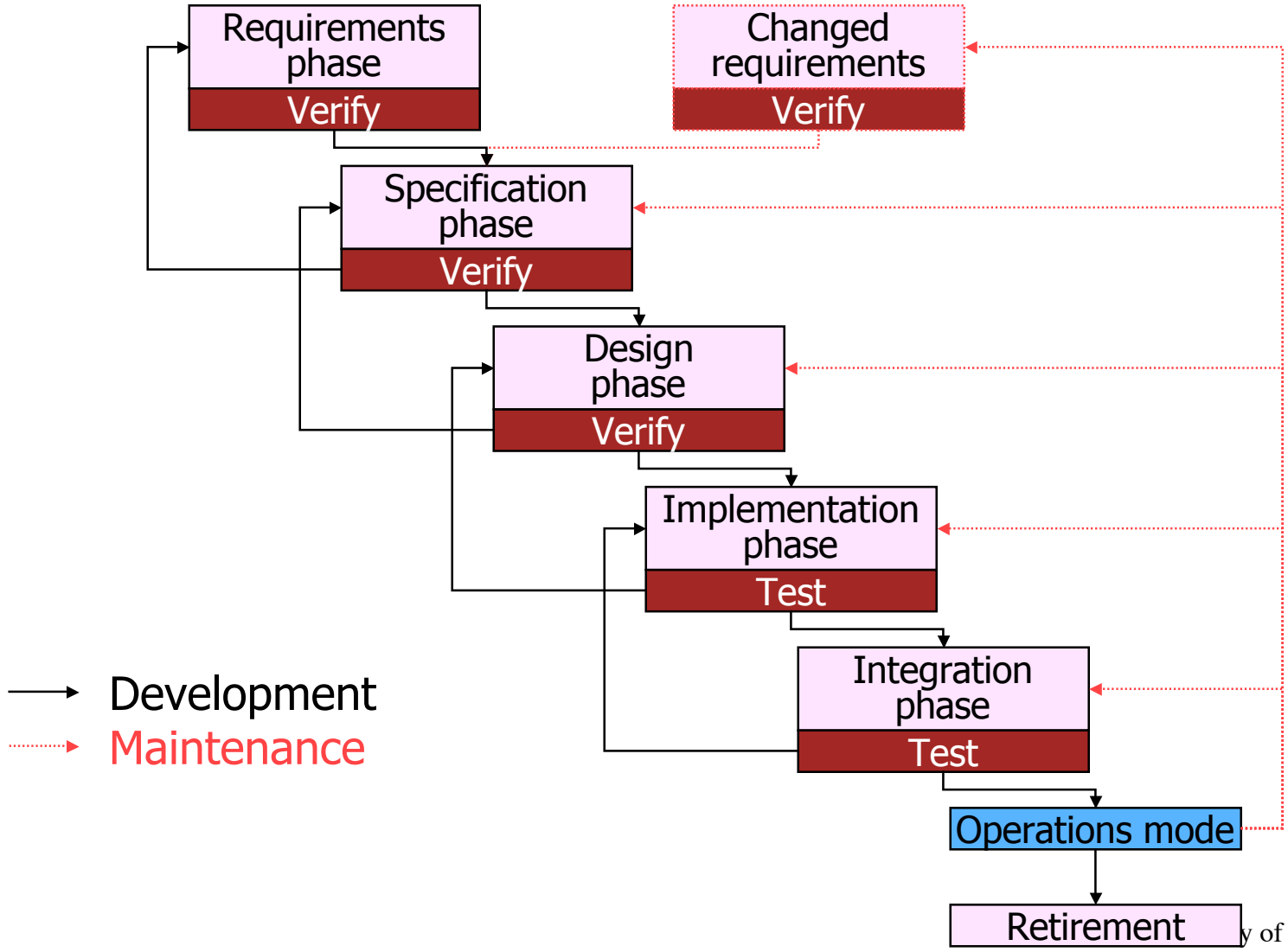
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

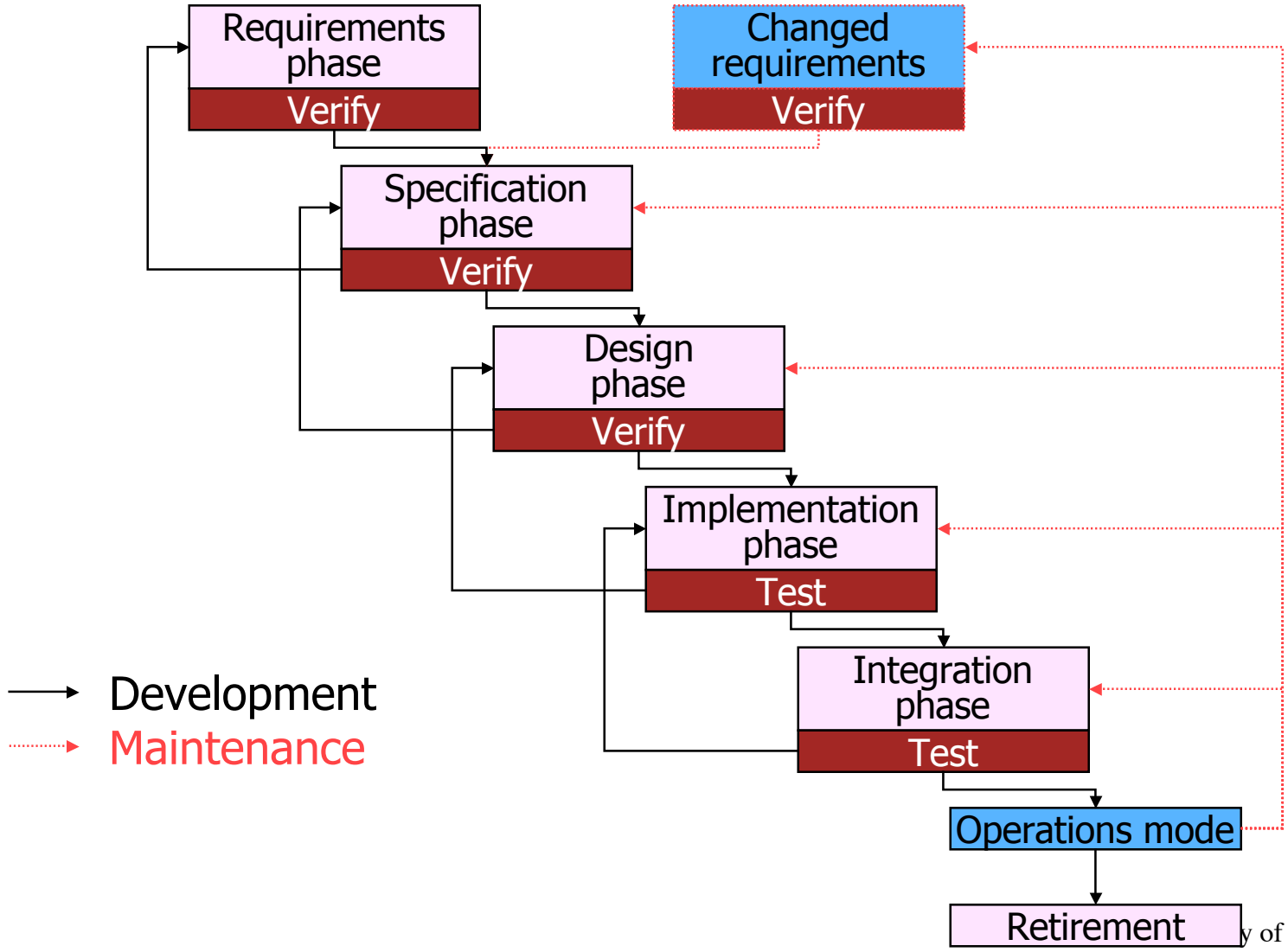
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

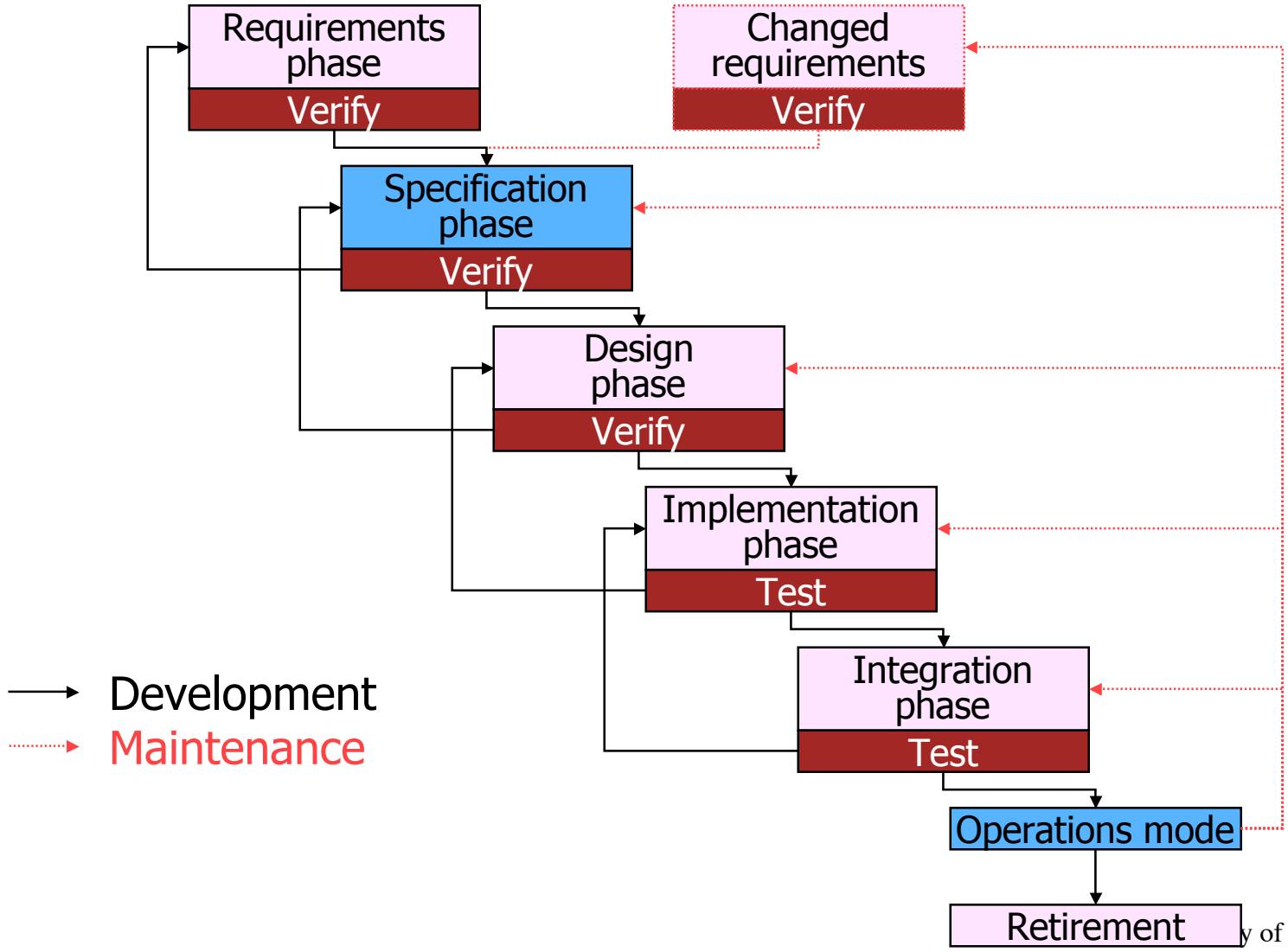
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

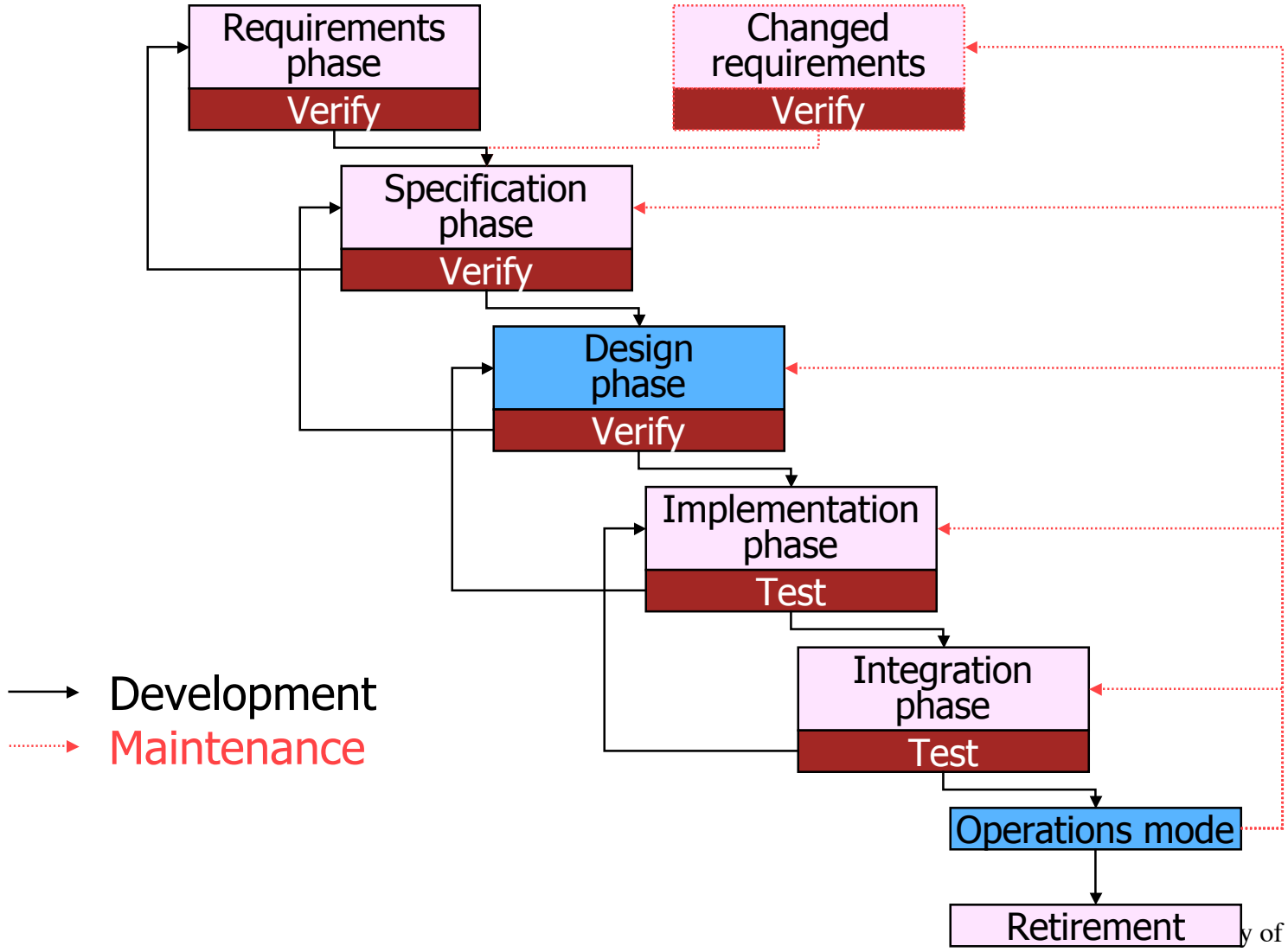
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

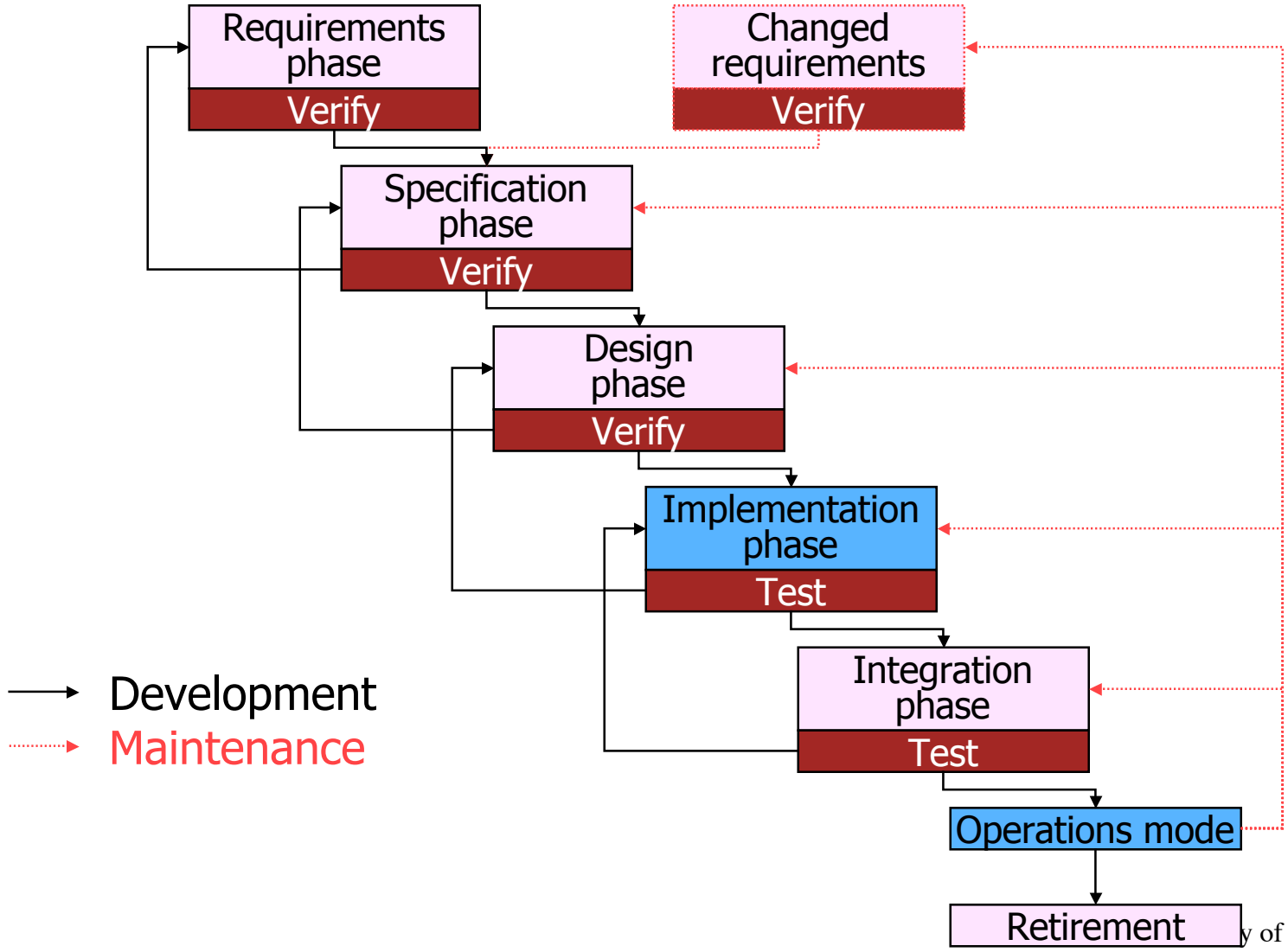
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

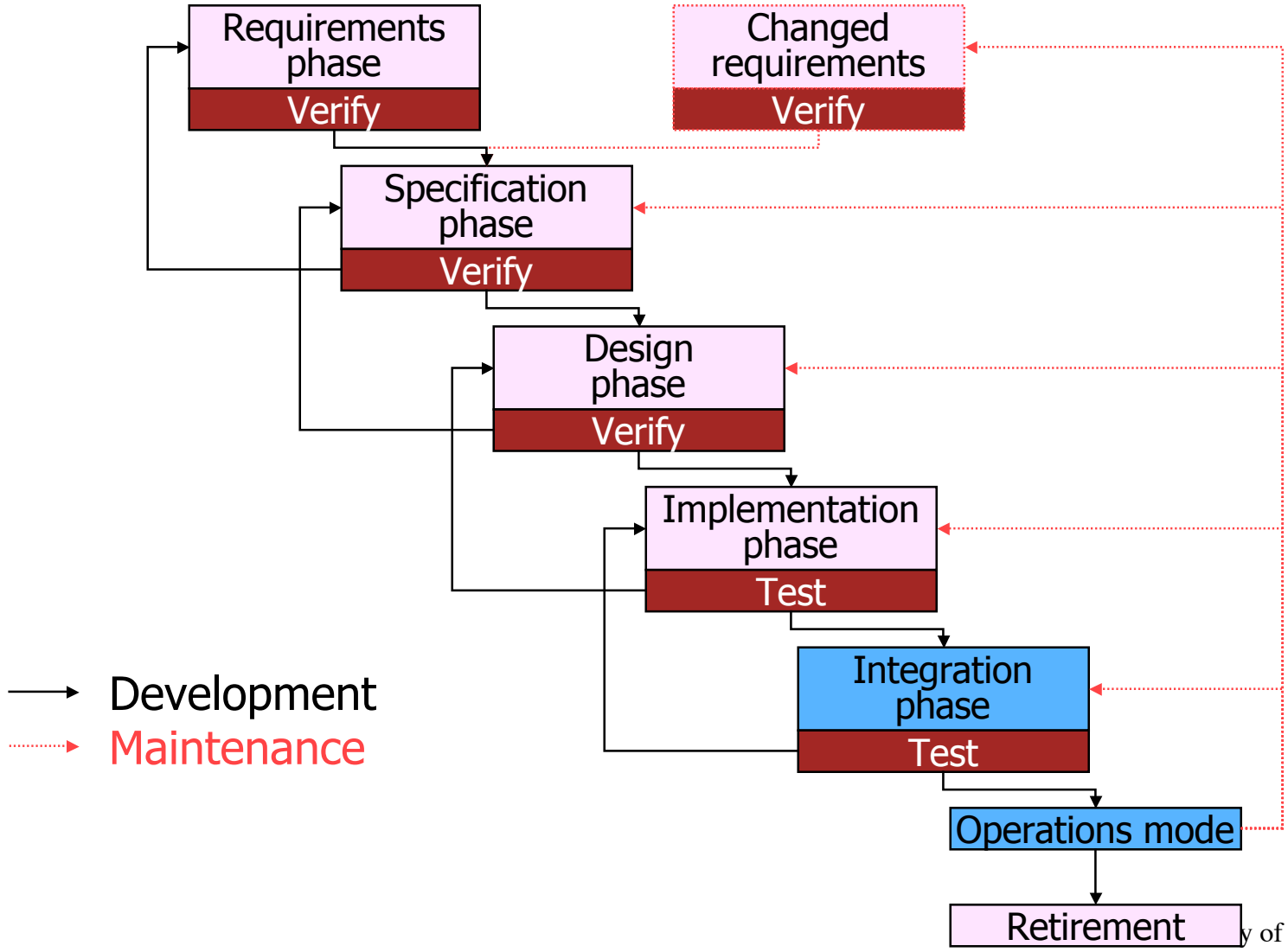
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

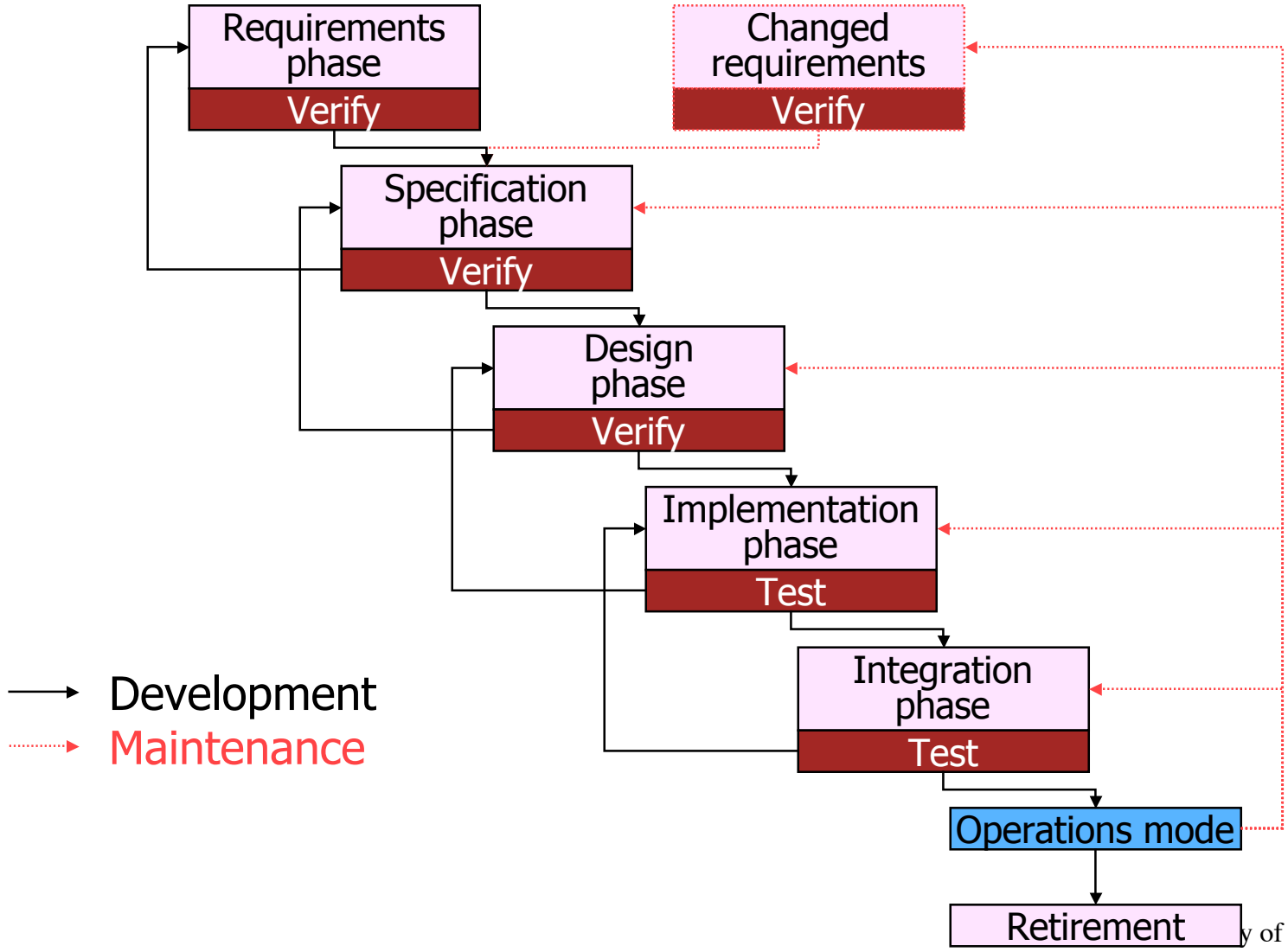
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

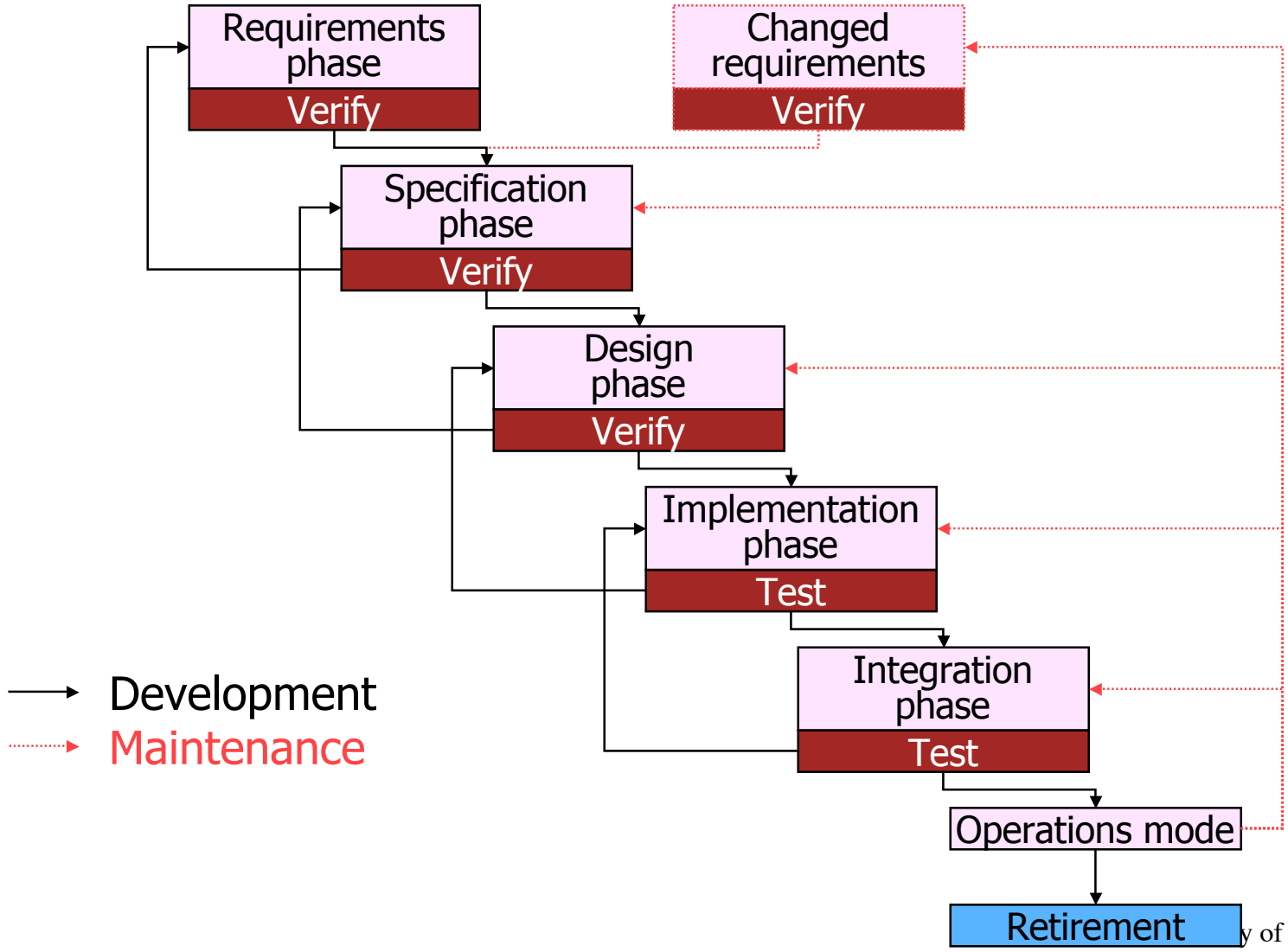
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

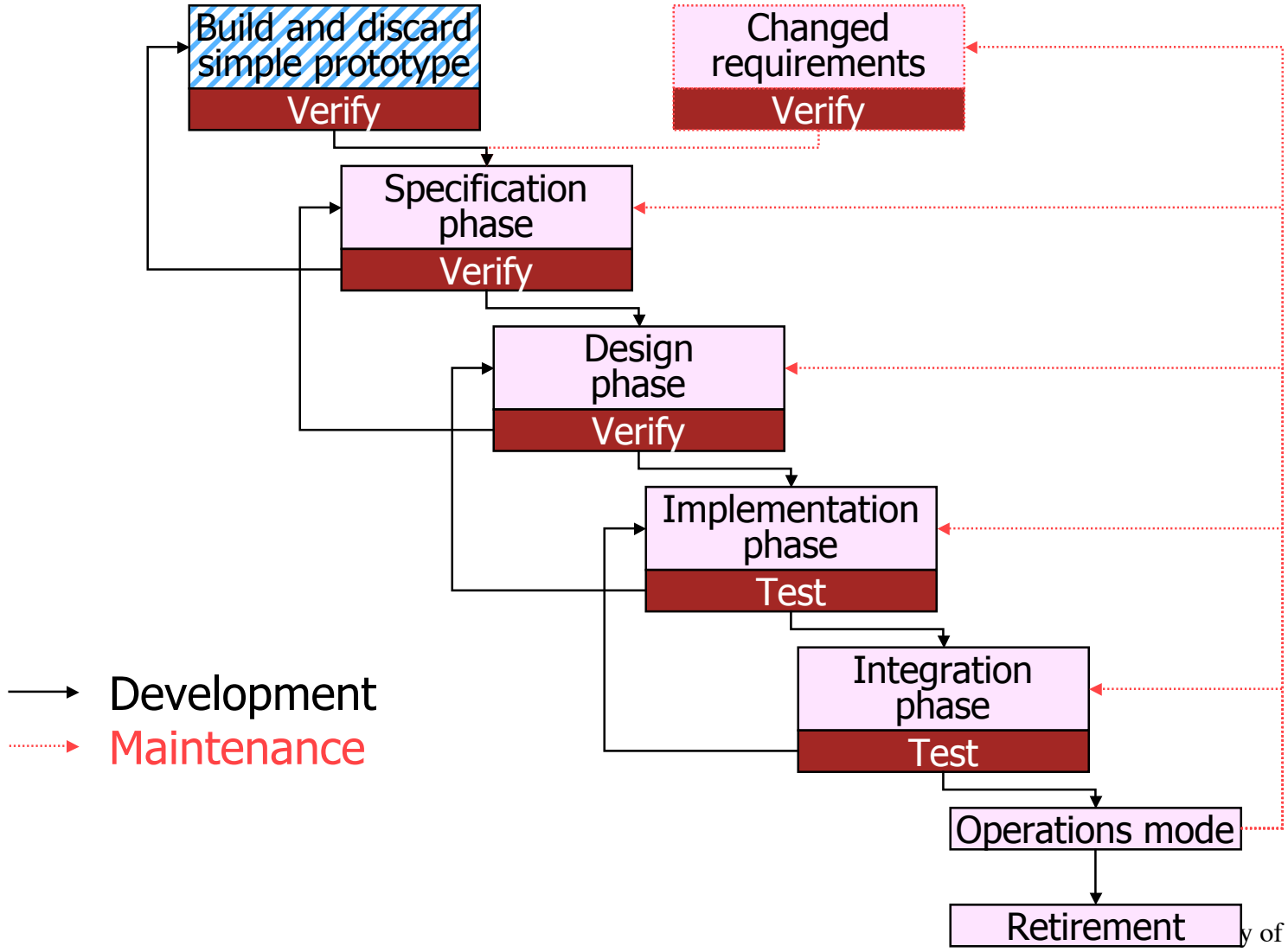
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

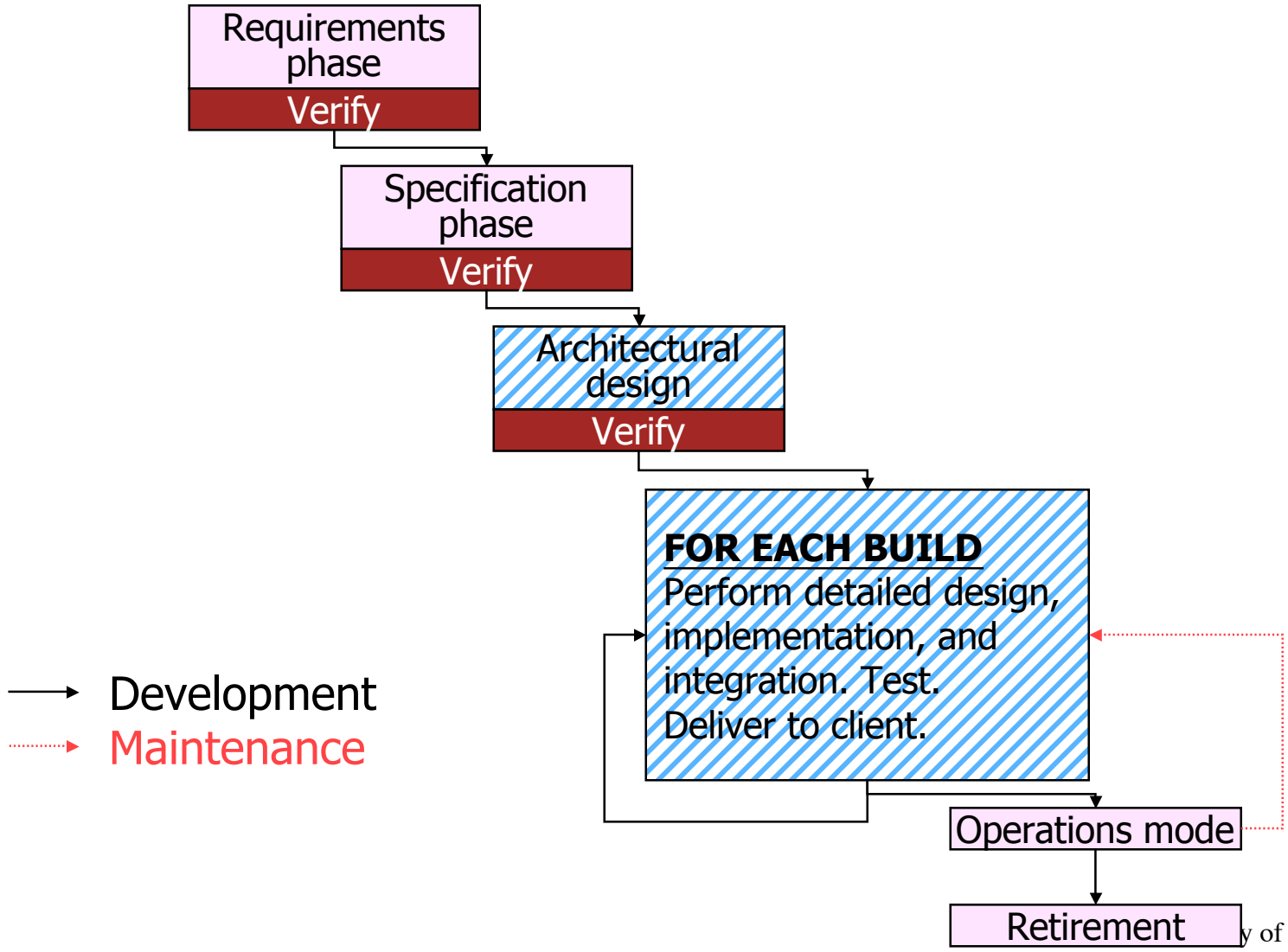
Waterfall


Requirements phase
Verify
Changed requirements
Verify
Specification phase
Verify
Design phase
Verify
Implementation phase
Test
Integration phase
Test
Operations mode
Retirement
Development
Maintenance

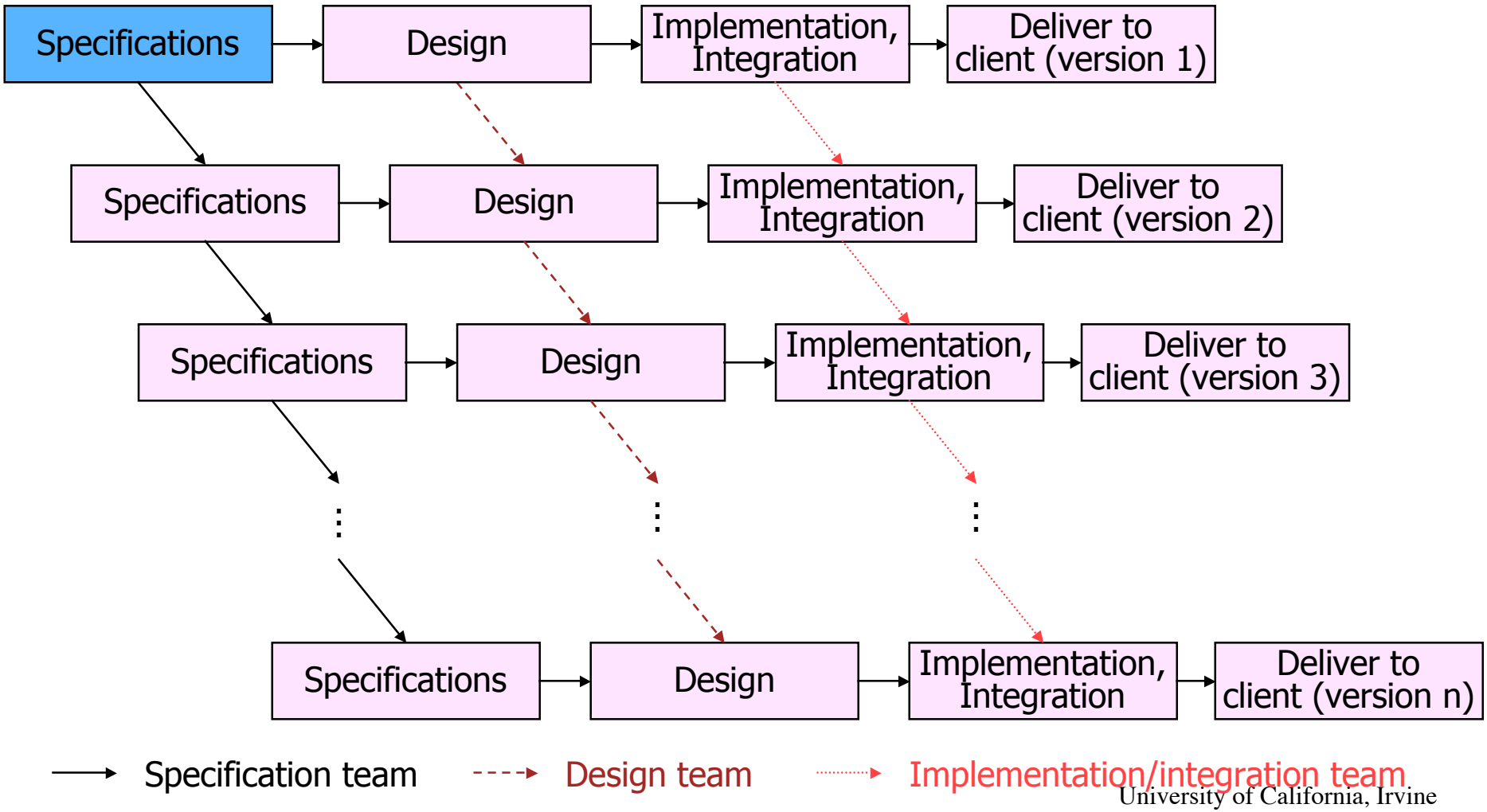
Rapid Prototyping



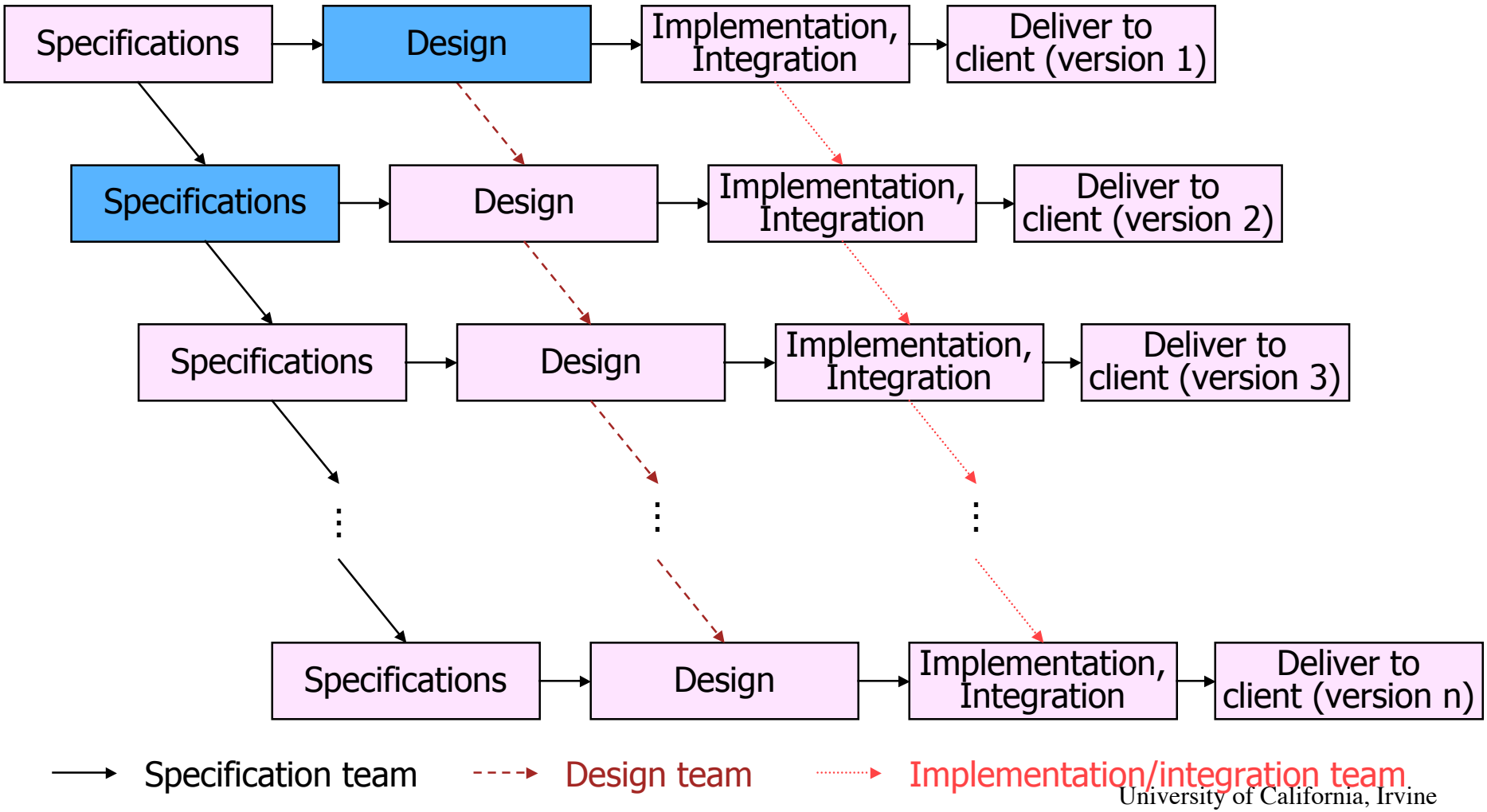
Incremental



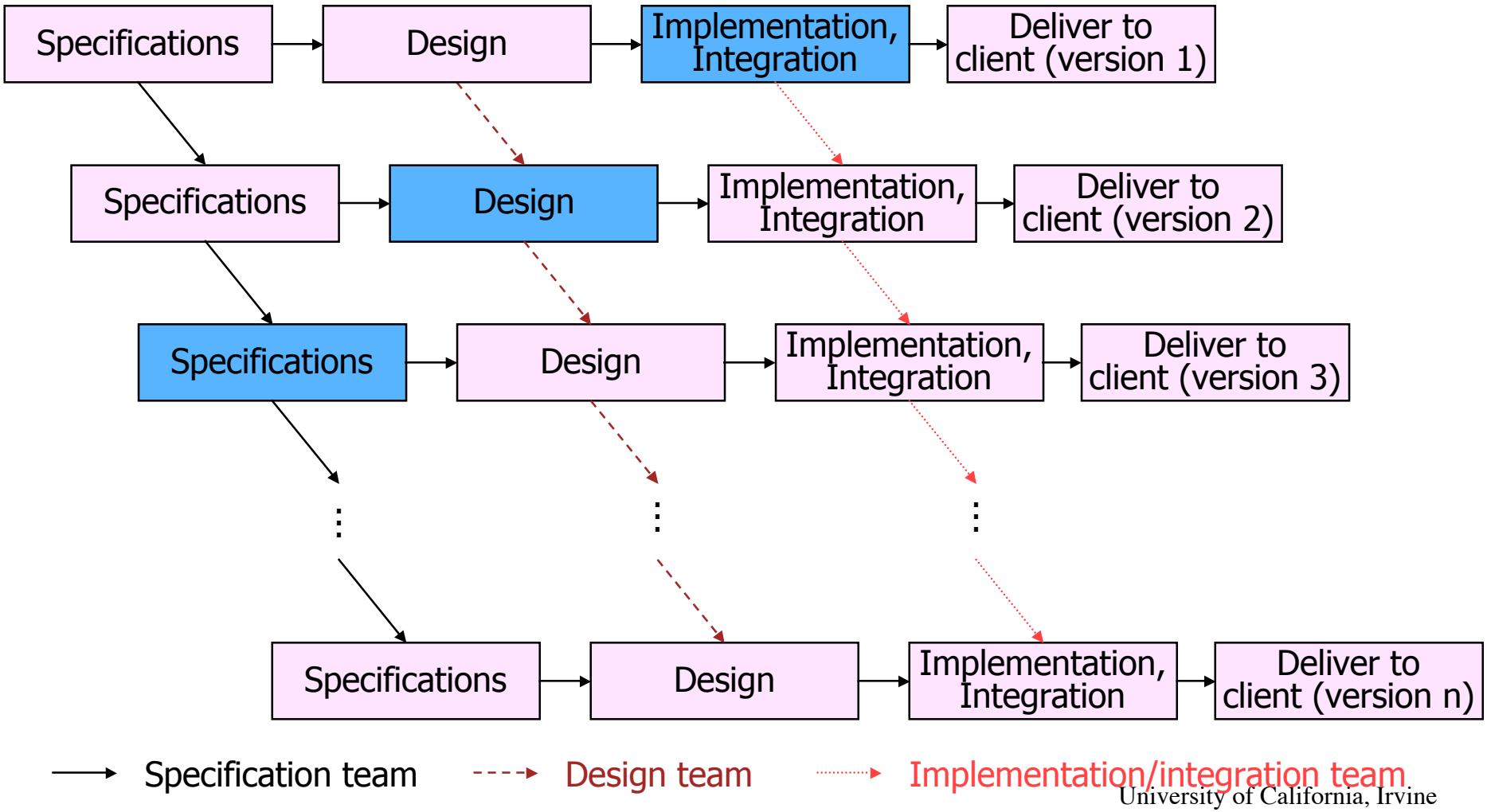
Synchronize-and-Stabilize



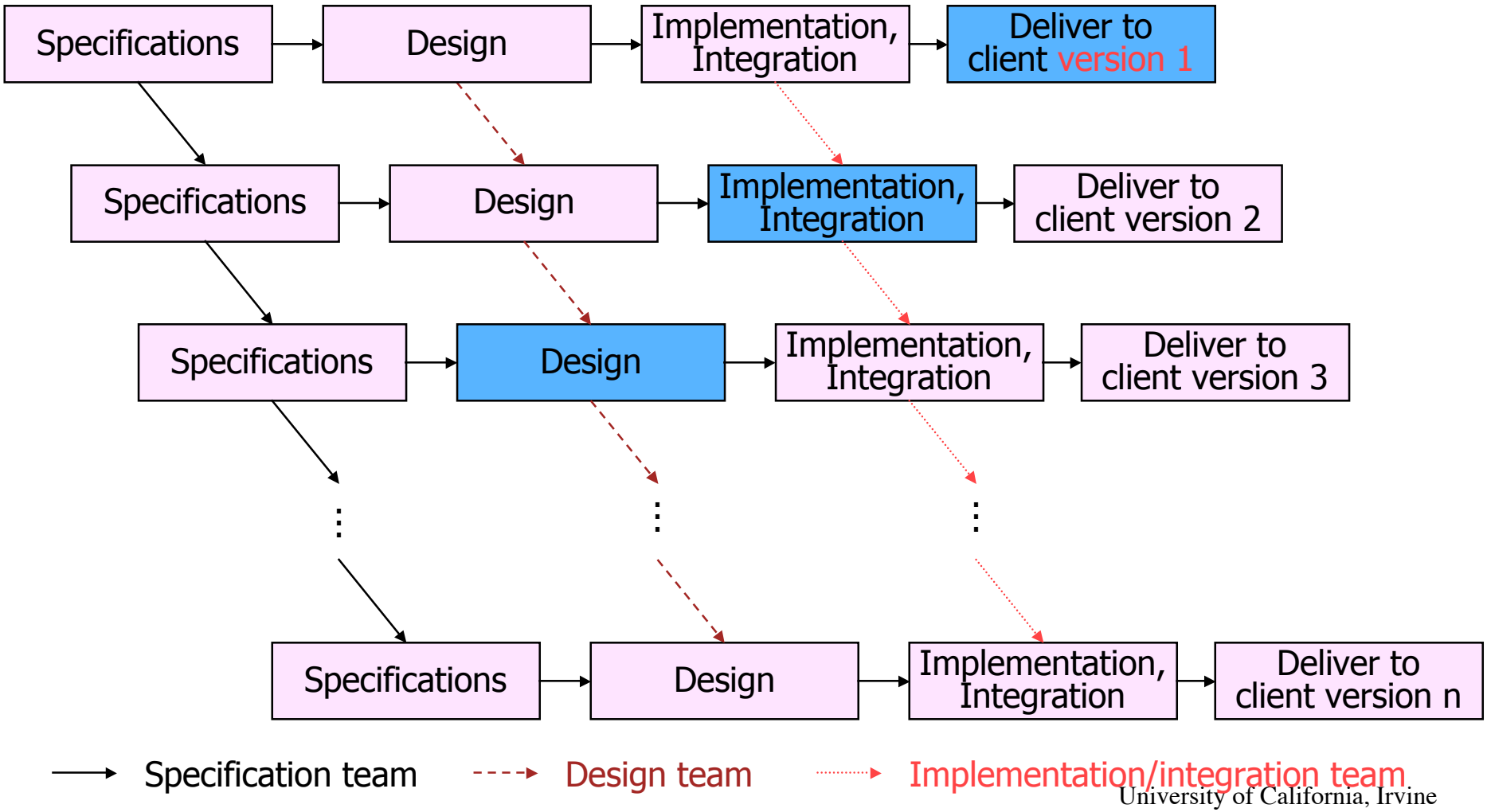
Synchronize-and-Stabilize



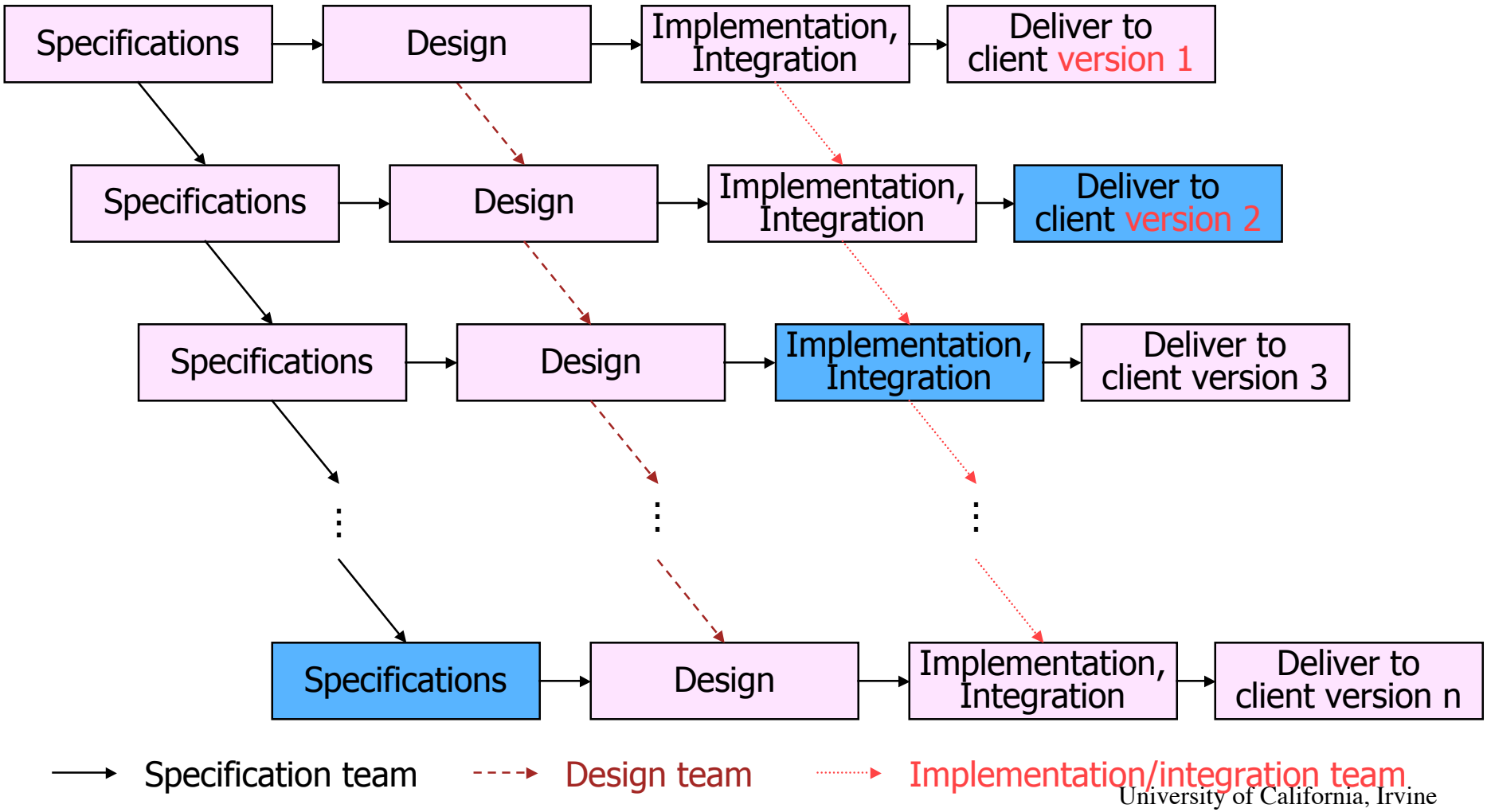
Synchronize-and-Stabilize



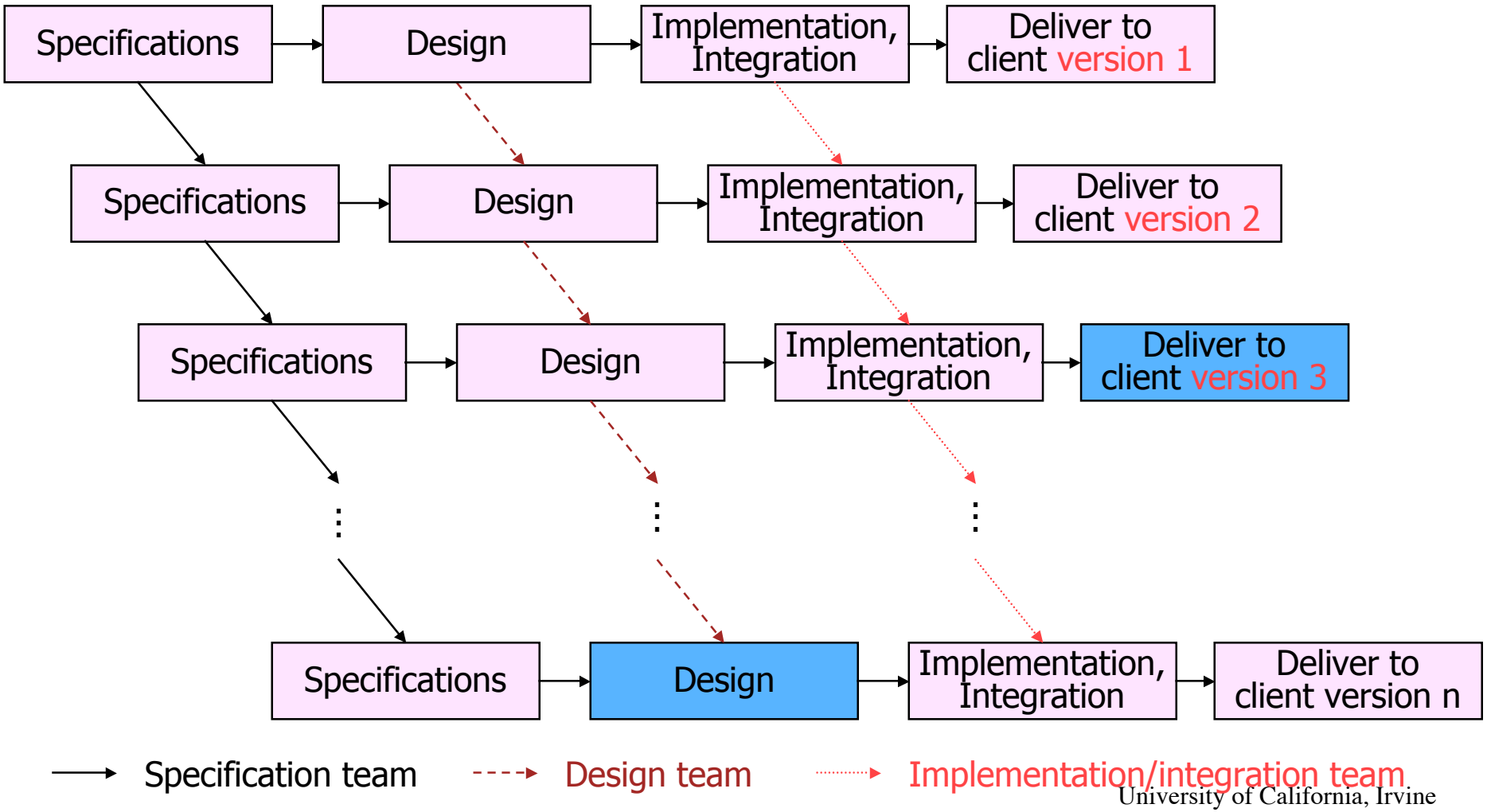
Synchronize-and-Stabilize



Synchronize-and-Stabilize



Synchronize-and-Stabilize



Lifecycle Models

- Waterfall Model (Winston Royce)
  - Centered on defining documents that describe intermediate products
  - User feedback and changes accommodated as an afterthought
- Spiral Model (Barry Boehm)
  - Iterative development model
  - Centered on risk analysis
  - Directly includes prototyping and user feedback

Spiral Model

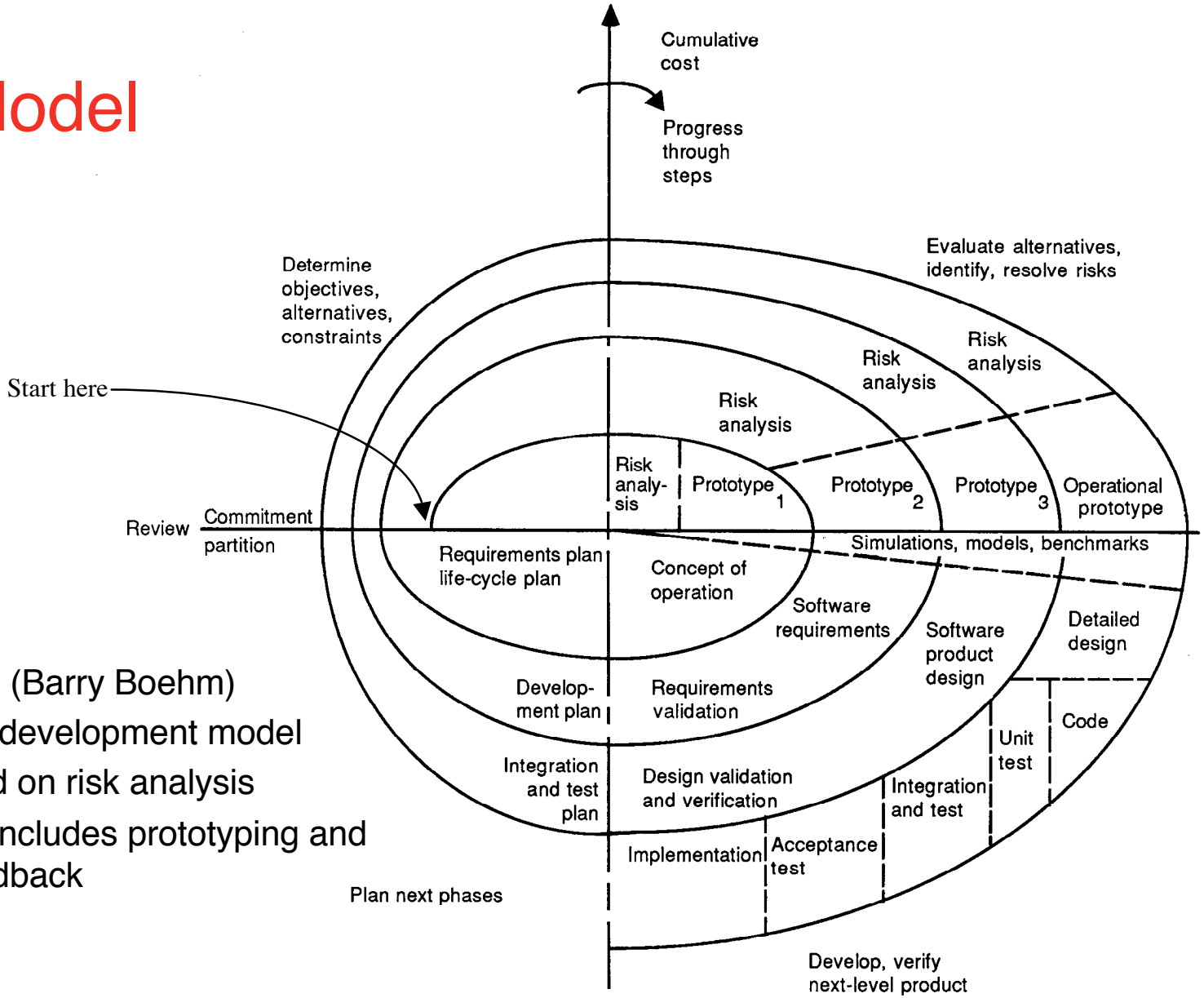


Figure 2. Spiral model of the software process.

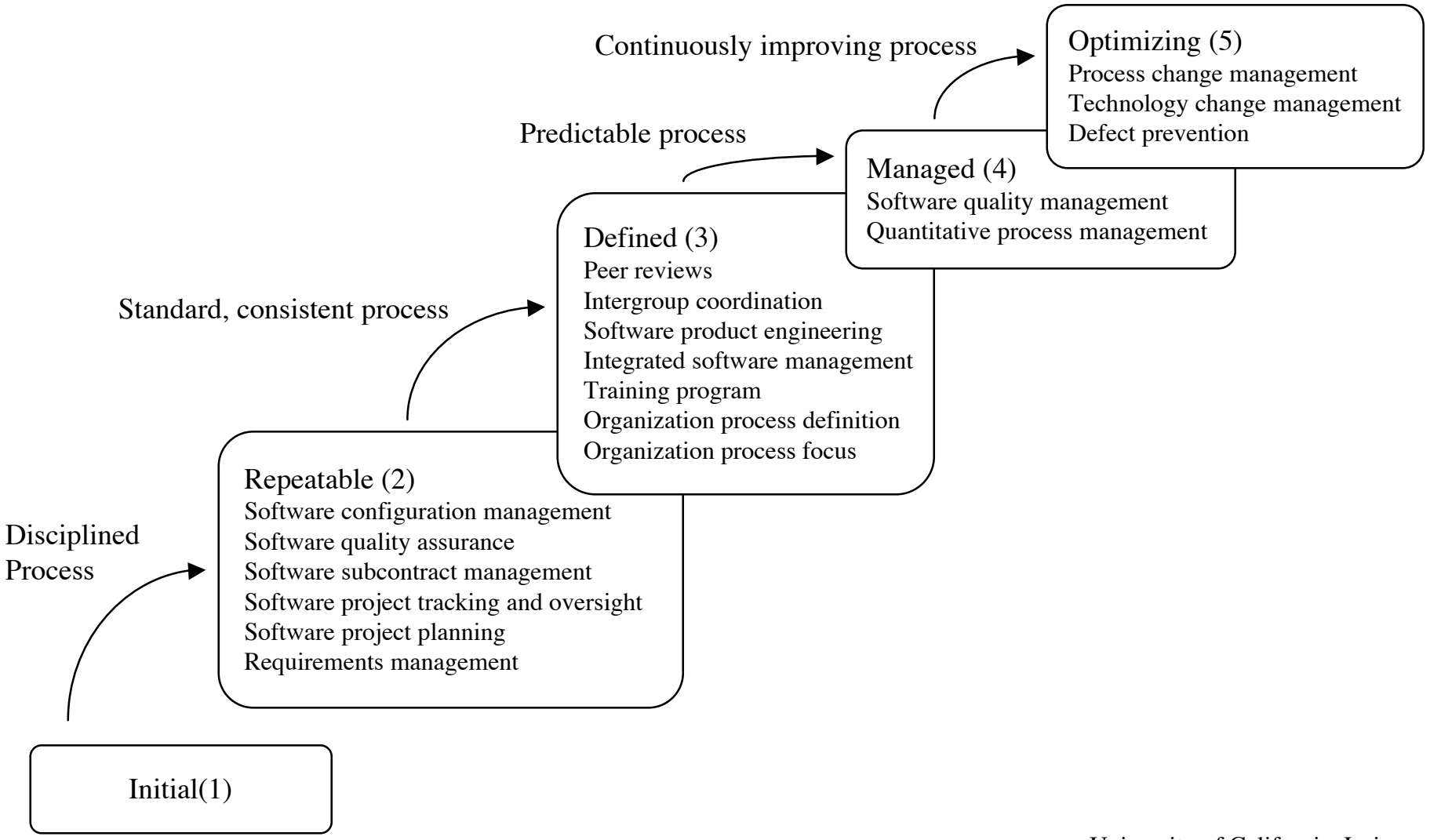
- Spiral Model (Barry Boehm)
  - Iterative development model
  - Centered on risk analysis
  - Directly includes prototyping and user feedback

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

Boehm's Top Ten Software Risks

1. Personnel shortfalls
2. Unrealistic schedules and budgets
3. Developing the wrong software functions
4. Developing the wrong user interface
5. "Gold plating"
6. Continuing stream of requirements changes
7. Shortfalls in externally furnished components
8. Shortfalls in externally performed tasks
9. Real-time performance shortfalls
10. Straining computer-science capabilities

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

Distribution of Software Costs: the Small Matter of Maintenance…

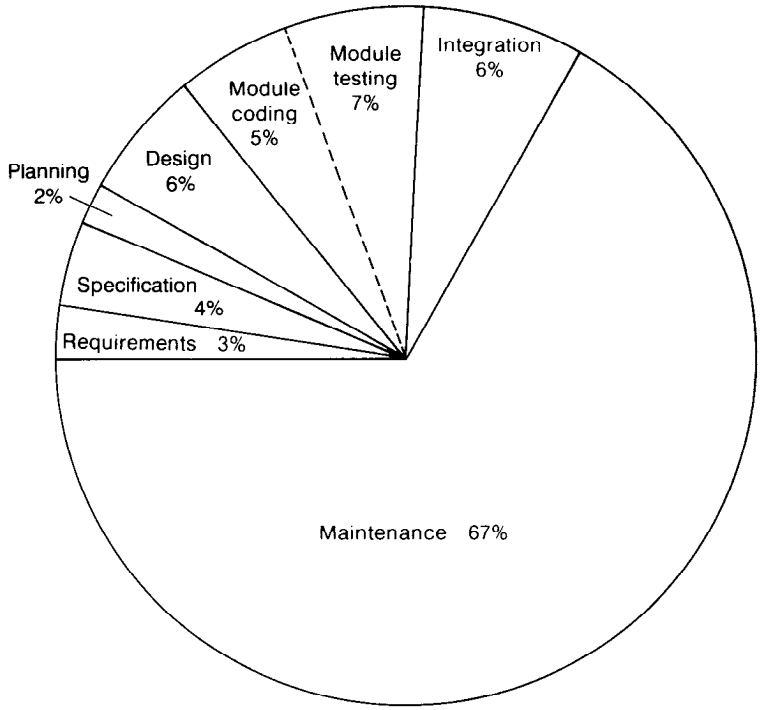


Figure 1.2 Approximate relative costs of the phases of the software process.

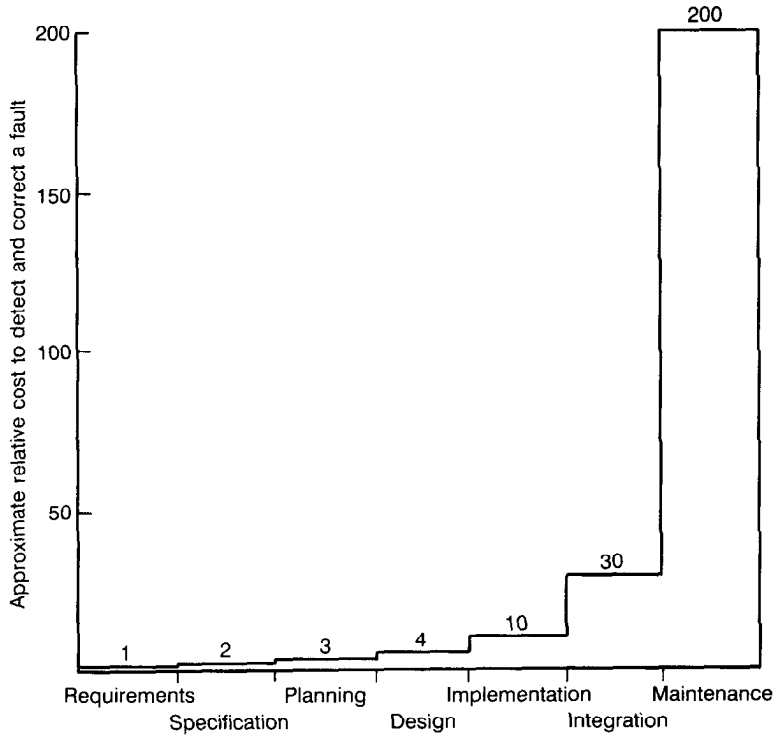


Figure 1.4 Points on solid line of Figure 1.3 plotted on a linear scale.

ICS 52 Software Life Cycle

- Requirements specification
  - Has been: Interview customer (TA)
  - This time: Based on examination of an existing product
  - Focus on “what”, not “how”
- Architectural and module design
  - Has been: Based on provided “official” requirements specification
  - This time: Based on examination and extension of an existing product
  - Focus on interfaces
- Implementation
  - Has been: Based on provided “official” design
  - This time: Extend existing product
  - Focus on good implementation techniques
- Testing
  - Has been: Based on provided “official” implementation
  - This time: You guessed it…
  - Focus on fault coverage and discovery