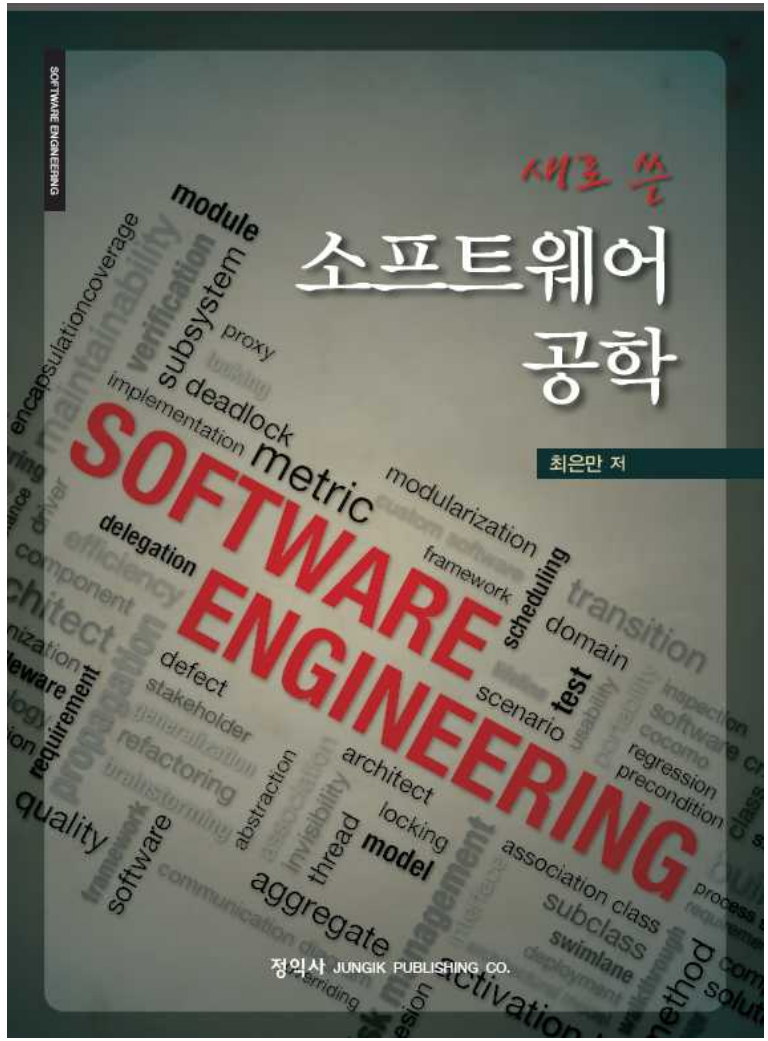
소프트웨어 공학 개론



강의 16: 품질

최은만
동국대학교 컴퓨터공학과

학습 목표

- 소프트웨어 품질
- 품질 측정 방법
- 품질 보증 활동
- 확인 및 검증 기법
- 프로세스 개선
- 품질 보증 도구

품질 보증

- 소프트웨어 품질 보증(software quality assurance)
  - 개별 작업과 병행하여 품질에 관한 작업, 즉 표준을 정하고, 품질을 보증하는 작업, 품질 표준에 맞는지 체크하는 작업이 이루어져야 함
- 보호활동(umbrella activity)
  - 프로젝트 관리, 문서화, 품질 보증 등의 작업
- 품질 보증(quality assurance)
  - 소프트웨어 프로젝트의 프로세스와 프로덕트에 대한 품질을 관리하고 향상시키는 작업
- 품질 관리(quality control)
  - 생산된 제품이 요구사항에 부합되는지를 확인하고 잘못되었을 때 원인을 규명하여 설계나 생산 및 조립 방법을 수정

11.1 소프트웨어 품질

- 소프트웨어 품질
  - 복합적인 요소가 포함된 다중 개념(multi-dimensional concept)

- 소프트웨어를 사용하는 관점, 개발하는 관점, 발주하는 관점 모두 중요하게 생각하는 품질 특성이 다름

품질의 의미

- 넓은 범주에서 보면 고객만족의 차원으로 생각

- 좁은 의미에서 보면 개발자의 입장으로 국한
  - 개발자가 생각하는 품질 : 소프트웨어가 요구를 만족하느냐 못하느냐

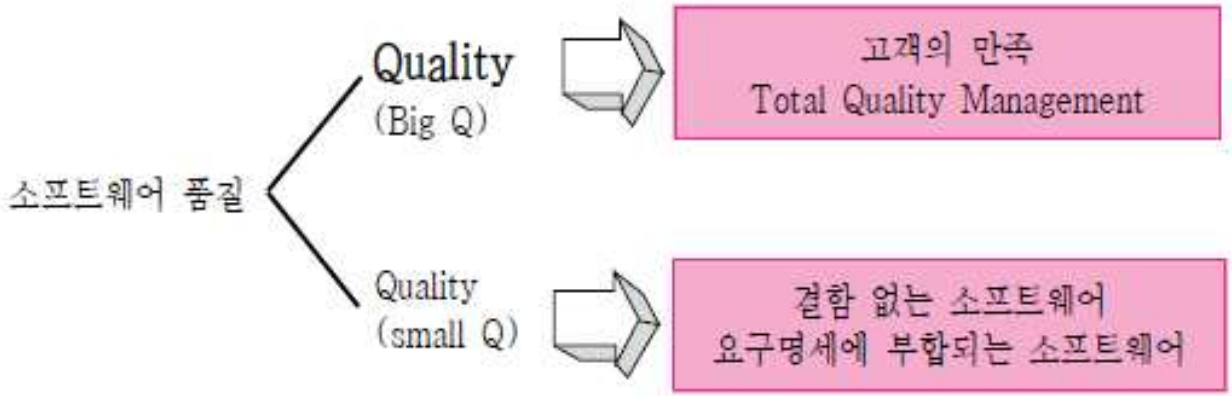


그림 11.1 ▶ 두 가지 개념의 소프트웨어 품질

품질 속성

- 소프트웨어 품질 속성(quality attribute)
  - 소프트웨어 품질 관련 측면을 나타내고 특징짓는 속성

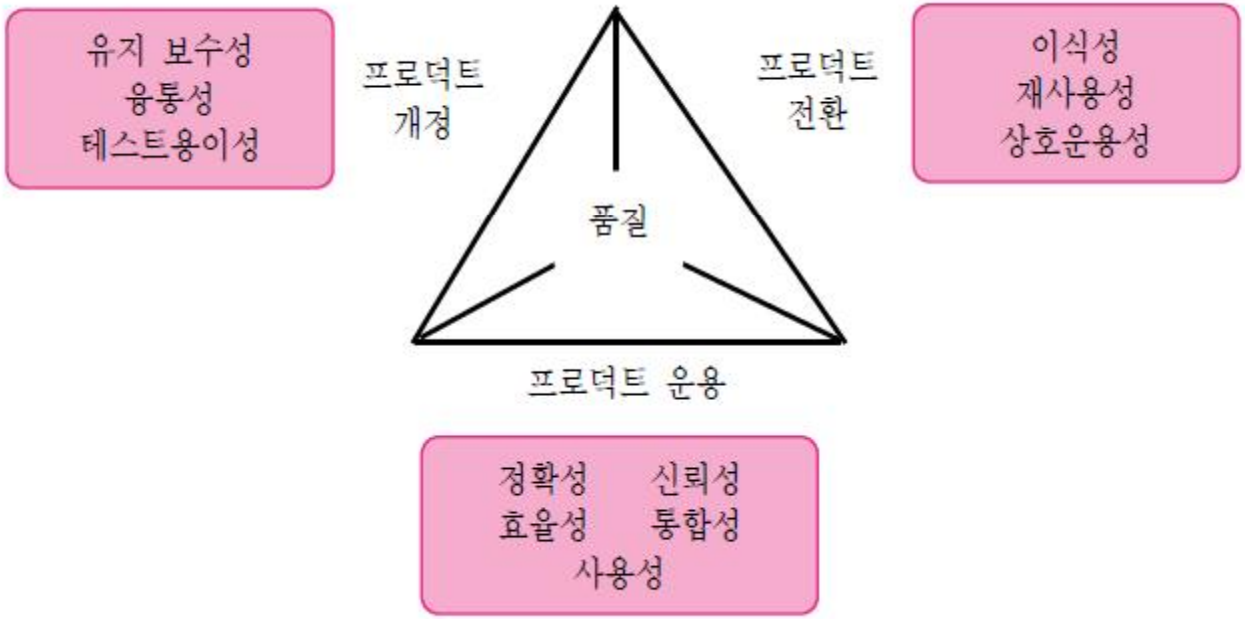
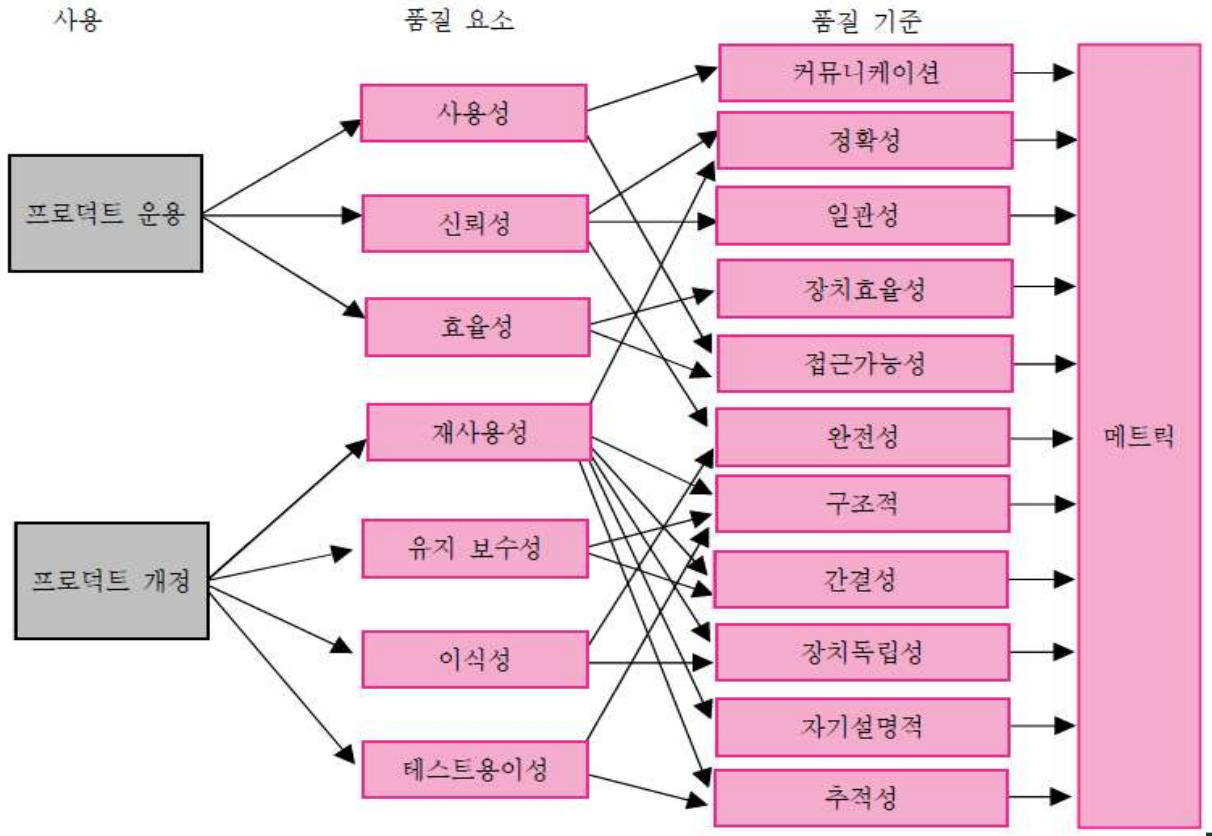


그림 11.2 ▶ 소프트웨어 품질 속성

품질 속성

- 소프트웨어의 품질의 특성은 세가지 차원이 존재
  - 품질 요소(factor) : 사용자에 의한 외부 관점
  - 품질 기준(criteria) : 개발자 측면의 내부 관점
  - 메트릭 차원 : 품질을 제어



품질 속성

- **ISO/IEC 9126**
  - 소프트웨어가 가질 수 잇는 여러 가지 품질 특성을 정의
  - ISO에서는 여섯 가지의 품질 특성을 정의
- **ISO와 IEEE에서는 품질 속성의 계층을 다르게 정의**

기능성
- 적절성
- 정확성
- 상호운용성
- 적합성
- 보안

신뢰성
- 성숙성
- 회복성
- 고장인내성

사용용이성
- 학습성
- 이해성
- 운용성

효율성
- 시간 효율
- 자원 효율

유지보수성
- 안정성
- 분석가능성
- 변경가능성
- 시험용이성

이식성
- 설치가능성
- 대체가능성
- 적응성
- 적합성

효율성
- 시간 경제성
- 자원 경제성

기능성
- 완벽성
- 정확성

신뢰성
- 보안
- 호환성
- 상호운용성

유지보수성
- 정확성
- 확장가능성
- 시험용이성

이식성
- 하드웨어 독립
- 소프트웨어 독립
- 설치용이성
- 재사용성

신뢰성
- 결함 없음
- 오류 허용성
- 가용성

사용용이성
- 이해용이성
- 학습용이성
- 운용성
- 의사소통성

품질 속성

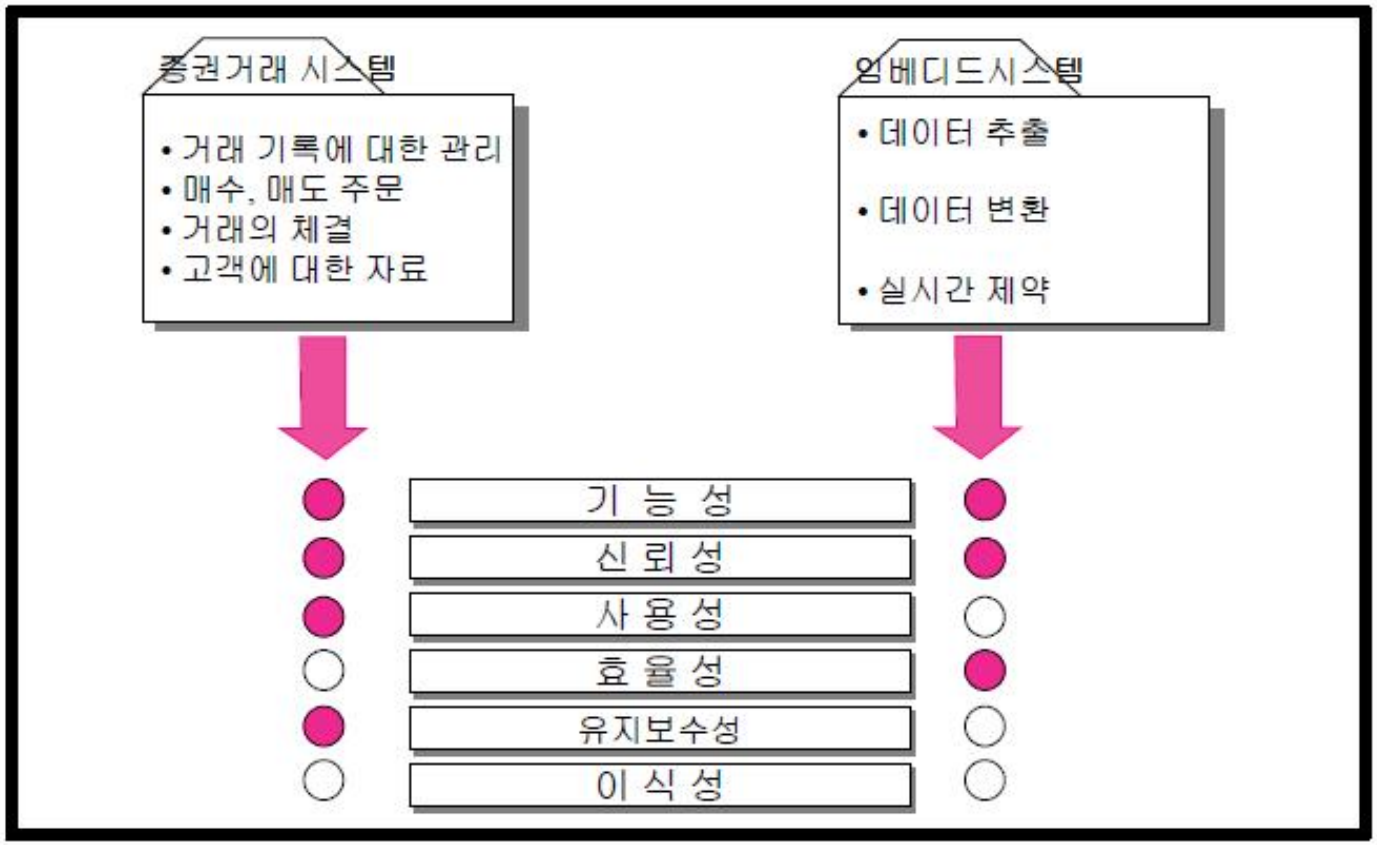
- 신뢰성(reliability) – 소프트웨어에 요구된 기능을 명시된 조건 하에 실행하여 정확하고 일관성 있는 결과를 생성하는 능력
- 강인성(robustness) – 소프트웨어에 요구된 기능을 어렵거나 예외적인 환경에서 수행할 능력
- 효율성(efficiency) – 소프트웨어에 요구된 기능을 최소의 시간과 자원을 사용하여 원하는 결과를 생성하는 능력
- 상호운용성(interoperability) – 소프트웨어가 다른 소프트웨어와 정보를 교환하는 능력
- 유지보수성(maintainability) – 소프트웨어가 수리 및 진화될 수 있는 성질

품질 속성

- 테스트가능성 – 소프트웨어에 요구된 또는 적용될 수 있는 모든 형식의 평가
  - 인스펙션, 동료검토, 화이트박스 테스팅, 블랙박스 테스팅 등
- 이식성(portability) – 소프트웨어가 여러 운영 환경 및 플랫폼에서 실행될 수 있도록 변형이 가능한 성질
- 재사용성(reusability) – 소프트웨어가 확장이나 커스텀화 없이 유사한 또는 다른 배경에서 사용될 수 있는 성질
- 모듈성(modularity) – 소프트웨어가 모듈 컴포넌트의 통합이나 조정을 쉽게 만드는 성질

소프트웨어 유형과 품질

- 소프트웨어 유형에 따른 품질 특성 중요도의 차이



11.2 품질 측정과 메트릭

- 소프트웨어 측정(software measurement)
  - 소프트웨어 속성의 객관적이고 정량적인 평가

- 소프트웨어 메트릭(software metric)
  - 표준화된 소프트웨어 측정 방법

품질 측정과 메트릭의 유용성

- 품질 측정과 메트릭
  - 요구분석, 설계, 구현, 문서화가 포함된 소프트웨어의 정량적인 평가
- 지표의 정의와 사용
  - 지표 – 상대적 의미가 있는 값의 범위
- 중요한 부분에 자원을 투입
- 유사 프로젝트와 시스템을 정량적으로 비교
- 개선의 정량적인 평가
- 기술의 객관적인 평가
- 프로세스 개선의 객관적인 평가

전통적인 품질 메트릭

- **전통적인 품질 메트릭의 일부**

품질 메트릭	타입	설명	계산	
요구의 비모호성 (Q1)	R	총 요구의 수(n_r) 대비 검토인이 요구에 대하여 동일한 의미적 해석을 하는 요구의 개수 (n_{ui})	Q1= n_{ui}/n_r	
요구의 완전성 (Q2)	R	상태와 자극의 모든 가능한 조합의 수($n_s \times n_i$) 대비 요구에 명시된 고유한 기능의 수(n_u)의 비율 n_s 는 상태의 수이며 n_i는 요구에 표시된 입력과 자극의 수	Q2= $n_u/(n_s \times n_i)$	
요구의 정확성 (Q3)	R	요구 총 개수(n_r= n_c+n_{nv})대비 검증된 정확한 요구(n_c)의 비율. n_{nv}은 아직 정확한 요구로 검증되지 않는 요구의 수	Q3=$n_c/n_r=n_c/(n_c+n_{nv})$	
요구의 일관성 (Q4)	R	요구 총 개수(n_r) 대비 상충되지 않는 해석을 가진 요구($n_r- n_{ci}$)의 비율. n_{ci} 는 검토인에 의하여 상충된 요구라고 알려진 요구의 개수	Q4=$(n_r-n_{ci})/n_r$	
팬 인	D	호출한 모듈의 개수. 팬 인이 높은 모듈은 • 고장의 핵심 • 너무 많은 책임이 배정됨 • 변경되면 다른 많은 모듈에 영향을 줌		
팬 아웃	D	주어진 모듈에 의하여 호출된 모듈의 수. 팬 아웃이 너무 높으면 • 너무 많은 다른 모듈을 요구하기 때문에 재사용이 어려움 • 다른 모듈의 수정에 의하여 영향을 받음 • 다른 모듈과 상호작용 때문에 이해하기 어려움		

전통적인 품질 메트릭

- **전통적인 품질 메트릭의 일부**

품질 메트릭	타입	설명	계산	
요구의 비모호성 (Q1)	R	총 요구의 수(n_r) 대비 검토인이 요구에 대하여 동일한 의미적 해석을 하는 요구의 개수 (n_{ui})	Q1= n_{ui}/n_r	
요구의 완전성 (Q2)	R	상태와 자극의 모든 가능한 조합의 수($n_s \times n_i$) 대비 요구에 명시된 고유한 기능의 수(n_u)의 비율 n_s 는 상태의 수이며 n_i는 요구에 표시된 입력과 자극의 수	Q2= $n_u/(n_s \times n_i)$	
요구의 정확성 (Q3)	R	요구 총 개수(n_r= n_c+n_{nv})대비 검증된 정확한 요구(n_c)의 비율. n_{nv}은 아직 정확한 요구로 검증되지 않는 요구의 수	Q3=$n_c/n_r=n_c/(n_c+n_{nv})$	
요구의 일관성 (Q4)	R	요구 총 개수(n_r) 대비 상충되지 않는 해석을 가진 요구($n_r- n_{ci}$)의 비율. n_{ci} 는 검토인에 의하여 상충된 요구라고 알려진 요구의 개수	Q4=$(n_r-n_{ci})/n_r$	
팬 인	D	호출한 모듈의 개수. 팬 인이 높은 모듈은 • 고장의 핵심 • 너무 많은 책임이 배정됨 • 변경되면 다른 많은 모듈에 영향을 줌		
팬 아웃	D	주어진 모듈에 의하여 호출된 모듈의 수. 팬 아웃이 너무 높으면 • 너무 많은 다른 모듈을 요구하기 때문에 재사용이 어려움 • 다른 모듈의 수정에 의하여 영향을 받음 • 다른 모듈과 상호작용 때문에 이해하기 어려움		

전통적인 품질 메트릭

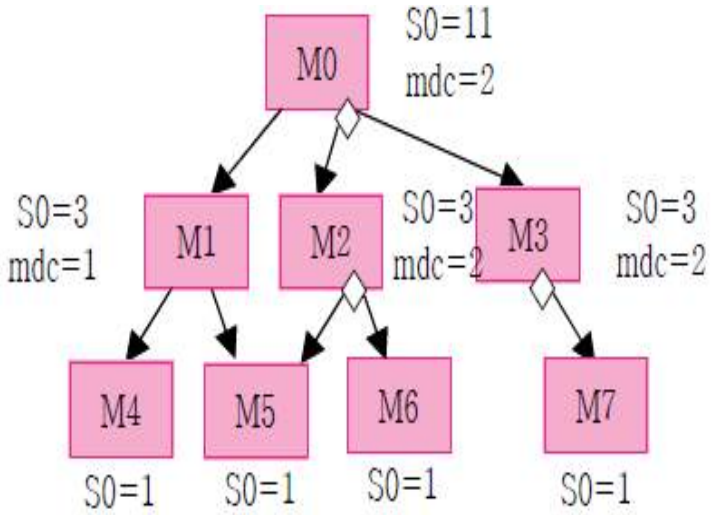
- 요구 메트릭(R)
  - 요구의 비모호성(unambiguity)
  - 요구 완전성 메트릭 – 요구 명세서에 기술된 시스템의 상태와 시스템에 대한 외부자극이 완전하다는 가정에 근거
  - SRS – 시스템의 모든 가능한 상태와 모든 가능한 외부자극을 포함

$$f(state, stimulus) \rightarrow (state, response)$$

  - f함수가 완벽하게 매핑된다면 SRS는 완벽한 것으로 간주

전통적인 품질 메트릭

- 설계 메트릭
  - M0-M7 – 모듈
  - 화살표 – 모듈 호출
  - 다이아몬드 화살표 – 분기 호출



- 모듈 설계 복잡도(mdc(M))
  - mdc(M) = d + 1
  - d : M이 가진 다이아몬드의 수
  - 다이아몬드 : 이진 조건의 분기

- 선택 복잡도(S0)
  - S0(leaf) = 1, 각 단말 노드는 하나의 서브트리
  - S0(M) = $\sum_{i=1}^{n} S0(M_i) + mdc(M)$
  - M : M_i(i=1,2,...n) 모듈들을 호출

전통적인 품질 메트릭

- M4~M7의 S0 = 1
  - 모두 단말 노드

- M1 모듈복잡도
  - S0(M1) = S0(M4)+S0(M5)+mdc(M1) = 1+1+1 =3
    - M1은 분기가 없기 때문에 mdc 값이 1

- 모듈 설계 복잡도
  - S0(M) = N_{dm}+N_{adb}
  - N_{dm} : 모듈의 개수
  - N_{adb} : 선택적 모듈을 호출하는 분기의 수

전통적인 품질 메트릭

- 구현 및 시스템 메트릭

- 구현 메트릭
  - LOC 메트릭 : 원시코드의 줄을 세는 것
  - 싸이클로매틱 복잡도 메트릭 : 프로그램을 통과하는 독립된 경로의 개수 이며 필요한 테스트의 횟수

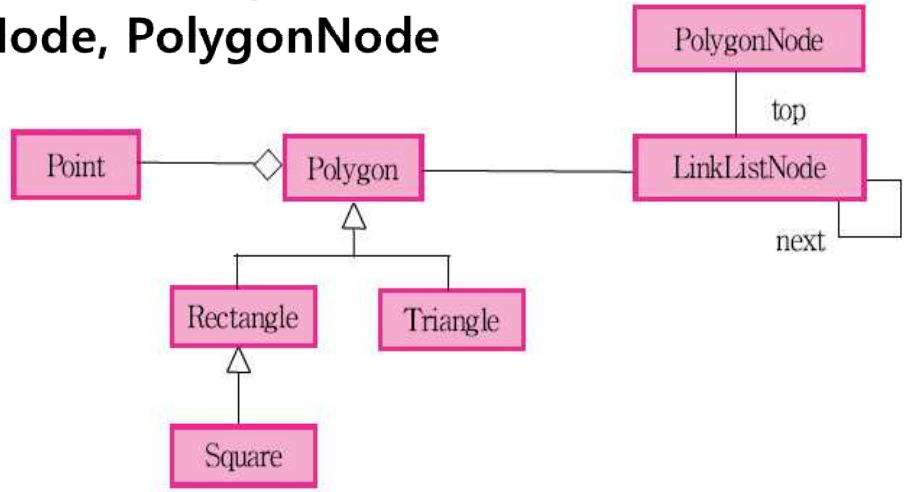
- 시스템 메트릭
  - 신뢰도 메트릭
    - MTBF = MTTF + MTTR
    - MTBF(Mean Time Between Failure) : 고장 사이의 평균 시간
    - MTTF(Mean Time To Failure) : 고장 까지의 평균시간
    - MTTR(Mean Time To Repair) : 수리 평균시간

객체지향 소프트웨어의 품질 메트릭

- Chidamber & Kamerer의 메트릭

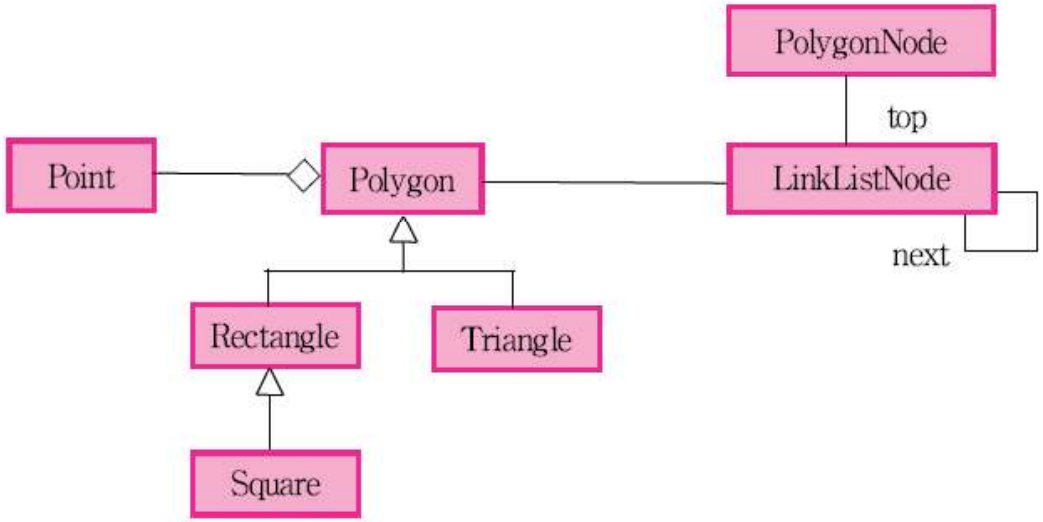
- 클래스 당 가중 메소드(Weighted Method per Class)
  - WMC(C) = $C_{m1} + C_{m2} + \cdots + C_{mn}$
  - C_{mn} , i=1,2,..n : 클래스 C에 있는 메소드의 복잡도

- 자식 노드의 개수(Number of Children)
  - 해당 클래스의 상속 구조에서 직계 자식 클래스의 수
  - NOC(C) = |{C':C'는 C의 직계 자식 클래스}|
  - NOC=0 : Point, Square, LinkNode, PolygonNode
  - NOC=1 : Rectangle, Triangle
  - NOC=2 : Polygon



객체지향 소프트웨어의 품질 메트릭

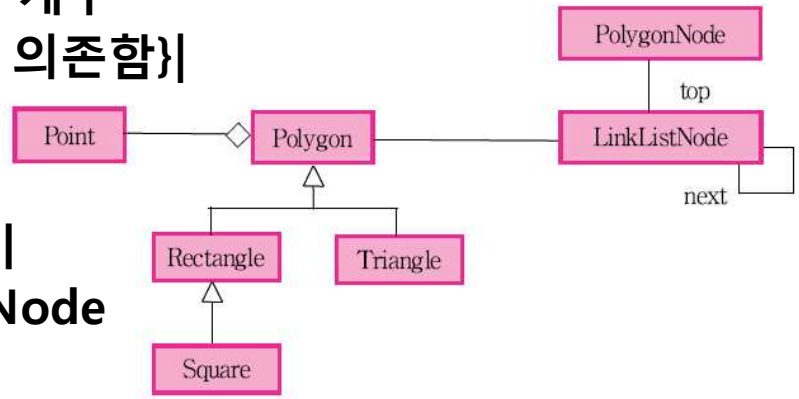
- 상속 트리의 깊이(Depth of Inheritance Tree)
  - 상속 트리의 루트로부터 해당 클래스까지 가장 싶은 상속 경로
  - DIT(C) = 0, 부모 클래스가 없는 경우
  - DIT(C) = 1+max({DIT(C'): C' 는 해당 클래스의 직계 부모 클래스})



  - DIT=0 : Point, Polygon, LinkListNode, PolygonNode 클래스
  - DIT=1 : Rectangle, Triangle 클래스
  - DIT=2 : Square 클래스

객체지향 소프트웨어의 품질 메트릭

- 객체 클래스 사이의 결합(Coupling Between Object Classes)
  - 해당 클래스가 의존하고 있는 클래스의 개수
  - CBO(C) = |{C':C 클래스가 C' 클래스에 의존함}|
  - 의존 관계 : 상속과 집합 관계를 포함
  - CBO=0 :
  - CBO=1 : Point와 Polygon 클래스 사이
  - CBO=2 : Polygon은 Point와 LinkListNode 영향을 받음
  - CBO=3 : LinkListNode는 자기 자신과 Polygon, PolygonNode의 영향을 받음



  - CBO가 클수록 이해하고, 테스트하고, 유지보수 및 재사용이 어렵다

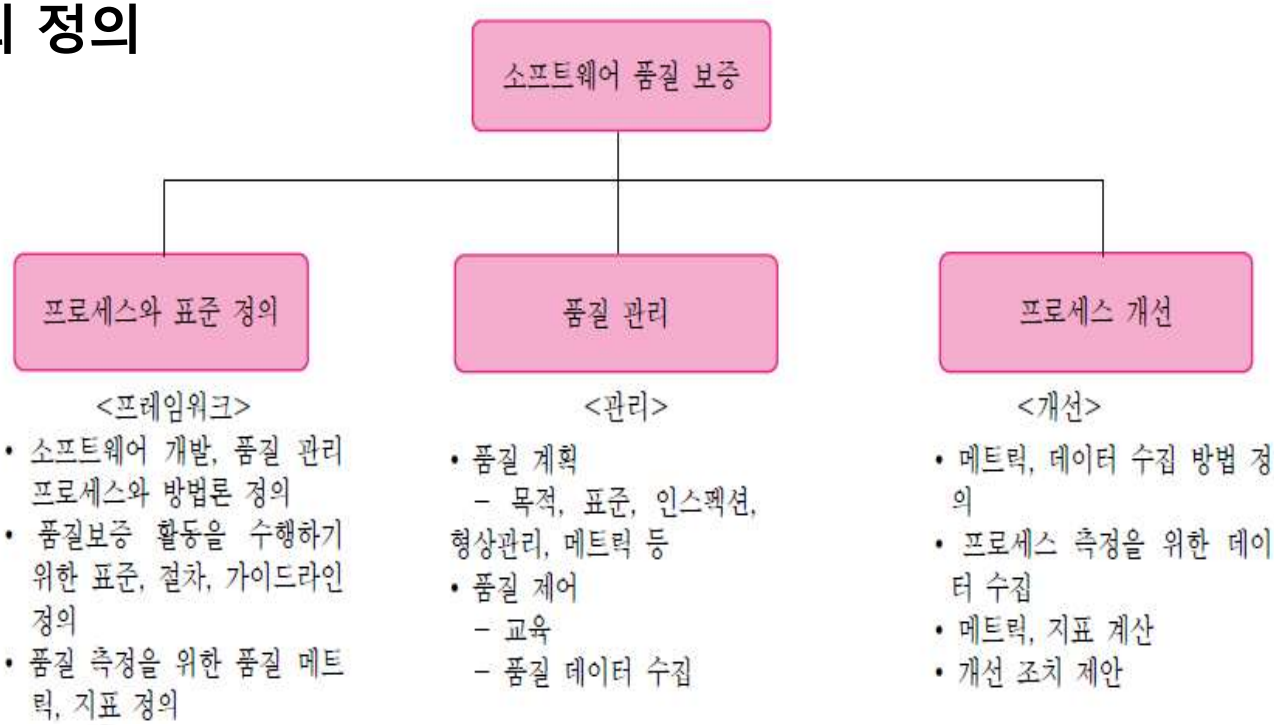
객체지향 소프트웨어의 품질 메트릭

- 클래스의 책임(Response for a Class)
  - 클래스의 메소드 개수 + 그 클래스의 메소드가 호출하는 메소드의 개수
  - RFC(C) = |{m:m은 클래스의 메소드 또는 C클래스의 메소드에 의하여 호출되는 메소드 m}|

- 메소드 응집 결핍(Lack of Cohesion in Methods)
  - 해당 클래스의 속성을 공유하지 않는 메소드 쌍의 수
  - LCOM=n*(n-1)/2-2*|{m_i,m_j): m_i와 m_j가 C의 속성을 공유한다}|
    - n : 클래스C의 메소드 개수
  - LOCM값이 크면 클래스 안에 있는 요소들 사이에 공하는 속성이 없고 클래스가 모듈로써 응집도가 떨어진다

11.3 품질보증 활동

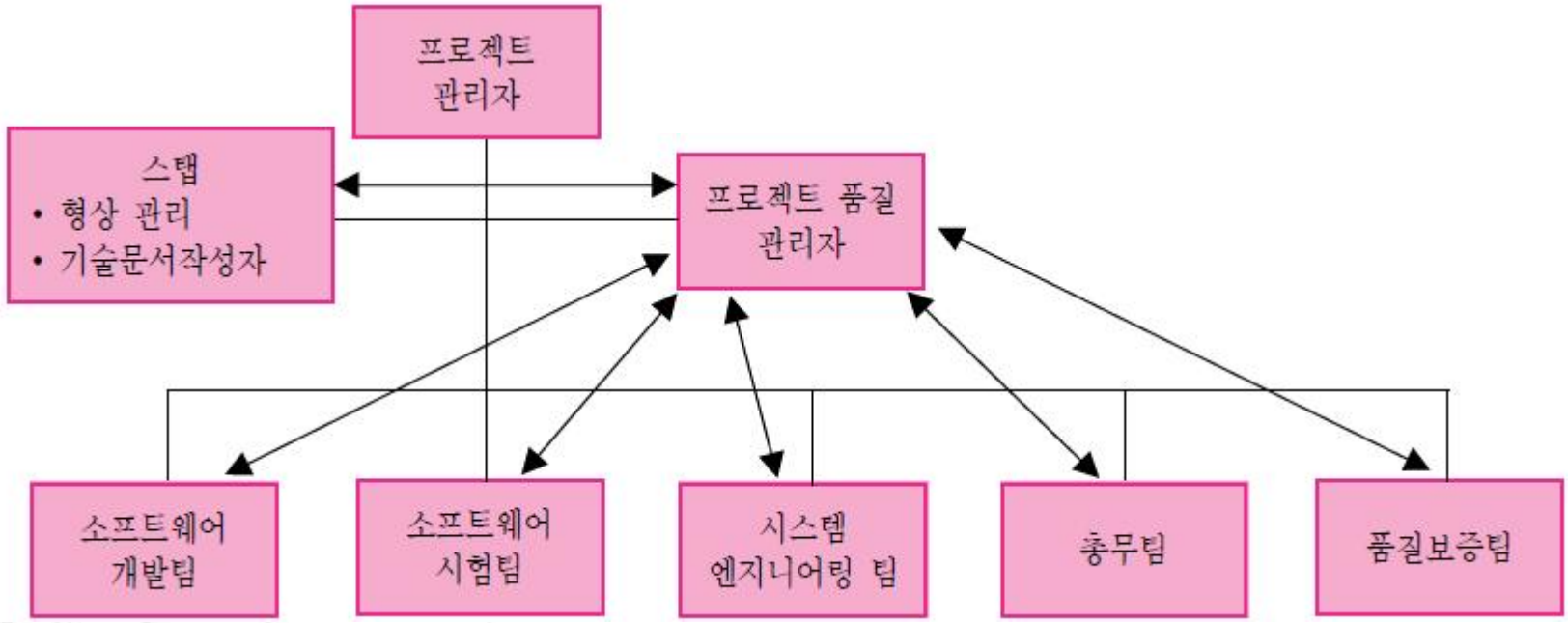
- 품질보증
  - 소프트웨어 제품이나 아이템이 정해진 요구에 적합하다는 것을 보장하는 데 필요한 계획적이고 체계적인 활동
- 프로세스와 표준의 정의
- 품질 관리
- 프로세스 개선



품질 보증 조직

- 관리적 활동
  - 개발 조직의 표준화 방법론을 잘 따르도록 하는 것

- 기술적 활동
  - 방법론을 잘 정의하는 것



프로세스와 표준 정의

- 전체 조직을 위하여 소프트웨어 품질 보증을 위한 프레임워크를 개발
  - 소프트웨어 개발, 품질 관리 프로세스 및 방법론의 정의
  - 개발 주기 동안 품질보증 작업을 수행할 표준, 절차, 가이드라인의 정의
  - 품질 측정과 평가를 위한 품질 메트릭, 지표 정의

- 프로세스와 방법론의 정의
  - 전통적인 프로세스에서 필요한 품질보증 활동

소프트웨어 개발 작업	전통적인 프로세스의 품질보증 활동
소프트웨어 요구 분석	1. 요구 명세 검토
	2. 요구와 관련된 메트릭과 지표 평가
	3. 프로토타이핑
	4. 모델 검토
	5. 소프트웨어 인수 시험 계획
소프트웨어 설계	1. 소프트웨어 설계 리뷰, 인스펙션, 워크스루
	2. 설계 관련 메트릭과 지표 평가
	3. 사용 사례 중심 검토
	4. 모델 체킹
	5. 소프트웨어 통합 테스팅 계획
소프트웨어 구현	1. 코드 리뷰, 인스펙션, 워크스루
	2. 정적 분석 체킹
	3. 메트릭과 지표에 근거한 평가
	4. 코딩 표준
통합 및 시스템 테스팅	1. 통합 테스팅
	2. 인수 테스팅
소프트웨어 운용 및 유지보수	1. 변경 분석 및 제어
	2. 소프트웨어 리엔지니어링
	3. 리그레션 테스팅

프로세스와 표준 정의

- 품질보증 표준과 절차의 정의

- 품질보증 표준
  - 기관의 장기 및 단기 목표와 품질 목표를 포함한 소프트웨어 공학 목표로부터 도출
  - ISO9001이나 IEEE품질 모델과 같은 품질 모델을 기초하여 정의

- 품질보증 절차
  - 소프트웨어 검토 및 확인 절차와 소프트웨어 형상관리 절차를 정의하여 명확하게 문서화

프로세스와 표준 정의

- 메트릭과 지표의 정의
- 프로젝트의 프로세스와 프로덕트 측면을 측정하는데 사용될 메트릭
- 지표는 관리자와 개발팀이 프로세스 개선이 어떻게 진행되고 있는지 파악하는데 도움을 준다

품질관리

- 품질계획
  - 각 프로젝트 초기에 이루어짐
  - 목적
  - 관리
  - 표준과 관례
  - 리뷰와 감리
  - 형상관리
  - 프로세스, 방법론, 도구, 기술
  - 메트릭, 지표
- 품질 보증 계획(IEEE730)

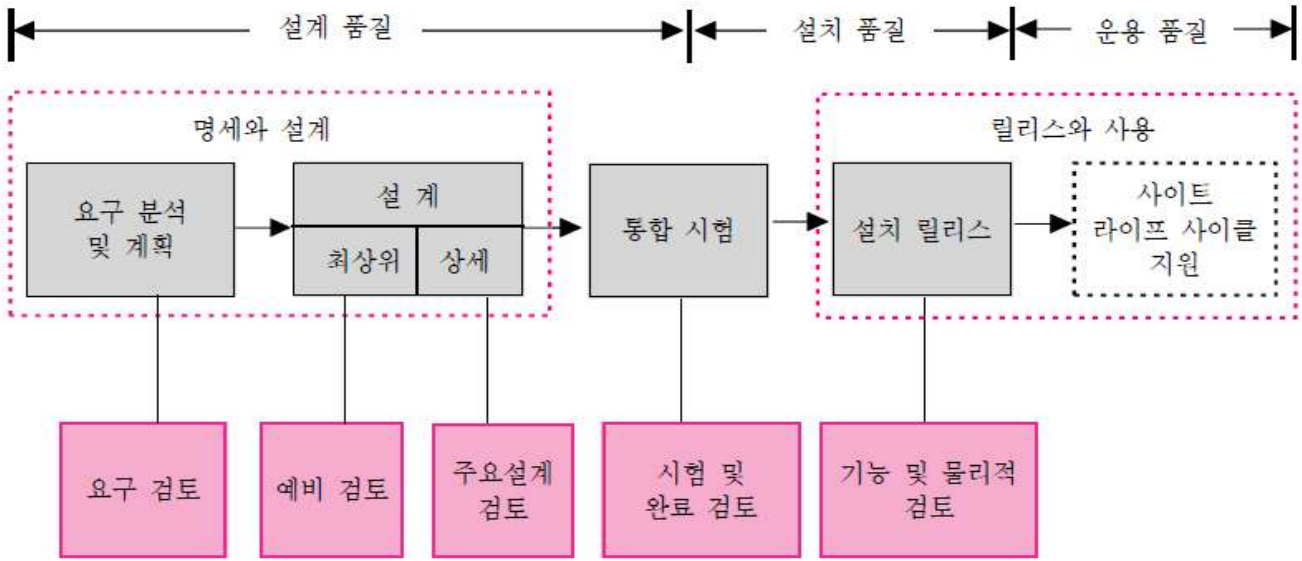
1. 계획의 목표
2. 관련 문서
3. 관 리
   - 3.1 조 직
   - 3.2 작 업
   - 3.3 책 임
4. 문서화
   - 4.1 목 적
   - 4.2 기본적으로 요구되는 문서
5. 표준, 실습, 관례, 메트릭
   - 5.1 목 적
   - 5.2 내 용
6. 검토와 검사
   - 6.1 목 적
   - 6.2 요구되는 검토
7. 테스트
8. 문제 보고 및 수정 작업
9. 도구, 기술, 방법
10. 코드 관리
11. 미디어 관리
12. 공급자 관리
13. 기록 취합, 관리, 보관

품질관리

- 품질보증 제어
  - 계획이 정확하게 실행되고 있는지 확인하는 것
  - 개발자가 품질보증 활동을 수행하도록 도와주는 것
  - 품질 관련 데이터를 모아 데이터베이스 사용하여 관리
  - 관리자에게 프로세스 개선을 위한 제안을 하고 수용된 제안이 적절히 실현되고 프로세스에 녹아들었는지 확인

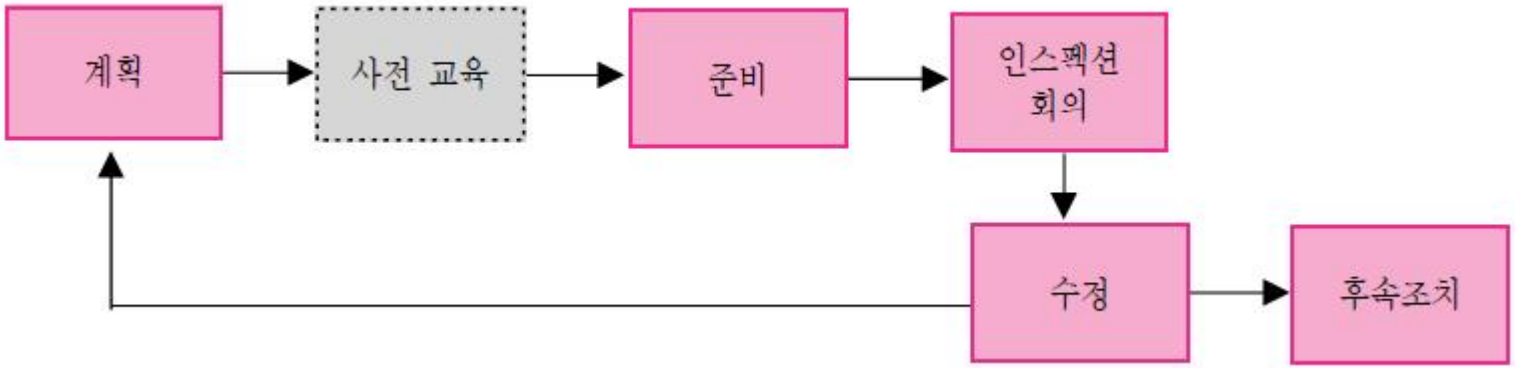
11.4 확인 및 검증 기법

- 확인(verification)
  - 프로덕트(소프트웨어)를 올바로 만들었는가?
- 검증(validation)
  - 올바른 프로덕트(소프트웨어)를 만들었는가?
- 품질 보증을 위한 검토 작업



인스펙션

- 품질 개선과 비용 절감을 위한 기법으로 사용
- 프로덕트를 공통되는 오류, 변칙, 표준이나 관례의 부적합 리스트와 체크해 보는 작업
- 인스펙션 과정



워크스루

- 단순한 테스트 케이스를 이용하여 프로덕트를 수작업으로 수행
- 개발자가 포함된 기술 스텝들이 참석

동료 검토

- 프로덕트를 동료가 검토
- 모든 소프트웨어 프로젝트 결과물에 대하여 적용
- 사람이 판단할 필요가 있는 문제점을 파악하는데 효과적

11.5 프로세스 개선

- CMM
- ISO의 SPICE

CMM

- 미래의 고객이 소프트웨어 공급업자가 어떤 점이 부족하며 어떤점이 강한지 발견하기 위하여 평가하는 기준
- 소프트웨어 개발자 스스로 프로세스 능력을 평가하고 개선의 방향을 설정하는 것

레 벨	초 점	주요 프로세스 영역	결과
5 Optimizing (최적단계)	계속적인 개선	· 프로세스 변경 관리 · 기술 변경 관리 · 결함 방지	생산성과 품질
4 Managed (관리단계)	프로덕트 및 프로세스 품질	· 소프트웨어 품질 관리 · 정량적 프로세스 관리	
3 Defined (정의단계)	엔지니어링 프로세스	· 조직 프로세스 집중 · 조직 프로세스 정의 · 동료 검토 · 교육 프로그램 · 그룹간 협력 · 소프트웨어 프로덕트 엔지니어링 · 통합 소프트웨어 관리	
2 Repeatable (반복단계)	프로젝트 관리	· 소프트웨어 프로젝트 계획 · 소프트웨어 프로젝트 추적 및 감독 · 소프트웨어 하청 관리 · 소프트웨어 품질 보증 · 소프트웨어 형상 관리 · 요구 관리	
1 Initial (초보단계)	영웅적 개인		위 험

CMM 레벨 1 : Initial

- 소프트웨어 개발 과정이 임시 방편이며 무질서한 상태이다.
- 프로세스가 정확히 정의되어 있지 않고 개인의 능력에 의존한다.
- 조직 내에 관리 부재로 소프트웨어의 품질 손상 우려가 있다.
- 심각한 경우는 프로젝트가 계획된 절차를 뛰어넘어 코딩과 테스트 작업으로 전환한다.
- 납기일을 맞추기 위하여 프로세스의 품질 보증 작업을 무시한다.
- 프로세스의 입력과 과정이 정해져 있지 않아 작업 결과의 예측이 불가능하다.
- 유사한 작업에 대한 프로젝트가 반복하여 진행되더라도 개발 생산성이나 품질에 큰 차이가 난다.

CMM 레벨 2 : Repeatable

- 비용, 일정, 기능에 대한 예측과 추적이 가능하다.
- 요구되는 프로세스 원칙들이 지켜져서 같은 문제에 대한 개발이 반복될 때 쉽게 성공이 가능하다.
- 조직은 소프트웨어 프로젝트 관리를 위한 정책을 입안하고 절차들을 실행한다.
- 프로젝트 관리자는 소프트웨어 비용과 일정, 기능을 추적할 수 있고 문제가 발생하면 바로 인식한다.
- 소프트웨어 요구와 이를 만족하기 위하여 개발된 결과물은 반드시 체크하고 일치하는지 확인한다.
- 소프트웨어 프로젝트 표준이 잘 정의되어 있고 조직이 표준을 잘 준수하고 있는지 확인한다.
- 원리 원칙을 준수한다.

CMM 레벨 3 : Defined

- 관리와 엔지니어링을 위한 소프트웨어 프로세스가 문서화되어 규격으로 만들어져 있다.
- 기관 전체가 표준 소프트웨어 프로세스를 따른다.
- 기관 내에 소프트웨어 엔지니어링 프로세스 그룹(Software Engineering Process Group)이 조직되어 소프트웨어 프로세스를 전담한다.
- 표준화와 일관성이 잘 지켜진다.
- 소프트웨어 엔지니어링과 관리 활동이 안정적이며 성공적으로 되풀이 될 수 있다.
- 개발 공정, 비용, 일정, 기능이 통제되고 있고 소프트웨어 품질에 대하여 추적이 가능하다.

CMM 레벨 4 : Managed

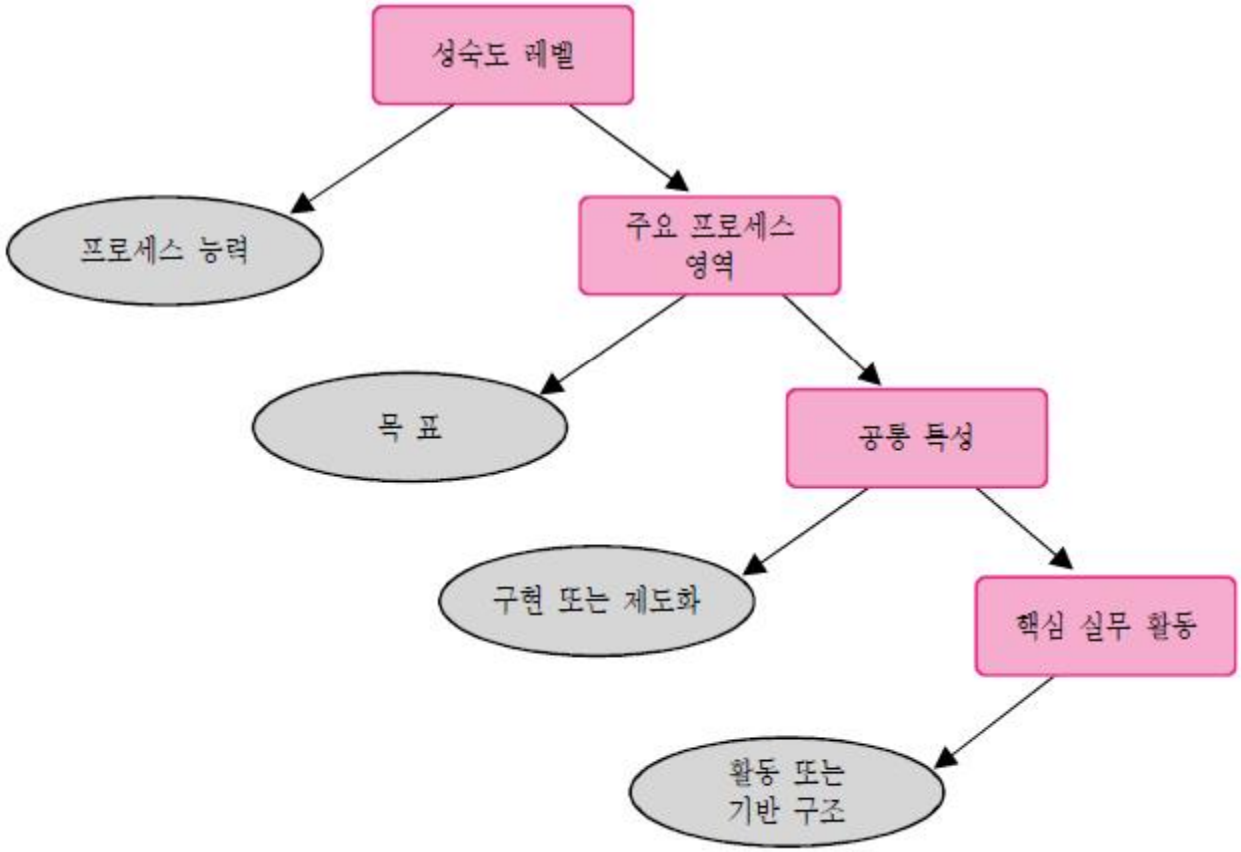
- 소프트웨어 프로세스와 프로덕트의 품질에 대한 자세한 **측정**이 이루어진다.
- 조직은 소프트웨어 프로덕트와 프로세스에 대한 **정량적인 목표**를 세운다.
- 프로세스 수행 성과의 편차를 허용하는 범위 안으로 줄여 프로덕트와 프로세스를 관리한다.
- 프로세스가 측정되고 정해진 범위 내에서 이루어지므로 프로세스 성과를 정확히 **예측**할 수 있다.

CMM 레벨 5 : Optimizing

- 프로세스에 대한 정량적인 피드백과 획기적인 아이디어와 기술로 지속적으로 프로세스를 향상시킨다.
- 전체조직이 지속적인 프로세스 개선을 위하여 초점을 맞춘다.
- 소프트웨어 프로세스의 효율성에 대한 데이터를 사용하여 새로운 기술의 도입과 소프트웨어 프로세스의 변경에 의한 비용과 수익을 분석한다.
- 최고의 소프트웨어 엔지니어링 기술을 찾아내고 이를 이용하여 혁신을 추구한다.

CMM

- **CMM의 구조**



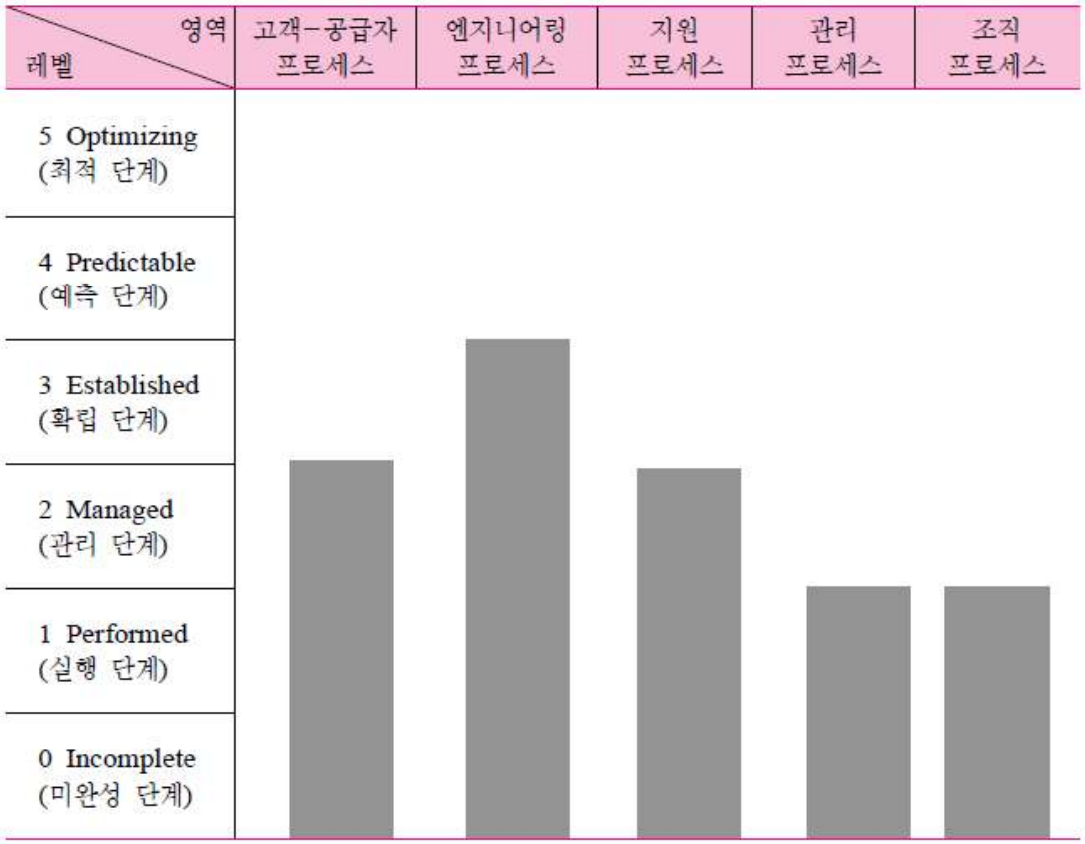
CMM 핵심 실무 활동(Key Practices)

- 필요한 업무 활동을 자세히 표현
- 가장 기본적이고 중요한 기반구조 및 활동을 나타냄
- 예) 프로젝트 계획 영역의 핵심 실무 활동
  - 비용산정, 라이프사이클의 정의, 일정과 투입 자원에 대한 계획의 문서화
- 핵심 실무활동에 대하여 CMM은 활동 항목만 나열하고 있으며 이를 어떤 방법으로 수행하여야 하는지는 언급하지 않고 있다 [Paulk, 1994].

- **SPICE**

  (Software Process Improvement and Capability dEtermination)

  - **소프트웨어 프로세스 평가를 위한 구제 표준**



SPICE의 이차원 모델

- 고객 공급자 프로세스
  - 예) 발주, 공급자 선정, 고객 인수, 요구사항 도출, 공급, 운영 등
- 엔지니어링 프로세스
  - 예) 요구분석, 설계 및 실험, 구축, 통합 등
- 지원 프로세스
  - 문서화, 형상관리, 품질보증, 검증, 확인, 검토 등 개발활동을 지원하는 프로세스
- 관리 프로세스
  - 예) 프로젝트 관리, 품질 관리, 위험 관리 등
- 조직 프로세스
  - 예) 프로세스의 정의, 심사, 개선, 인적자원 관리, 기반구조, 측정, 재사용

SPICE vs CMM

- 유사점
  - 프로세스 심사를 위한 참조모형을 제공
    - 개발 성숙도에 따라 차별화된 수준을 정의
    - 각 수준의 특징을 제시하여 기관의 수준을 판단 기준 제공

- 차이점
  - 성숙도 레벨과 심사 영역의 구분
    - CMM: 레벨 1부터 5까지 5개의 성숙도 주순을 정의
    - SPICE: 레벨 0부터 5까지의 6개의 수준으로 나누어 정의
  - 능력 평가 방법
    - CMM: 어떤 기관의 프로세스 능력을 여러 분야에 걸쳐 평가하여 하나의 레벨로 평가하는 일차원적인 구조
    - SPICE: 각 프로세스 영역마다 능력에 대한 평가를 별도로 할 수 있는 이차원적인 구조

11.6 품질보증 도구

- 정적분석도구

- 메트릭 도구

도구 이름	설명
Analyst4j	이클립스 기반의 Java 언어를 위한 메트릭 및 품질 분석 도구
Checkstyle	Java 언어를 위한 코딩 표준 체크 도구. 어떤 코딩 표준도 체크할 수 있도록 조정할 수 있다.
Chidamber와 Kemerer Java 메트릭	Java 언어를 위한 Chidamber와 Kemerer의 메트릭 계산 도구
Eclipse 메트릭	Eclipse IDE를 위한 메트릭 계산 및 의존 분석 플러그-인.
FindBugs	Java 코드의 버그를 찾는 정적분석 도구
Rational Logiscope	코드 리뷰를 자동화하고 오류 가능성 있는 모듈을 감지하는 품질보증 도구
Jindent	Java 프로그램을 위한 코드 포매터. Javadoc 커멘트를 생성하고 완성시킴
McCabe IQ Developers Edition	정적 분석을 실시하고ㄴ 아키텍처를 보여주는 품질측정 도구. 코드의 복잡한 부분을 강조하고 버그와 취약점을 파악한다.
PMD	잠재적인 문제를 찾는 Java 원시코드 분석 도구

Questions?