2장. 소프트웨어 프로세스

- 소프트웨어 시스템을 명세화, 설계, 구현, 그리고 시험하기 위한 활동들의 집합

개요

- 소프트웨어 프로세스 모델들을 소개
- 여러 다른 프로세스 모델들을 설명하고, 언제 그 모델들을 사용할 수 있는지를 기술.
- 요구 공학, 소프트웨어 개발, 테스팅, 그리고 진화에 대한 개략적 프로세스 모델을 설명.
- 소프트웨어 프로세스 활동들을 지원하는 CASE 기술을 소개

다루는 주제

- 소프트웨어 프로세스 모델
- 프로세스 반복(Process iteration)
- 소프트웨어 명세화(specification)
- 소프트웨어 설계와 구현
- 소프트웨어 확인(validation)
- 소프트웨어 진화(evolution)
- 자동화된 프로세스 지원

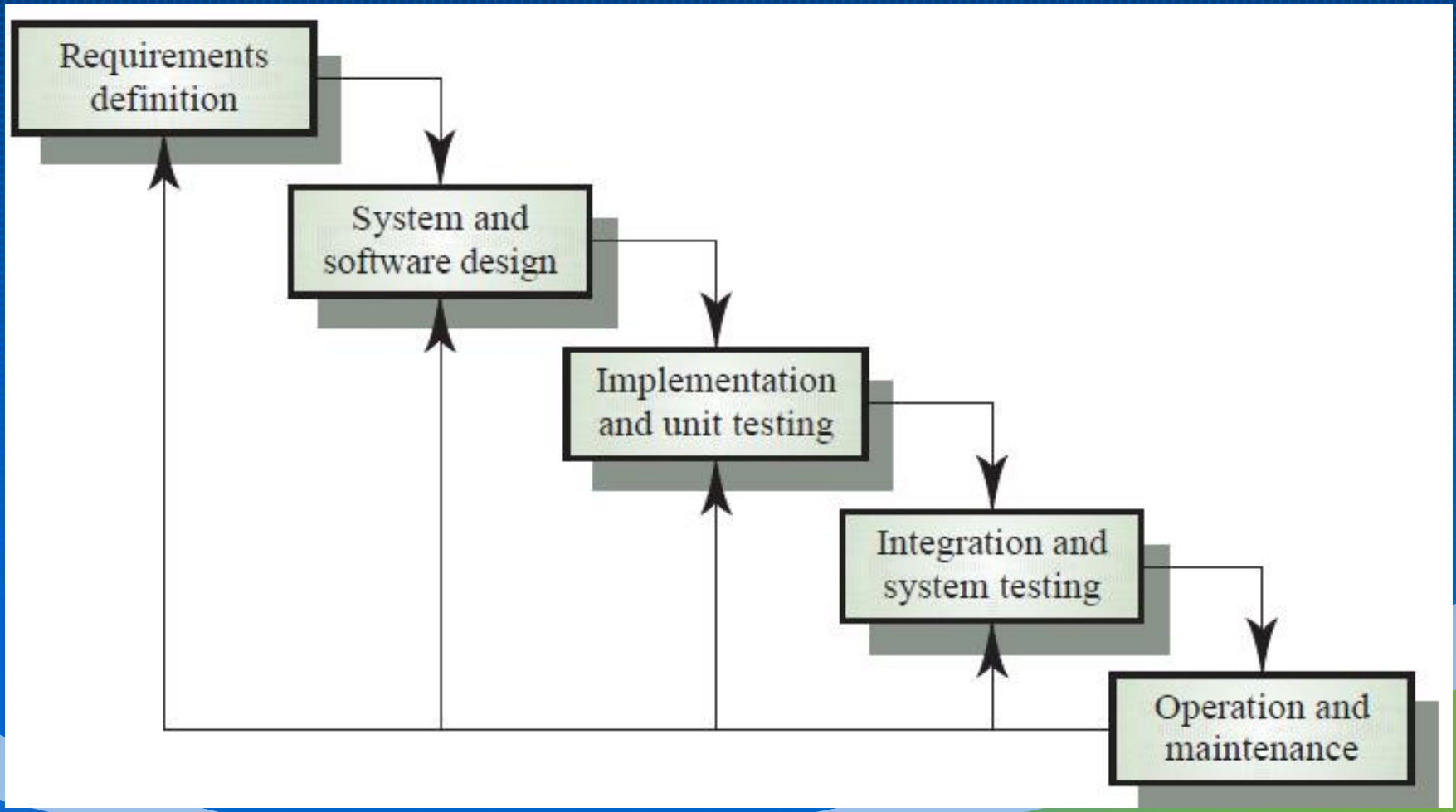
소프트웨어 프로세스

- 소프트웨어 시스템을 개발하는데 필요한 활동들의 구조화된 집합
  - 명세화(Specification)
  - 설계(Design)
  - 확인(Validation)
  - 진화(Evolution)
- 소프트웨어 프로세스 모델은 공정에 대한 추상적 표현으로, 프로세스에 대한 설명을 나타냄.

일반적인 소프트웨어 프로세스 모델들

- 폭포수(waterfall) 모델
  - 명세화와 개발이 독립적이고 구분되는 단계
- 진화적 개발(Evolutionary 개발)
  - 명세화와 개발이 섞여있음
- 정형적(Formal) 시스템 개발
  - 수학적인 시스템 모델이 구현으로 공식적으로 변환됨.
- 재사용 기반 개발
  - 시스템이 기존의 컴포넌트들로부터 조립됨.

폭포수 모델


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

폭포수 모델 단계(phases)

- 요구사항 분석과 정의
- 시스템과 소프트웨어 설계
- 구현과 단위(unit) 테스팅
- 통합과 시스템 테스팅
- 작동과 유지보수(maintenance)
- 폭포수 모델의 단점은 프로세스가 지나간 후에 변경사항을 수용하기가 어렵다.

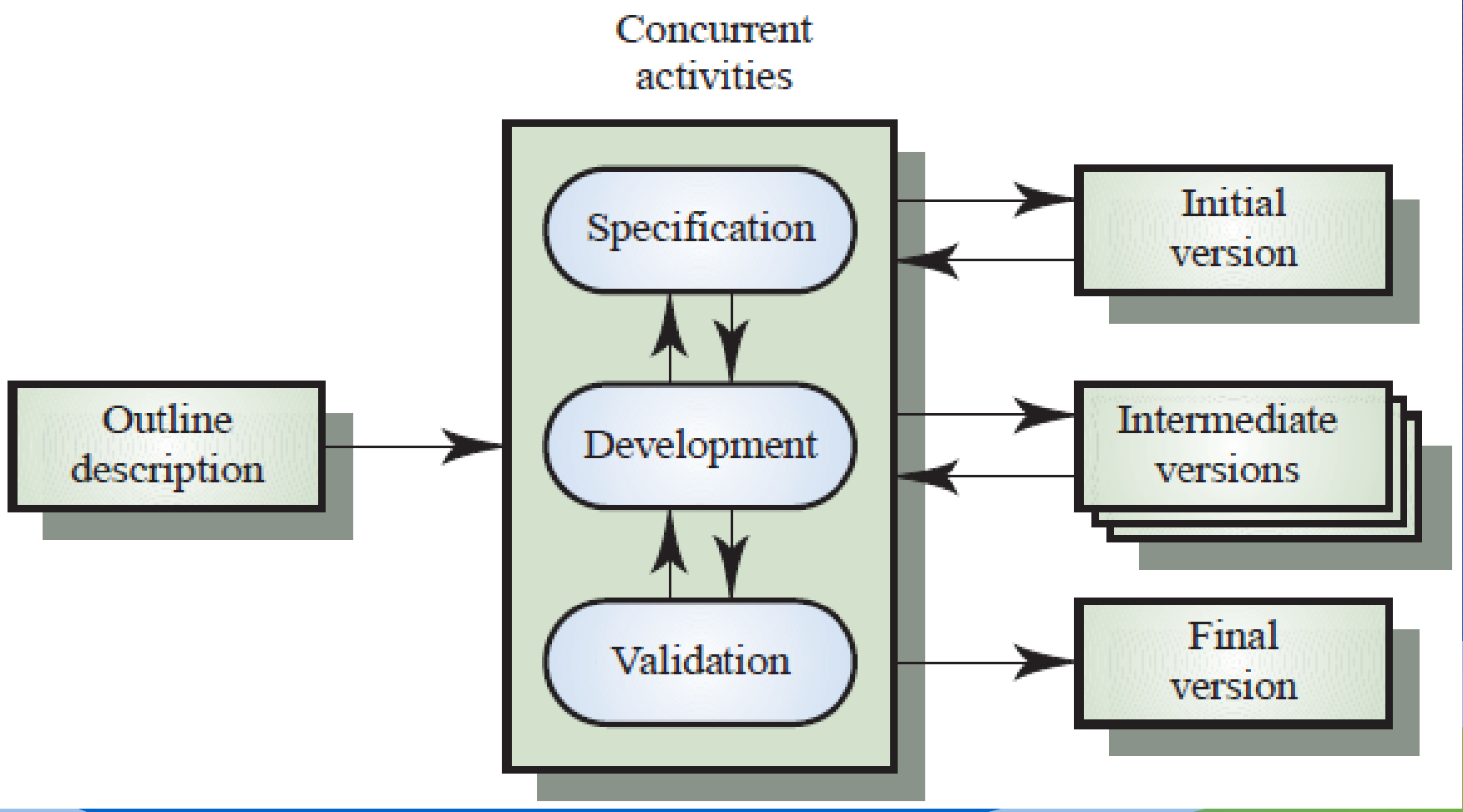
폭포수 모델 문제점

- 프로젝트를 개별적인 단계들로 분할하는것의 비융통성
- 이것이 변경하는 고객의 요구사항에 응답하는 것을 어렵게 함.
- 그러므로, 이 모델은 요구사항이 잘 정의될 때에만 적절함.

진화적 개발

- 탐험적(Exploratory) 개발
  - 고객과 함께 작업하여 초기의 개략적 명세서로부터 최종 시스템을 개발하는 것이 목적. 요구사항이 잘 정의된 경우에 적합함.(상세한 명세서를 작성할 수 없을때)
- Throw-away 프로토타이핑
  - 시스템 요구사항을 이해하는 것이 목적임. 요구사항을 잘 이해하지 못할 때 적함함.

진화적 개발(계속)


Concurrent activities
Specification
Development
Validation
Outline description
Initial version
Intermediate versions
Final version

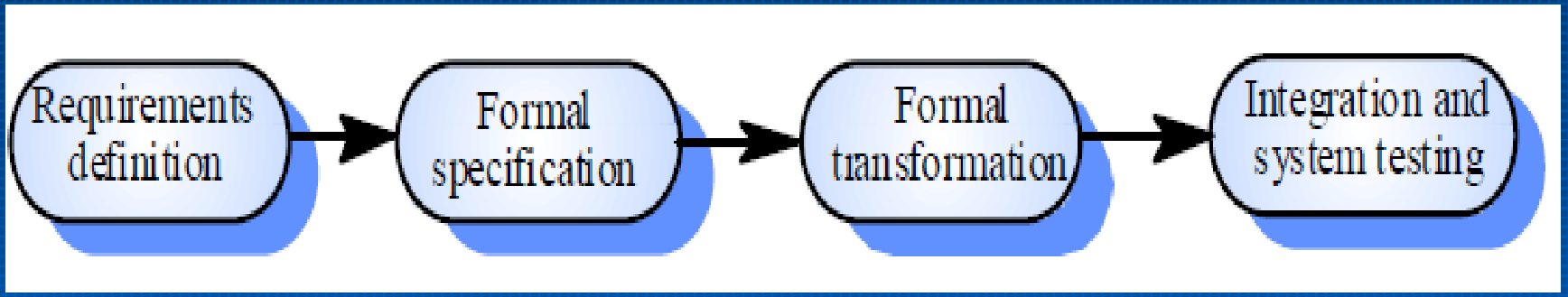
진화적 개발

- 문제점
  - 프로세스 가시성의 부족
  - 종종 시스템의 구조가 나쁘게 됨.
  - 특별한 기술(빠른 프로토타이핑을 위한 언어의 선택과 같은)이 필요할 수도 있음.
- 적용성(Applicability)
  - 작은 또는 중간 규모의 대화식 시스템
  - 대형 시스템의 일부분(예, 사용자 인터페이스)
  - 수명이 짧은 시스템

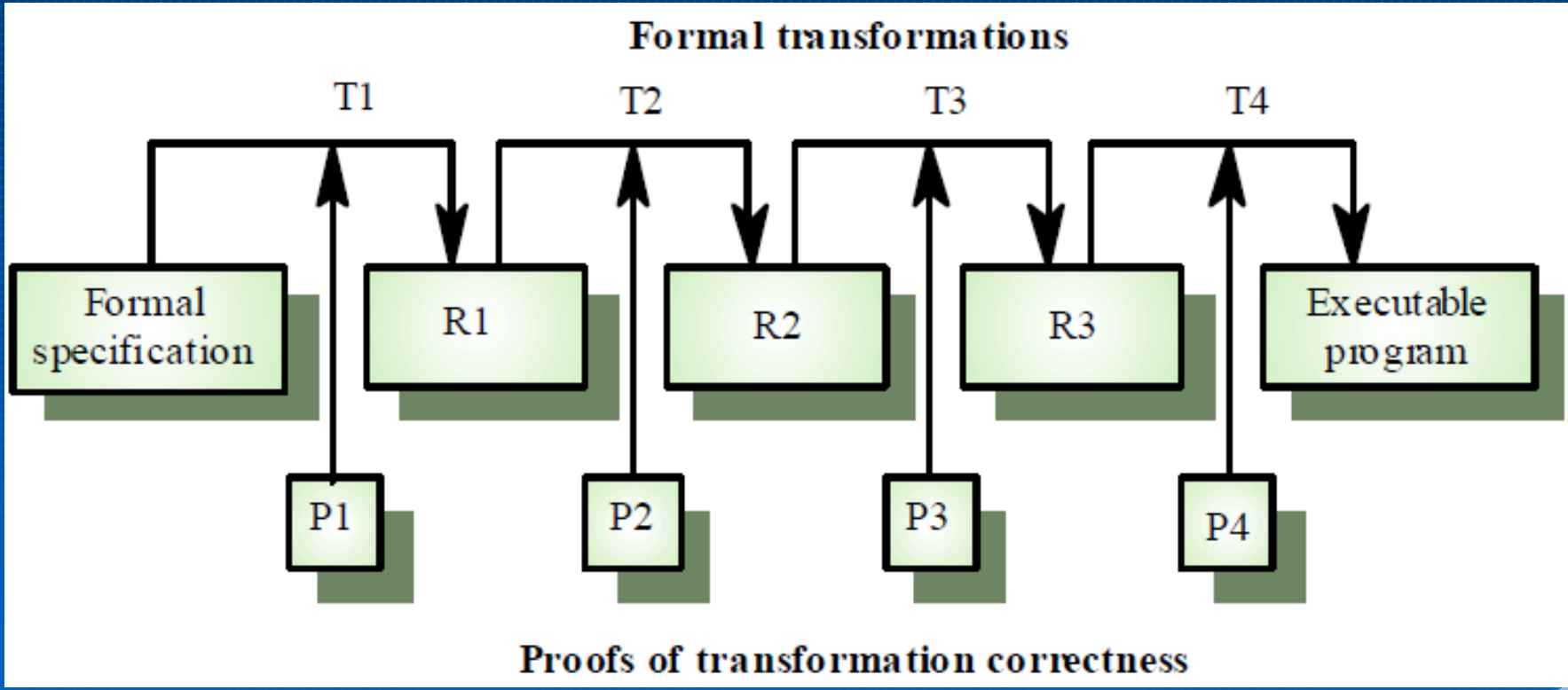
정형적 시스템 개발

- 수학적 명세서로부터 다양한 표현방식을 거쳐 실행가능한 프로그램으로 변환하는 것에 기초
- 변환은 정확성을 유지함(correctness-preserving) 따라서 프로그램이 명세서에 적합하다는 것을 보이는 것은 매우 쉽다.
- 소프트웨어 개발의 청정실(Cleanroom) 접근법에 포함됨.

정형적 시스템 개발


Requirements definition
Formal specification
Formal transformation
Integration and system testing

정형적 변환


Formal transformations
T1
T2
T3
T4
Formal specification
R1
R2
R3
Executable program
P1
P2
P3
P4
Proofs of transformation correctness

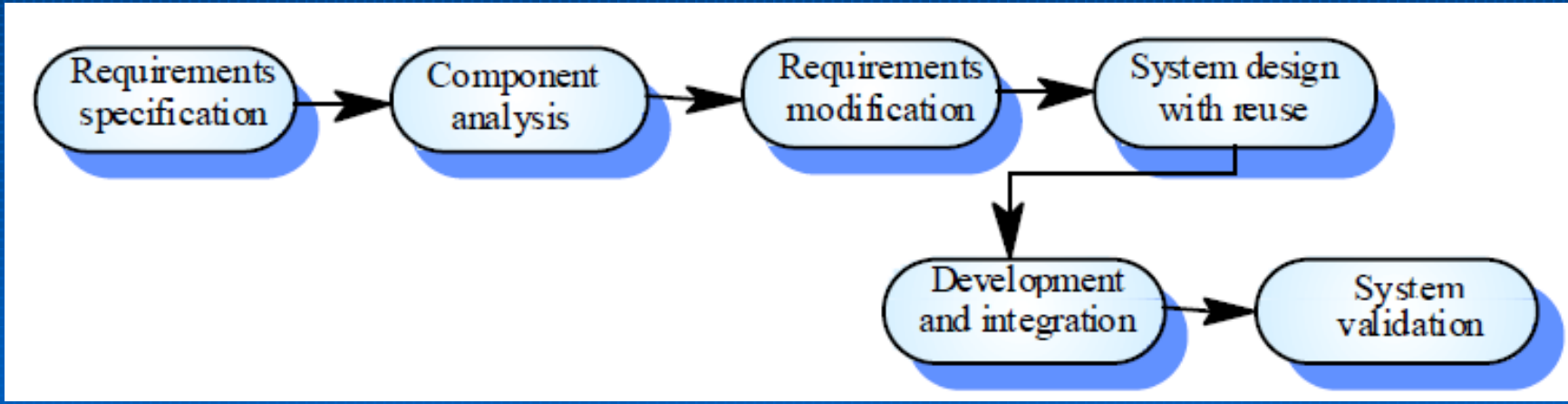
정형적 시스템 개발

- 문제점
  - 전문적인 기술과 함께 기술을 적용하기 위한 교육이 필요.
  - 시스템의 어떤 부분(예를 들면 사용자 인터페이스)들을 정형적으로 명세화하는 것이 어려움.
- 적용성
  - 시스템이 가동되기 전에 안전도 또는 보안이 확실하게 이루어져야하는 중요한(Critical) 시스템

재사용 중심 개발

- 시스템들이 기존의 컴포넌트들 또는 COTS (Commercial-off-the-shelf) 시스템으로부터 통합되는 것과 같은 체계적인 재사용에 기초
- 프로세스 단계
  - 컴포넌트 분석
  - 요구사항 수정
  - 재사용을 고려한 시스템 설계
  - 개발과 통합
- 이 접근법은 점점 중요해지고 있다. 그러나 아직도 이 방법에 대한 경험이 부족한 실정임.

재사용 중심 개발


Requirements specification
Component analysis
Requirements modification
System design with reuse
Development and integration
System validation

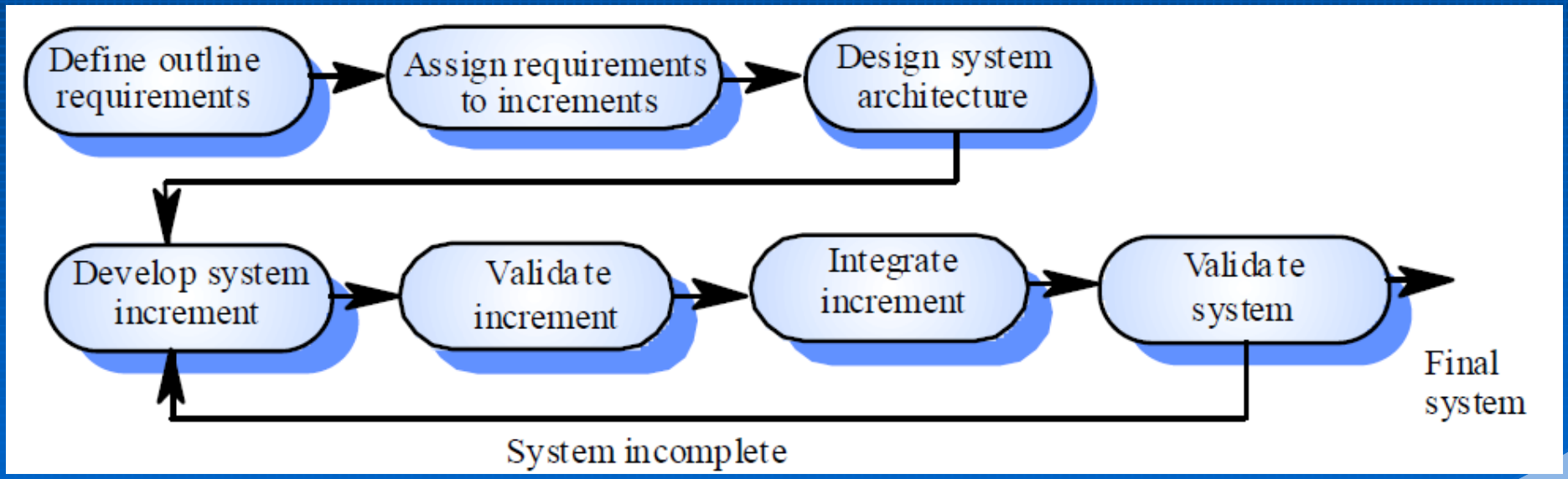
프로세스 반복

- 시스템 요구사항은 프로젝트가 진행하는 동안 항상 진화한다. 따라서 초기 단계를 재작업하는 프로세스 반복은 항상 대형 시스템을 위한 프로세스의 일부분이다.
- 반복은 일반적인 프로세스 모델의 어떤 것에도 적용될 수 있다.
- 2개의 (관련된) 접근법
  - 점진적(Incremental) 개발
  - 나선형(Spiral) 개발

점진적 개발

- 시스템을 한번에 인도하는 것이 아니라, 개발과 인도가 여러 번으로 나누어진다. 각각의 증가분에서는 요구되는 기능들을 인도한다.
- 사용자의 요구사항들은 우선순위가 매겨지며, 높은 우선순위를 갖는 요구사항들을 먼저 인도한다.
- 증가분의 개발이 시작되면, 요구사항에는 변동이 없게되며, 다음번 증가분에 대한 요구사항이 계속해서 진화하게 된다.

점진적 개발


Define outline requirements
Assign requirements to increments
Design system architecture
Develop system increment
Validate increment
Integrate increment
Validate system
Final system
System incomplete

점진적 개발의 이점

- 고객에게 필요한 부분이 각각의 증가분마다 인도될 수 있다. 따라서 시스템의 기능이 초기에 이용 가능하다.
- 초기의 증가분이 나중의 증가분에 대한 요구사항을 이끌어내는 프로토타입의 역할을 한다.
- 전체적인 프로젝트 실패에 대한 위험이 적음.
- 가장 높은 우선순위를 갖는 시스템 서비스는 많은 테스팅을 거치게 된다.

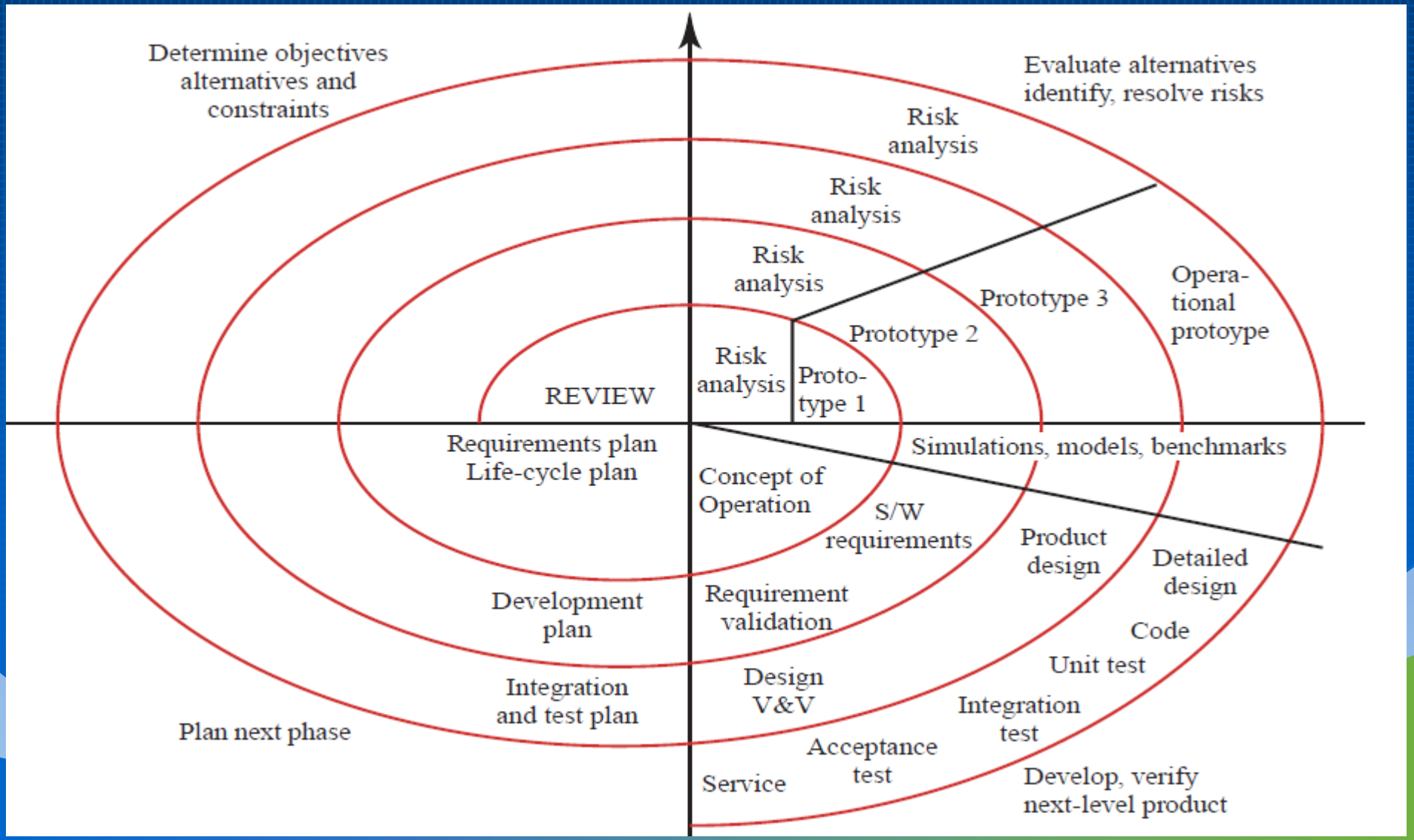
극단적(Extreme) 프로그래밍

- 매우 적은 기능성을 가진 증가분의 개발과 인도를 기초로 하는 개발에 대한 새로운 접근법
- 계속적인 코드 향상, 개발 팀에 사용자의 포함, 그리고 pairwise 프로그래밍에 의존함.

나선형(spiral) 개발

- 프로세스를 백트랙킹을 가진 활동들의 순서가 아닌 나선형으로 표현.
- 나선내의 각각의 루프는 프로세스내의 단계를 나타냄.
- 프로세스 내내 위험(Risk)이 명백하게 측정되고 해결된다.

소프트웨어 프로세스의 나선형 모델



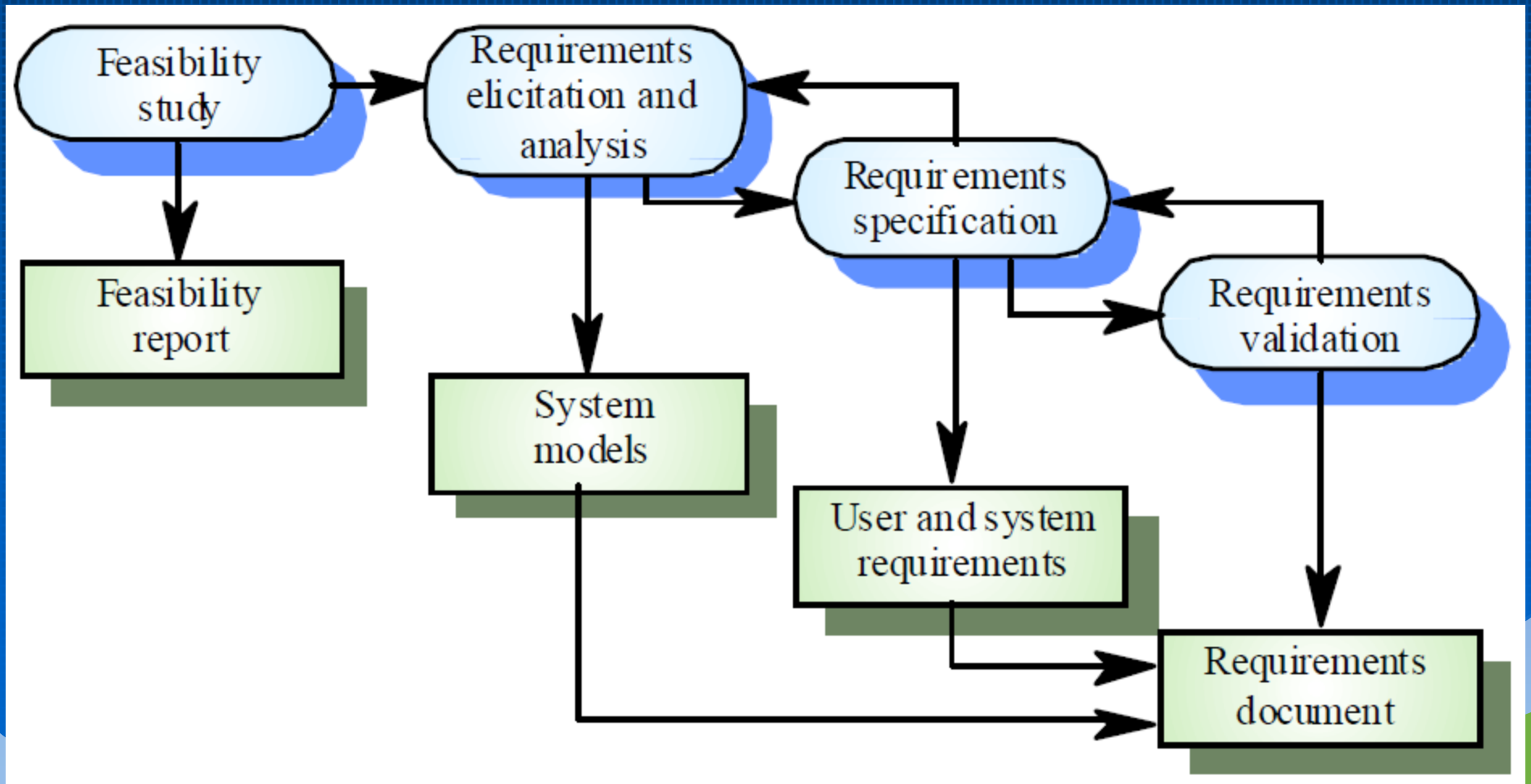
나선형 모델의 섹터

- 목표 설정
  - 프로젝트 단계를 위한 특정한 목적을 식별함.
- 위험 측정과 감소
  - 위험들을 측정하고, 주요한 위험들을 줄이기 위한 활동들이 위치함.
- 개발과 확인
  - 시스템에 대한 개발 모델을 선택. 일반적인 모델 중 어떤 것이라도 상관 없음.
- 계획수립
  - 프로젝트 검토, 나선형의 다음 단계에 대한 계획.

소프트웨어 명세화

- 어떤 서비스가 필요하고, 시스템의 동작과 개발에 대한 제한사항을 설립하는 과정.
- 요구(Requirements) 공학 프로세스
  - 가능성(Feasibility) 연구
  - 요구사항 유도와 분석
  - 요구사항 명세화
  - 요구사항 확인

요구 공학 프로세스


Feasibility study
Requirements elicitation and analysis
Requirements specification
Requirements validation
Feasibility report
System models
User and system requirements
Requirements document

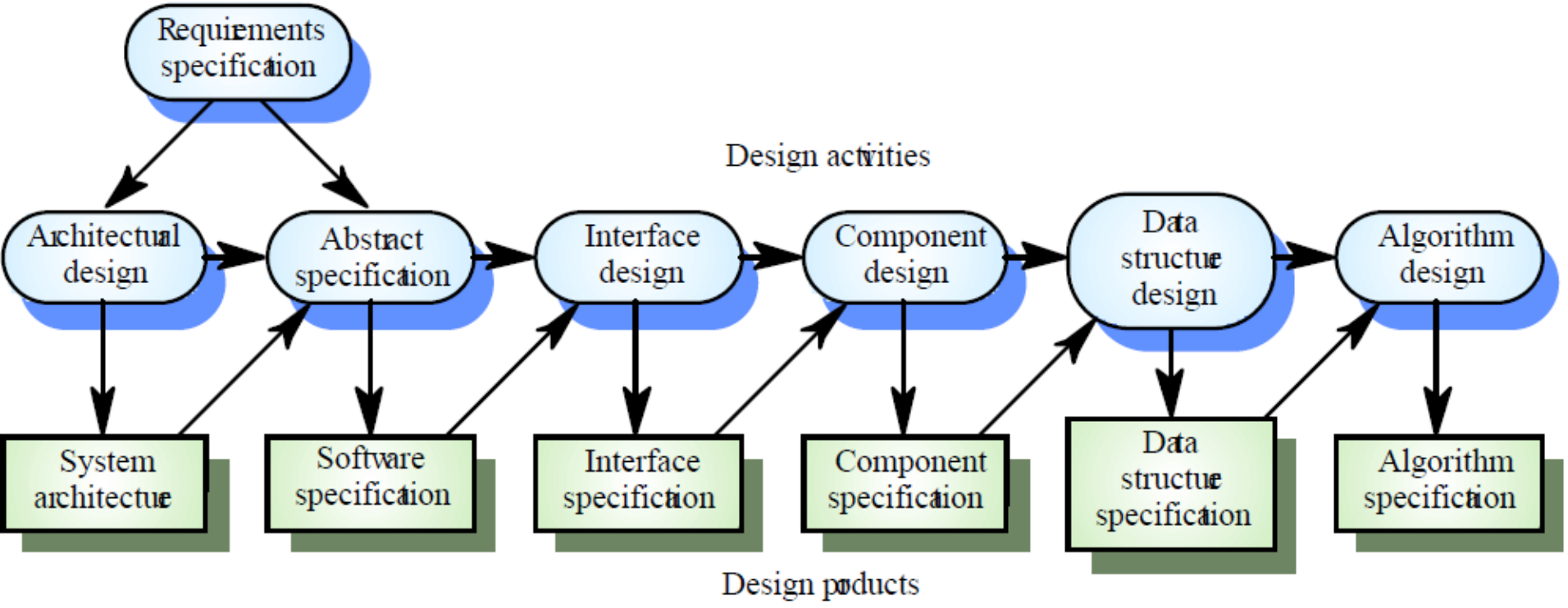
소프트웨어 설계와 구현

- 시스템 명세서를 실행가능한 시스템으로 변환하는 프로세스
- 소프트웨어 설계
  - 명세서를 실현하는 소프트웨어 구조를 설계
- 구현
  - 이 구조를 실행가능한 프로그램으로 변환.
- 설계와 구현 활동은 밀접하게 관련되어 있고, 혼재될 수 있다.

설계 프로세스 활동

- 구조(Architectural) 설계
- 추상적인 명세화
- 인터페이스 설계
- 컴포넌트(부품) 설계
- 자료 구조 설계
- 알고리즘 설계

소프트웨어 설계 프로세스



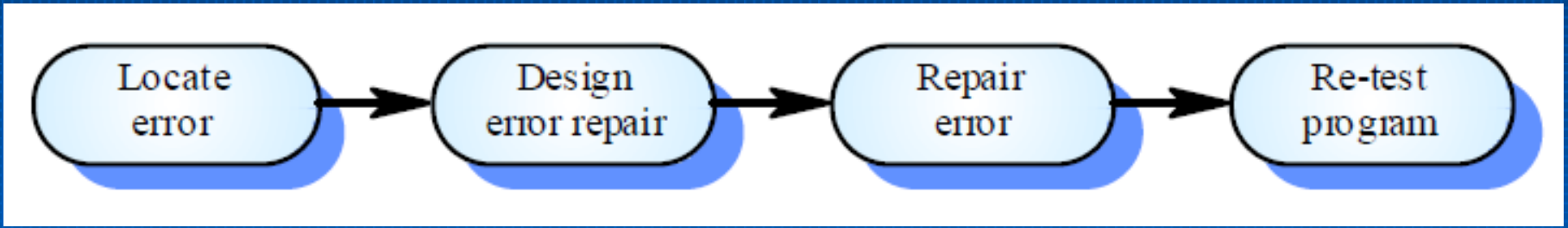
설계 방법

- 소프트웨어 설계물을 개발하기 위한 체계적인 접근법
- 설계물은 대체로 여러 그래픽 모델들로 문서화된다.
- 가능한 모델들
  - 자료 흐름(Data-flow) 모델
  - 개체-관계-속성(Entity-relation-attribute) 모델
  - 구조적 모델
  - 객체 모델

프로그래밍과 디버깅

- 설계를 프로그램으로 변환하고 프로그램에 있는 오류들을 제거.
- 프로그래밍은 개별적인 활동 – 일반적인 프로그래밍 프로세스는 없음.
- 프로그래머들은 프로그램에 있는 오류들을 찾아내기 위해 프로그램 테스팅을 수행함. 그리고 이 오류들을 디버깅 프로세스에서 제거함.

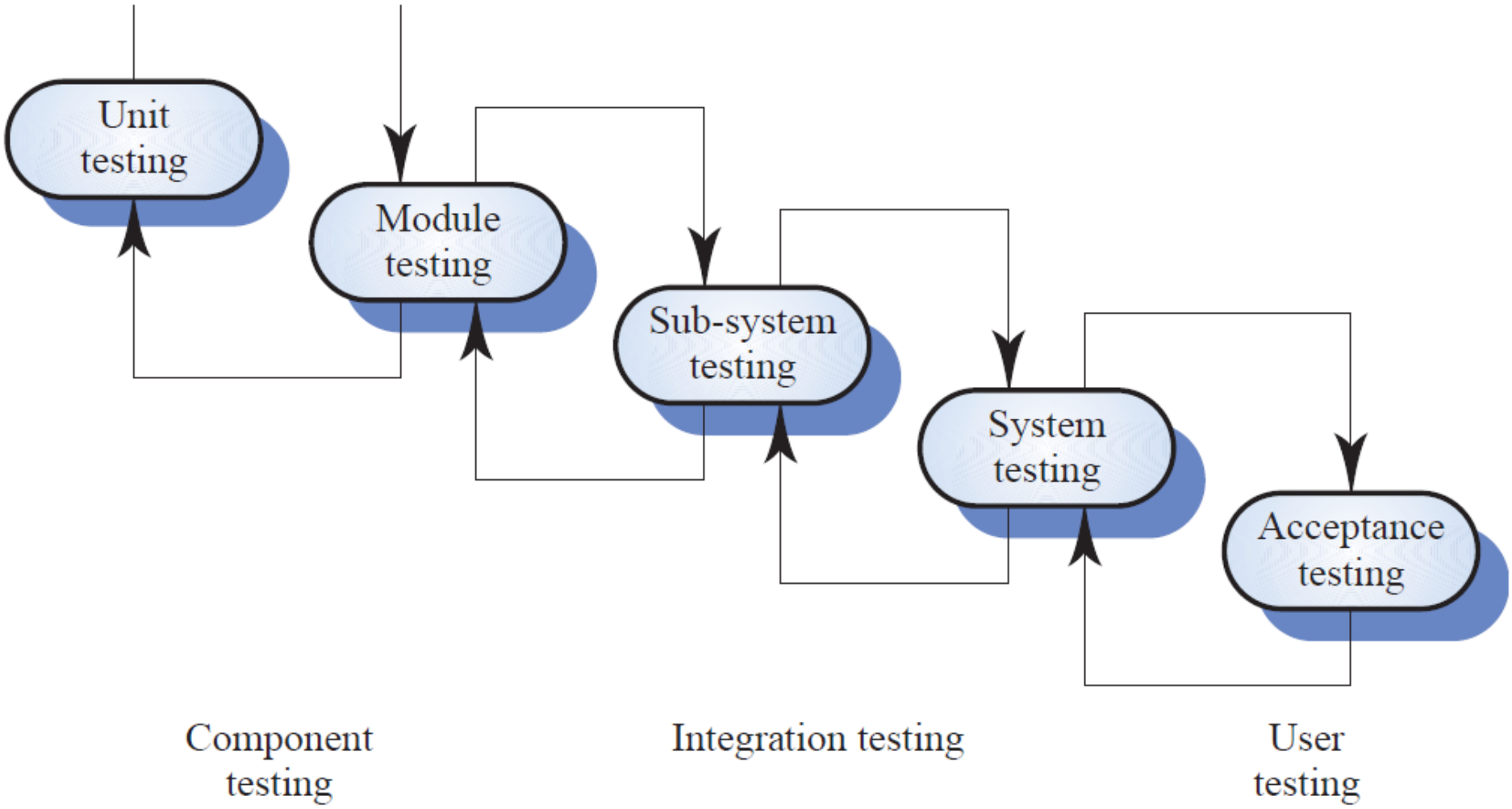
디버깅 프로세스


Locate error
Design error repair
Repair error
Re-test program

소프트웨어 확인

- 검증과 확인(Verification and validation)은 시스템이 명세서에 적합한지와 고객의 요구사항을 만족하는지를 보이는 것임.
- 검사와 검토 과정, 그리고 시스템 테스팅을 포함함.
- 시스템 테스팅은 시스템에 의해 처리되는 실제 데이터에 대한 명세서로부터 유도된 테스트 케이스들을 가지고 시스템을 실행시키는 것임.

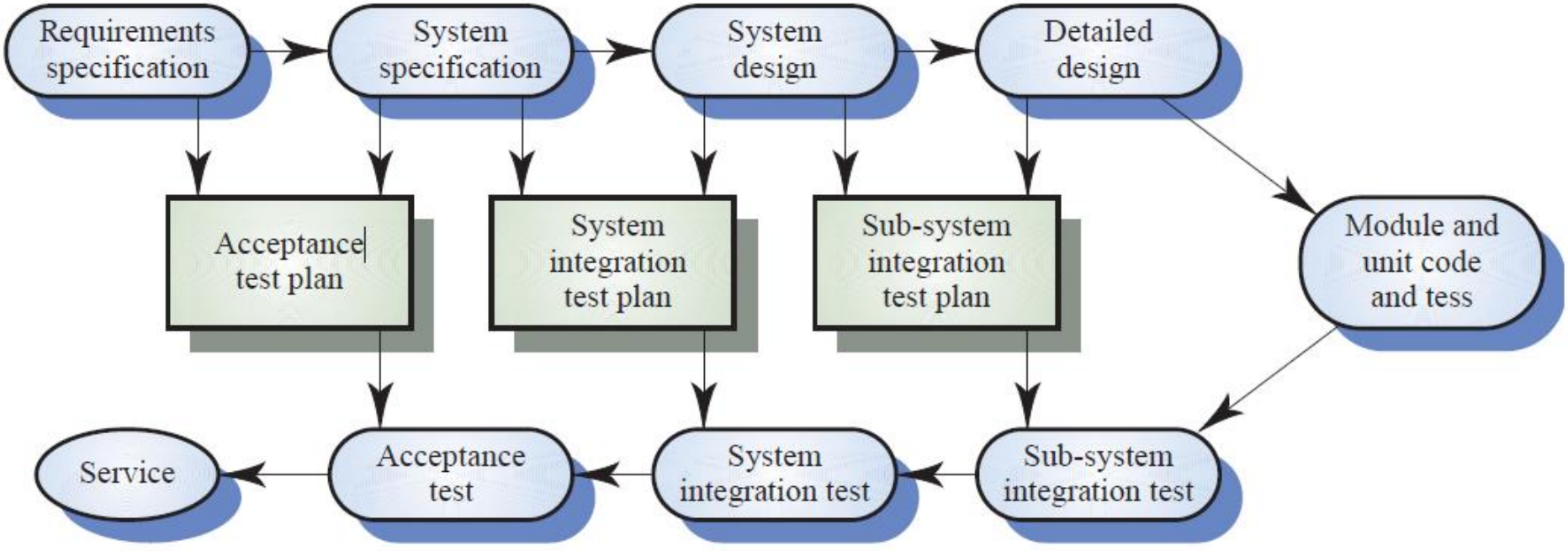
테스팅 프로세스


Unit testing
Module testing
Sub-system testing
System testing
Acceptance testing
Component testing
Integration testing
User testing

테스팅 단계

- 단위 테스팅
  - 개별적인 부분들을 시험.
- 모듈 테스팅
  - 관련된 부분들의 모음을 시험함.
- 서브시스템 테스팅
  - 모듈들을 서브시스템으로 통합하고 시험함. 초점은 인터페이스 테스팅임.
- 시스템 테스팅
  - 전체로서의 시스템 시험. 중요한 성질들에 대한 시험.
- 인수 테스팅
  - 인수가능성을 검사하기 위한 고객 데이터로의 시험

테스팅 단계

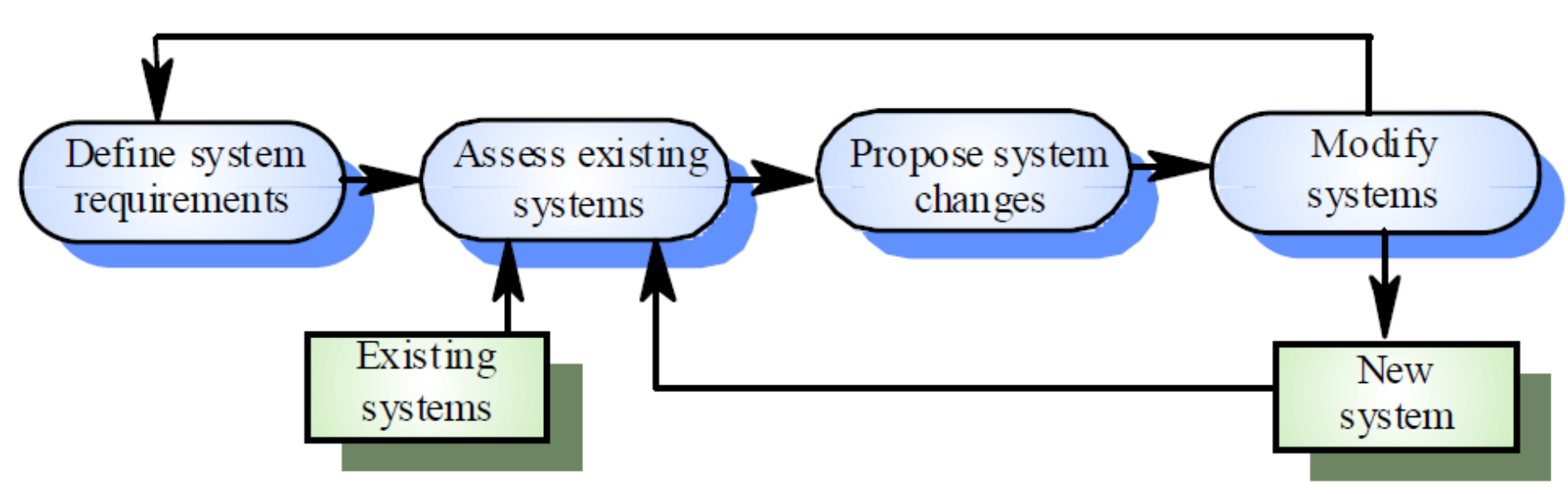


alpha-test와 beta test의 차이는?

소프트웨어 진화

- 소프트웨어는 본래가 유연하고(flexible) 변할 수 있다.
- 비즈니스 환경에 따라 요구사항이 변화하면, 비즈니스를 지원하는 소프트웨어 또한 반드시 진화하여야 한다.
- 비록 개발과 진화(유지보수) 사이에는 경계가 있지만, 완전히 새로운 시스템의 수가 점점 적어짐에 따라 이 경계도 점차로 구분이 모호해져 감.

시스템 진화


Define system requirements
Assess existing systems
Propose system changes
Modify systems
Existing systems
New system

자동화된 프로세스 지원 (CASE)

- Computer-aided Software engineering (CASE)는 소프트웨어 개발과 진화 프로세스를 지원하는 소프트웨어이다.
- 활동 자동화(Activity automation)
  - 시스템 모델 개발을 위한 그래픽 편집기
  - 설계 개체들을 관리하기 위한 자료 사전(Data dictionary)
  - 사용자 인터페이스 구축을 위한 그래픽 UI 생성기
  - 프로그램 오류 발견을 지원하는 디버거
  - 프로그램의 새로운 버전을 생성하기 위한 자동 번역기

CASE 기술

- CASE 기술은 소프트웨어 프로세스에서 상당한 개선을 가져왔다. (비록 개선의 정도가 예측했던 것에는 못미치지만)
  - 소프트웨어 공학은 창조적인 생각을 요구한다
    - 이것을 자동화하기에는 쉽지 않다.
  - 소프트웨어 공학은 대형 프로젝트를 위한 팀 활동이고, 팀원간의 상호작용에 많은 시간이 소요된다. CASE 기술이 이런 것들을 정말로 지원하지는 못하고 있다.

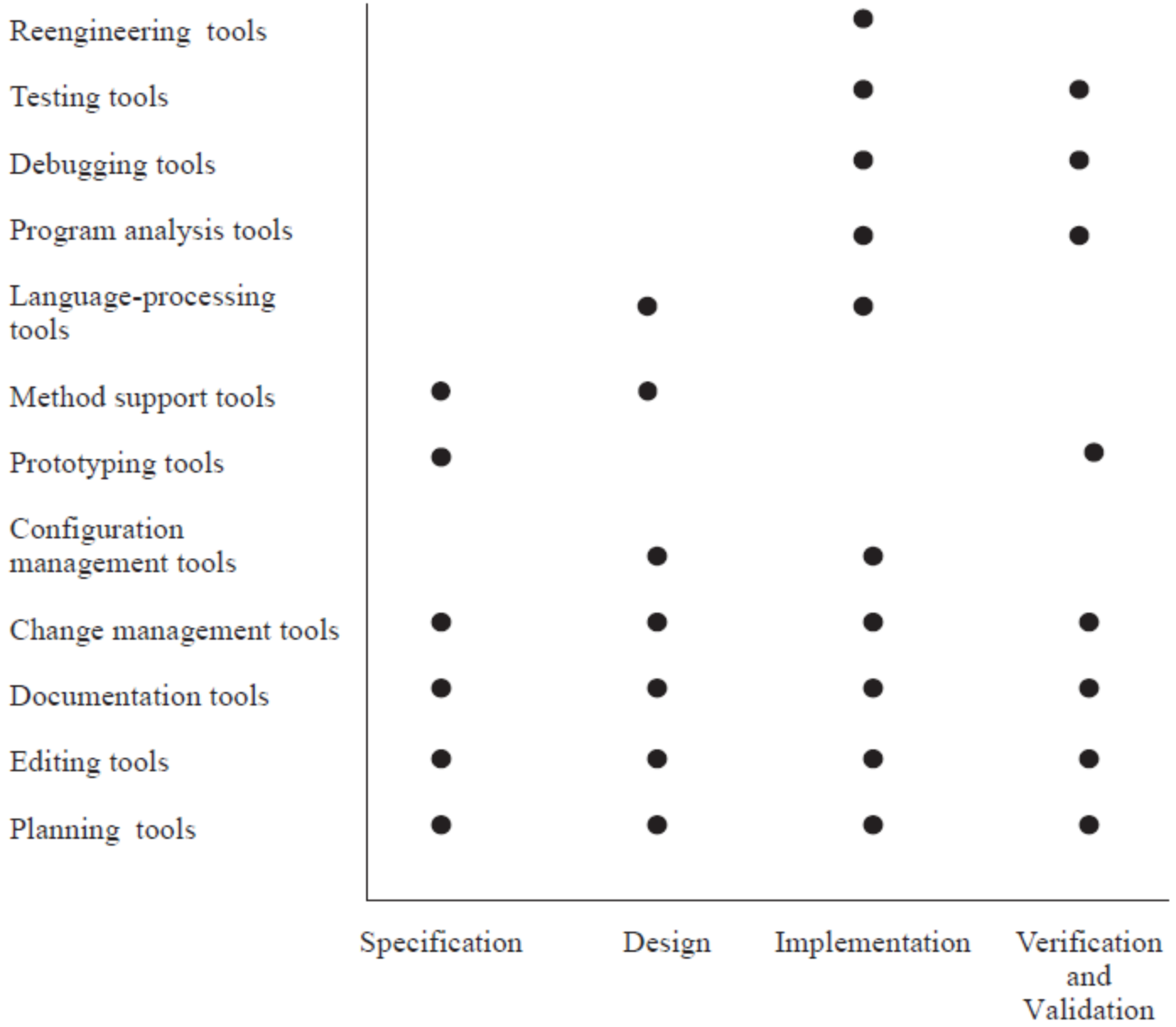
CASE 분류

- 분류는 여러 유형의 CASE 도구들과 프로세스 활동들에 대한 지원 방법을 이해하는 데 도움을 준다.
- 기능적인 관점
  - 도구들의 특정한 기능에 따라 도구들을 분류함.
- 프로세스 관점
  - 도구들이 지원하는 프로세스 활동들에 따라 분류함.
- 통합의 관점
  - 도구들이 통합된 단위(정도, 구조)로서 분류함.

기능적인 분류

Tool type	Examples
Planning tools	PERT tools, estimation tools, spreadsheets
Editing tools	Text editors, diagram editors, word processors
Change management tools	Requirements traceability tools, change control systems
Configuration management tools	Version management systems, system building tools
Prototyping tools	Very high-level languages, user interface generators
Method-support tools	Design editors, data dictionaries, code generators
Language-processing tools	Compilers, interpreters
Program analysis tools	Cross reference generators, static analysers, dynamic analysers
Testing tools	Test data generators, file comparators
Debugging tools	Interactive debugging systems
Documentation tools	Page layout programs, image editors
Re-engineering tools	Cross-reference systems, program re-structuring systems

활동에 따른 분류


Reengineering tools
Testing tools
Debugging tools
Program analysis tools
Language-processing tools
Method support tools
Prototyping tools
Configuration management tools
Change management tools
Documentation tools
Editing tools
Planning tools
Specification
Design
Implementation
Verification and Validation

CASE 통합

- 도구
  - 설계 일관성 검사, 텍스트 편집과 같은 개별적인 프로세스 작업들을 지원.
- 워크벤치
  - 명세화 또는 설계와 같은 하나의 프로세스 단계를 지원, 대체로 여러 도구들이 통합되어 있음.
- 환경
  - 전체 소프트웨어 프로세스의 전부 또는 상당한 부분을 지원함. 대체로 여러 개의 통합된 워크벤치들을 포함함.

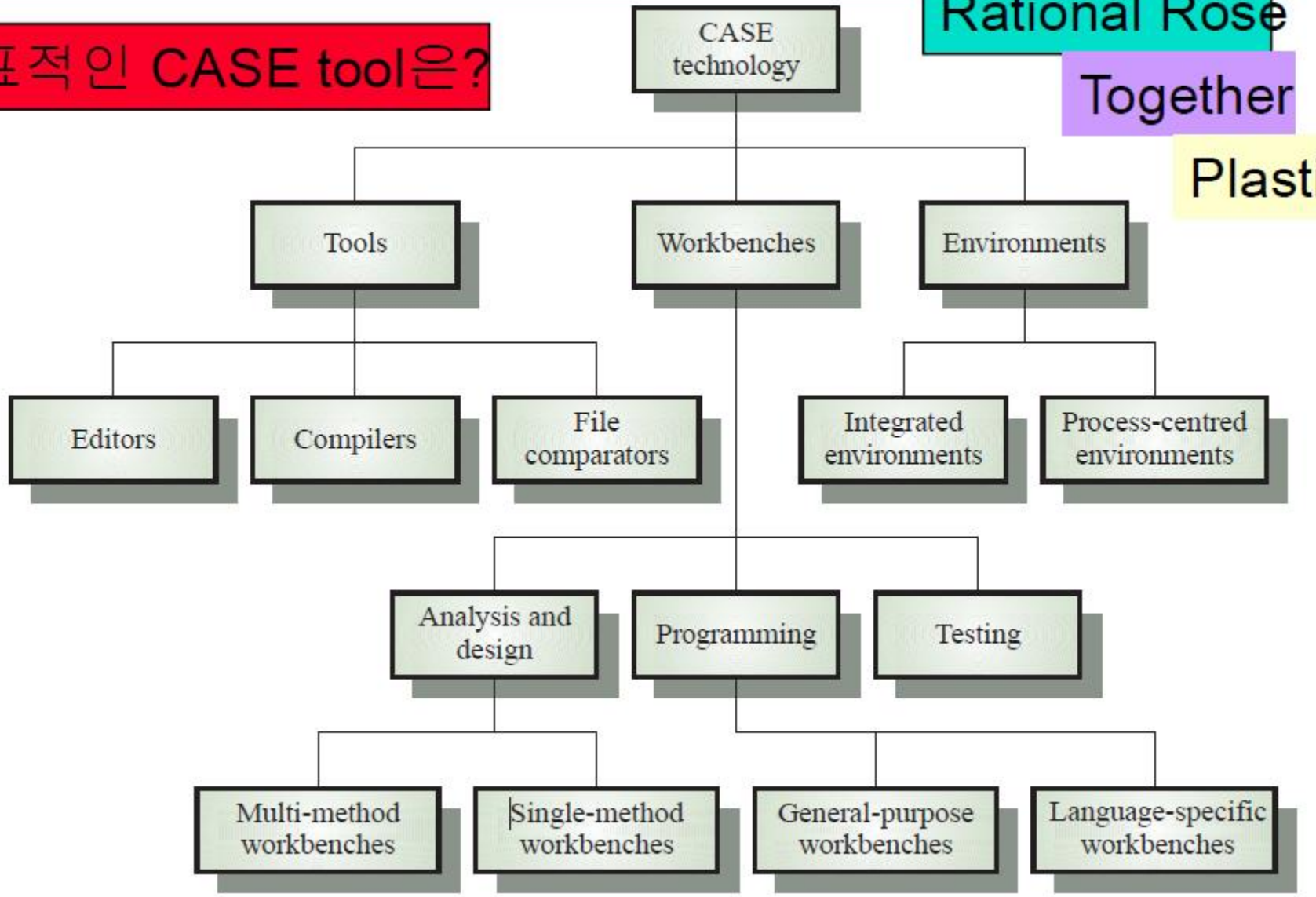
도구, 워크벤치, 환경

대표적인 CASE tool은?

Rational Rose

Together

Plastic



Key points

- 소프트웨어 프로세스는 소프트웨어 시스템을 생산하고 진화하는데 관련되는 활동들이다. 이것들은 소프트웨어 프로세스 모델로 표현된다.
- 일반적인 활동으로는 명세화, 설계와 구현, 확인과 진화 등이다.
- 일반적인 프로세스 모델들은 소프트웨어 프로세스의 구조를 기술한다.
- 반복적인 프로세스 모델들은 소프트웨어 프로세스를 활동들이 반복되는 것으로 기술한다.

Key points(계속)

- 요구 공학은 소프트웨어의 명세서를 개발하는 프로세스이다.
- 설계와 구현 프로세스는 명세서를 실행가능한 프로그램으로 변환하는 프로세스이다.
- 확인과 검증은 시스템이 명세서와 사용자 요구사항을 만족하는지를 검사하는 것을 포함한다.
- 진화는 시스템이 사용된 이후에 그것을 변경하는 것에 관련된다.
- CASE 기술은 소프트웨어 프로세스 활동들을 지원한다.