L/O/G/O

# 1장. 서론(introduction)

소프트웨어공학에 대한 소개

# 목적

- 소프트웨어 공학에 대한 소개와 그것의 중요성에 대한 설명
- 소프트웨어 공학에 관한 주요 질문들에 대한 답변을 정리
- 도덕적(ethical) 그리고 전문적인 이슈들을 소개하고, 그것들이 소프트웨어 공학자들에게 중요한 이유에 대해

# 다루는 토픽

- 소프트웨어 공학에 관한 FAQ
- 전문적 그리고 도덕적 책임감

# 소프트웨어 공학

- 모든 선진 국가들의 경제는 SW에 의존함.
- 점점 많은 시스템들이 SW에 의해 제어됨.
- 소프트웨어 공학은 전문적인 S/W 개발을 위한 이론과 방법 그리고 도구들을 다룸
- 모든 선진 국가들에서는 소프트웨어에 대한 지출 비용이 GNP의 상당한 부분을 차지한다.

# 소프트웨어 비용

- 소프트웨어 비용이 시스템 비용의 대부분을 차지하는 것이 일반적임. PC 상의 SW 비용은 종종 HW 비용보다 많음.
- 소프트웨어 개발에 드는 비용보다 이를 유지보수하는 비용이 많이 차지함. 긴 수명을 가진 시스템인 경우에는 개발 비용의 5-6 배(several) 정도일 수도 있음.
- 소프트웨어 공학은 cost-effective 소프트웨어 개발을 추구함.

# SE의 등장 배경과 발전

- SW Crisis
- Spaghetti code
- Structured Methodology
  - Sequence, Selection, Iteration
- Object Oriented Methodology
  - Encapsulation, Inheritance, Polymorphism
- CBSE, WBSE

# 소프트웨어 공학에 대한 FAQ

- 소프트웨어는 무엇인가?
- 소프트웨어 공학은 무엇인가?
- 소프트웨어 공학과 컴퓨터 과학과의 차이는 무엇인가?
- 소프트웨어 공학과 시스템 공학과의 차이는 무엇인가?
- 소프트웨어 프로세스는 무엇인가?
- 소프트웨어 프로세스 모델은 무엇인가?

# 소프트웨어 공학 FAQ (계속)

- 소프트웨어 공학의 비용은 무엇인가?
- 소프트웨어 공학 방법(method)은 무엇인가?
- CASE (Computer-Aided Software Engineering)는 무엇인가?
- 좋은 소프트웨어의 성질은 무엇인가?
- 소프트웨어 공학이 직면하고 있는 주요 과제(challenge)는 무엇인가?

# 소프트웨어는 무엇인가?

- 컴퓨터 프로그램들과 관련된 문서
- 소프트웨어 프로덕트들은 특정한 고객들을 위해 개발될 수도 있고 범용적인 시장을 위해 개발될 수도 있다.
- 소프트웨어 프로덕트
  - 범용적 - 광범위한 여러 고객들에게 팔기 위해 개발됨.
  - 주문(맞춤형) - 고객의 요구사항에 따라 하나의 고객을 위해 개발됨.

# 소프트웨어 공학은 무엇인가?

- 소프트웨어 공학은 소프트웨어 생산의 모든 측면에 관심을 가지는 공학 훈련(분야)이다.
  SE is an engineering discipline !
- 소프트웨어 공학자는 그들의 작업에 대하여 체계적(**systematic**)이고 조직적(**organized**)인 접근 방법을 채택하여야 한다. 그리고 해결해야할 문제, 개발 제약조건과 이용가능한 자원에 따라 적절한 도구들과 기술을 사용하여야 한다.

# 소프트웨어 공학과 컴퓨터 과학의 차이는 무엇인가?

- 컴퓨터 과학은 이론과 기초(fundamental)에 관심을 갖는다; 소프트웨어 공학은 유용한 소프트웨어를 개발하고 인도하는 실질적인 것에 관심을 갖는다.
- 컴퓨터 과학 이론들은 현재 소프트웨어 공학에 대한 완전한 토대의 역할로서는 불충분하다.

# 소프트웨어 공학과 시스템 공학의 차이는 무엇인가?

- 시스템 공학은 하드웨어, 소프트웨어 그리고 프로세스 공학을 포함한 컴퓨터 기반의 시스템 개발에 대한 모든 측면에 관심을 가진다. 소프트웨어 공학은 이 프로세스의 일부분이다.
- 시스템 공학자는 시스템 요구사항, 구조적 설계, 통합과 배치(deployment)에 참여한다.

# 소프트웨어 프로세스는 무엇인가?

- 소프트웨어 개발 또는 진화를 목적으로하는 활동들의 집합
- 모든 소프트웨어 프로세스들에 있는 일반적인 활동들:
  - 명세화(Specification) - 시스템이 무엇을 해야하고 개발 제약조건들은 무엇인가
  - 개발 - 소프트웨어 시스템의 생산
  - 확인(Validation) - 소프트웨어가 고객이 원하는 것인가를 검사하는 것.
  - 진화(Evolution) - 변하는 요구사항에 따라서 소프트웨어를 변경하는 것.

# 소프트웨어 프로세스 모델은 무엇인가?

- 소프트웨어 프로세스의 단순화된 표현 (특정한 측면을 나타내기 위한)
- 프로세스 측면들의 예
  - 웍플로우 측면 - 활동들의 순서
  - 자료-흐름 측면 - 정보 흐름
  - 역할/활동 측면 - 누가 무엇을 하는가
- 일반적인 프로세스 모델
  - 폭포수(Waterfall)
  - 진화적 개발(Evolutionary development)
  - 정형적 변환(Formal transformation)
  - 재사용 가능한 부품들로부터의 통합

# 소프트웨어 공학의 비용은 무엇인가?

- 대략 비용의 60%는 개발 비용, 40%는 테스팅 비용임. 주문 소프트웨어의 경우는, 종종 진화 비용이 개발 비용을 초과함.
- 개발하는 시스템의 유형과 성능, 시스템 신뢰도와 같은 시스템 성질들의 요구사항에 따라 비용이 달라짐.
- 비용의 분포는 사용하는 개발 모델에 따라 다름.

# 소프트웨어 공학 방법은 무엇인가?

- 시스템 모델, 표기법, 규칙, 설계 권고사항과 프로세스 지침을 포함하는 소프트웨어 개발에 대한 구조적인 접근법
- 모델 설명
  - 만들어져야 하는 그래픽 모델들에 대한 설명
- 규칙
  - 시스템 모델들에 적용되는 제한조건
- 권고사항
  - 좋은 설계 습관에 대한 충고
- 프로세스 지침.
  - 활동(작업)들의 순서에 대한 지침

# CASE (Computer-Aided Software Engineering)는 무엇인가?

- 소프트웨어 프로세스 활동들에 대한 자동화된 지원을 제공할 목적을 가진 소프트웨어 시스템. CASE 시스템들은 종종 메소드 지원을 위해 사용됨.
- Upper-CASE
  - 요구분석, 설계와 같은 초기의 프로세스 활동들을 지원하기 위한 도구들
- Lower-CASE
  - 프로그래밍, 디버깅, 테스팅과 같은 후반부 활동들을 지원하기 위한 도구들.

# 좋은 소프트웨어의 성질은 무엇인가?

- SW는 고객이 필요로하는 기능과 성능을 제공해야 한다. 그리고 유지보수가 쉬어야하고, 믿을수 있어야 하며, 사용하기가 용이하여야 한다.
- 유지보수성(Maintainability)
  - SW는 변경하는 요구를 만족하기 위해 진화되어야 한다.
- 신빙성(Dependability)
  - 소프트웨어는 믿을 수 있어야 한다.
- 효율성(Efficiency)
  - 소프트웨어는 시스템 자원을 낭비하지 않아야 한다.
- 유용성(Usability)
  - 소프트웨어는 의도하고 있는 사용자들에게 쓸만하여야 함.

# 소프트웨어 공학이 직면하고 있는 주요 과제는 무엇인가?

- 레가시 시스템, 증가하고 있는 다양성(diversity), 인도 기간의 단축에 대한 요구 등에 대한 대처
- 레거시 시스템(Legacy systems)
  - 오래되었지만 가치있는 시스템들을 관리하고 갱신함.
- 이질성(Heterogeneity)
  - 시스템들이 분산되고 HW와 SW들이 결합된 것을 포함.
- 인도(Delivery)
  - 소프트웨어의 빠른 공급에 대한 압력이 증가함.

# 전문적이고 도덕적인 책임감

- 소프트웨어 공학은 단순히 기술적인 기교의 응용보다는 광범위한 책임감을 포함한다.
- 소프트웨어 공학자가 전문가로서 존경받으려면 정직하고 도덕적으로 책임감있게 행동하여야 한다.
- 도덕적 행동은 단순히 법을 준수하는 것 이상이다.

# 이슈: 전문적인 책임감

- *신임(Confidentiality )*
  - 공학자들은 고용주나 고객들로부터 신임받는 것을 존중하여야 한다.(공식적인 신임장을 받고 안받고는 상관없이)
- *적임성(Competence )*
  - 공학자들은 자신들의 능력 수준을 잘못 나타내어서는 아되다 자시의 능력을 넘어서는 작언들을 의식적으로 수용해서는 안된다.

# 이슈: 도덕적인 책임감

- *지적 재산권(Intellectual property rights )*
  - 공학자들은 특허, 저작권 등과 같은 지적 권리의 사용에 대한 법률을 잘 인지하고 있어야 한다. 고용주와 고객들의 지적 권리를 보호하는 것에 주의를 기울여야 한다.
- *컴퓨터 오작동(Computer misuse)*
  - 소프트웨어 공학자는 다른사람의 컴퓨터를 잘못 동작시키기 위해 자신의 기술을 사용하지 않아야 한다. 컴퓨터 오작동은 상대적으로 사소한 것(고용주의 머신상에서 게임을 하는것)부터 매우 심각한 것(바이러스의 유포)까지 여러 유형이 포함된다.

# ACM/IEEE 윤리규약

- 미국의 전문가 사회에서는 윤리적 실천사항에 대한 규약을 만들기 위해 공동작업을 하였다.
- 이 조직의 멤버들은 그들이 가입할 때 실천 규약에 서명한다.
- 규약은 전문적인 소프트웨어 공학자들(실무자, 교육자, 관리자, 감독자, 정책 입안자 뿐만아니라 훈련생, 학생들을 포함하여)이 만든 행동과 결정에 관련된 8개의 원리(Principle)를 포함한다.

# 윤리 규약 - 서문

- **서문(Preamble)**
  - 규약의 요약판은 높은 추상화 레벨에서의 포부(aspiration)들을 요약한다; full 버전에 포함된 구문들은 이런 포부들이 어떻게 소프트웨어 공학 전문가로서 수행해야 하는 방식들을 바꾸는가에 대한 예제들과 상세한 내용들을 제시한다. 이런 포부가 없다면, 상세한 내용들은 단순히 번잡적이고 지루해질 것이다; 상세한 내용이 없다면, 포부들은 좋은 소리이기는 하지만 빈 껍데기가 될 것이다; 포부와 상세한 내용 들이 합쳐져서 규약을 구성한다.
  - 소프트웨어 공학자들은 소프트웨어에 대한 분석, 명세화, 설계, 개발, 테스팅 그리고 유지보수가 가치있고 존경받는 직업이 되도록 실천할 것이다. 공공의 건강, 안녕 그리고 복지에 대한 실행과 함께, 소프트웨어 공학자들은 다음과 같은 8개의 원리를 고수해야 한다:

# 윤리 규약 - 원리

1. 공공(PUBLIC)
   - 소프트웨어 공학자들은 일관되게 공공의 이익을 위해 행동하여야 한다.
2. 고객과 고용주
   - 소프트웨어 공학자들은 공공의 이익과 함께 그들의 고객과 고용주들의 이익을 추구하는 범위하에서 행동하여야 한다.
3. PRODUCT
   - 소프트웨어 공학자들은 그들의 프로덕트와 관련된 변형물들이 가능한 최상위의 전문적인 표준을 만족하는 것을 보장해야 한다.

# 윤리 규약 - 원리 (계속)

4. 판단
   - 소프트웨어 공학자들은 그들의 전문적인 판단(judgement)에서 성실성과 독립성을 유지해야 한다.
5. 관리
   - 소프트웨어 공학 관리자와 리더들은 소프트웨어 개발과 유지보수에 대한 관리에서 윤리적 접근방법을 채택하고 추진하여야 한다.
6. 직업
   - 소프트웨어 공학자들은 공공의 이익과 함께 직업의 고결성과 평판도를 촉진시켜야 한다.

# 윤리 규약 - 원리 (계속)

## 7. 동료 (COLLEAGUES )

- 소프트웨어 공학자들은 동료들에게 공정하고 우호적이어야 한다.

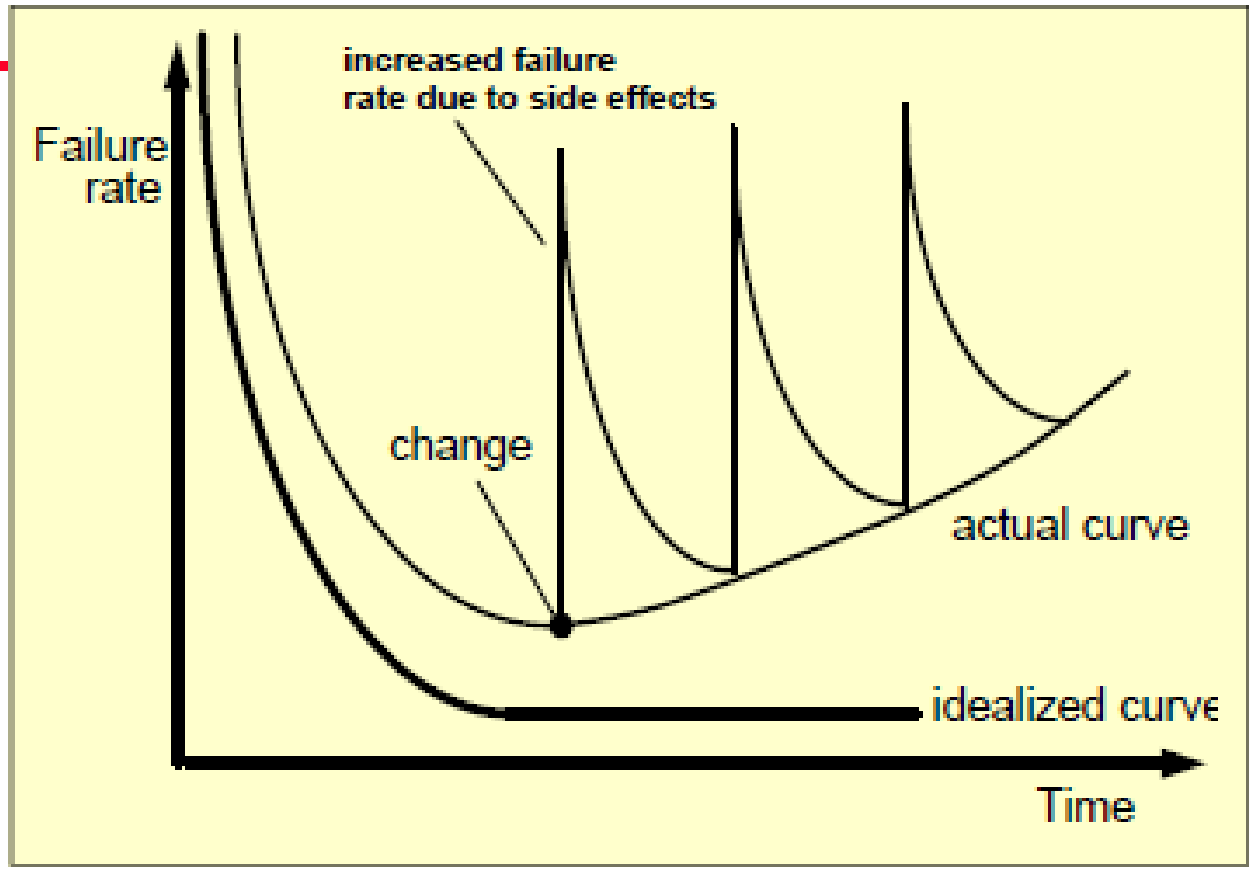
## 8. 자신(SELF)

- 소프트웨어 공학자들은 그들의 직업의 실행에 관하여 평생 학습에 참여하여야 하며 직업의 실행에 대한 윤리적 접근법을 추진하여야 한다.

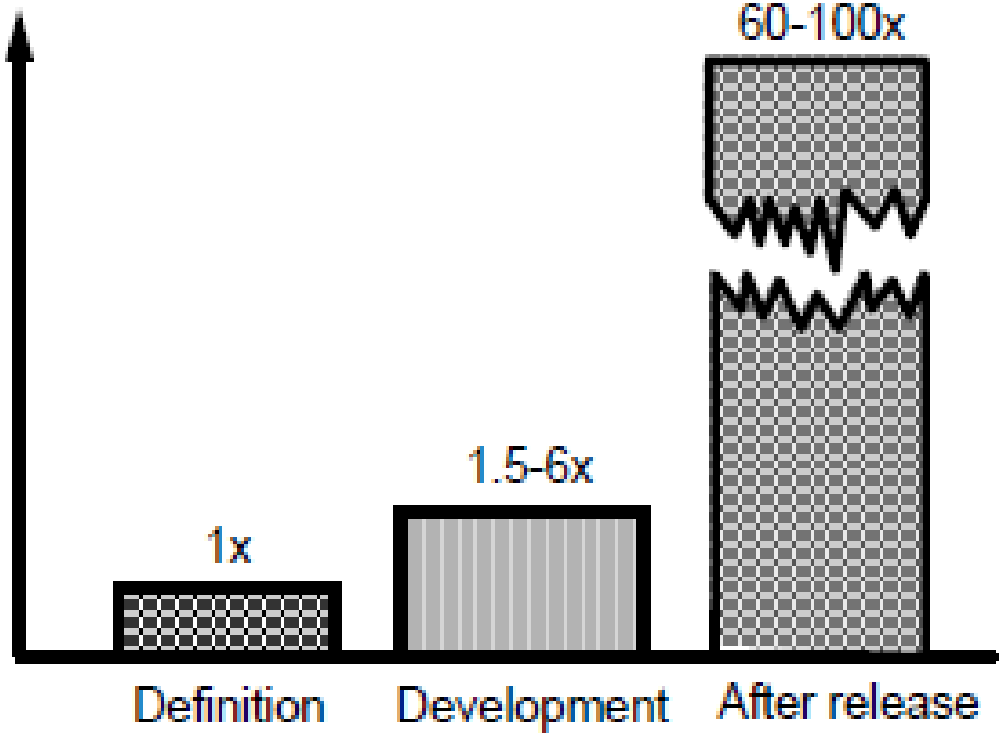
# 도덕적 딜레마

- 상급 관리자의 정책과 원리와의 불일치
- 고용주들이 비윤리적인 방식으로 행동하고 안전도가 중요한 시스템에 대해 테스팅이 끝나지 않은채 인도하는 경우가 있음.
- 군사 무기 시스템 또는 핵 시스템의 개발에의 참여

# Wear vs. Deterioration



# The Cost of Change



# Key points

- 소프트웨어 공학은 소프트웨어 생산의 모든 측면에 관심을 가지는 공학 훈련(분야/engineering discipline)이다.
- 소프트웨어 프로덕트는 개발된 프로그램, 관련된 문서들로 구성된다. 프로덕트의 중요 성질에는 유지보수성, 신뢰성, 효율성, 그리고 사용가능성이 있다.
- 소프트웨어 프로세스는 소프트웨어 프로덕트들을 개발하는데 관련된 활동들로 구성된다. 기본적인 활동에는 소프트웨어 명세화, 개발, 확인과 진화이다.
- 방법들은 소프트웨어를 생산하는 구조적인 방법들이다. 여기에는 필요한 활동들과 활동들의 순서, 사용되는 표기법, 만들어야 하는 시스템 설명에서 지켜야할 규칙과 설계 지침 등이 포함된다.

# Key points (계속)

- CASE 도구는 소프트웨어 프로세스에 있는 일상적인 활동들(설계 다이아그램의 편집, 다이아그램 일관성 검사, 실행된 프로그램 테스트들에 대한 추적 관리 등)을 지원하는 소프트웨어 시스템이다.
- 소프트웨어 공학자는 책임감을 가져야 한다. 그것은 단순히 기술적인 이슈에만 국한되지 않는다.
- 전문가 사회에서는 그들의 멤버들에게 기대되는 행동의 표준을 설정하는 실천 규약을 공표하였다.