Chapter 2
Software Processes

Objectives

- software process 및 process models의 개념 이해
- 3종류의 generic process models 이해
- requirements engineering, software development, testing and evolution에 관련된 activities 간략 소개
- Process와 변경(change)
- RUP(Rational Unified Process) 소개

Topics covered

- Software process models
- Process activities
- Coping with change
- The Rational Unified Process
  - An example of a modern software process.

The software process

- Software process란?
  - A structured set of activities required to develop a software system
  - Specification, Design, Validation, Evolution 포함
- A software process model이란?
  - an abstract representation of a process.
  - a description of a process from **some particular perspective**.

Software process descriptions

- the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- 그외 다음을 포함.
  - Products: the outcomes of a process activity
  - Roles: reflect the responsibilities of the people involved in the process
  - Pre- and post-conditions: statements that are true before and after a process activity has been enacted or a product produced.

Plan-driven and agile processes

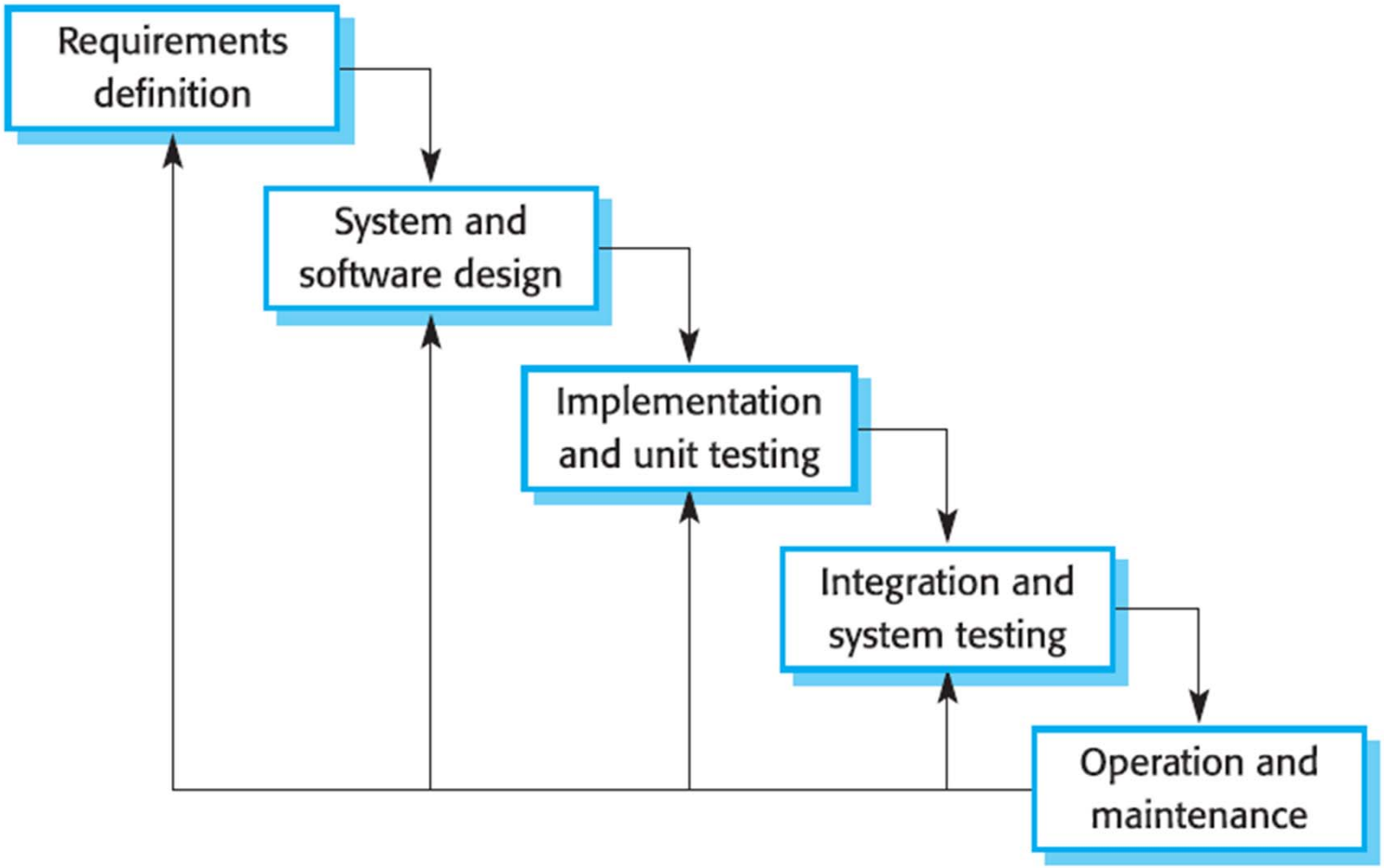
- Plan-driven processes
  - all of the process activities are planned in advance and progress is measured against this plan.
- Agile processes
  - planning is incremental
  - customer요구사항 변경으로 인한 프로세스 변경이 용이함.
- 대부분의 practical process에서는 두 방식이 혼합되어 쓰임.
  - There are no right or wrong software processes.

2.1 Software process models

Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development.
- Incremental development
  - Specification, development and validation are interleaved.
- Reuse-oriented software engineering
  - The system is assembled from existing components.
- 이들의 여러 변형이 존재함
  - (예) formal development + a waterfall-like process

Waterfall model



Waterfall model phases

- Requirements analysis and definition
  - 시스템 services, constraints, goals 결정 → 시스템 spec.이 됨
- System and software design
  - 시스템 전체 아키텍처를 수립하여, 요구사항들을 HW/SW에 할당.
  - 시스템 구성요소의 기능 및 관계 명시
- Implementation and unit testing
  - SW design이 프로그램으로 구현.
- Integration and system testing
- Operation and maintenance
  - lifecycle중 가장 길다.
  - 에러 수정, 기능 향상 등.
- 각 단계의 결과
  - approved documents

Waterfall model의 특징

- 장점
  - process가 visible
  - 다음의 경우에 적합함
    - the requirements are well-understood and changes will be fairly limited during the design process.
  - 그러나!
    - Few business systems have stable requirements.

Waterfall model의 특징

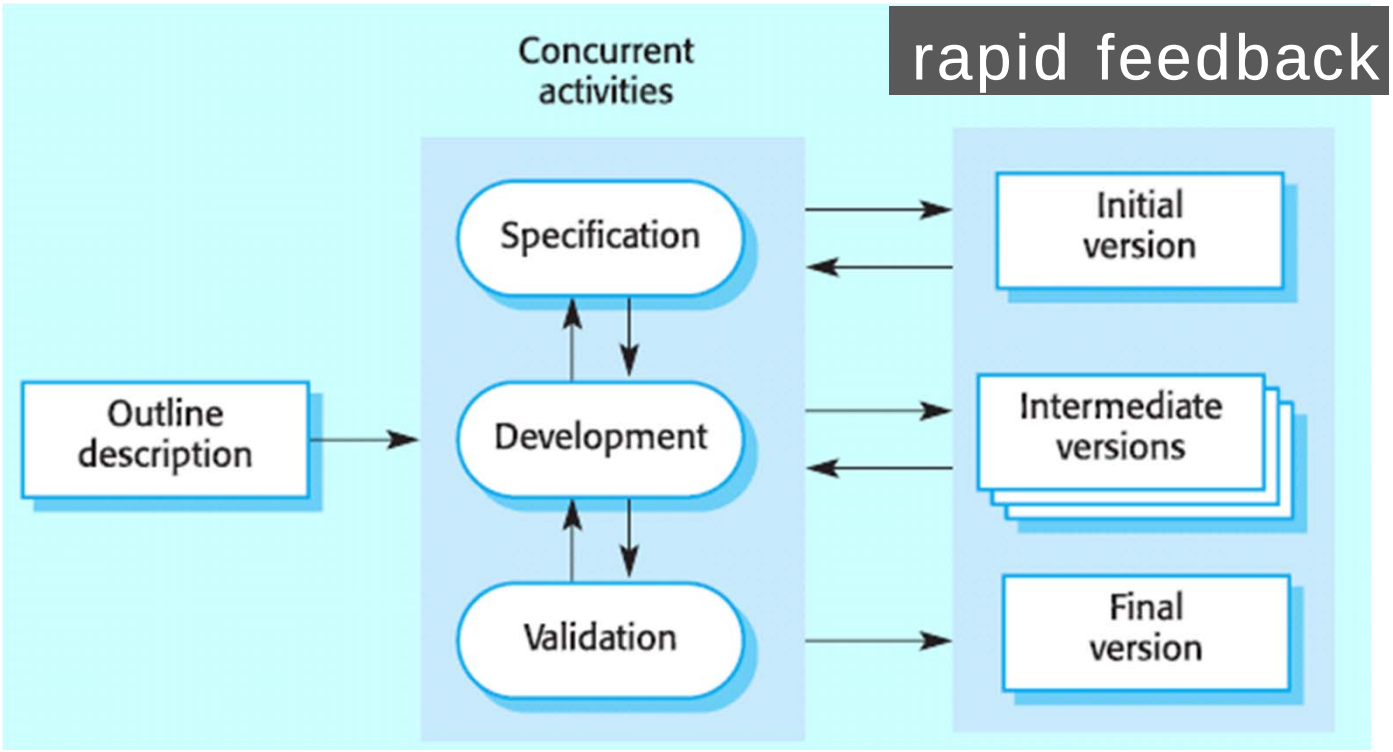
- 문제점
  - 프로세스가 진행 중일 때 변화를 수용하기 어렵다
    - One phase has to be complete before moving onto the next phase.
    - Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
    - product의 visiblity가 낮음

- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

Variant of Waterfall model

- Formal system development
  - SW spec.의 수학적 모델이 생성됨.
  - 엄격한 safety, reliablilty, security를 요하는 시스템에 적합.
  - specialized expertise 필요.

Incremental development



Incremental development benefits

- 초기 increment에 important/urgent functionality 포함
  → 빠른 evaluation가능
- customer의 요구사항 변경 반영비용 감소
  - waterfall모델에 비해 재 작업해야 하는 분석, 문서화 작업량 작음
- 기 개발된 결과물에 대해 customer feedback이 용이
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- Customer에게, 보다 빠른 delivery/deployment가 가능.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

- Business, E-commerce, personal system에 적합.

Incremental development problems

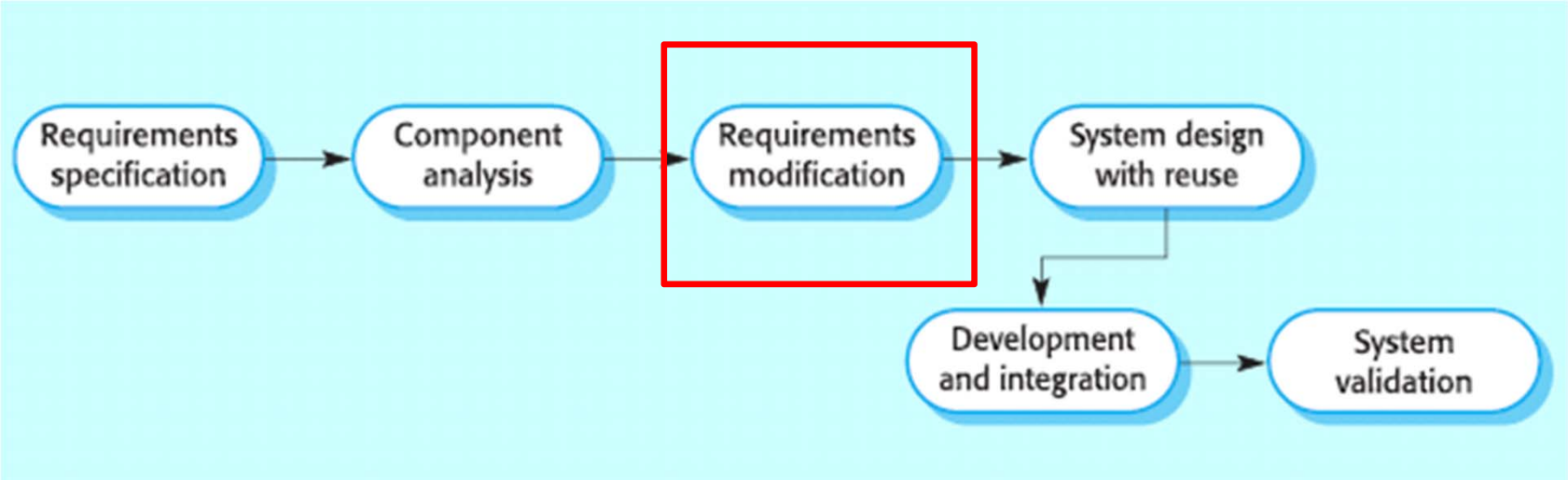
- Process가 invisible.
  - Managers need regular deliverables to measure progress.
  - If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- 새로운 increment가 추가될 때마다 시스템 구조가 degrade됨.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

- 여러 팀들이 시스템의 각 부분을 나누어 개발하는 Large, complex, long-life 시스템에는 부적합.

Reuse-oriented software engineering

- 기존에 존재하는 components나 COTS(Commercial-off-the-shelf) systems을 integration하는 체계적인 재사용에 기반.
- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- This approach is becoming increasingly used as component standards have emerged.

Reuse-oriented development



Types of software component

- Web services
  - developed according to service standards and which are available for remote invocation.
- Collections of objects
  - developed as a package to be integrated with a component framework such as .NET or J2EE.
- Stand-alone software systems (COTS)
  - configured for use in a particular environment.

Reuse-oriented development의 특징

- 장점
  - cost, risk 감소
  - fast delivery
- 단점
  - requirement compromises 요함
  - SW evolution의 control문제

2.2 Process activities

Process activities

- 실제 SW 프로세스는...
  - inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities
  - (1) Software specification
  - (2) Software design and implementation
  - (3) Software validation
  - (4) Software evolution

(1) Software specification

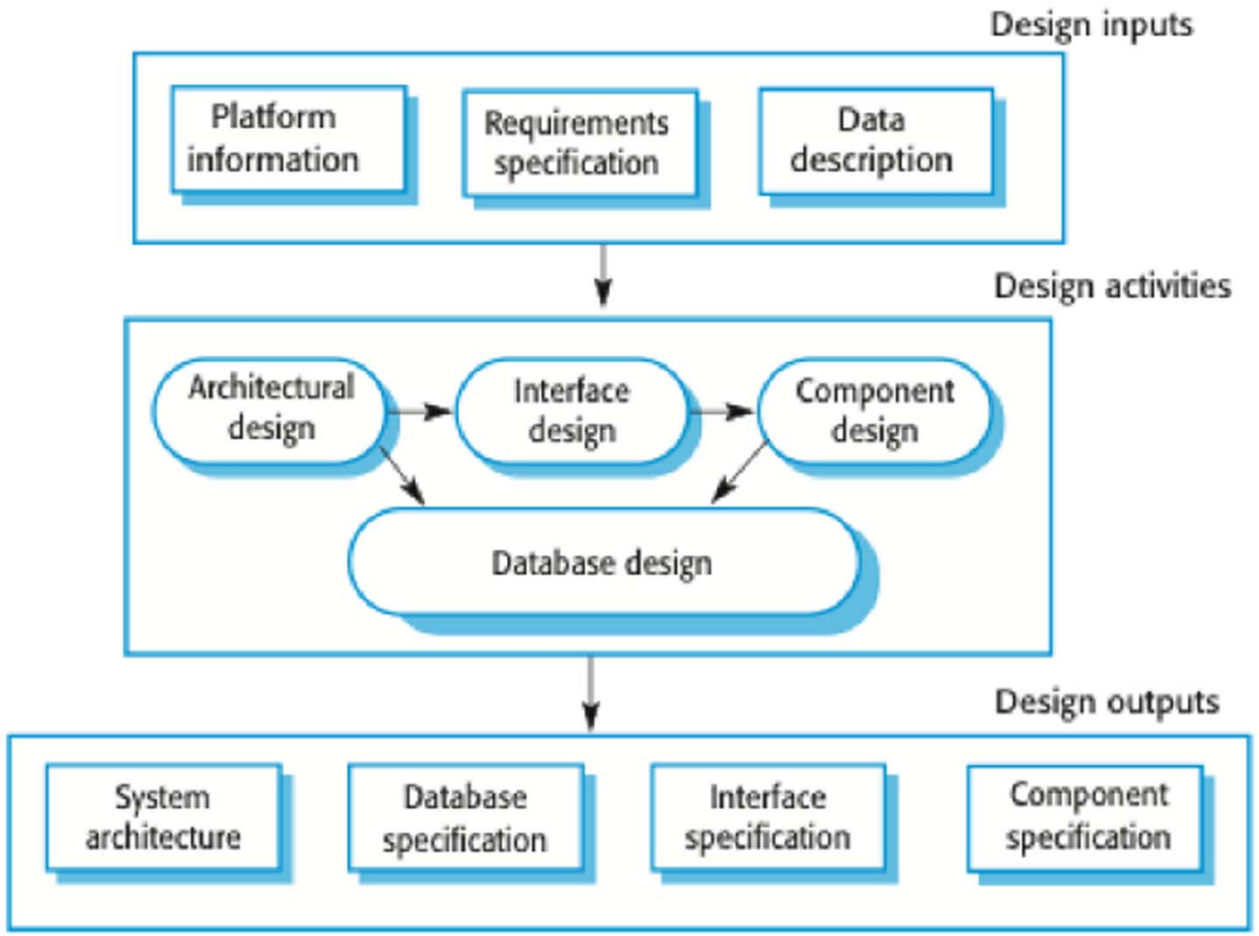
- 어떤 서비스가 필요하고, 시스템 동작 및 개발에서의 제약조건 (constraints) 을 규명하는 프로세스
- Requirements engineering process
  - Feasibility study(타당성 조사)
  - Requirements elicitation and analysis
  - Requirements specification
  - Requirements validation.

(2) Software design and implementation

- system specification을 executable system으로.
- Software design
  - Design a software structure that realises the specification
- Implementation
  - Translate this structure into an executable program

- 설계 및 구현은 밀접히 관련되어 있으며, ‘interleaved’가능.

A general model of the design process



Design activities

- Architectural design
  - identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- Interface design
  - define the interfaces between system components.
- Component design
  - take each system component and design how it will operate.
- Database design
  - design the system data structures and how these are to be represented in a database.

Programming and debugging

- Translating a design into a program and removing errors from that program.
- Programming is a personal activity - there is no generic programming process.
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process.

(3) Software validation

- Verification and validation (V & V) is intended to show that
  - a system conforms to its specification and
  - a system meets the requirements of the system customer.
  - “스펙에 맞게 개발되었나?” & “정말 고객이 원하는 것인가?”

- Involves checking and review processes and system testing.
- System testing: executing the system with test cases

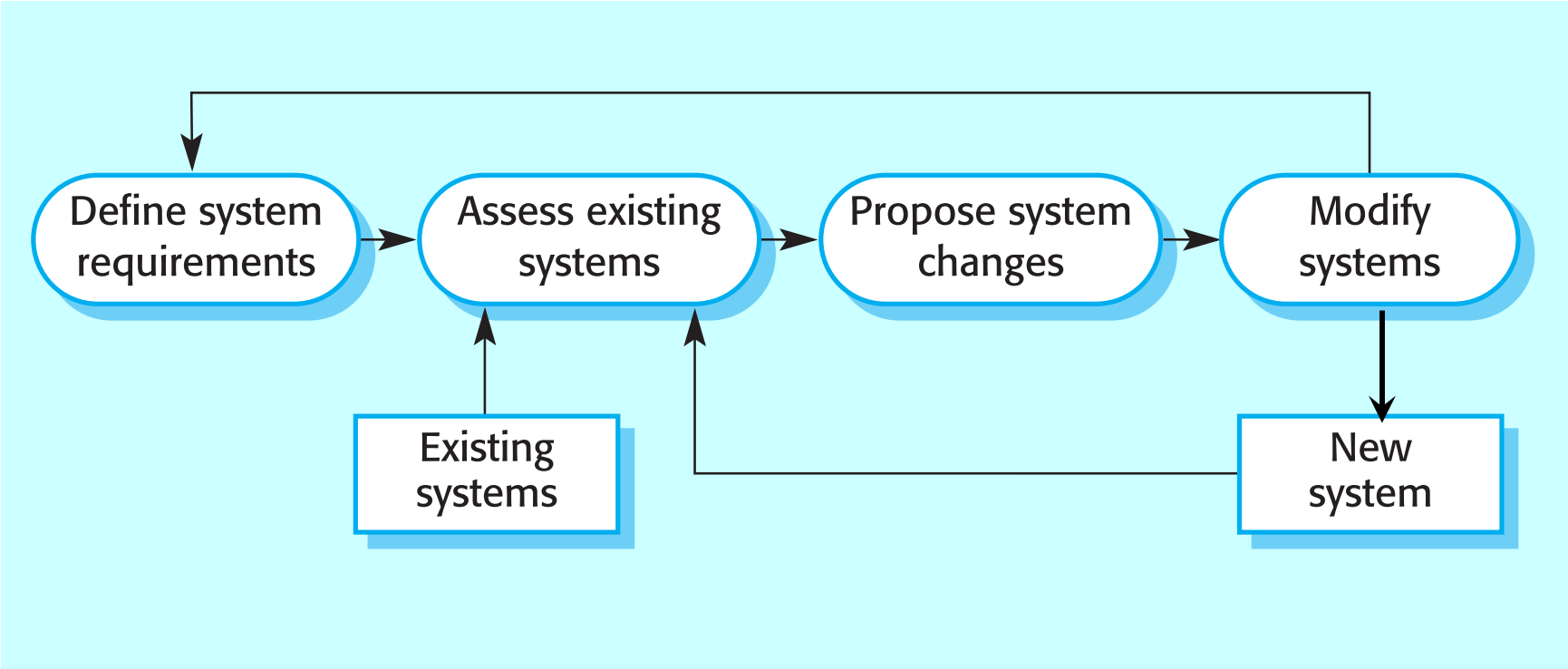
Testing stages

- Development or component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- Acceptance testing
  - Testing with customer data to check that the system meets the customer's needs.

(4) Software evolution

- Software is inherently flexible and can change. 왜?
  - 비즈니스 환경의 변화에 따른 요구사항 변화 → 소프트웨어 변화 요구
- development 와 evolution(maintenance)의 경계가 허물어지고 있다.
  - Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process where SW is continually changed over its lifetime in response to changing requirements and customer needs.

System evolution



2.3 Coping with changes

Coping with change

- Change is inevitable in all large software projects.
  - **Business changes**
  - **New technologies**
  - **Changing platforms**
- Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality

Reducing the costs of rework

- Change avoidance
  - the software process includes activities that can anticipate possible changes before significant rework is required.
  - (예) 시스템의 주요 features를 customer에게 보여주는 prototype system을 개발
- Change tolerance
  - the process is designed so that changes can be accommodated at relatively low cost.
  - incremental development
    - 변경을 increment로 구현, 이것이 안되면 only a single increment (a small part of the system) 이 수정

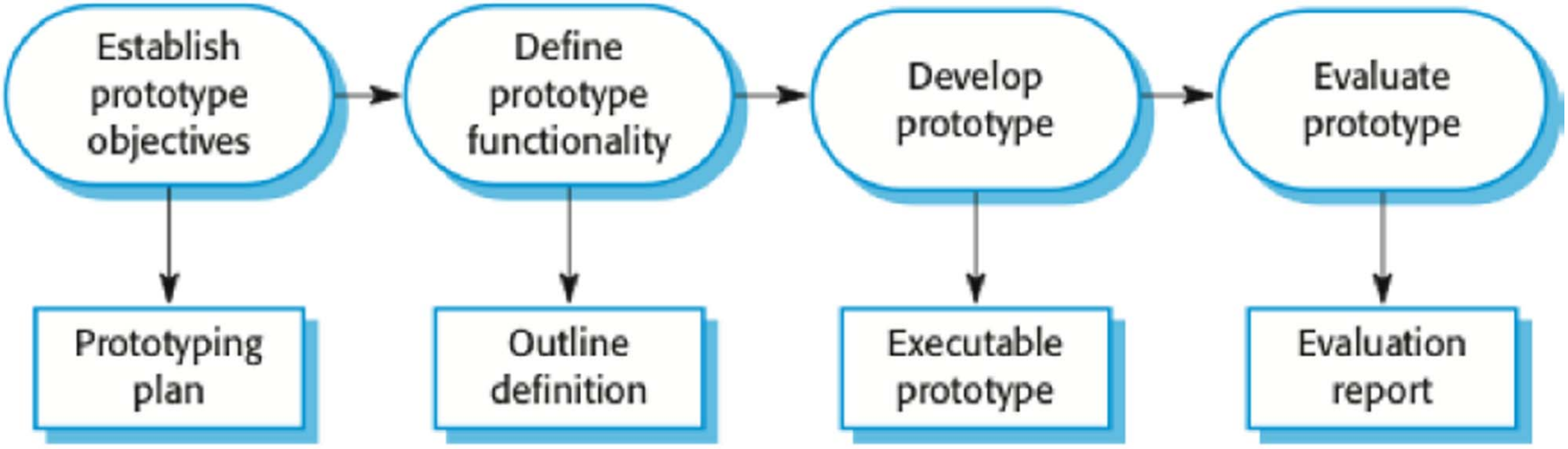
Software prototyping

- prototype: an initial version of a system used to demonstrate concepts and try out design options.
- A prototype이 사용되는 곳:
  - Requirements engineering process: help with requirements elicitation and validation
  - Design processes: explore options and develop a UI design
  - Testing process: run back-to-back tests.

Benefits of prototyping

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

The process of prototype development



Prototype development

- rapid prototyping languages or tools 이용.
- functionality 누락 가능
  - 프로토타입은 areas of the product that are not well-understood에 집중.
  - Error checking and recovery 은 제외 가능
  - non-functional 요구사항보다 functional 요구사항에 집중

Throw-away prototypes

- Prototypes은 개발 후 버려짐
  - they are not a good basis for a production system
  - It may be impossible to tune the system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will not meet normal organizational quality standards.

Process iteration

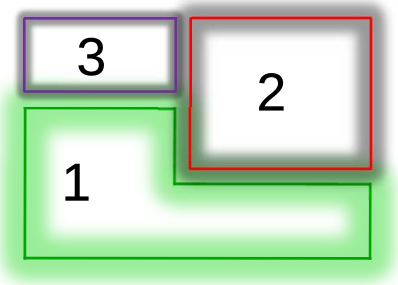
- System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems.
- Iteration can be applied to any of the generic process models.
- Two (related) approaches
  - Incremental delivery;
  - Spiral development.

Incremental development and delivery

- Incremental development
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
  - Normal approach used in agile methods;
  - Evaluation done by user/customer proxy.
- Incremental delivery
  - Deploy an increment for use by end-users;
  - More realistic evaluation about practical use of software;
  - Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

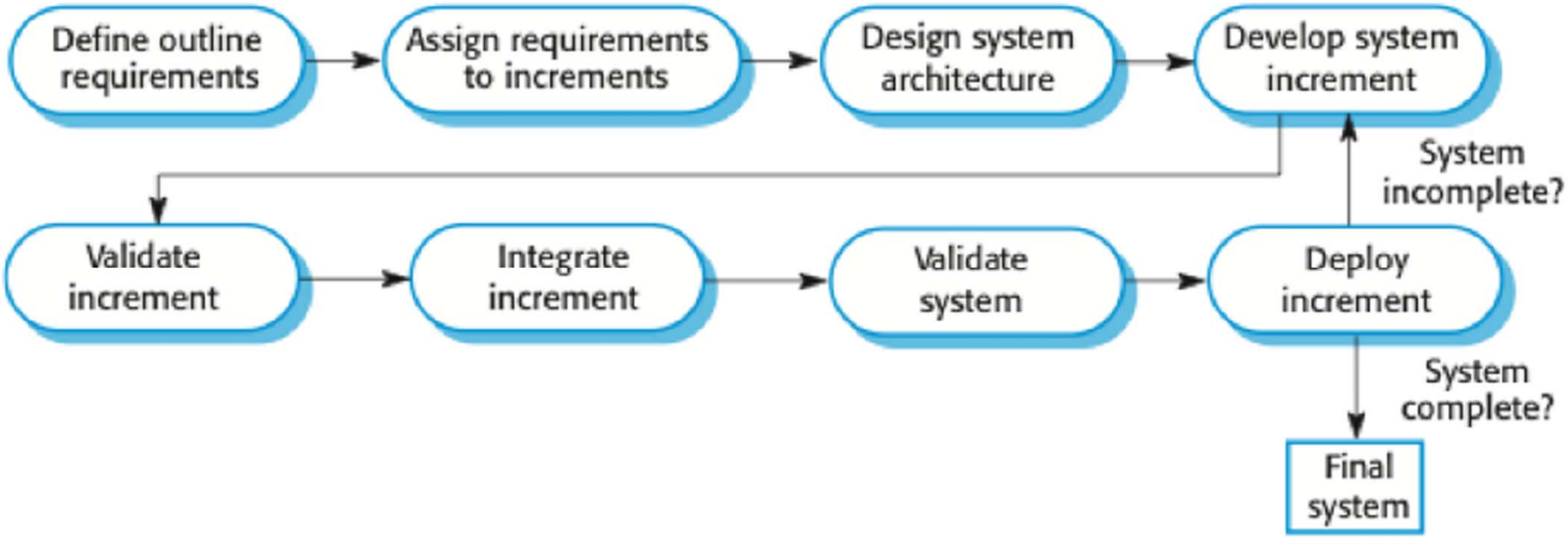
Incremental delivery

- 시스템을 한번에 deliver하지 않음
  - the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- 사용자 요구사항은 우선순위화→ 가장 높은 우선순위의 요구사항이 초기 increments에 포함
- 일단 increment개발이 시작되면→ 요구사항은 'frozen' (requirements for later increments can continue to evolve)



1→ 1+2 → 1+2+3

Incremental delivery



Incremental delivery advantages

- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

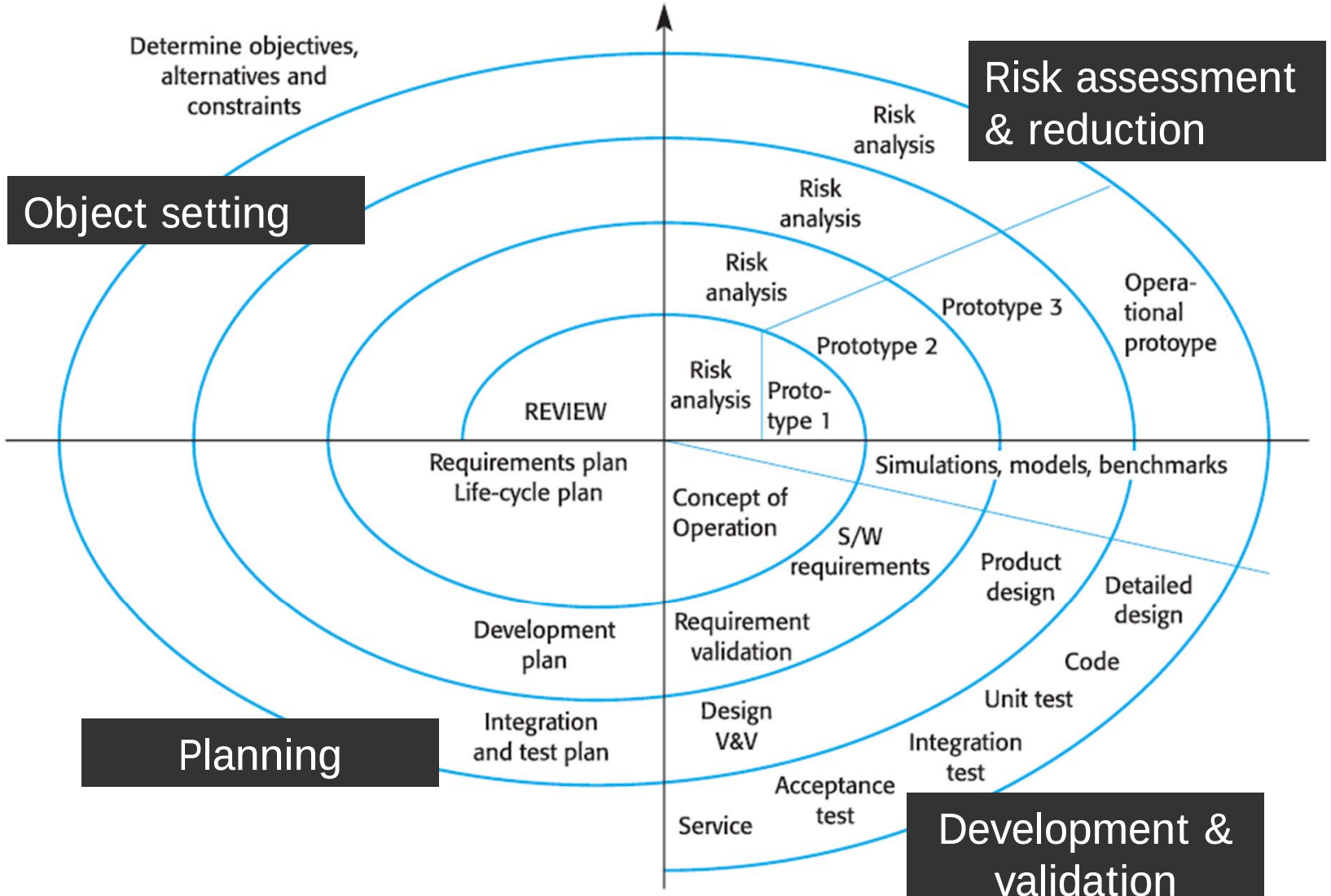
Incremental delivery problems

- Most systems require a set of basic facilities that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

Boehm's spiral model

- 프로세스가 나선형(spiral)으로 나타남: a sequence of activities with backtracking의 형태 아님.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

Spiral model of the software process



Spiral model sectors

- Objective setting
  - Specific objectives for the phase are identified.
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks.
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - The project is reviewed and the next phase of the spiral is planned.