Chapter 2 – Process Models

Overview

- The roadmap to building high quality software products is software process.
- Software processes are adapted to meet the needs of software engineers and managers as they undertake the development of a software product.
- A software process provides a framework for managing activities that can very easily get out of control.
- Modern software processes must be agile, demanding only those activities, controls, and work products appropriate for team or product.
- Different types of projects require different software processes.
- The software engineer's work products (programs, documentation, data) are produced as consequences of the activities defined by the software process.
- The best indicators of how well a software process has worked are the quality, timeliness, and long-term viability of the resulting software product.

Software Process

- Framework for the activities, actions, and tasks required to build high quality software
- Defines approach taken as software is engineered
- Adapted by creative, knowledgeable software engineers so that it is appropriate for the products they build and the demands of the marketplace

Generic Process Framework

- Communication
- Planning
- Modeling
- Construction
- Deployment

Umbrella Activities (applied throughout process)

- Software project tracking and control
- Risk management
- Software quality assurance
- Formal technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Process Flow

- Describes how each of the five framework activities, actions, and tasks are organized with respect to sequence and time
- *Linear process flow* executes each of the framework activities in order beginning with communication and ending with deployment
- *Iterative process flow* executes the activities in a circular manner creating a more complete version of the software with each circuit or iteration
- *Parallel process flow* executes one on more activities in parallel with other activities

軟體生命週期

- 軟體由產生至結束的演繹過程
- 軟體由一群程式碼模組與資料組成
  - 軟體如何產生?
  - 軟體運作時會因什麼緣故而作改變?
  - 軟體退休前該作何處理?
- 分7 phases(階段): requirements, specification, design, implementation, integration, maintenance, and retirement.
- 前五階段:軟體發展

軟體生命週期與軟體流程
(Software life cycle and software process)

Requirements Phase (1 of 3)

- 任務：界定軟體的目的、應有用途、及限制條件
- 軟體產生的動機: Client tries to use the product to solve his problem.
  - client與 user 的差別
  - 解決什麼問題？問題產生的背景
  - 用此軟體後，希望應得到什麼效益？(效益衡量指標及計算方式)
- 了解問題才能提出適切解決方案
- 明確投資效益分析有助雙方共識

Requirements Phase (2 of 3)

- 主要工作:
  - 瞭解client 的動機及問題
  - 界定與此軟體相關之使用者、外部系統與外部設備
  - 界定使用者、外部系統與設備希望此軟體系統作什麼事(功能性需求 functional requirement), 及軟體運作及發展過程有何限制條件(非功能性需求 nonfunctional requirement,如 response time, security等等)
- Determine what the client needs, *not* what the client wants

Requirements Phase (3 of 3)

- 產出物：(user) requirement document
  - 其內容應包括那些？
  - 非功能性需求可能有那些項目？
  - 用文字描述,使客戶能充分了解與確認
- 關鍵因素:
  - 客戶問題描述是否清楚,是否可用電腦解決
  - 使用者、外部系統、外部設備的充分掌握
  - 使用者、外部系統、設備要系統作的事可否清楚描述,是否有相互衝突或矛盾的情況
  - 如何引導相關者說出其需求

Specification phase(1 of 4)

- 制定軟體系統應有的功能使其可解決客戶的問題
  - 訂定系統提供的功能服務(service),以可滿足使用者、外部系統、與外部設備的功能性需求,且可滿足非功能性需求限制條件←(系統分析師所提的服務應可解決客戶所提的需求)
- Determine what the system should do.（功能）
- Computer : data processing machine(資料）
  - Input/output data types, format, volume ‘frequency, etc., should be considered,.

Specification phase(2 of 4)

- 產出物: Specification document
  - Explicitly describes the functionality of the product (what the product is supposed to do) and lists any constraints that the product must satisfy
  - Inputs to and required outputs of the product should be included.
  - Part of contract (system requirement)
  - User manual and operation manual 是依據specification document而撰寫.
- 需使客戶可了解.

Specification phase(3 of 4)

- Specifications must be traceable to requirement and must not be
  - ambiguous
  - incomplete
  - contradictory
  - And not have phrases like “optimal,” or “98% complete”
- Once the specifications have been signed off, the software product management plan (SPMP) is drawn up.
  - Schedule, resource (human, equipment, budget etc), milestones
- Specification Phase Documentation
  - Specification document (specifications)
  - SPMP

Specification phase(4 of 4)

- 客戶需由specification document判斷答案的正確性
  - 了解 “How software handle the job”？
  - Software will meet the requirement.
- 開發者須向客戶詳細解釋 software “將做什麼事”、”如何操作”、及“這些功能與需求的對應關係.
- 描述“問題的解決方案”

Design phase (1 of 2)

- Design: determine *how* the product is to do it
- The running environment of the software system and the programming language should be determined before the software system is started to design.
- When the software is complicated, it had better decompose it into modules.
- Design phase consists of two phases:
  - Architecture design: Decompose the product into modules. Function, signature, and deployment of the module will be defined. Control flow of modules should also be defined.

Design phase (2 of 2)

- Design phase consists of two phases:
  - Detailed design: Define processing logic (data structures and algorithm) of each module.
- Architecture should be flexible and extensible.
  - 影響Testability and maintainability of the software
- Design decisions should be documented.
- Nonfunctional requirements (response time, security, etc.) should be incorporated into the design.
- 產出物: design document
  - How to describe architecture and processing logic
- Traceable to requirements and specification

Implementation Phase

- Implement the detailed design of modules in code
- Verify the correctness of modules
- Document
  - Source code of each module
  - All test cases, test procedures, test environment, the expected result and actual result of each module

Integration Phase(1 of 3)

- Check that the modules combine together correctly to achieve a product that satisfies the specification.
- 包含Integration testing, product testing, 及 acceptance testing
- 各模組正確, 不能保證組合後一定正確
- 若將各模組整合在一起而作測試, 很難發現錯誤的所在處
- Integration testing
  - 分批整合, 並作測試
  - Integration test plan:整合次序、測試環境、測試重點
  - 注重module interfaces的正確性測試
  - 由開發者執行
- Integration testing Documentation
  - Commented source code
  - Test Plan
  - Testing environment, test cases, test procedures, expected and actual result of each integration

Integration Phase(2 of 2)

- Product testing (α-test)
  - Performed by SQA (Software Quality Assurance,第三者) group after completing integration testing
  - The functionality and constraints of the product as a whole is checked against the specification
  - 依據specification來規劃各種可能情況的測試

Integration Phase(3 of 3)

- Acceptance testing(β-test)
  - Test the delivered software at the client site using actual data
  - 確認確認無誤,才能驗收使用或銷售
  - 產品的品質與實用性

Maintenance Phase (1 of 3)

- Once the client has accepted the software, the software is in the maintenance Phase.
- 軟體在客戶端正常運作及使用
- Software will be modified due to error, platform change, and functional enhancement.
  - Corrective maintenance
  - Enhancement maintenance
  - Codes of some modules will be changed← New version
- The most money is devoted to this phase
  - 使用時間長久

Maintenance Phase (2 of 3)

- Maintenance Phase Documentation
  - Record of all changes made with reasons
  - Configuration management:儲存所有軟體模組及其版本
  - Regression test cases
- Regression testing
  - Once the programmer has determined that the desired changes have been implemented, the product must be tested against previous test cases to make certain that the functionality of the rest of the product has not been compromised.

Maintenance Phase (3 of 3)

- The problem of lack of design documentation
  - Difficult to find and fix bugs, to add new functions, and to port to new platform
- 有作Configuration management,才能複製發生的問題

Retirement

- Good software is maintained
- Sometimes software is rewritten from scratch
  - Software is now un-maintainable because
    - A drastic change in design has occurred
    - The product must be implemented on a totally new hardware/operating system
    - Documentation is missing or inaccurate
    - Hardware is to be changed—it may be cheaper to rewrite the software from scratch than to modify it
- 移到新軟體上使用,需將原有資料庫與檔案轉移到新系統上，必要時需作轉檔。

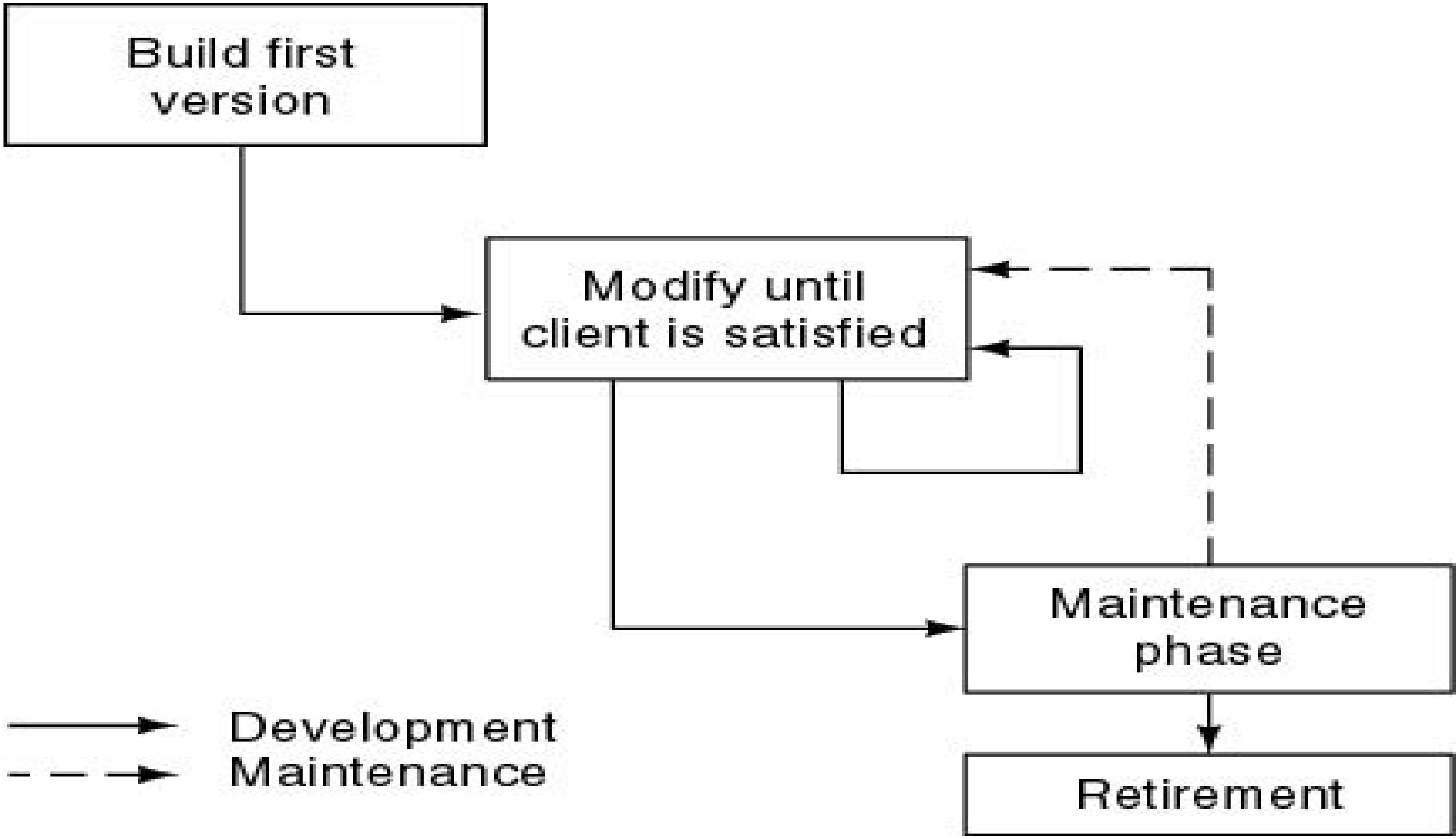
軟體生命周期模式
(SOFTWARE LIFE-CYCLE MODELS)

前言

- 偏重軟體生命周期前五個phases (軟體的產生 software development)的發展模式，亦稱軟體發展模式(software development model)
- 新軟體發展模式演進的主因:
  - Previous learned experience (i.e. improve shortage, productivity, and quality)
  - New software technology (programming languages, development concept, technique, Method, etc.)

- 主要模式：
  - Build-and-fix model
  - Waterfall model
  - Rapid prototyping model
  - Incremental model
  - Synchronize-and-stabilize model
  - Spiral model
  - Object-oriented life-cycle models

Build and Fix Model (1 of 2)

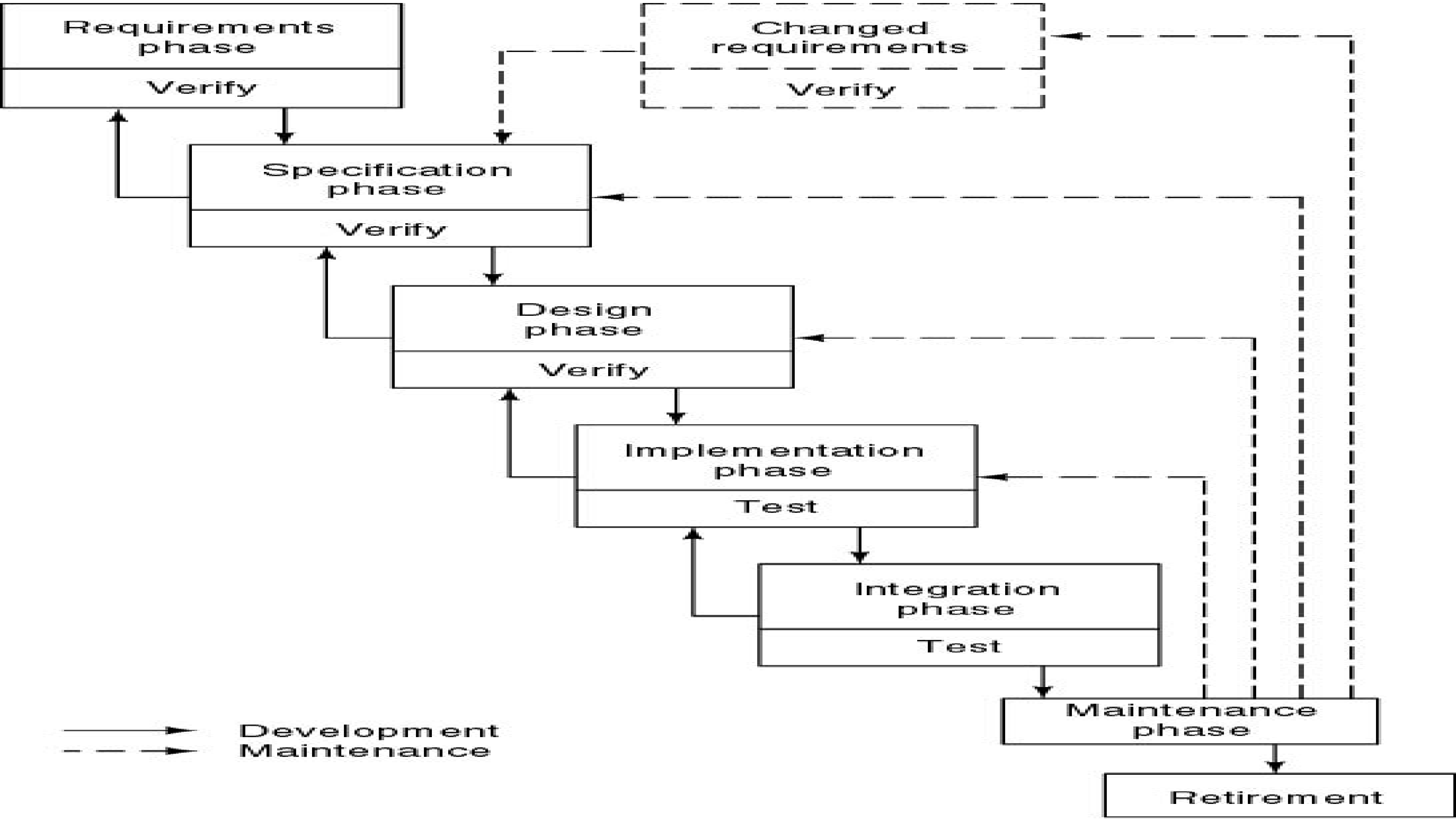


Build and Fix Model (2 of 2)

- The product is constructed without any specification or any attempt at design.
- Rework as many times as necessary to satisfy client.
- 適用於小系統(100 to 200 lines of code)及個人單獨發展
- 問題:
  - 缺乏發展制度及規劃,由錯誤中學習
  - 靠成品與客戶溝通,猜測客戶需求
  - 不適用眾人合作發展,且缺乏效率

→應有階段及檢核點觀念,方能循序且系統化發展

Waterfall Model (1 of 3)



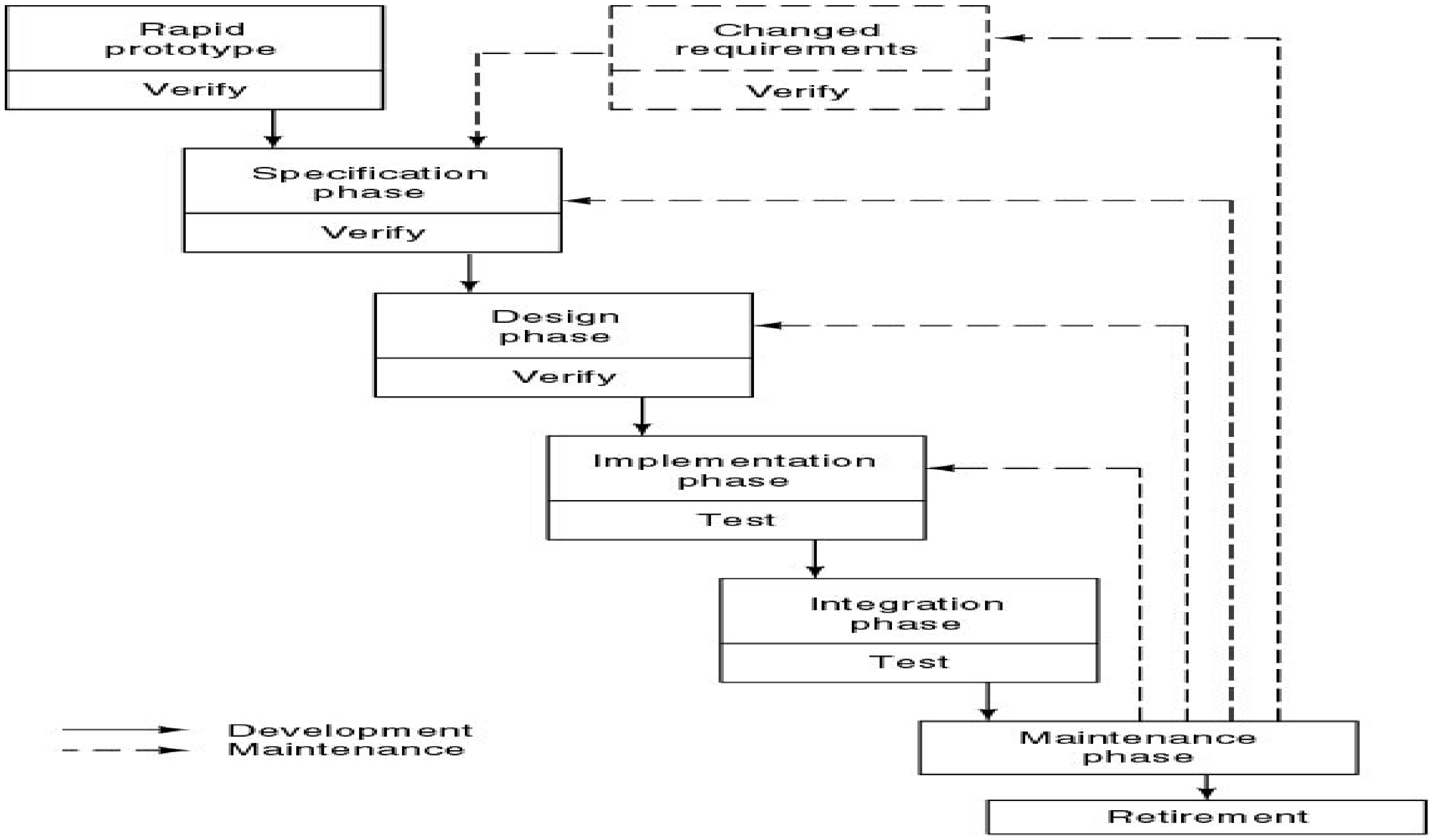
Waterfall Model (2 of 3)

- 依life cycle階段,一階段工作完成後才進入下階段
- Characterized by
  - Documentation-driven
  - Feedback loops (by formal review)
- Activities, rules of team work and documentation standard of each phase are defined. → Engineering approach
- Development methods and processes are proposed
  - Structured programming, structured design, structured analysis
  - Functional- or data-oriented approaches

Waterfall Model (3 of 3)

- Advantages
  - Documentation complete
  - Maintenance easier
- Disadvantages
  - Client can see the product only after the entire product has been coded
  - Client不易由specification掌握未來產品的性能 →共識性不足, 易生爭議及反悔
  - 一階段未通過則不能往下走,人員易閒置
- Suitable to the project with previous experience. ←有樣板可參考, 較易產生共識及減少錯誤發生

Rapid Prototyping Model (1 of 4)



Rapid Prototyping Model (2 of 4)

- Experience learned
  - Requirement capture is the most difficult job.（講不清楚, 未提到…）
  - If requirement or specification is not fixed, design team and coding team will be idle.
  - Wrong requirement leads wrong design and code, all development should be started again.
- Solution: Use prototype to help users to list requirements
  - Static prototype (由紙畫出輸出入畫面, 以解釋未來運作）
  - Working prototype: code or executable specification

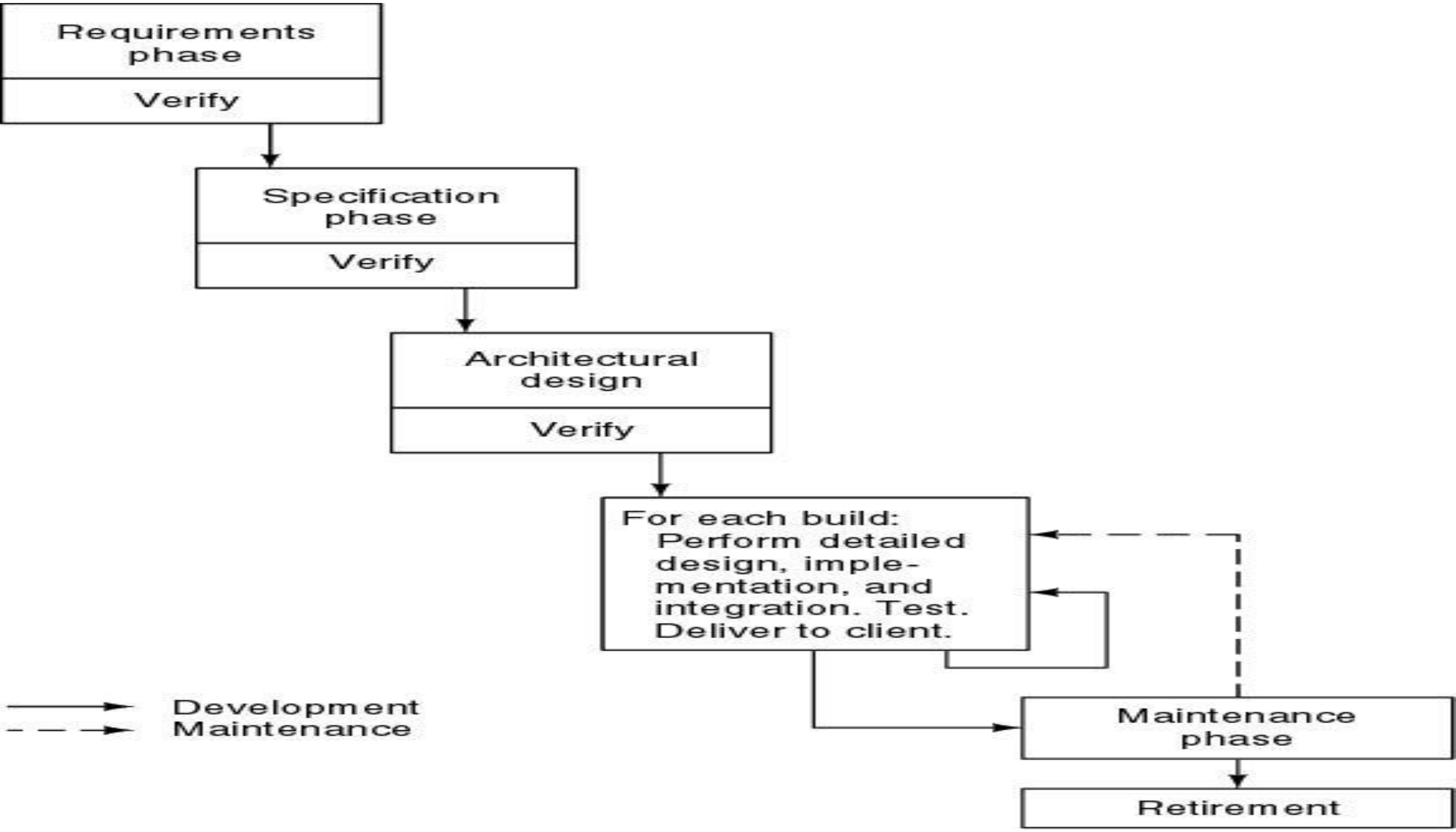
Rapid Prototyping Model (3 of 4)

- A rapid prototype is a working model that is functionally equivalent to a subset of the product.
- Once the working model has been accepted by client, the development is almost straightforward.
- Rapid prototyping may replace specification phase, never the design phase
- Advantage
  - Easy to get true requirements and to define specification
  - “Rapid”

Rapid Prototyping Model (4 of 4)

- Disadvantage
  - The working prototype will be discard.
  - Nonfunctional requirements are not easy to be implemented and verified in prototype.
  - The architecture may not be good.（只為驗証規格的正確性, 系統架構未深入分析及設計, 工程師若繼續沿用, 產生不良後果）
- Suitable to exploring project(創新型專案）
  - 需求及規格較不易明確訂定, 客戶與發展者需對需求與規格有必要多溝通以求共識

Incremental Model (1 of 6)



Incremental Model (2 of 6)

- Experience learned
  - Executable specification language suitable for all applications is not easy designed.
  - Software is built, not written. (Software is constructed step by step, in the same way that a building is constructed.
- User is difficult to list all requirements at one time, but is easy to list most important and frequently used requirements. →incomplete specification issue
- 先作部份成品供客戶使用,驗證其正確性及發覺不足之處

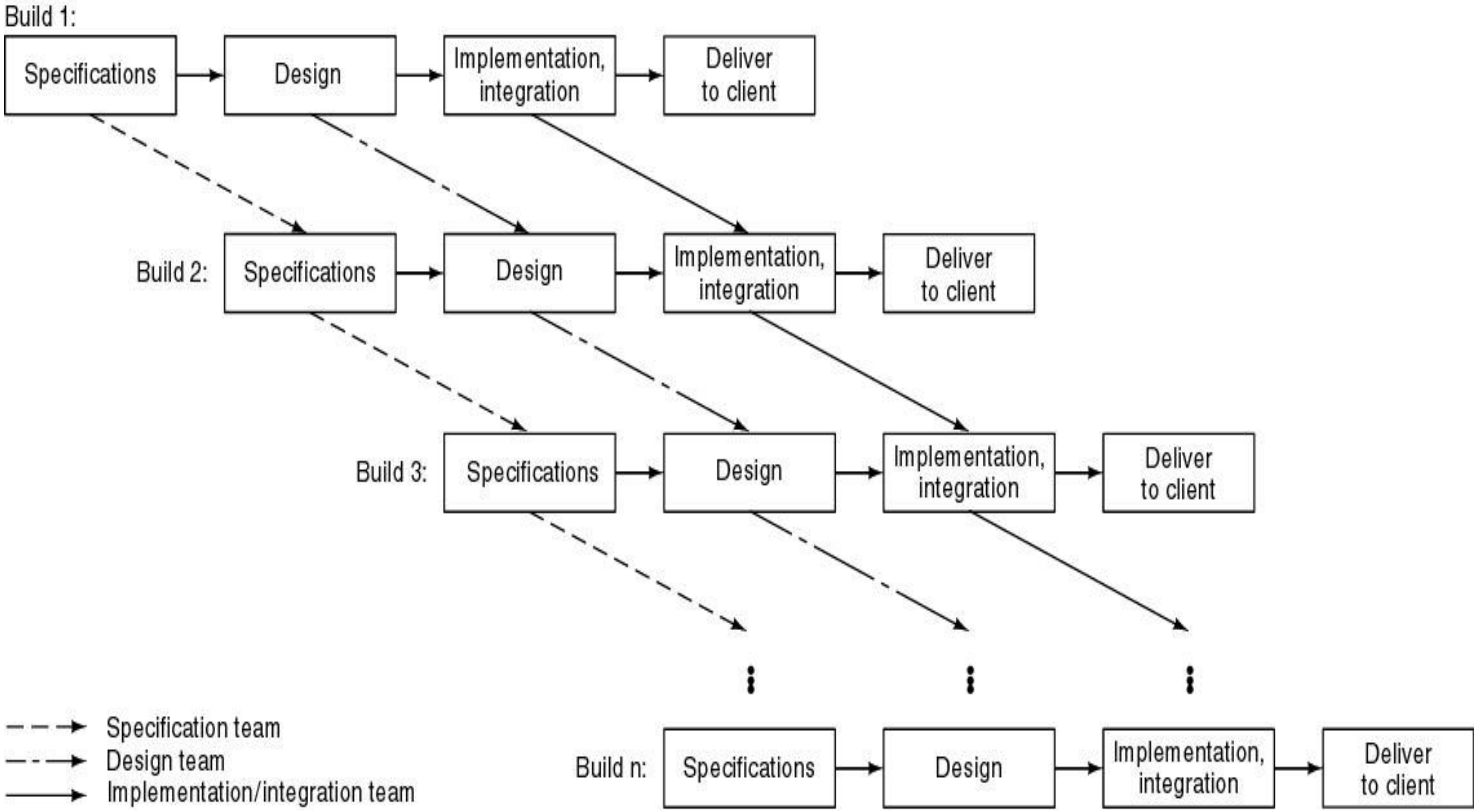
Incremental Model (3 of 6)

- Waterfall, rapid prototyping models
  - Operational quality complete product at end
- Divide project into *builds,* where a build consists of code pieces from various modules interacting together to provide a specific functional capability (i.e. meet some requirements). The software product is designed, implemented, integrated, and tested as a series of incremental builds（逐漸增加功能）.
- Incremental model
  - Operational quality portion of product within weeks
  - 10 to 50 builds for a typical product

Incremental Model (4 of 6)

- 己有的build(s).可提供客戶試用以確定符合其需求,因此用戶可及早看到成果。→Smaller capital outlay, rapid return on investment
- Still need a long time to capture requirements and to define specification (user is difficult to list all requirements at one time, but is easy to list 50% requirements)
  - User will list other requirements incrementally after he use the build(s).
- Need open architecture—maintenance implications
- 可平衡人員運用及加速發展的運作方式

Incremental Model (5 of 6)



Incremental Model (6 of 6)

- Problems
  - Build-and-fix danger →若 build錯誤太多, 導致 control of the whole process can be lost
  - Contradiction in terms: View the product as a whole, but simultaneously view the product as a sequence of builds

  →Risky incremental model,偏離焦點
- 需有明確發展規劃, 及控制build的正確性

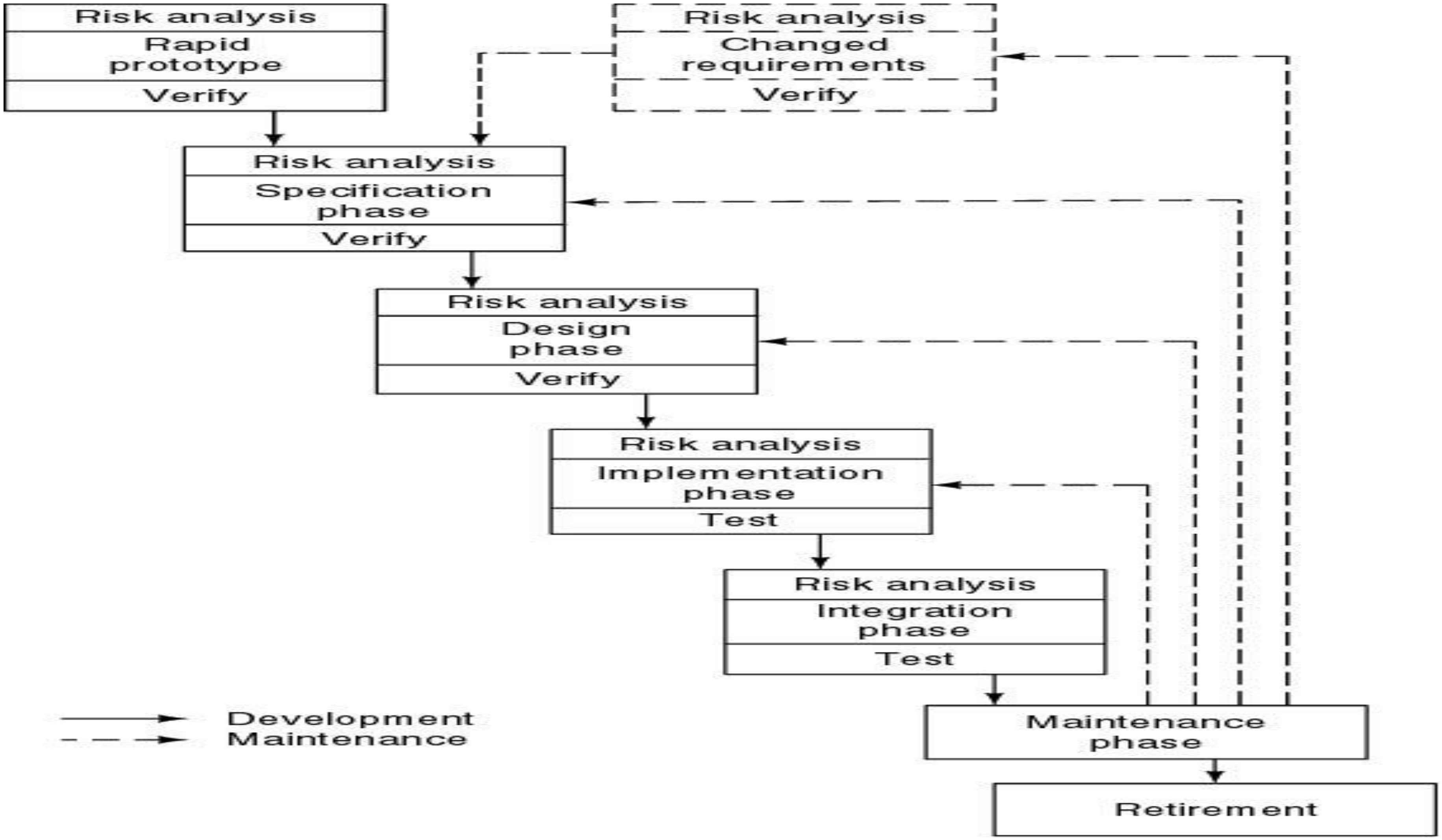
Synchronize-and Stabilize Model (1 of 2)

- Microsoft’s life-cycle model
- Requirements analysis—interview potential customers
- Draw up specifications
- Divide project into 3 or 4 builds. The first build consists of the most critical features, the second build consists of the next most critical features, and so on.
- Each build is carried out by small teams working in parallel

Synchronize-and Stabilize Model (2 of 2)

- At the end of the day: synchronize (test and debug)
- At the end of the build: stabilize (freeze build)
- Advantage: Components always work together
  - Get early insights into operation of product
- Can be used even if specification is not complete.
- Only one final build is selected.
- 適用於Exploring project (尤其是software package): need creative idea, use parallel group to create new idea.
- Good but not perfect process model
  - Not all projects are successful
  - High Cost

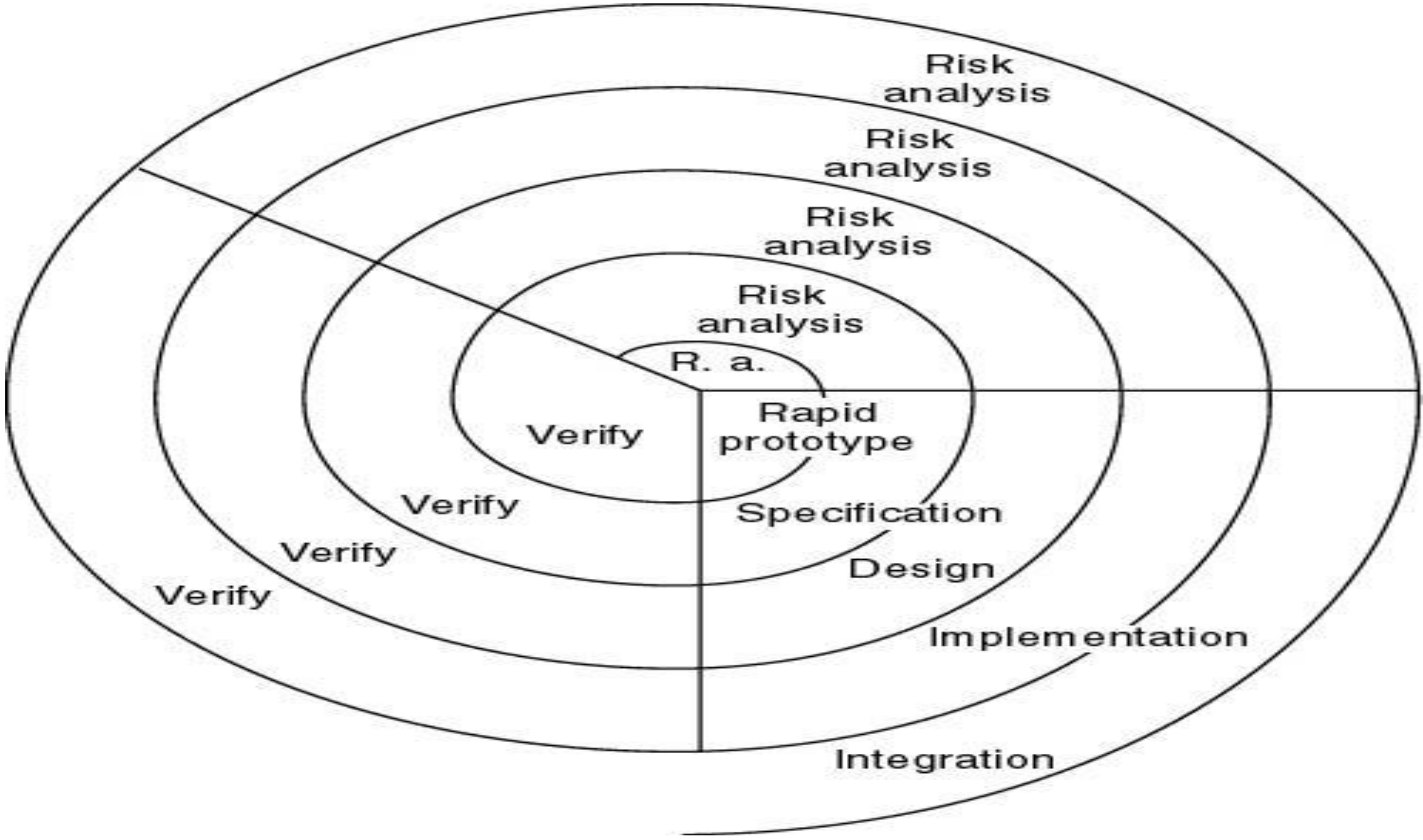
Spiral Model (1 of 5)



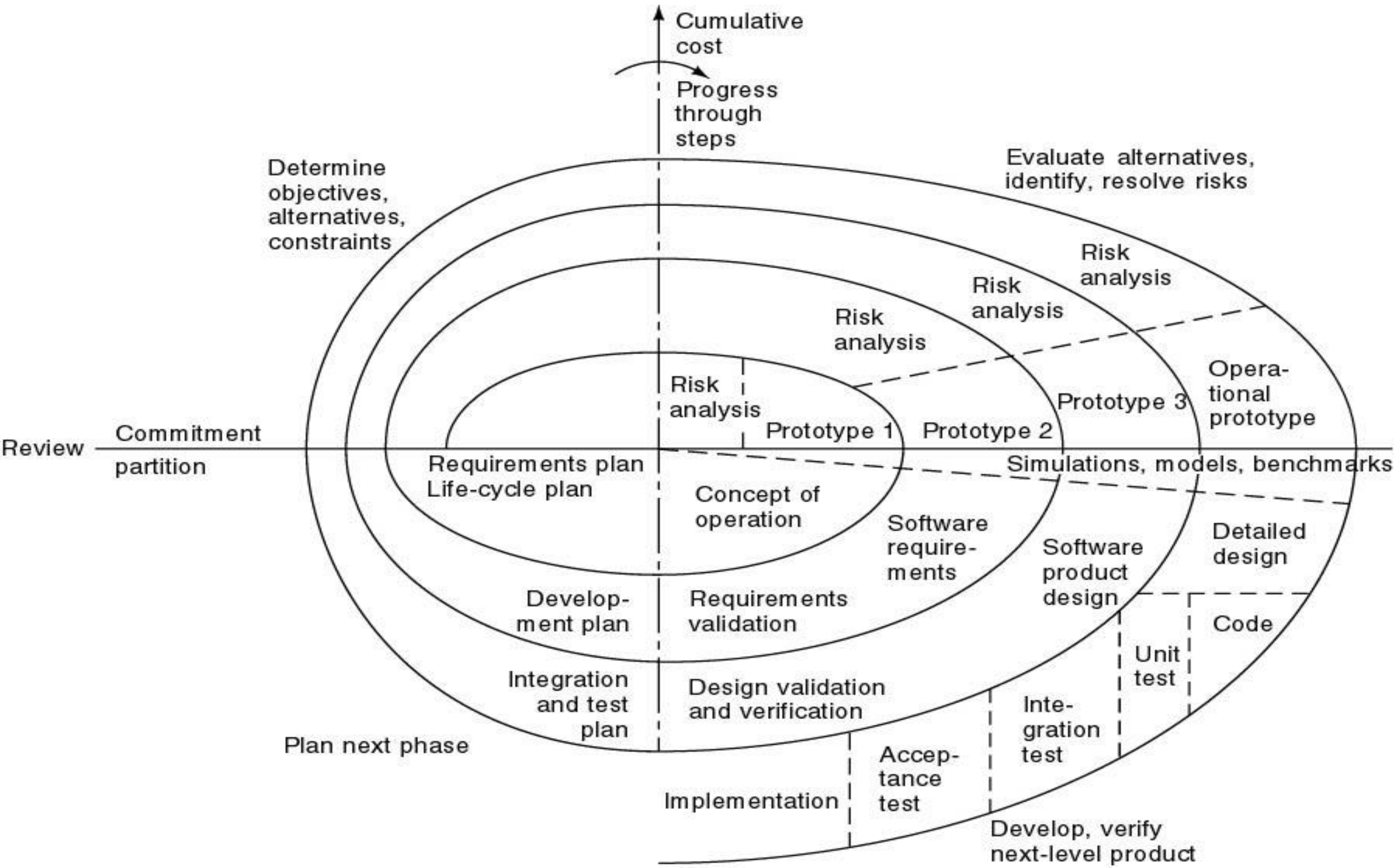
Spiral Model (2 of 5)

- Exploring project: many uncertain/unknown factors.
- Simplified form
  - Waterfall model plus risk analysis
- Precede each phase by
  - Objectives and constraint
  - Alternatives
  - Risk analysis
- Follow each phase by
  - Evaluation
  - Planning of next phase
- If risks cannot be resolved, project is immediately terminated → risk-driven approach

Spiral Model (3 of 5)



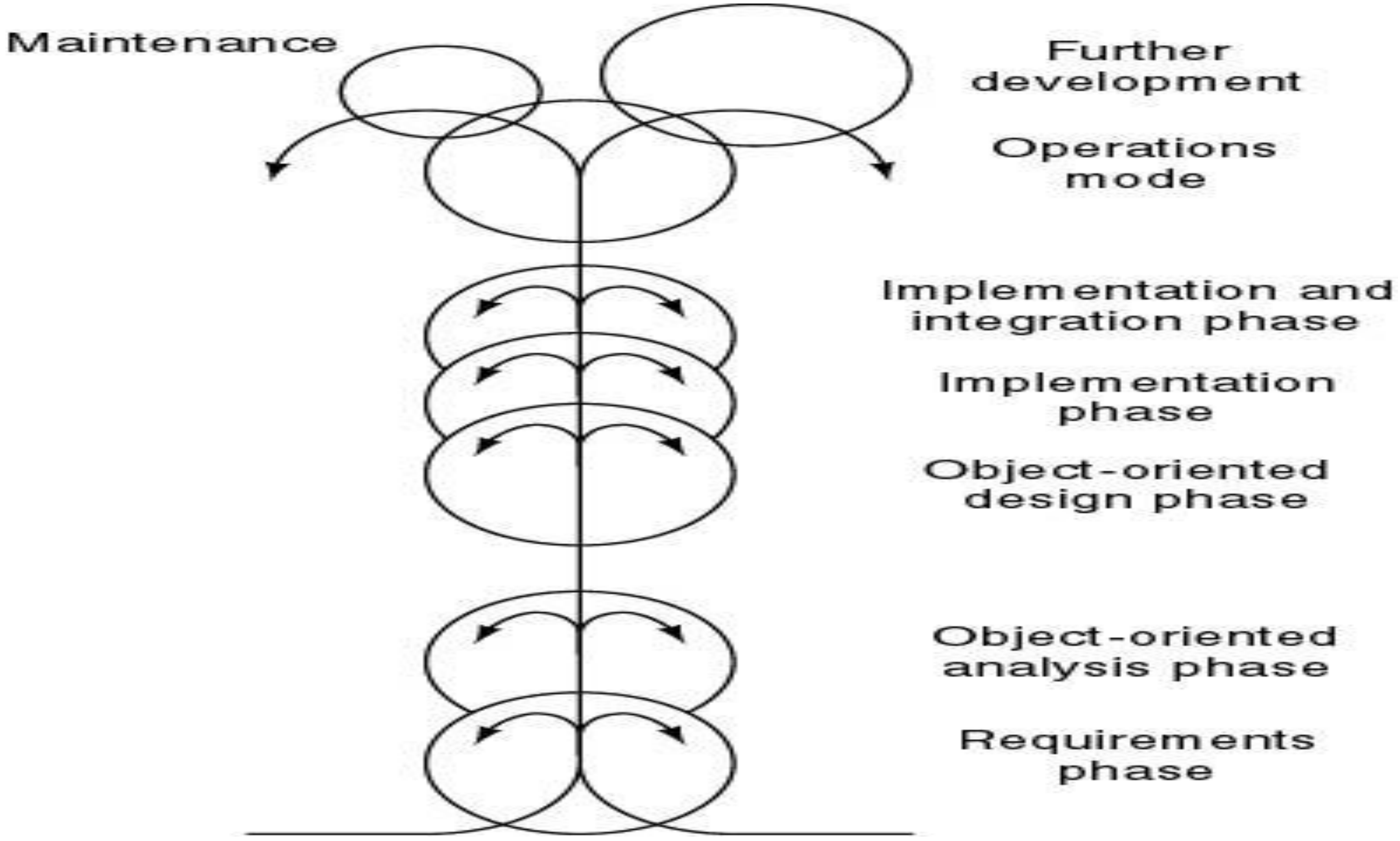
Spiral Model (4 of 5)



Spiral Model (5 of 5)

- Radial dimension: cumulative cost to date
- Angular dimension: progress through the spiral
  - 適用於前瞻性大型專案(如武器系統、太空計畫)
- Strengths
  - Easy to judge how much to test
  - No distinction between development and maintenance
- Weaknesses
  - For large-scale software only
  - For internal (in-house) software development only

Object-Oriented Life-Cycle Models (1 of 3)


Maintenance
Further development
Operations mode
Implementation and integration phase
Implementation phase
Object-oriented design phase
Object-oriented analysis phase
Requirements phase

Object-Oriented Life-Cycle Models (2 of 3)

- Object- oriented paradigm
  - 物件導向程式由一群 classes 組成, 程式執行是一群 objects 交互運作。
- 物件導向軟體發展是Seamless development:
  - 先定義class specification, 確認系統正確性, 寫code時可繼續沿用
  - 設計時, classes 是漸進式定義及發展
    →no waste effort
  - Easy to develop incrementally
- Iterative development
  - “build” concept is easily adapted

Object-Oriented Life-Cycle Models (3 of 3)

- Need for iteration within and between phases
  - Fountain model (噴泉式模型)
  - Recursive/parallel life cycle
  - Unified software development process
- All incorporate some form of
  - Iteration
  - Parallelism
  - Incremental development
- Danger
  - CABTAB (Code A Bit, Test A Bit)

Impact of OO approach

- Class: function and data should be consider together
- Component concept: an independent module.
- Reuse of component
  - component repository → software reuse concept
  - component-based development
- Importance of documentation standard

Conclusions

- Each life-cycle model has its own strengths and weaknesses
- Criteria for deciding on a model include
  - The organization
  - Its management
  - Skills of the employees
  - The nature of the product
- 軟體產品需求與先前產品需求的類似性及產品需求的不確定性影響模式之選擇
- 只有構想，各階段的發展方法仍需提出。

The Software Process (1 of 4)

- The way we produce software.→眾人合作工作模式
- Incorporate the software life-cycle model, CASE (Computer-Aided Software Engineering) tools, the individuals, and existing building software.
- Encompass the activities, techniques, and tools used to develop software
- 包含驗證工作
- Different organizations have different processes!

The Software Process (2 of 4)

- NO testing phase
  - Testing (quality assurance) is an activity performed throughout software production
- Every phase must be verified before starting the next phase.
- Every phase must be fully documented before starting the next phase.
  - Keep all artifacts and decision
  - Standard documentation content and format

The Software Process (3 of 4)

- Why?
  - Postponed documentation may never be completed
  - The responsible individual may leave or change job
  - The product is constantly changing—we need the documentation to find where we should change.
- Verification
  - Performed at the end of each phase

The Software Process (4 of 4)

- Validation
  - Performed before delivering the product to the client
- 如何管理軟體的發展專業?
  - 牽涉多人同時發展,個人的延遲與錯誤將對他人造成影響
  - “流程的完整度”與“流程監控與危機管理機制”的相互配合