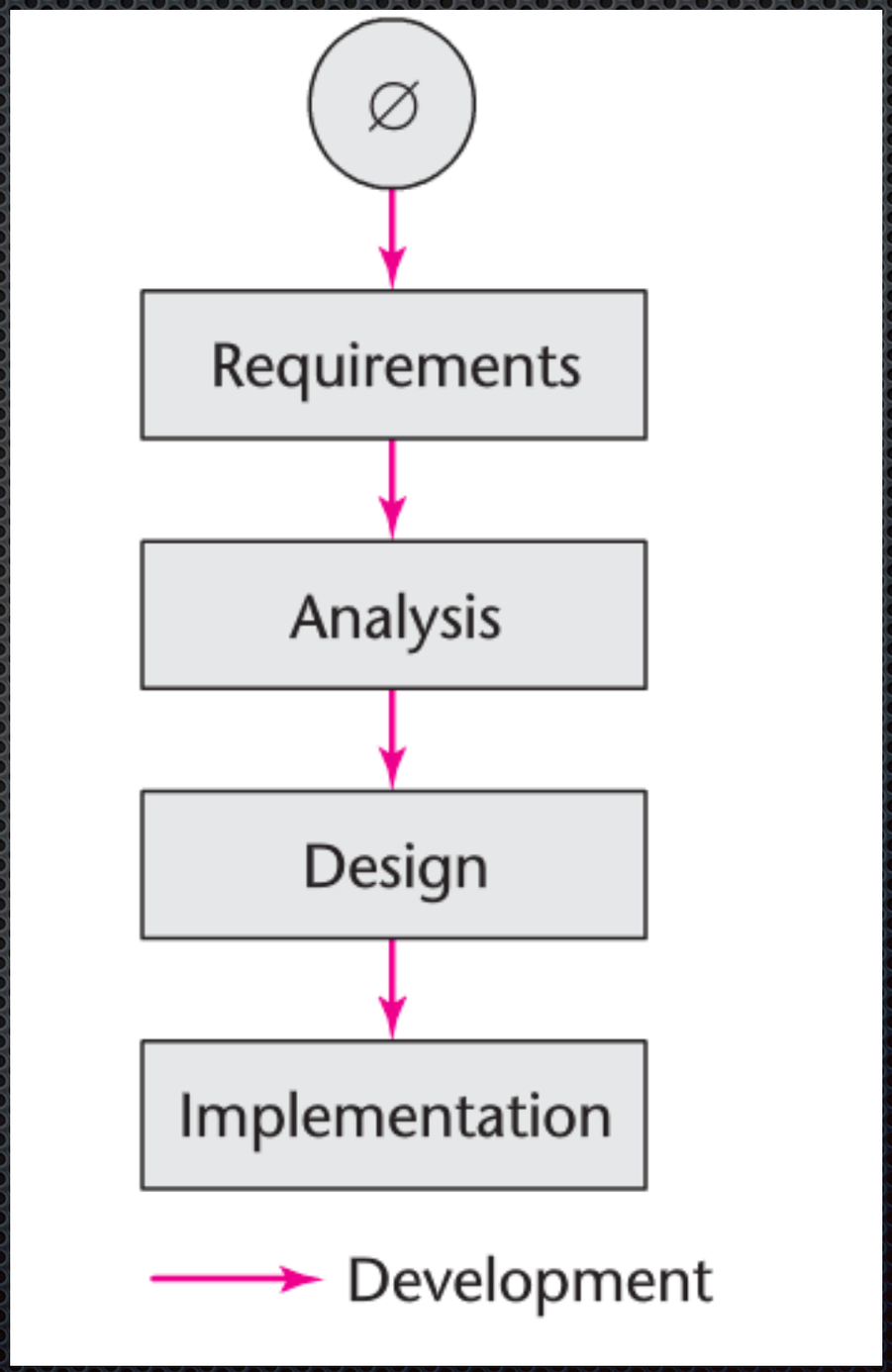
Software Life Cycle Models

CAC 430

Object Oriented and Classical Software Engineering, 8th Edition

- Stephen Schach

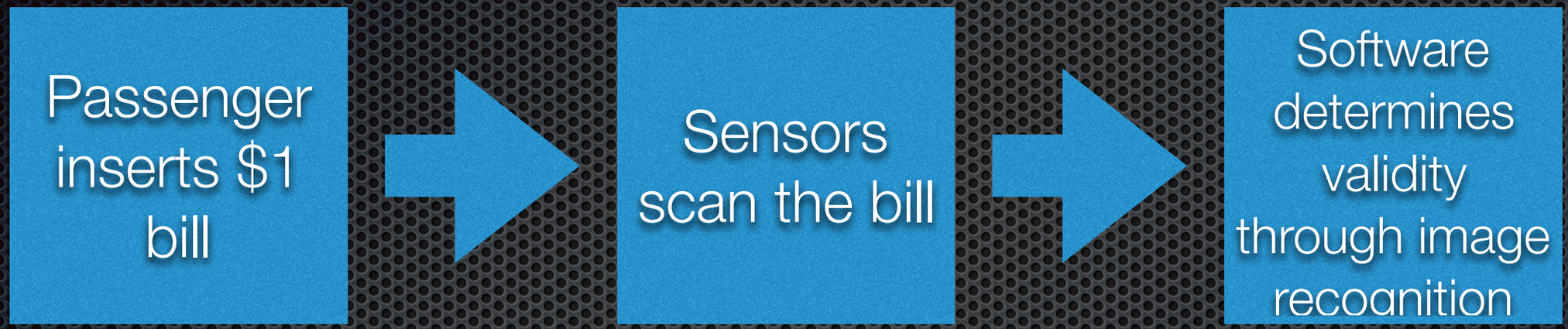
Software Development In Theory



Winburg Mini Case Study

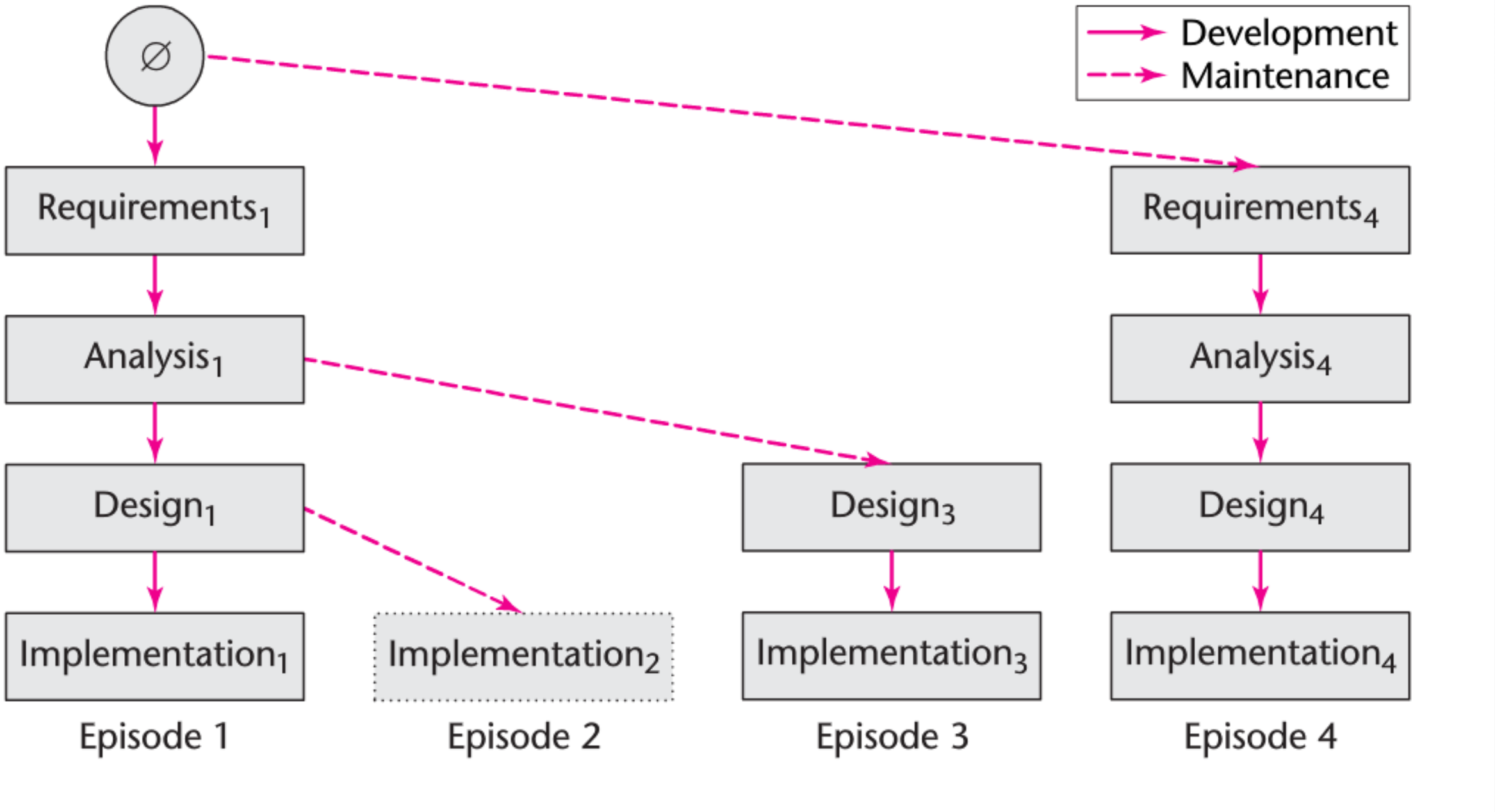
- Mayor convinces the city to set up a public transportation system in Winburg, Indiana
- Commuters encouraged to “park and ride” - $1 per ride
- Each bus has a fair machine that accepts only dollar bills

Winburg Mini Case Study



- Machine must be accurate, 98%
- Must be rapid, less than 1 sec

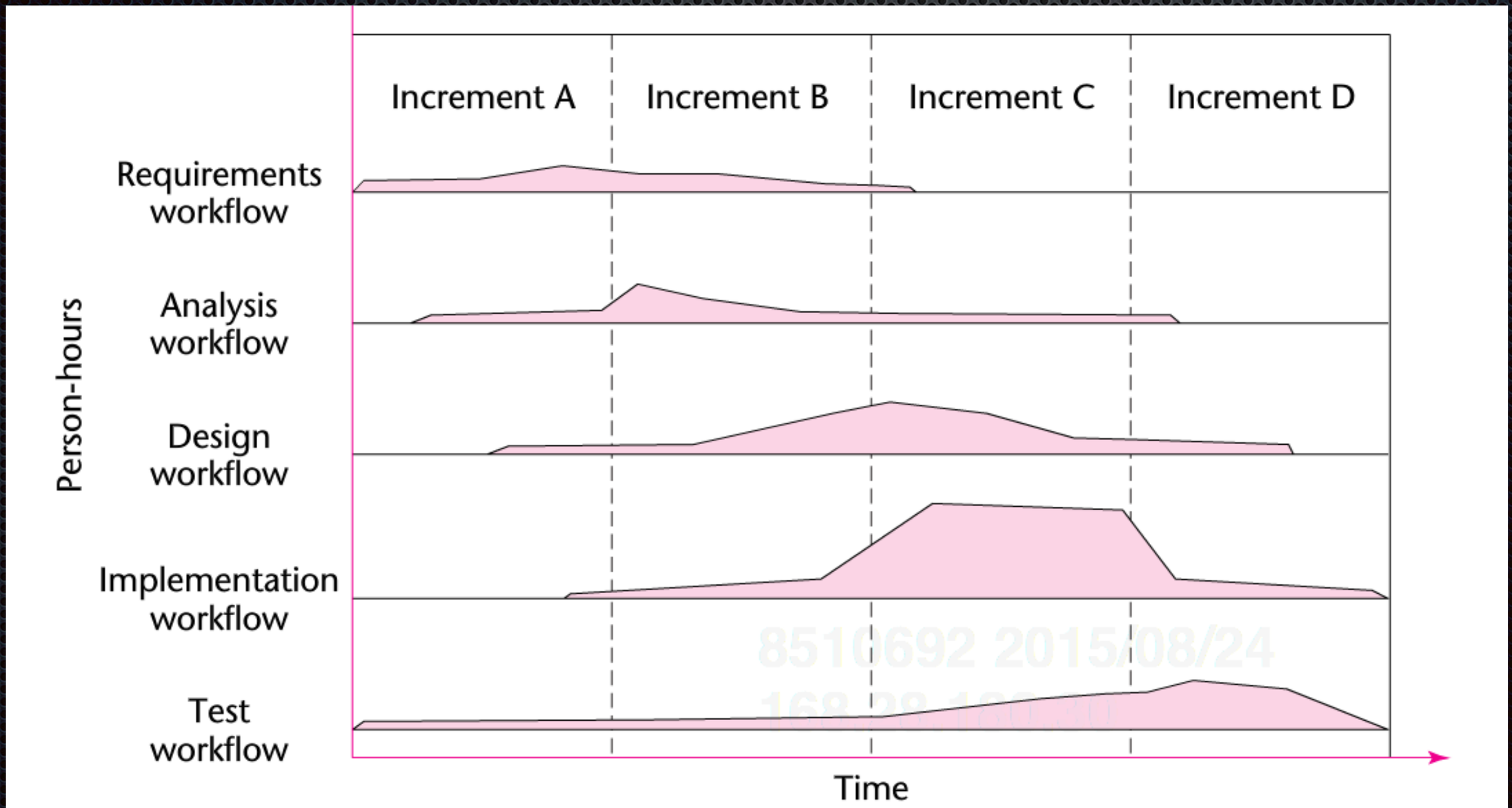
Winburg Mini Case Study



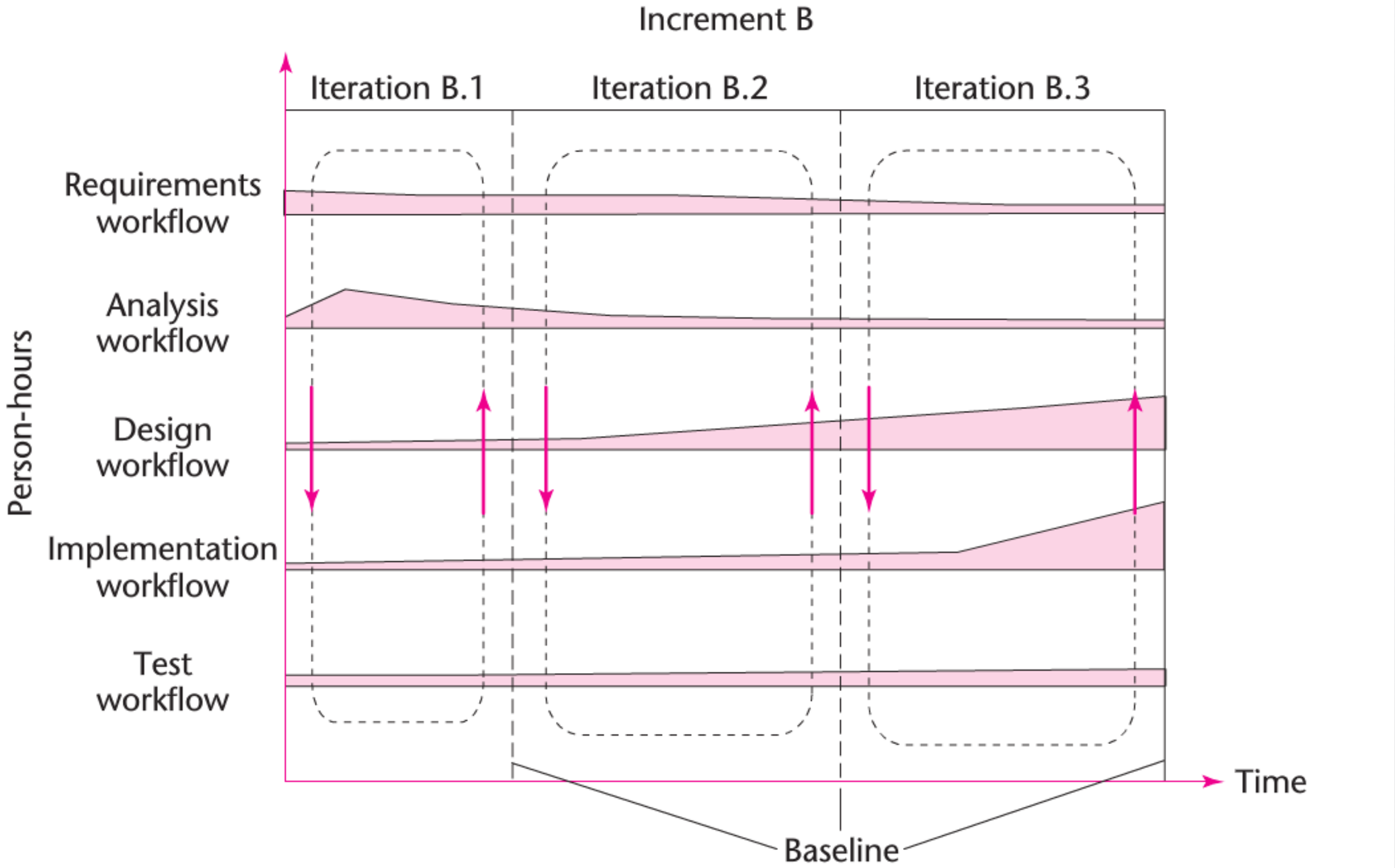
Terminology

- Feature Creep
- Moving-Target Problem
- Regression Fault
- Iterative Life-Cycle Models
- Incremental Software Development
  - Miller's Law

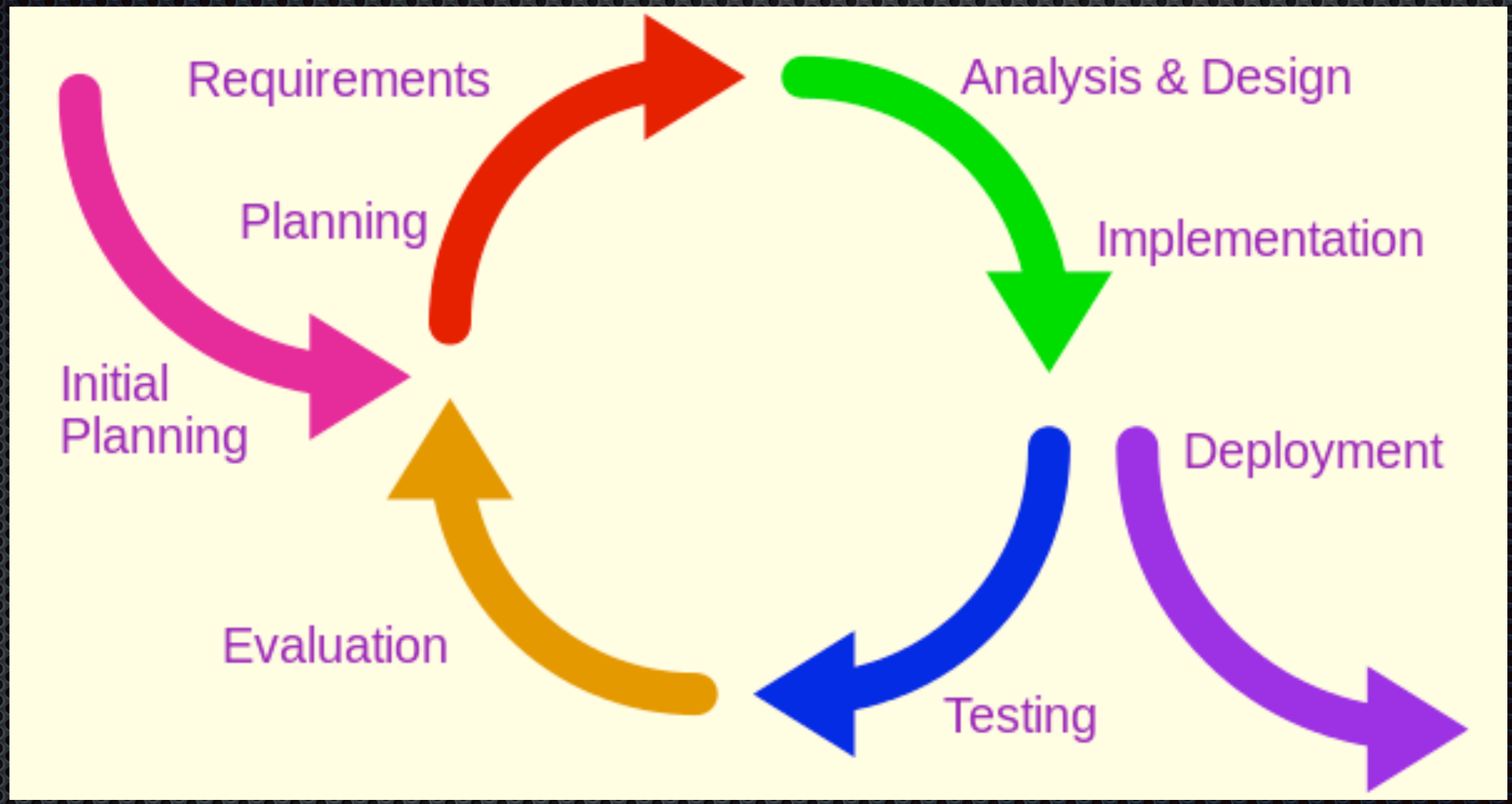
Incremental Software Development Example



…with Iteration

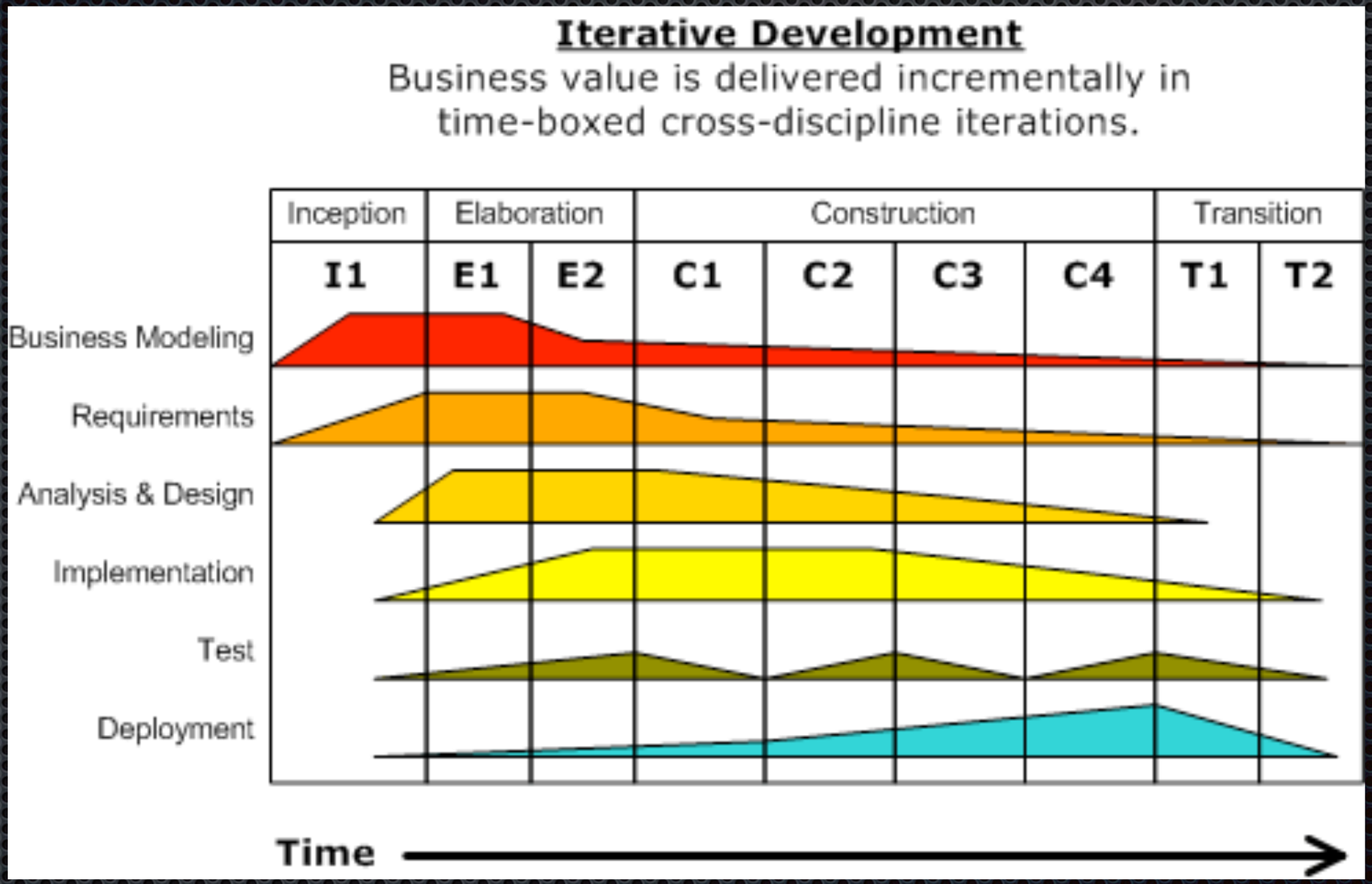


Incremental and Iterative Development



https://en.wikipedia.org/wiki/Iterative_and_incremental_development

Incremental and Iterative Development

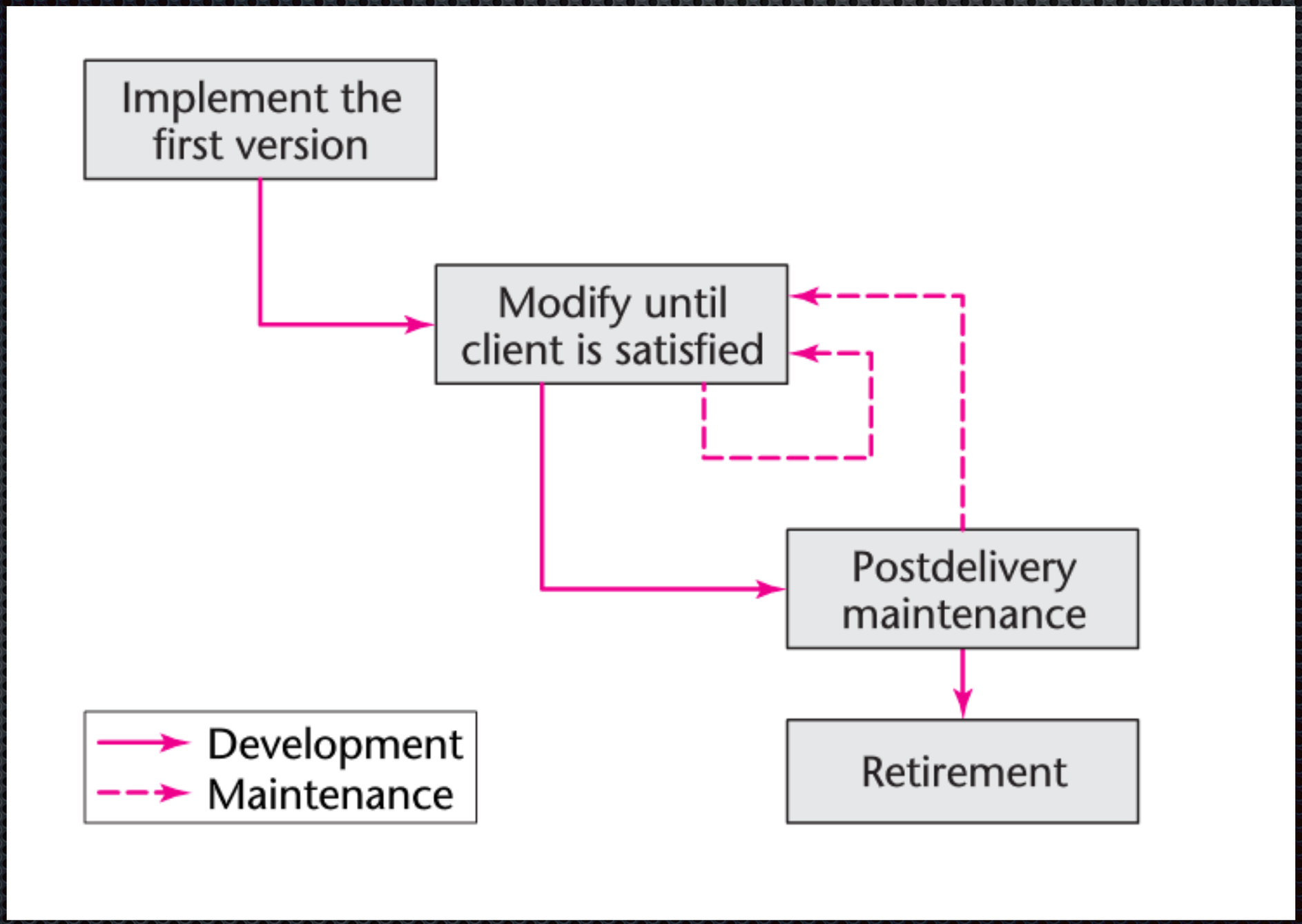


https://en.wikipedia.org/wiki/Iterative_and_incremental_development

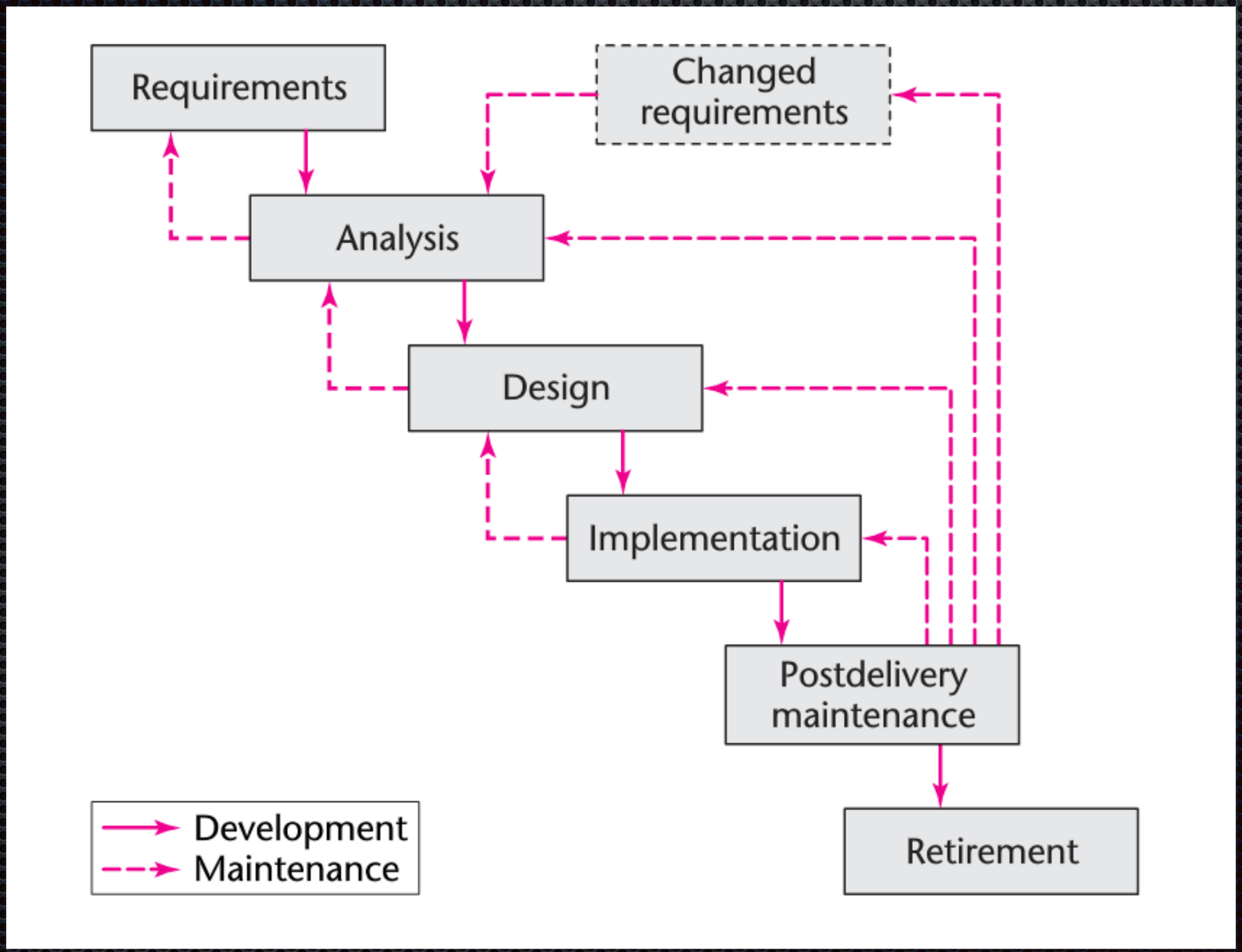
Iterative-and-Incremental Model

- Multiple opportunities are offered for checking that the software product is correct
- The robustness of the underlying architecture can be determined relatively early in the life cycle
- The iterative-and-incremental model enables us to mitigate risks early
- We always have a working version of the software
- There is empirical evidence that the iterative-and-incremental life cycle works

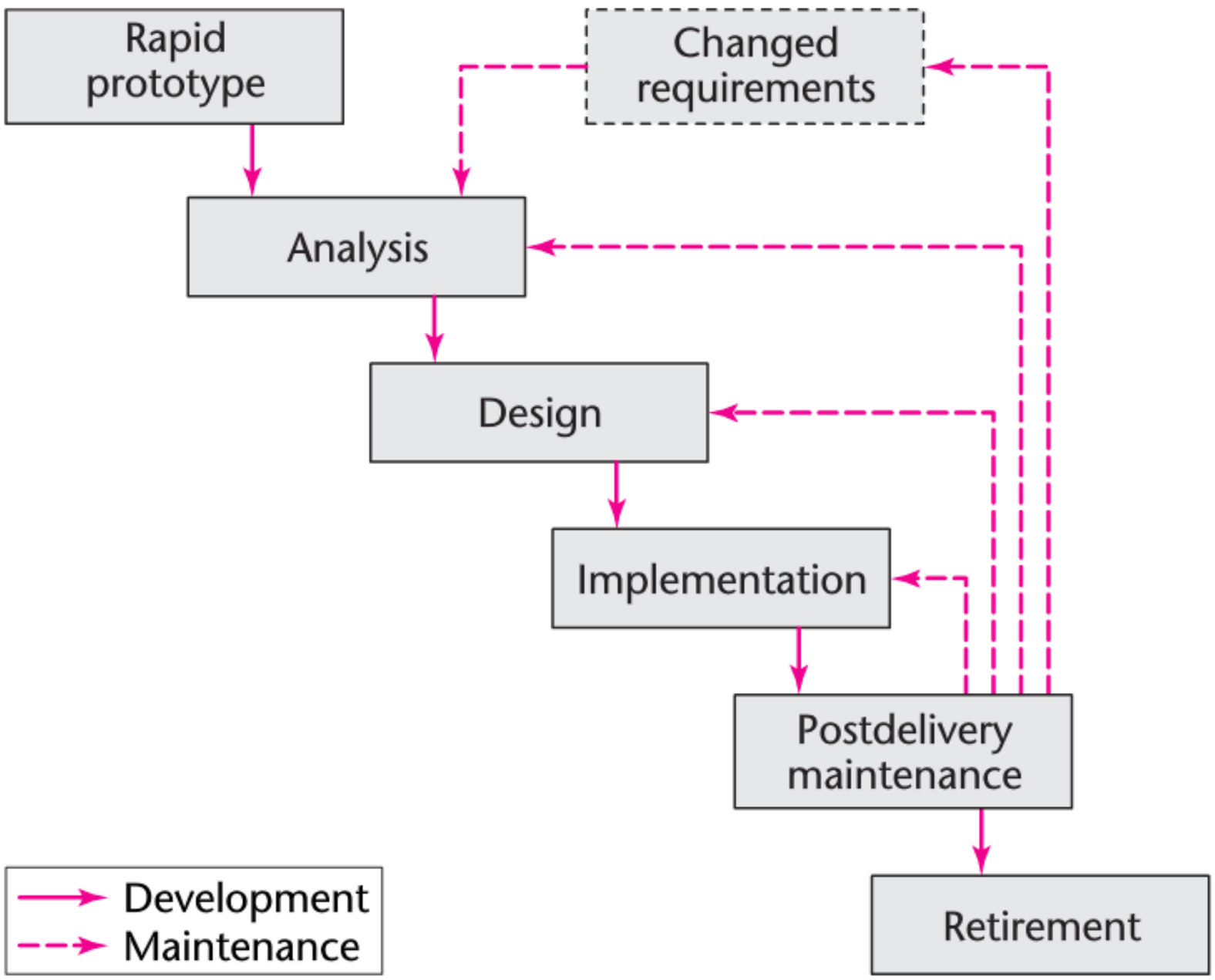
Code-and-Fix Model



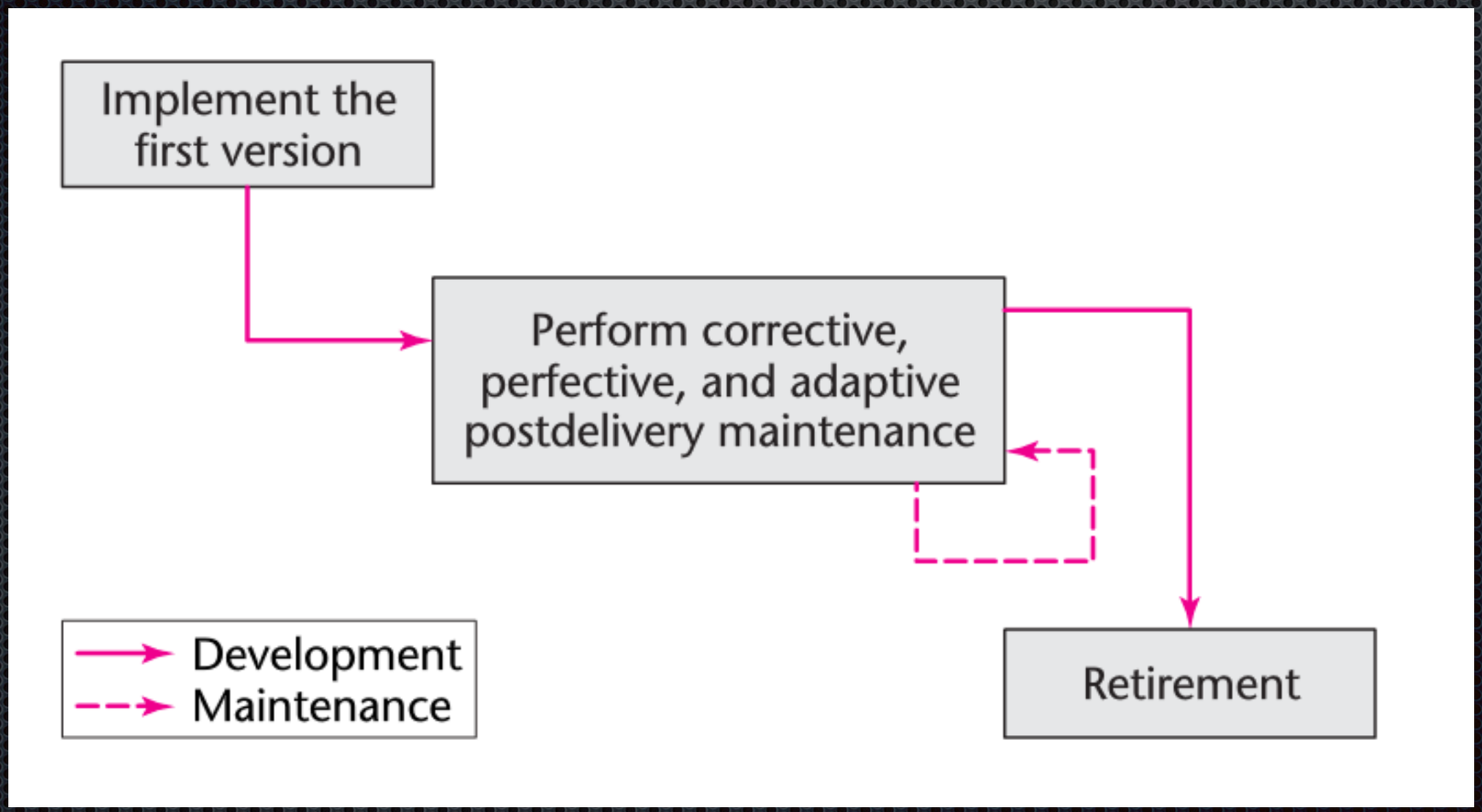
Classic Waterfall



Rapid-Prototyping



Open-Source Software Model



Open-Source vs. Closed-Source

- Closed-Source: failure reports
- Open-Source: fault reports
- Open-Source: Volunteers (core group and peripheral group)
- Closed-Source: scheduled releases checked by SQA
- Open-Source: “Release early. Release often.”

Agile Processes

- Manifesto for Agile Software Development
- Extreme Programming
  - Stories/Features
  - Client chooses features to start with
  - Test Driven Development (TDD)
  - Pair Programming

Agile Processes

- Timeboxing
  - Set amount of time for each iteration
- Stand Up Meeting
  - What have I done since yesterday's meeting?
  - What am I working on today?
  - What problems are preventing me from achieving this?
  - What have we forgotten?
  - What did I learn that I would like to share with the team?

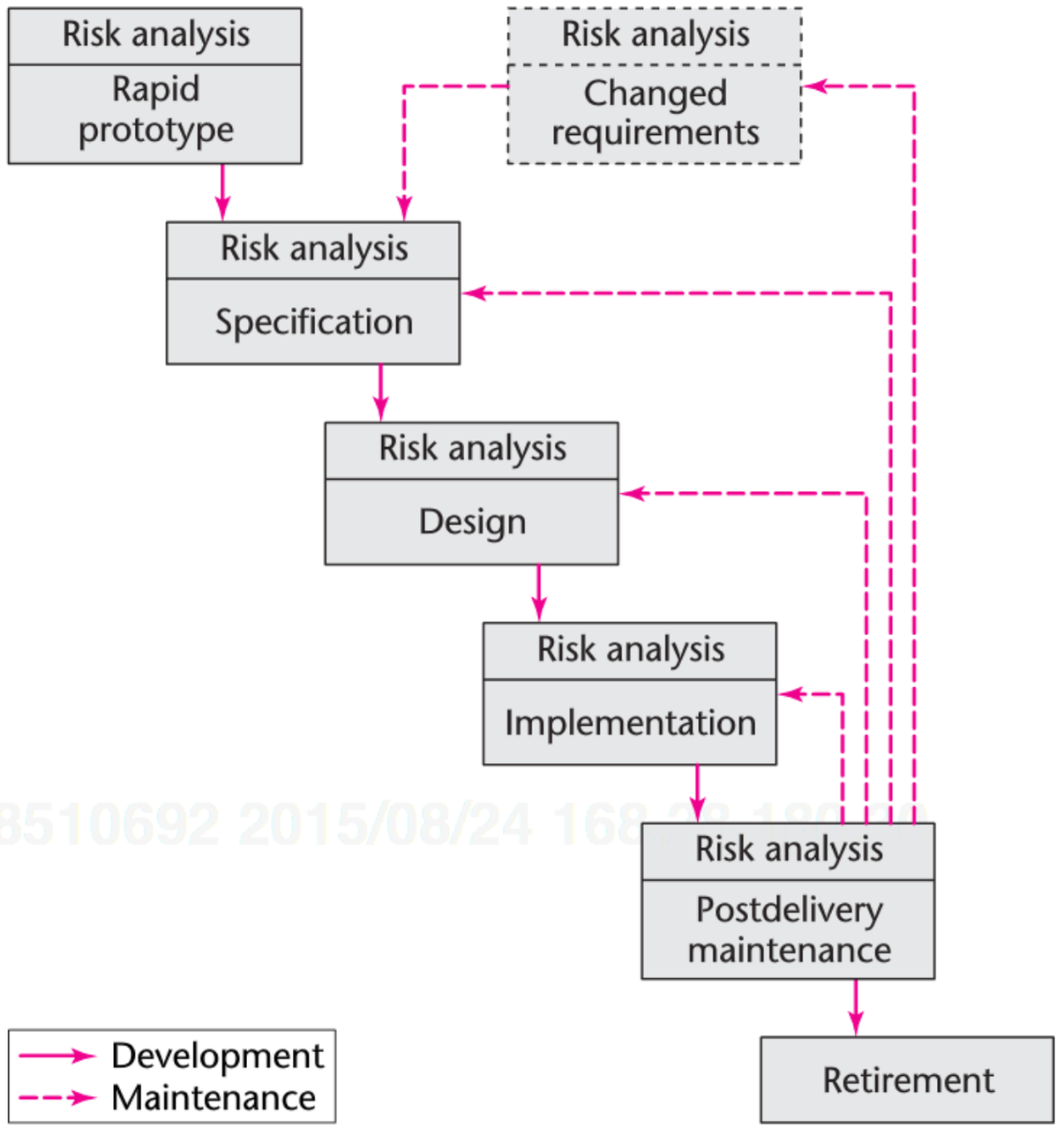
Synchronize-and-Stabilize

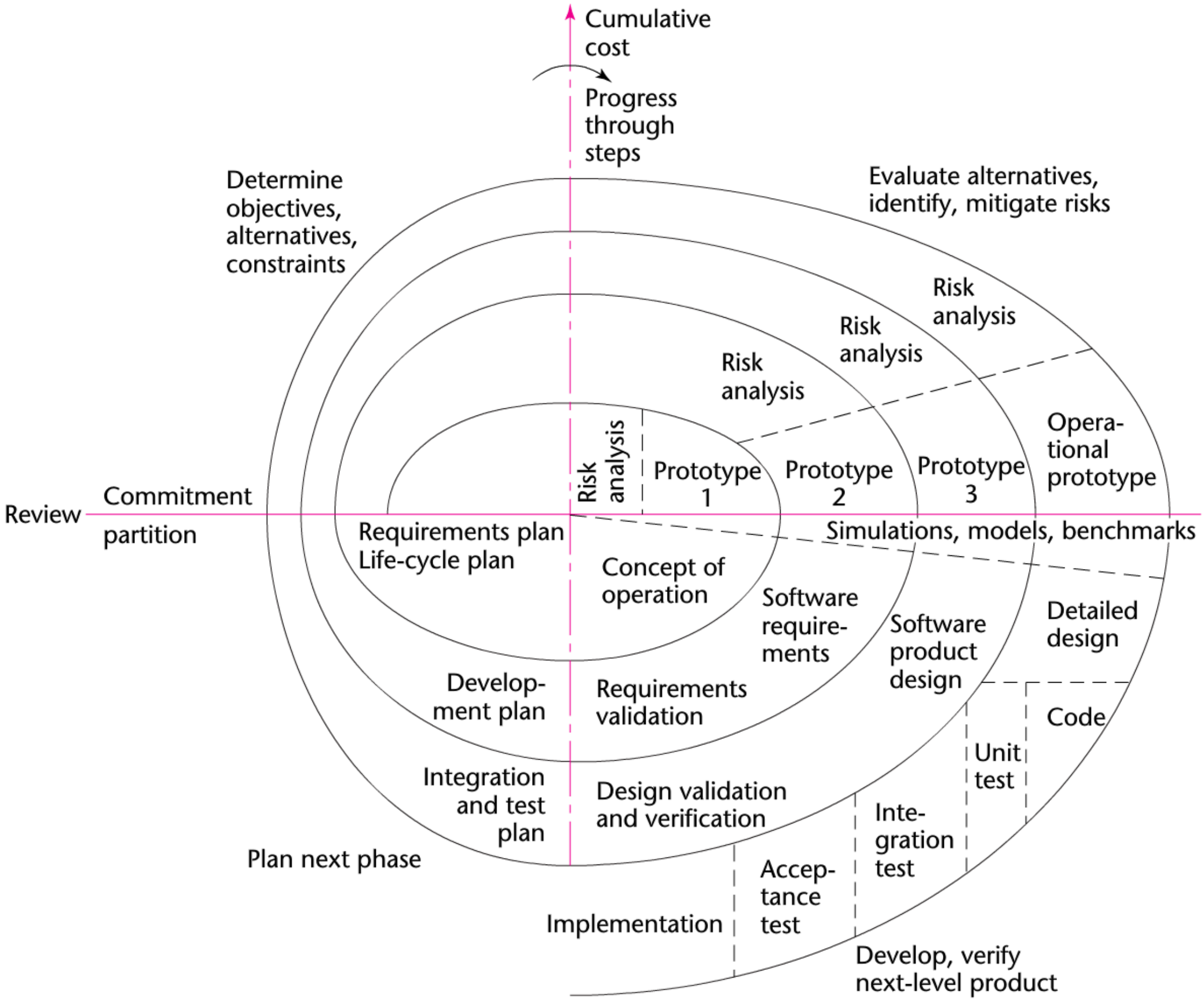
- Interview clients - prioritize features
- Specification document
- Work broken up into 3-4 builds
- Work is **synchronized** at the end of each day
- At the end of each build, the work is **stabilized**
- Once stabilized, the build is frozen

Windows 2000: 30 million lines of code, 3000 programmers

Spiral Life-Cycle Model

FIGURE 2.12
A simplified version of the spiral life-cycle model.




Cumulative cost
Progress through steps
Determine objectives, alternatives, constraints
Evaluate alternatives, identify, mitigate risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Prototype 1
Prototype 2
Prototype 3
Operational prototype
Commitment partition
Review
Requirements plan
Life-cycle plan
Concept of operation
Simulations, models, benchmarks
Software requirements
Software product design
Detailed design
Development plan
Requirements validation
Code
Unit test
Integration and test plan
Design validation and verification
Integration test
Plan next phase
Acceptance test
Implementation
Develop, verify next-level product

Life-Cycle Model	Strengths	Weaknesses
Evolution-tree model (Section 2.2)	Closely models real-world software production Equivalent to the iterative-and-incremental model	
Iterative-and-incremental life-cycle model (Section 2.5)	Closely models real-world software production Underlies the Unified Process	
Code-and-fix life-cycle model (Section 2.9.1)	Fine for short programs that require no maintenance	Totally unsatisfactory for nontrivial programs
Waterfall life-cycle model (Section 2.9.2)	Disciplined approach Document driven	Delivered product may not meet client's needs
Rapid-prototyping life-cycle model (Section 2.9.3)	Ensures that the delivered product meets the client's needs	Not yet proven beyond all doubt
Open-source life-cycle model (Section 2.9.4)	Has worked extremely well in a small number of instances	Limited applicability
Agile processes (Section 2.9.5)	Work well when the client's requirements are vague	Usually does not work
Synchronize-and-stabilize life-cycle model (Section 2.9.6)	Future users' needs are met Ensures that components can be successfully integrated	Appear to work on only small-scale projects Has not been widely used other than at Microsoft
Spiral life-cycle model (Section 2.9.7)	Risk driven	Can be used for only large-scale, in-house products Developers have to be competent in risk analysis and risk resolution

Your Turn

- Homework activity
- Wednesday: Requirements/Case study