

소프트웨어 공학 & 테스팅

경북대학교 IT대학 컴퓨터학부
이우진 교수

교재

- 소프트웨어 공학 관련 책
 - IAN Sommerville, Software Engineering, -10th Edition, Addison-Wesley, 2016
- 소프트웨어 테스트 관련 책
 - Paul C. Jorgensen, Software Testing, A Craftsman's Approach, fourth edition, CRC Press, 2014
 - 소프트웨어 테스트 관련 Open Source Tool

수업 진행

- 강의
 - 소프트웨어 테스트 기법
- 실습
 - 수업 중 오픈소스 자동화 도구 실습
 - 테스트 케이스 생성 기법(Black-box, White box)
 - 자동 단위 테스트(JUnit)
 - 시스템 테스트 도구
- 개인별 프로젝트
 - Use Case Diagram, Class Diagram, 테스트의 충실도 평가

평가 방법

- 평가 방법
 - 중간고사 (40%)
 - 기말고사 (40%)
 - 개인별 프로젝트 (20%)

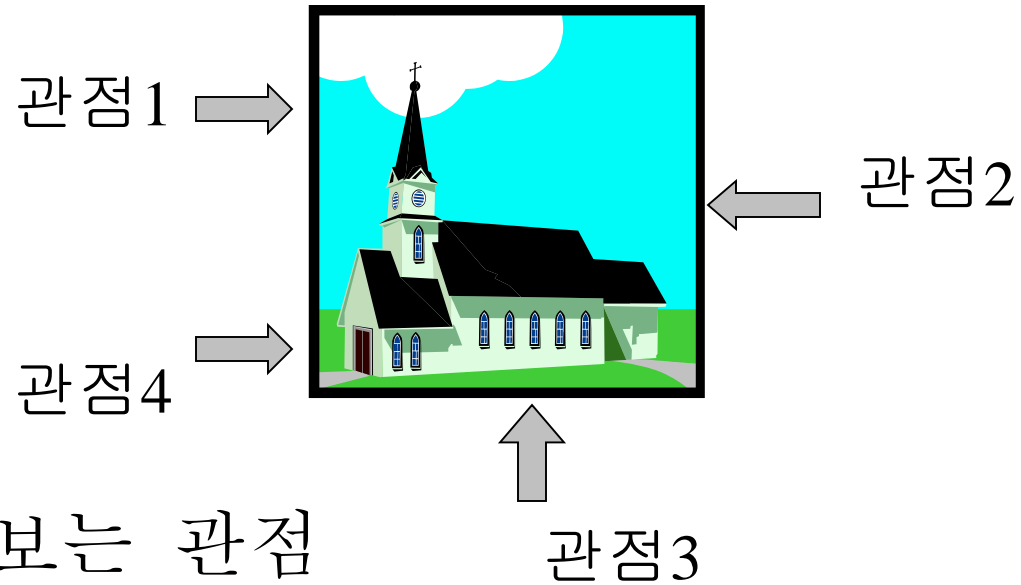
Software 개요

$I \rightarrow \boxed{P} \rightarrow O$

관점의 차이

- 건축물을 바라보는 관점

- 시공자
- 거주자
- 설계사
- 작업 인부, ...



- 소프트웨어를 바라보는 관점

- 의뢰자
- designer
- programmer
- Users, ...

소프트웨어

- 소프트웨어란?

- 프로그램과 프로그램의 개발, 운용, 수정 및 기능 확장을 위해 필요한 모든 정보
- 소프트웨어는 프로그램의 동적인 의미
 - 프로그램은 형식 언어(formal language)로 표현된 지적 구축물

소프트웨어의 특성

- 소프트웨어의 **특성**
 - 비가시성(Invisibility)
 - 개념적, 무형적
 - 구조가 코드 내에 내재
 - 복잡성(Complexity)
 - 개발 과정이 복잡
 - 대상 업무, 대상 시스템이 난해
 - 순응성(Conformity), 변경 가능성(Changeability)
 - 요구나 환경 변화에 적절히 변형 가능
 - 복제 가능성(Duplicability)
 - 테스트의 어려움(Intestability)

소프트웨어의 유형(I)

- 소프트웨어 분류 체계

- 기능

- 시스템 소프트웨어, 응용 소프트웨어

- 개발 과정

- newly-built, 재사용(reuse), modification and adaptation of existing software, porting

- 하드웨어 환경

- multiprocessor, ...

- 요구되는 신뢰도

- critical,, non-critical

- 소프트웨어 크기

- programming-in-the-small, programming-in-the-large....

- 데이터 특성

- 그래픽, 이미지, 텍스트, 음성,

소프트웨어의 유형(II)

- **정보시스템(Information Systems)**

- 대량의 데이터의 분류, 저장, 검색에 관심
- 컴퓨터 초심자들도 데이터베이스에 쉽게 접근할 수 있도록 인터페이스 제공
- 시스템 사례
 - electronic fund transfer network
 - credit card authorization services
 - airline reservation systems
- 정보 시스템의 특성
 - 시스템이 정적(static)이지 않다.
 - 데이터 설계는 비중이 큰 반면 제어 구조는 비교적 간단하다.
 - 개발, 운용 및 유지 보수에 많은 노력과 비용이 든다.
 - 문서가 중요한 요소
- 경영 정보 시스템(MIS)
 - 조직에서의 경영, 의사 결정 등을 지원하기 위해 여러 정보를 효과적으로 제공하는 통합된 시스템
 - TPS, EDPS, DSS, EIS, SIS, Information Center, OA

소프트웨어의 유형(III)

- **내포 시스템(Embedded Systems)**
 - A system that is logically incorporated in a larger system whose primary function is not computation
 - large scale, long-lived
 - real-time response, fail-safe reliability
 - tend to be asynchronous, highly parallel, and distributed
 - 시스템 사례
 - 공정 제어(process control)
 - 비행기 유도 시스템(flight guidance)
 - 교환기 시스템(switching systems)

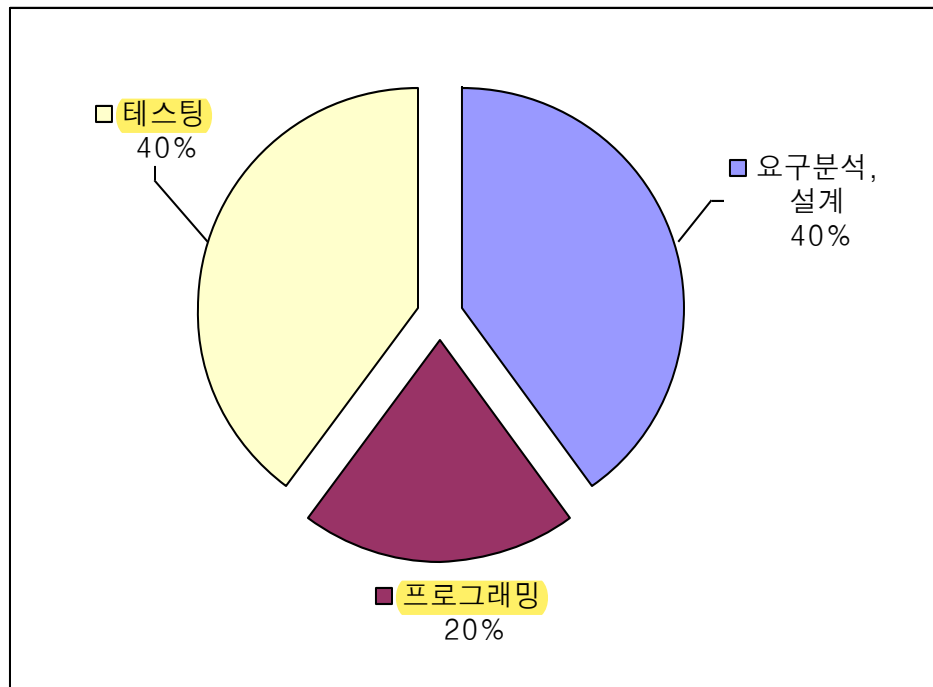
소프트웨어와 관련된 질문 (1)

- 질문 1 : 소프트웨어 시스템을 개발하는 데 드는 비용 중 프로그래밍(코딩)에 드는 비용은 어느 정도인가?
 - 가. 20%
 - 나. 30%
 - 다. 40%
 - 라. 50%

답변

- 가 (20%)

- 소프트웨어 시스템은 물리적이기보다는 논리적인 요소로 구성되기 때문이다. 건축의 경우 80 ~ 90%가 시공에 소요되나 소프트웨어 개발의 경우 약 20% 정도가 소요된다.



소프트웨어와 관련된 질문(2)

- 질문 2 : 중간 사이즈(50,000 라인 이하)의 소프트웨어 시스템을 개발할 때 한 프로그래머가 일년에 만드는 실행 코드(executable code)는 평균 몇 줄(line)이나 될까?
 - 가. 5,000 이하
 - 나. 5,000~10,000
 - 다. 10,000~15,000
 - 라. 15,000 이상

답변

- 가 (5,000 줄 이하)

- 한 프로그래머가 평균 하루에 10줄 정도의 프로그램을 작성한다고 알려져 있다.
 - $10\text{줄} * 5\text{일/주} * 50\text{주} = \text{약 } 2500\text{줄}$ 정도의 실행 코드 작성.
- 미국의 경우, 엔지니어 1인당 1년 계상액 : 약 1억원 (간접 비용 포함)
 - 1인당 코드 생성 능력 : 5,000줄 이내, 1줄당 코딩 비용 : 20,000 원
- 실행 코드 100,000줄의 소프트웨어 시스템 개발 비용?
 - 약 50억원 = 20억(개발 20%, 설계 20%) + 30억

소프트웨어와 관련된 질문(3)

- 질문 3 : 사용자에게 배달되는 소프트웨어의 실행코드 1,000 줄 당 예상 오류의 개수는?
 - 가. 4개 미만
 - 나. 4 ~ 6개
 - 다. 7 ~ 9개
 - 라. 10개 이상

답변

- 가 (4개 이하)
 - 개발 과정에서 발견되는 오류 : 약 **50 ~ 60** 정도/**1000**줄 제품이 완료되어 **배달된 후 발**견되는 오류는 **평균 4개** 이하로 알려져 있다.

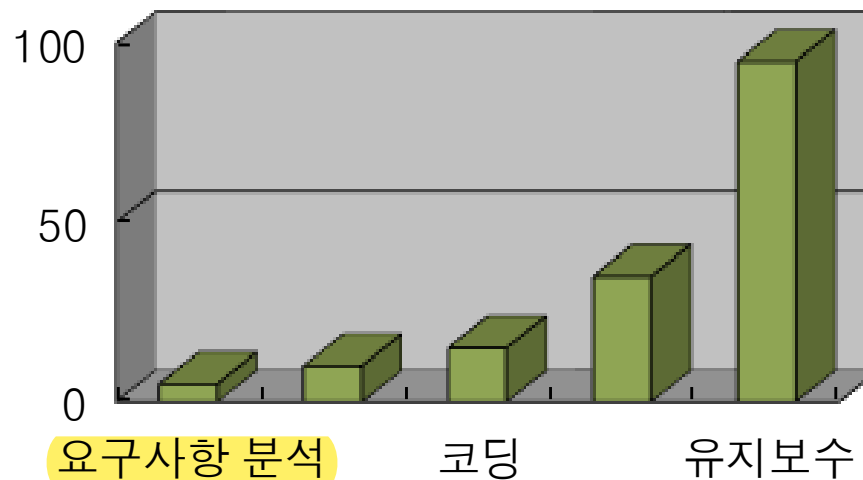
소프트웨어와 관련된 질문(4)

- 질문4 : 사용자가 발견하는 소프트웨어 시스템의 오류 중 발견하기 어려운 것은 어느 것인가?
 - 가. 설계의 오류
 - 나. 프로그래밍의 오류
 - 다. 제안서와 사용자 요구 사항에 대한 잘못된 이해
 - 라. 테스트의 오류

답변

- 다 (제안서와 사용자 요구 사항에 대한 잘못된 이해)
 - 소프트웨어 시스템의 개발에 있어서 가장 어려운 문제
 - 사용자가 무엇을 원하는지 명확하게 정의 내리는 일
 - 프로그래밍을 서둘러 시작할수록 더 오랜 시간이 걸려 늦게 끝난다

오류의 발견 시점과 요구되는 비용



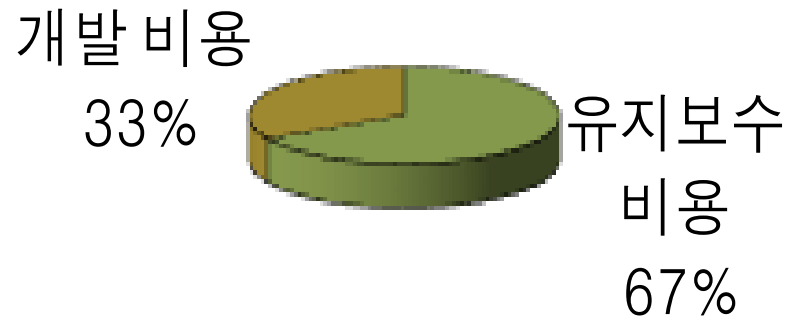
소프트웨어와 관련된 질문(5)

- 문제5 : 소프트웨어 시스템을 유지 보수하는 데 드는 비용이 개발 비용의 몇 배 정도일까?
 - 가. 0.5 배
 - 나. 1 배
 - 다. 1.5 배
 - 라. 2 배

답변

- 라 (2배)

- 유지 보수에 들어가는 비용은 제품이 얼마나 체계적으로 만들어졌느냐에 반비례한다.



소프트웨어에 대한 오해(1)

- 관리자의 오해

- 소프트웨어 개발에 관한 좋은 책들이 있고 책 안에 개발 표준과 단계가 제시되어 있어 우리에게 필요한 모든 것을 제공할 것이다.
- 개발자들에게 필요한 최신 기계(Workstation)나 CASE 도구를 도입하였으니 좋은 제품을 빠른 시일 내에 만들 수 있을 것이다.
- 엔지니어들이 요구 분석을 하고 있으면 생산적이지 못한 일을 하고 있다고 생각한다.
- 공정이 지연될 경우 인력을 더 투입하면 해결된다.

소프트웨어에 대한 오해(2)

- 고객의 오해

- 목표에 대한 개략적인 기술만 해놓으면 소프트웨어를 만드는데 충분하다. 세부적인 것은 나중에 채워 넣으면 된다.
- 사용자의 요구 사항은 계속 변하며 소프트웨어는 융통성이 있어 쉽게 변경을 수용할 수 있다.

소프트웨어에 대한 오해(3)

- 엔지니어의 오해

- 일단 프로그램이 만들어지고 작동하면 우리의 임무는 끝난다.
- 시스템을 작동시켜 보기 전까지는 품질을 평가할 방법이 없다.
- 프로젝트의 결과는 작동하는 프로그램뿐이다.

소프트웨어 위기

- 소프트웨어 개발의 문제


- 예산 초과(Cost Overruns)
- 개발 일정 지연(Late Delivery)
- 불충분한 성능(Inadequate Performance)
- 신뢰하기 어려운 품질(Unreliability)
- 유지 보수의 어려움(Impossible Maintenance)
- Prohibitive Maintenance Costs

- 소프트웨어 위기(Software Crisis)

- 컴퓨터 하드웨어의 급속한 발전과 컴퓨터의 대중화로 인해 소프트웨어의 수요가 급증하였으나 소프트웨어의 생산성과 생산 기술은 그에 미치지 못함으로 나타난 현상

	1965 대 1985	1983 년도 증가율
Increase in Demand	100 times	12 %
Productivity	2 times	4 %
Manpower Supply	10 times	4%

소프트웨어 공학이란 ?

- The disciplined application of engineering, scientific, and mathematical principles and methods to the economical production of **quality software** 
- 품질 좋은 소프트웨어를 최소의 비용으로 계획된 일정에 맞추어 개발하기 위하여 여러 가지 공학적 원리와 방법을 체계적으로 적용하는 것

소프트웨어 공학의 역사

- 소프트웨어 위기의 인식 ~1970
- 소프트웨어 공학의 탄생
 - high-level language programming
 - interactive programming
 - goto 논쟁 (1965)
- 소프트웨어 위기 해소책 모색 ~1980
- 소프트웨어 공학 학문으로 정착
- 하드웨어로부터 독립된 개념의 소프트웨어
 - software lifecycle
 - methodology and tools
 - programming-in-the-large
 - information hiding
 - structured design
- 소프트웨어 공학의 발전 ~ to date
 - software environment, software reuse, software factory
 - CASE (Computer Aided Software Engineering)
 - design patterns, software components, software architecture

소프트웨어 공학의 이슈들

- 저 수준의 소프트웨어 생산성 및 품질
- 노동 집약적인 소프트웨어 생산 과정
 - 가격 상승의 주요 원인
 - 향상된 기술에 둔감하다.
- 숙련된 소프트웨어 공학자의 부족
- 늦은 속도의 기술 이전(technology transfer)
- 소프트웨어 개발 기술의 개발
 - 소프트웨어 생산 과정의 자동화
 - 표준화된 부품을 이용한 생산
- 소프트웨어의 입지 확보
 - Software mind
 - 직업으로서의 소프트웨어 공학
 - legal protection and market development

소프트웨어 공학의 크기 요소

- **Programming-in-the-Small**

- 비교적 잘 정의된 문제의 구현에 관심
- 정확한 프로그램의 개발이 목표
- 명세(specifications)가 좀처럼 변경되지 않는다.

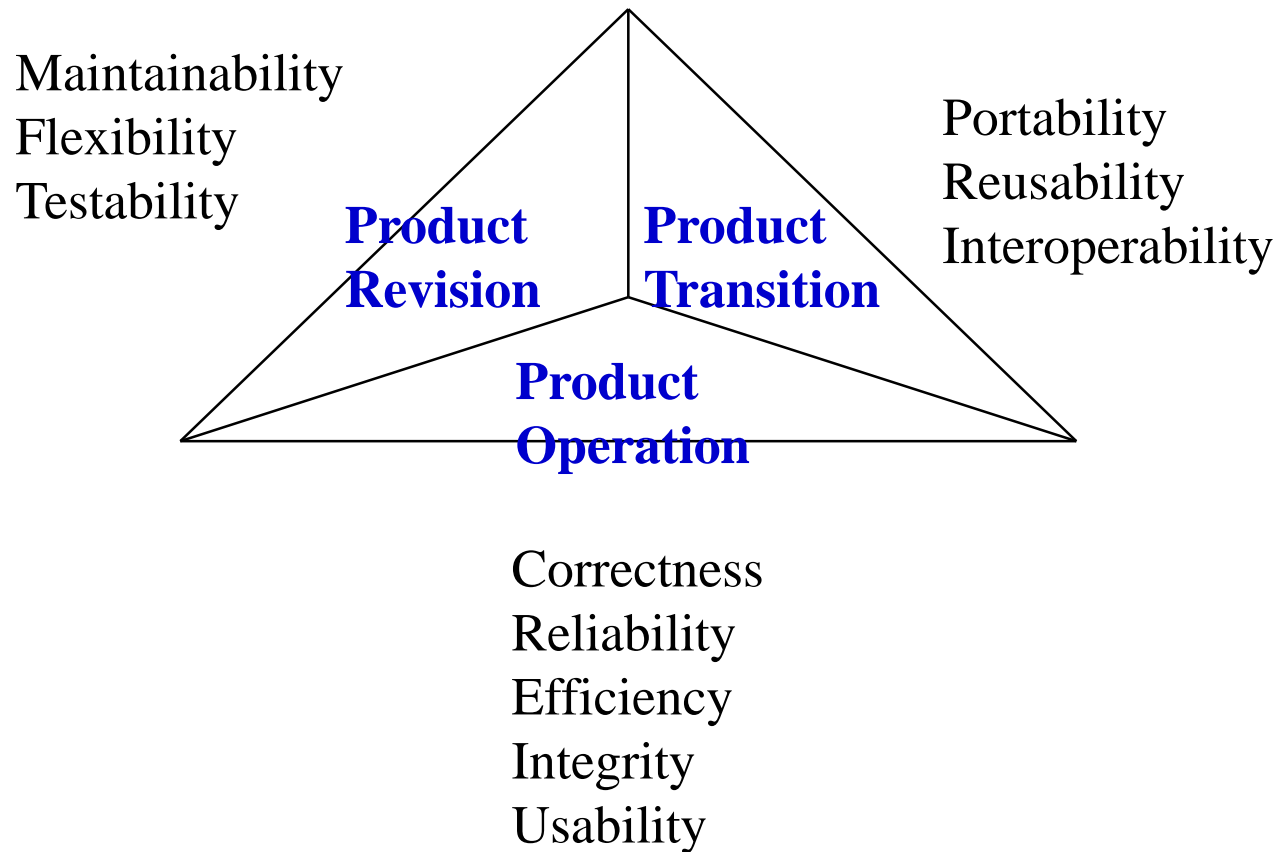
- **Programming-in-the-Large**

- Problem definition
- Changing requirements
- Interface between components
- System integration and testing
- Configuration management
- Maintenance, Documentation, Information management

- **Programming-in-the-Many**

- Communication의 문제 : 팀 원 간의 정보 교환
- Work assignment
- Quality assurance : 표준화(기법, 표기법)의 필요
- Project management : 계획 및 제어, 팀 조직, 팀원의 이적, 요구의 변화

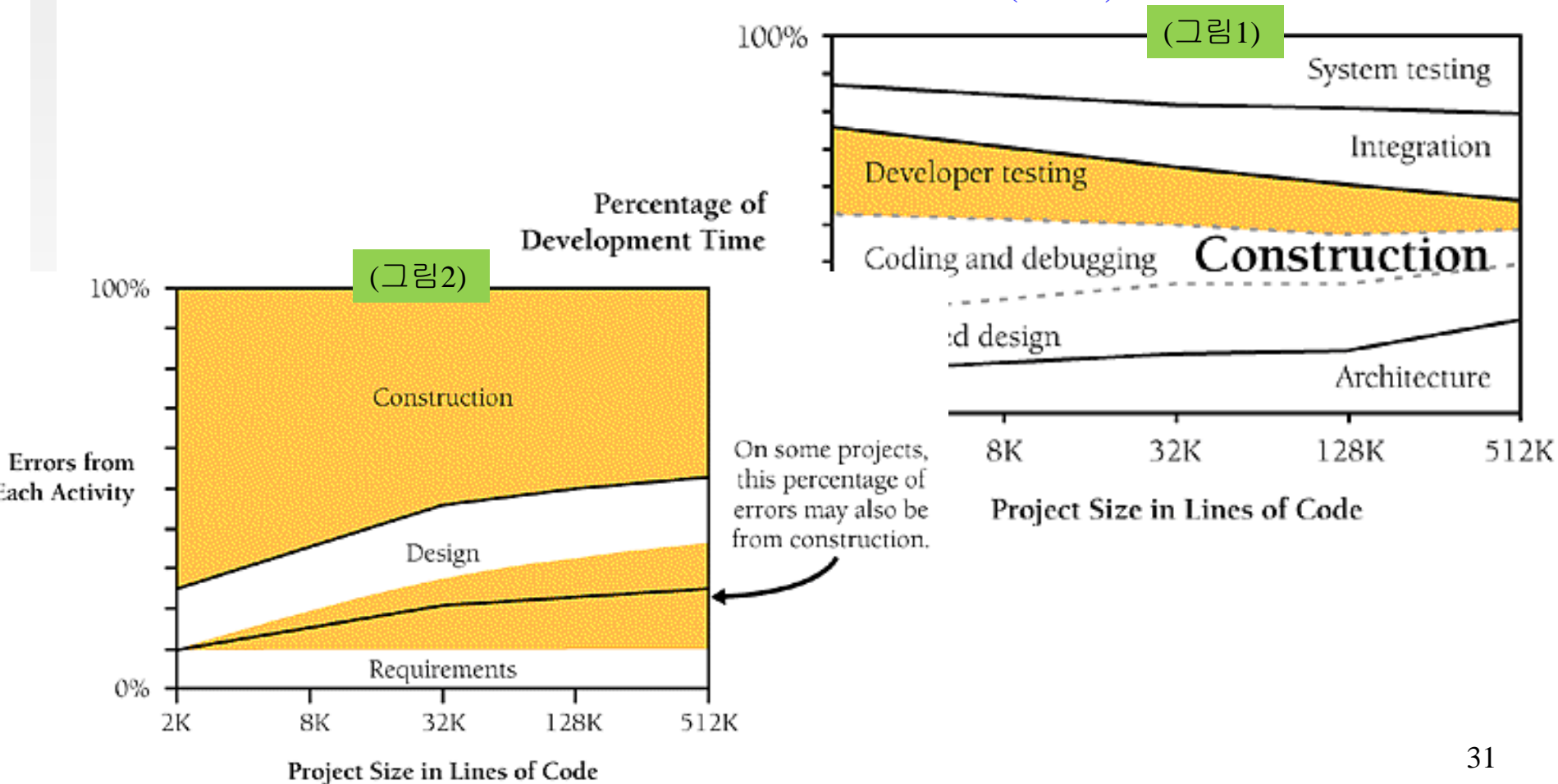
소프트웨어의 품질 요소



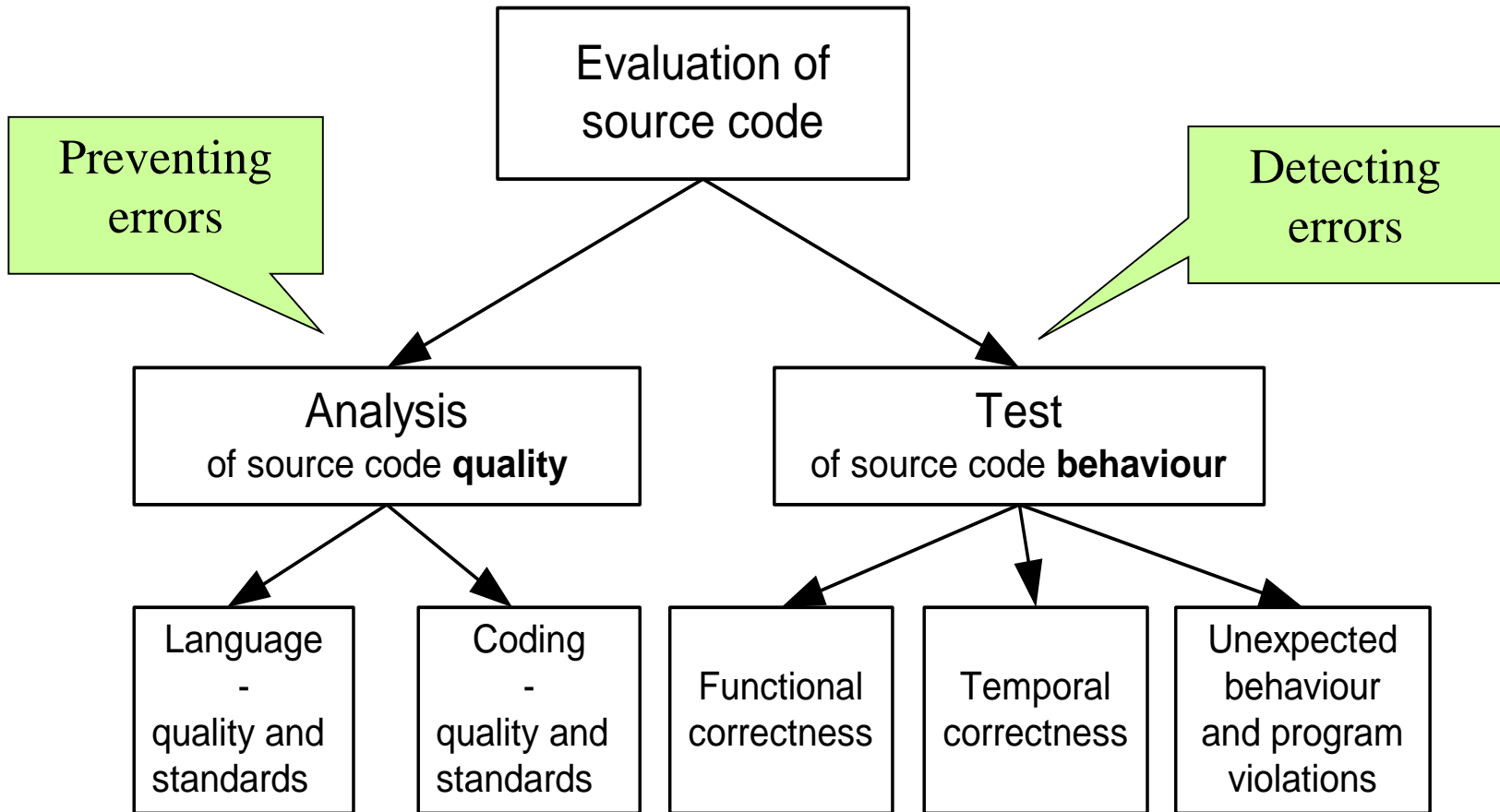
소프트웨어 테스트 개요

구현과 테스트

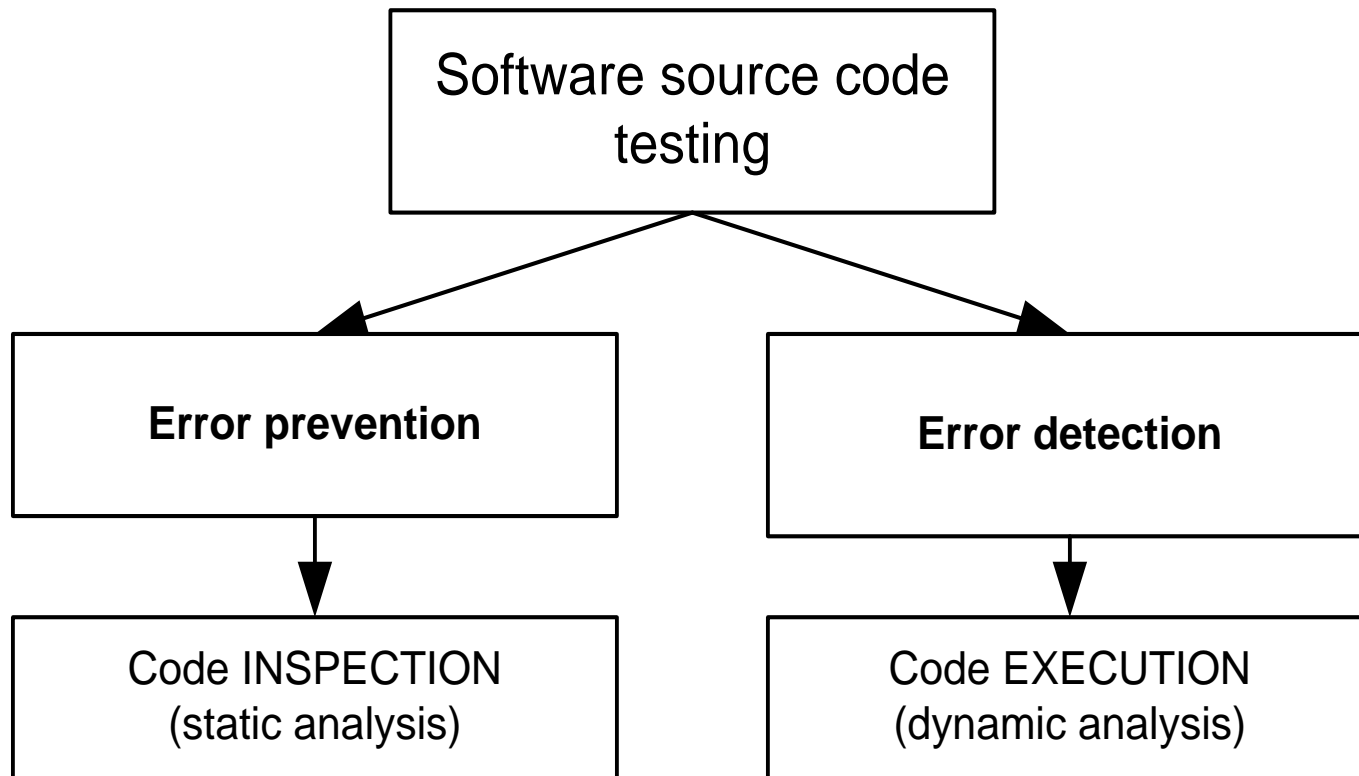
- 프로젝트의 규모가 클수록 구현과 개발자 테스트비중이 줄어듬 (그림1)
- 프로젝트에서 구현 오류의 비중이 가장 높음 (그림2)



SW 코드 검증 방법들



소스 코드 검증 기법



다양한 테스트 기법들의 오류 감지율

- 하나의 방법이 평균 75%이상인 것이 없음
- 오류 감지율을 높이기 위해 다양한 방법을 병행해 사용하여야 함

Removal Step	Lowest Rate	Modal Rate	Highest Rate
Source: Adapted from <i>Programming Productivity</i> (Jones 1986a), "Software Defect-Removal Efficiency" (Jones 1996), and "What We Have Learned About Fighting Defects" (Shull et al. 2002).			
Informal design reviews	25%	35%	40%
Formal design inspections	45%	55%	65%
Informal code reviews	20%	25%	35%
Formal code inspections	45%	60%	70%
Modeling or prototyping	35%	65%	80%
Personal desk-checking of code	20%	40%	60%
Unit test	15%	30%	50%
New function (component) test	20%	30%	35%
Integration test	25%	35%	40%
Regression test	15%	25%	30%
System test	25%	40%	55%
Low-volume beta test (<10 sites)	25%	35%	40%
High-volume beta test (>1,000 sites)	60%	75%	85%

Testing Life Cycle (V Model)

