Recommended Skills and Knowledge for Software Engineers

Steve Tockey

요약

1. **컴퓨터 과학**과 **소프트웨어 공학**은 구분되지만 관련된 주제라는 것.
2. **컴퓨터 과학**과 **소프트웨어 공학**의 정의를 명확하게 하는 것.
3. 소프트웨어 공학자들에게 추천 **지식**과 **기술**을 제공.
4. 표준화된 소프트웨어 공학 교육과정에 쓰여서 소프트웨어 산업 발전에 도움.

1. 서론

- 산업체에서 **컴퓨터 과학에** 관련된 알맞은 지식과 기술 **체계**를 구성하는 것은 적절한 합의가 있다.
- Computing Science Accreditation Board : 컴퓨터 과학 인증 프로그램을 위한 규정을 발행
- 소프트웨어 공학과 컴퓨터 과학의 관계에 관한 합의(agreement)가 적음.
- 소프트웨어 공학을 위한 알맞은 기술과 지식의 구성에 대한 합의 (agreement)가 적음.
- 결과적으로 소프트웨어 공학 학위를 딴 학생들, 또는 같은 이름의 학위지만 다른 기관에서 취득한 학생들의 기술과 지식은 **광범위**하고 **다양**
- 이것은 고용주들이 진정한 소프트웨어 공학 학위를 평가하기 어렵게 만든다.

2. 컴퓨터 과학 VS 소프트웨어 공학

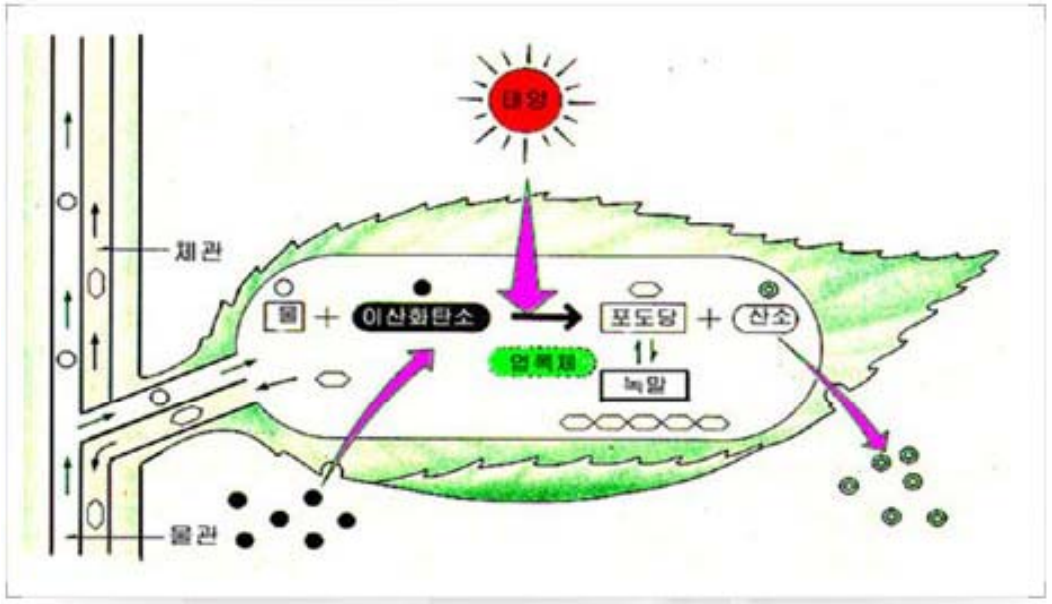
- **<과학과 공학의 일반적인 정의>**
- **과학** – 연구의 대상으로서 체계화된 지식의 일부분 과학적 방법을 통해 시험 또는 획득된 일반적인 법칙의 작용 또는 일반적인 믿음의 지식 체계.

- **공학** - 경제적으로 인류의 이익을 위해 자연의 힘과 물질의 활용 방법을 발전시키는 것을 판단하는데 적용하는 연구와 경험, 실습으로 얻게 된 수학,과학적 전문지식 <ABET>

- ABET(Accreditation Board of Engineering and Technology) – 미국 대학들의 공학 기술 인증제도.

2. 컴퓨터 과학 VS 소프트웨어 공학

- 화학 **과학** : 우주에서 볼 수 있는 현상을 설명하기 위해 화학적 프로세스를 더 잘 이해하는 것처럼 화학 지식을 위해 화학 지식을 확장하는 것에 중점



< 잎에서 발생하는 광합성 >

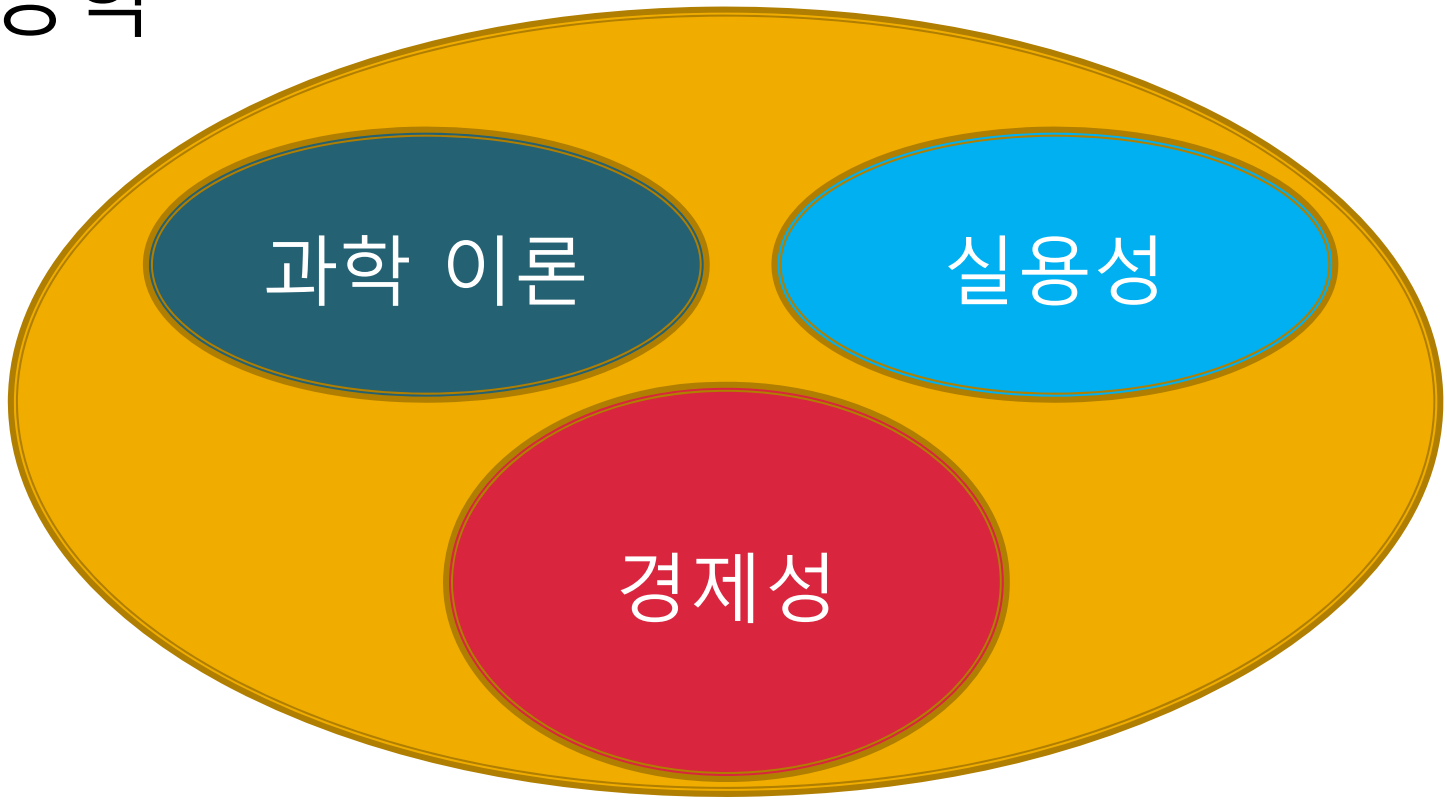
2. 컴퓨터 과학 VS 소프트웨어 공학

- 화학 **공학** : 경제적인 이해와 함께 압력 용기의 설계, 열의 낭비 제거 메커니즘 같은 화학 프로세스의 **실용성**에 중점을 두며 **화학 과학으로부터 온 지식을 가지고** 사람들의 **요구를 채워 주는 것**.

< 원자력 발전소 >

2. 컴퓨터 과학 VS 소프트웨어 공학

- 공학



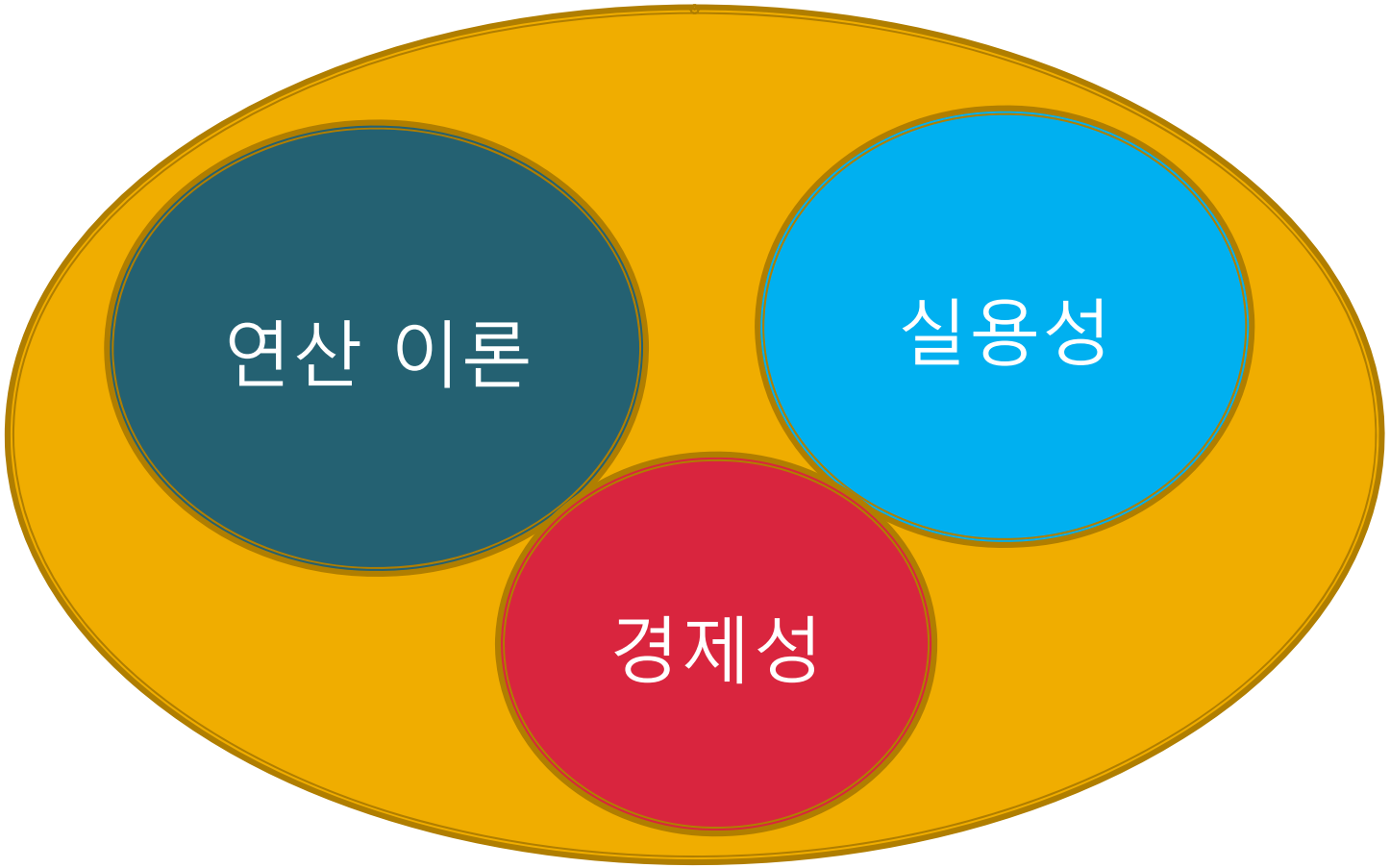
2. 컴퓨터 과학 VS 소프트웨어 공학

- <과학의 일반적인 정의로 부터>
- **컴퓨터 과학** = 학문의 한 분야로서 **연산(computing)**에 대한 체계화되어 있는 지식의 일부분.

- <공학의 일반적인 정의로 부터>
- **소프트웨어 공학** = **경제적으로 인류의 이익을** 위해 **연산 시스템의 활용 방법을** 증가하기 위해 학업, 경험, 실험, 실습을 통해 얻게 된 **수학, 연산 과학 지식**.

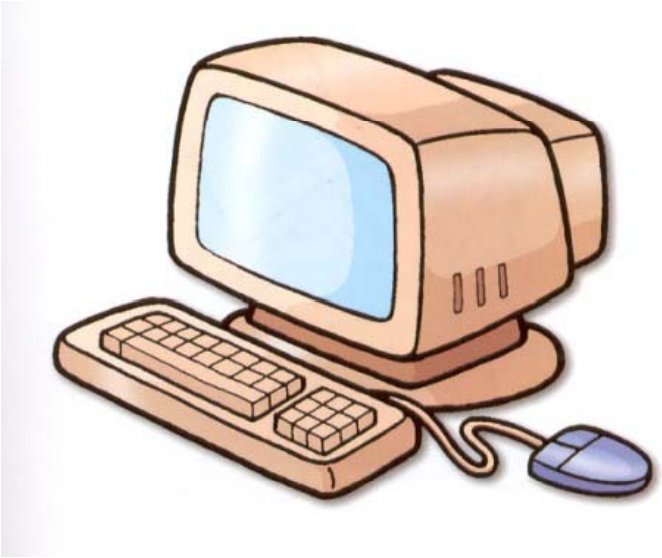
2. 컴퓨터 과학 VS 소프트웨어 공학

- 소프트웨어 공학



2. 컴퓨터 과학 VS 소프트웨어 공학

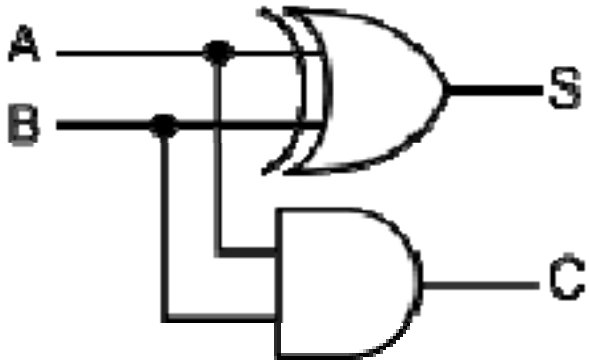
- 컴퓨터 과학과 소프트웨어 공학의 공통점
  - 컴퓨터, 연산, 소프트웨어를 다룬다.



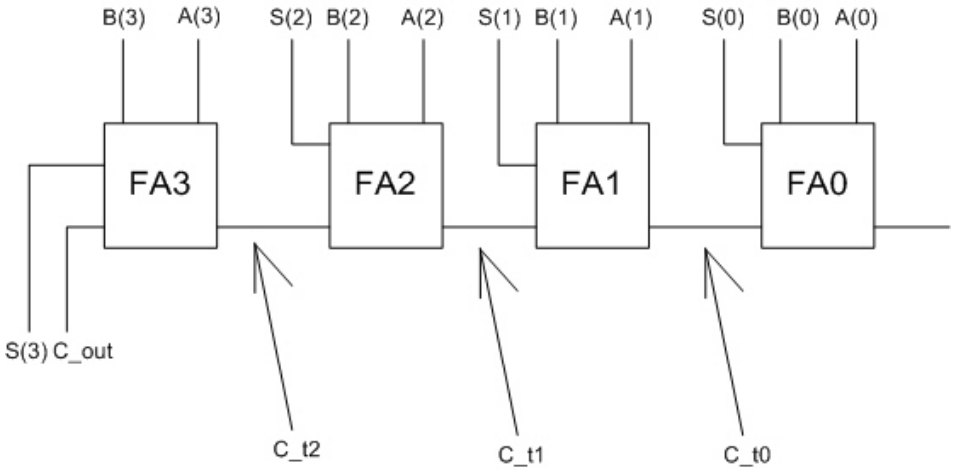
2. 컴퓨터 과학 VS 소프트웨어 공학

- 컴퓨터 과학과 소프트웨어 공학의 차이점

  컴퓨터 과학 : 컴퓨터 지식 그 자체로서 중요. 지식으로 서의 컴퓨터, 연산, 소프트웨어에 대한 지식과 그 지식에 대한 확장에 관련.



< Half- Adder>



<4비트 가산기>

2. 컴퓨터 과학 VS 소프트웨어 공학

- 소프트웨어 공학 : 경제적인 연산(Computing) 시스템과 효율적인 관리를 의해서 특별하게 제작, 디자인된 **실질적인 목적**의 소프트웨어, 컴퓨터, 연산의 응용(application)에 관련.

<휴대폰>

<MP3플레이어>

<구글의 모바일 OS 안드로이드(가칭)>

3. 추천 소프트웨어 공학 기술과 지식

- **기술** – 발전된 능력이나 행동(aptitude)
- **지식** – 공부, 연구, 관찰, 경험에 의해 얻게 된 사실, 생각.
- 소프트웨어 공학 협회의 능력 신장 모델, 산업체와 대학에서 소프트웨어 경험이 있는 20세 이상의 프로그래머가 공학자들에게 지식과 기술을 추천.
- **비용-효율적인** 연산 시스템의 설계, 구현, 유지가능.
- 공학자들에게 좀더 professional 한 실무가능.
- 반대로 이러한 지식과 기술이 없으면 **비용-비효율적인 소프트웨어 결과물**.
- 소프트웨어 조직에서 이러한 지시과 기술을 가진 공학자는 그렇지 못한 공학자 보다 좀 더 가치 있는 사람으로 여겨짐.

3. 추천 소프트웨어 공학 기술과 지식

- 모든 소프트웨어 공학자들이 추천된 모든 내용을 아는 숙달된 사람이 되어야 하는 것은 이상이다.
- 일반적으로 자신의 관심사는 깊고 자세히 ,나머지는 핵심 지식들만 다양하고 간단하게 알면 된다.
- 소프트웨어 개발시에는 각 소프트웨어 개발마다 관련된 기술과 핵심 지식을 아는 능숙한 사람이 각 팀에 적어도 한 명 필요.
- 전체로서 팀은 팀원 각 개인들의 힘을 증가 시킨다.

3.1 연산 이론

- Dr. Richard Feyman (세계 최고의 물리학자)
  - "어떻게 그렇게 훌륭한 많은 생각들을 할 수 있게되었습니까?"
  - " 쓸모 없게된 생각들은 **버리고 가능한 많은 생각**들을 한 것"
- 소프트웨어 공학자들에게 **연산에 관한 중요한 관점**을 제시.
- 기타 다른 방법 보다 다양하고 많은 설계 방법 제시
- 기타 다른 방법 보다 빠르게 잘못된 이론으로 인해 동작하지 않게 제안된 디자인의 확인과 폐기

3.1 연산 이론

- ------- **1. 추천 연산 이론** -------
- Programming language concepts
- Data Structure concepts
- Database system concepts
- Relational algebra
- Operating System concepts
- Software architectures
- Computer architectures
- **Automata theory and Petri nets**
- Computability theory and **Turing machine theory**
- **Complexity theory**
- Linguistics and parsing theory
- Computer graphics
- Set theory
- Predicate logic <술어 논리>
- Formal proofs <형 증명>
- Induction <귀납법>

3.1 연산 이론

- **Petri-net** - 분산 시스템 환경의 기술을 위한 수학적 모델링 중에 하나.

- **Turing machine:** 추상적 기계로써 충분한 기억장소와 처리시간이 주어지고 문제를 해결할 수 있는 절차(알고리즘)가 주어지면 문제를 일반화하여 처리할 수 있는 도구.
  Turing Test: 기계가 인간과 얼마나 비슷하게 대화할 수 있는지를 기준으로 기계에 지능이 있는지를 판별하고자 하는 테스트 앨렁 튜링이 제안.

- **복잡성 이론** – 복잡성 이론은 주어진 문제를 해결하기 위해 연산(computatoin)을 수행하는 동안 요구되는 자원과 관련된 연산 이론의 한 부분. 가장 일반적인 자원은 문제의 해결을 수행하기 위해 필요한 단계들의 **시간**과 필요한 **메모리의 양**이다.

3.2 소프트웨어 실무(PRACTICE)

--- 소프트웨어 제품 공학의 추천 기술과 지식---

- Requirements, analysis, and equirements engineering
- Software design
- Code optimization and semantics preserving transformations
- Human-computer interaction, and usability engineering
- Specific programming languages
- Debugging techniques
- Software-software and software-hardware integration
- Product family engineering techniques and reuse techniques
- CASE/CASE tools

3.2 소프트웨어 실무(PRACTICE)

- **--- 추천 소프트웨어 품질 보장 (Software Quality Assurance SQA) 기술과 지식 ---**

- Task kick-off, previews, and readiness reviews
- **Peer reviews**, inspections, and walk-throughs
- Software project audits (감사)
- Requirements tracing/**Quality Function Deployment(QFD**)
- Software testing techniques
- Proofs of correctness
- Process definition and process improvement techniques.
- Statistical process control
- Technology innovation

3.2 소프트웨어 실무(PRACTICE)

- **< 추천 소프트웨어 품질 보장 기술과 지식 >**

- **Quality Function Deployment**(QFD) - 품질 기능 전개 - 제품 개념 정립, 설계, 부품 계획 그리고 생산 계획과 판매까지 모든 단계를 통해 **고객의 요구가 최종 제품과 서비스에 충실히 반영**되도록 하여 **고객의 만족도를 극대화 하는데 초점**을 맞추고 있는 방법론의 하나

- **Peer reviews** – 시스템이나 시스템 컴포넌트 또는 소프트웨어 프로그램의 결함이나 개선 사항을 발견하기 위하여 **개발 당사자를 제외한 주변 동료가** 시스템 문서 및 프로그램 코드(code)를 검토, 분석하고 **개선 사항을 제안**하는 작업.

3.2 소프트웨어 실무(PRACTICE)

- **<소프트웨어 제품 전개 기술과 지식>**
- 설계와 구현된 것 만으로서 소프트웨어는 불충분.
- 소프트웨어는 최종 사용자가 사용할 형식으로 패키지 되고 전달 되어야 함.
- 사용자는 제품에 대한 사용 방법, 유지, 수리 방법에 대한 도움을 요청 할 수 있음.

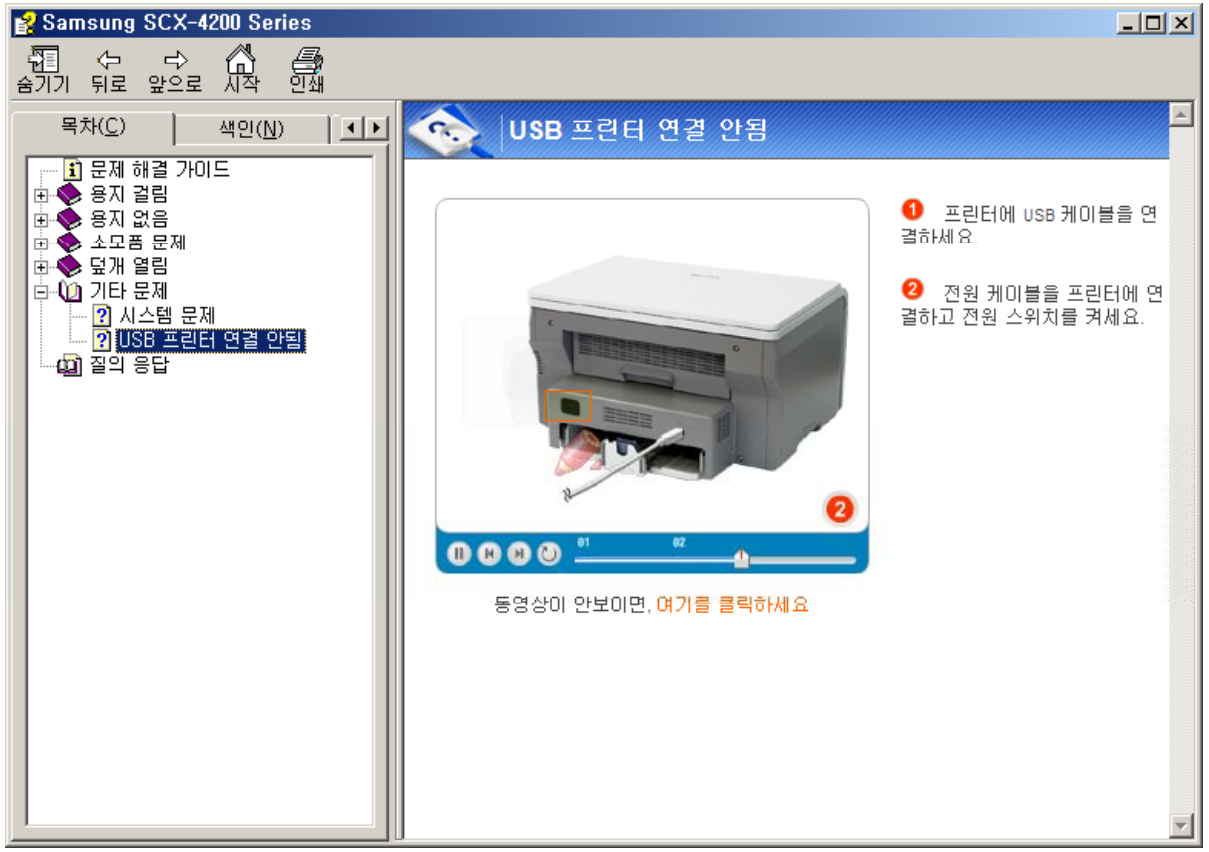
3.2 소프트웨어 실무(PRACTICE)

- **-- 추천되는 소프트웨어 제품 전개 기술과 지식 --**

- User documentation techniques (사용자 설명서 기술)
- Product packaging techniques(제품 포장 기술)
- System conversion techniques
- Customer support techniques
- General technology transfer issues

3.2 소프트웨어 실무(PRACTICE)

< Customer support techniques >



<삼성 프린터 도움말 프로그램>

3.2 소프트웨어 실무(PRACTICE)

- **< 소프트웨어 공학 경영 기술 과 지식 >**

- 소프트웨어 프로젝트는 회사에서 투자한 돈에 대한 최대한의 좋은 결과물을 얻기 위해 사람, 시간, 장비, 돈 기타 등등의 **자원의 협력**이 매우 중요하다.

- **소프트웨어 공학 경영은** 소프트웨어 조직과 소프트웨어 조직 주위의 기관들의 필요한 협력을 제공.

3.2 소프트웨어 실무(PRACTICE)

- **-- 추천되는 소프트웨어 공학 경영 기술과 지식 --**
- Risk assessment and risk management (위험 경영)
- Project planning (프로젝트 계획)
- Alternative software lifecycles
- Organizational structures (조직 구조)
- Organizational behavior (조직 활동)
- Project tracking and oversight (프로젝트 감시)
- Cost management, schedule management, and resource management (비용 ,계획, 자원 경영)
- Metrics, goal-question-metric paradigm, and measurement theory
- Configuration management and change management
- Supplier and subcontract management(공급자, 하도급 경영)
- Effective meeting skills
- Effective communication skills
- Negotiation skills (협상 기술)

3.3 공학 경제

- **경제** : 자원의 효율적이고 알뜰한 사용.
- **공학 경제** : 미시경제학에 적용되며 가장 근본적인 문제는 "제안된(proposed) 기술적 노력으로 제한된 자원을 투자하여 기업의 최대한의 **이익(interest)**을 내는것"
- 업무적 관점 : **이익(profit)**은 조직의 **목표**이자 **생존**이다. **근본적인 공학의 목표**는 **최소한의 지출**로부터 **최대한의 수익**(income)을 얻는 것, 즉 **이익의 최대화**이다.
- 정부적 관점 : 공학 경제는 매우 중요. **최소한의 세금**을 가지고 **최대한의 사람들에게 최대한의 복지 혜택**을 주는 것.

3.3 공학 경제

- Leon Lavy는 소프트웨어 공학의 타당성과 공학 경제에 관한 중요한 관점을 제시.
- 소프트웨어 경제학은 자주 프로그래밍 프로젝트에 관한 비용 평가로 착각.
- 경제는 선택의 과학
- 소프트웨어 경제학은 소프트웨어 프로젝트가 해야하는 선택들의 분석을 위한 모델과 방법을 제시해야한다.
- 프로젝트 기간을 잡아야 하는 경우에 이러한 모델과 분석들은 그 기간에 대한 선명해 해답을 제공.

3.3 공학 경제

- **-- 추천되는 공학 경제 기술과 지식 –**

- Time value of Money
- Economic equivalence (경제 등가)
- Inflation
- Depreciation
- Income taxes
- Decision making among alternatives (대안 결정)
- Decision making under risk and uncertainty (위험성 결정)
- Evaluating replacement alternatives (대안 대체 평가)
- Evaluating public activities (공적 업무 평가)
- Breakeven (균형점, 이익도 손해도 없는)
- Optimization (최적화)

3.4 고객과 업무 환경

- 소프트웨어 공학자가 고객에게 좋은 제품과 효율적인 서비스를 하기 위한 방법

  ➔ 자신의 제품이 고객의 업무에 어떻게 서비스되며 어떤 영향을 주는지에 대한 자세한 이해가 필요.

- Engineer에게 필요한 지식
  - 누가 고객이며 무엇이 그들의 일인가?
  - 그들은 우리의 어떤 제품을 사용하며 무엇을 서비스 하는가?
  - 언제, 어디서, 무엇 때문에 우리의 제품이 사용되며 서비스 되는가?
  - 우리의 제품이 처음 의도의 목적과는 다른 방법으로 서비스되고 사용되진 않는가? 그렇다면 그 이유는 무엇인가?
  - 어떻게 우리의 제품과 서비스가 고객의 비즈니스에 영향을 주는가?
  - 무엇이 제품과 서비스가 고객에 전달되는 능력에 제한과 규정을 두는가?

Table 7. Recommended customer and business environment skills and knowledge

- **-- 7. 추천되는 고객과 업무 환경 기술과 지식 --**

- Customer satisfaction assessment techniques(고객 만족 평가 기술)
- Competitive benchmarking techniques(경쟁사 벤치마킹 기술)
- Technical communication
- Intellectual property law (지적 소유권 관련법)
- Ethics and professionalism

4. SUMMARY

- Computer Science와 Software Engineering 의 차이점 과 공통점 그리고 관련되어 있는 방법.
- Software Engineer들에게 힘이 될 수 있는 여러 가지 기술과 지식을 추천
- 기술과 지식들을 가지고 졸업한 학생들이 소프트웨어 기관에 중대한 책임을 가진 위치에 빠르게 적응.
- 기관들은 그러한 소프트웨어 공학 학위의 진정한 가치에 감사할 것.