




2024-2학기







게임소프트웨어공학(GSE)

[3주차-1] 이론 3장 소프트웨어 개발 프로세스

0. PLC와 SDLC의 이해

1. 실현 가능성 분석
2. 전통적인 소프트웨어 프로세스
3. 애자일 및 XP 프로세스
4. 소프트웨어 프로세스 개선
5. 과제/진도/Q&A



이 강의자료는 여러분들을 위한 담당교수의 경험/생각과 한빛아카데미(㈜)에서 제공한 교재 자료를 활용해서 작성하였습니다

1

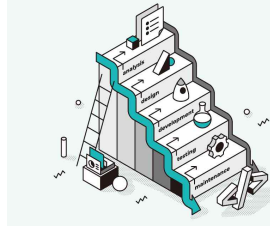
[이론 3장] 소프트웨어 개발 프로세스


주요 내용


1. PLC와 SDLC의 이해
2. 실현 가능성 분석
3. 전통적인 소프트웨어 프로세스
4. 애자일 및 XP 프로세스
5. 소프트웨어 프로세스 개선

학습 목표

1. 소프트웨어 개발 프로젝트가 실현 가능한지 진단하는 고려사항 이해
2. 소프트웨어 개발 프로세스 모델의 종류와 특성 이해 및 비교
3. 소프트웨어 개발 능력을 향상하기 위한 CMMI와 SPICE 이해
4. 소프트웨어 프로세스 관련 표준 이해






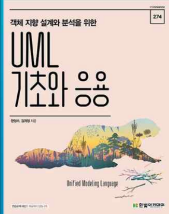


[이론 3장] 소프트웨어 개발 프로세스

2




이론교재
3장
**소프트웨어 개발
프로세스+@
(PLC+SDLC)**



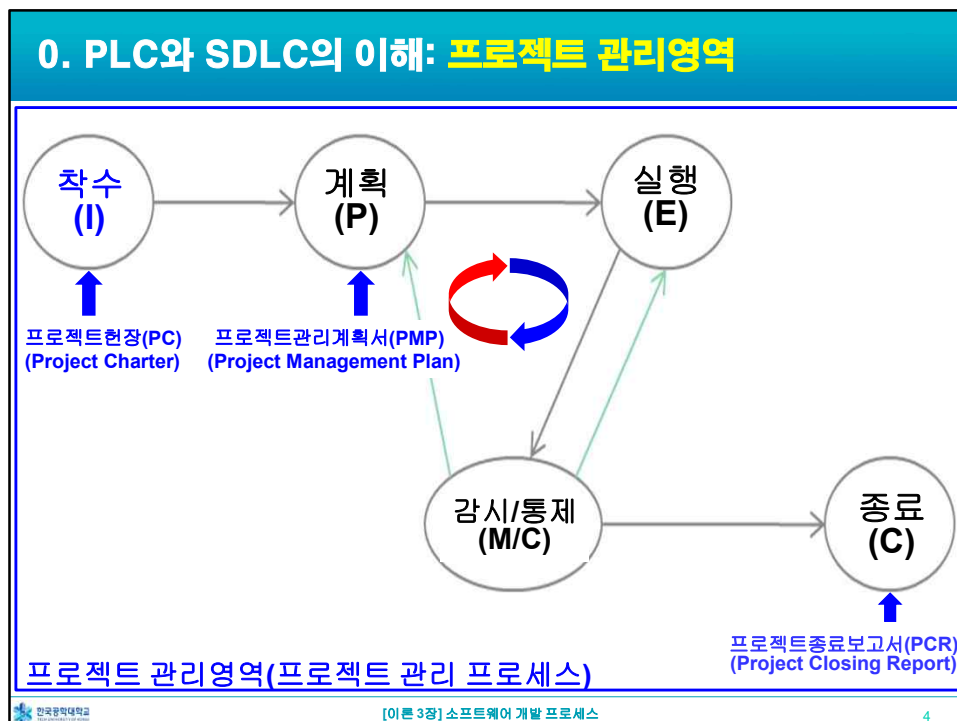

0. PLC와 SDLC의 이해

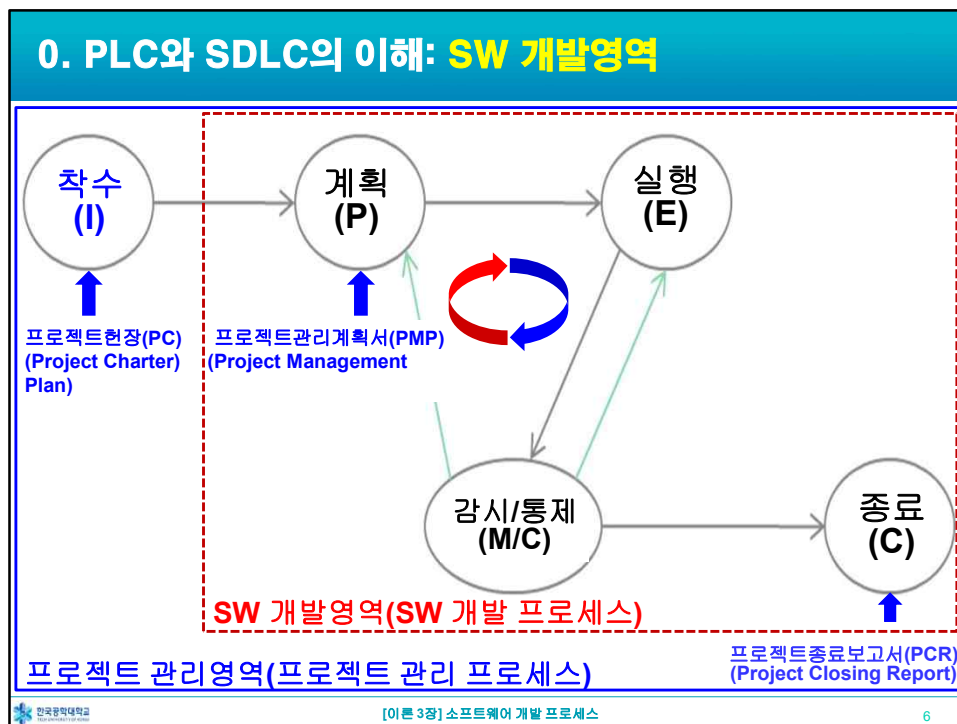
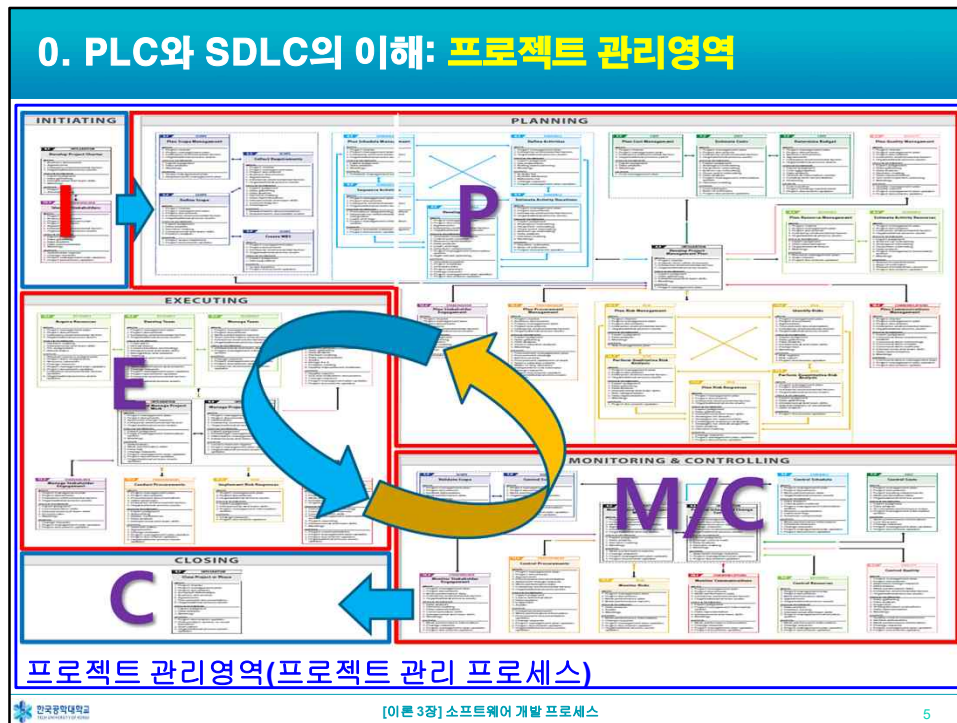
1. 실현 가능성 분석
2. 전통적인 소프트웨어 프로세스
3. 애자일 및 XP 프로세스
4. 소프트웨어 프로세스 개선
5. 과제/진도/Q&A

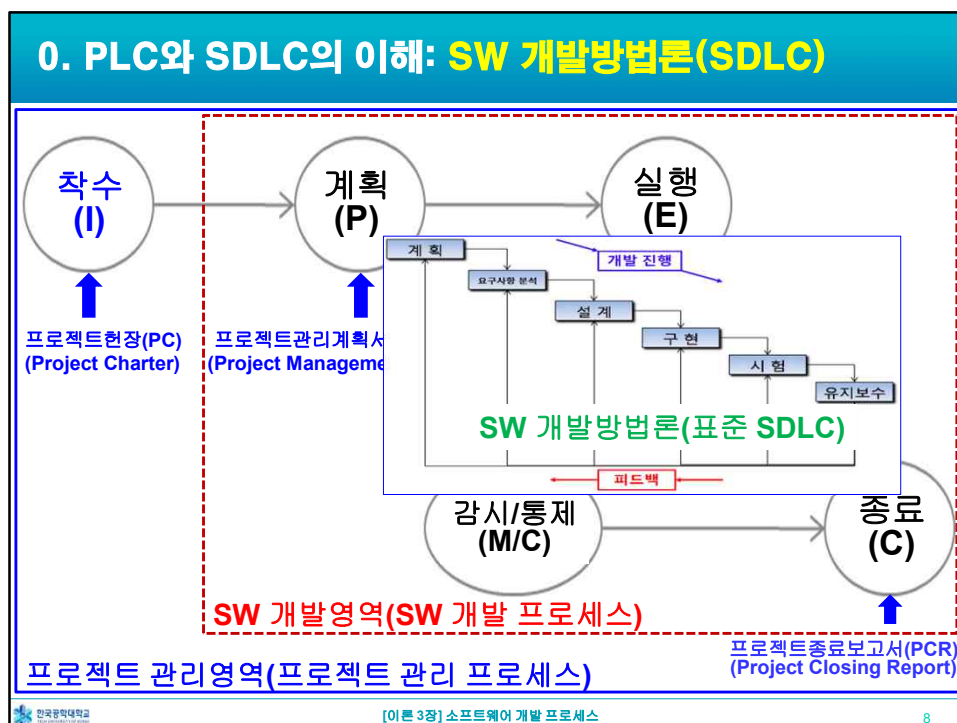
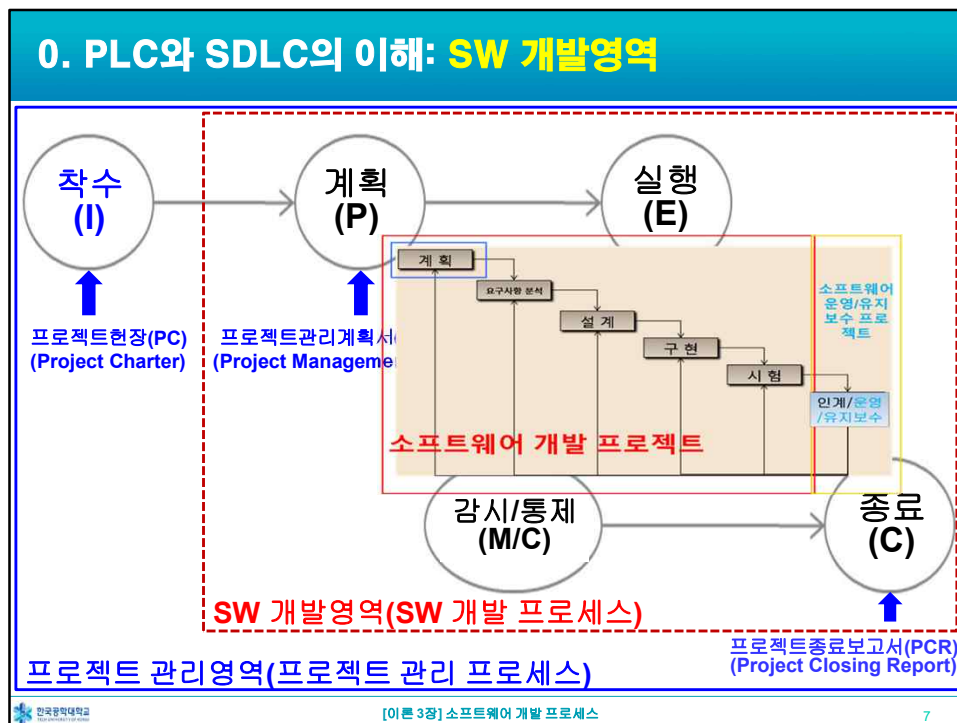


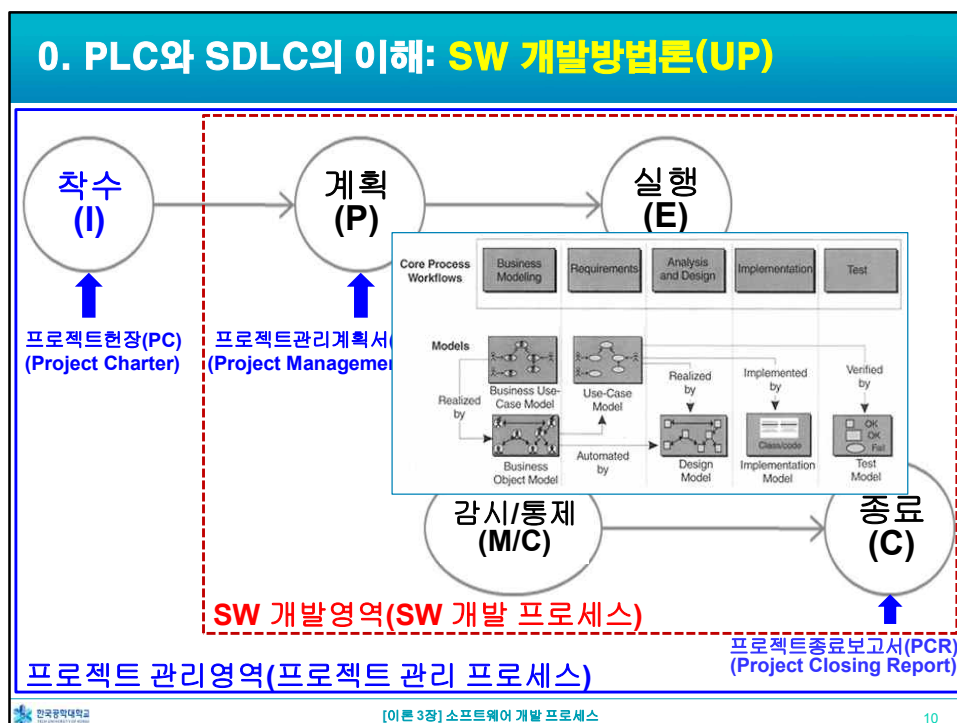
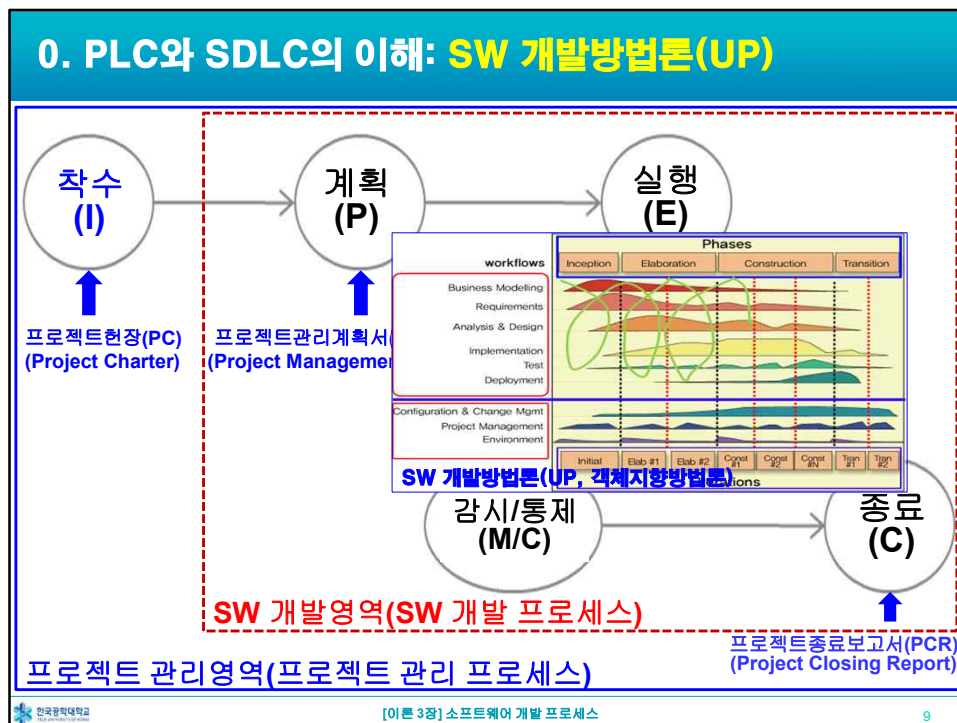
[이론 3장] 소프트웨어 개발 프로세스

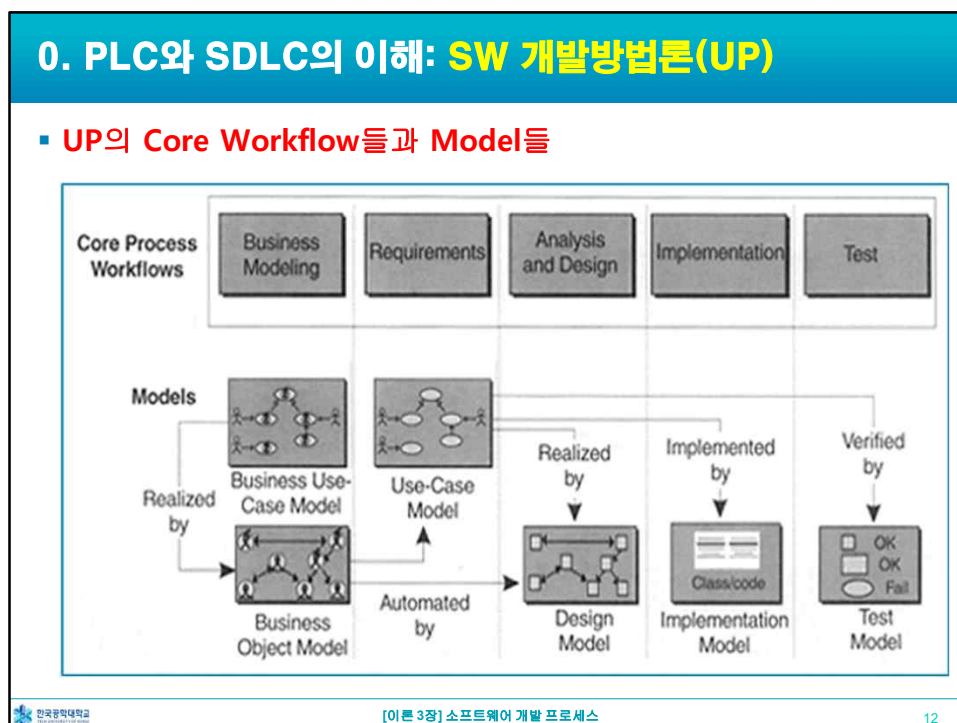
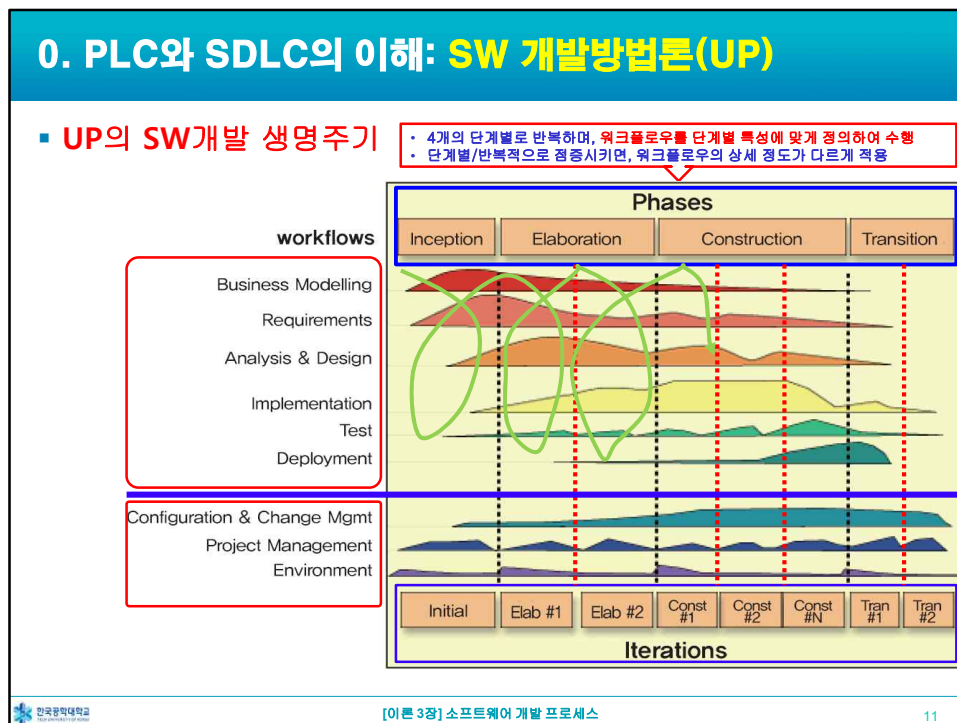
3

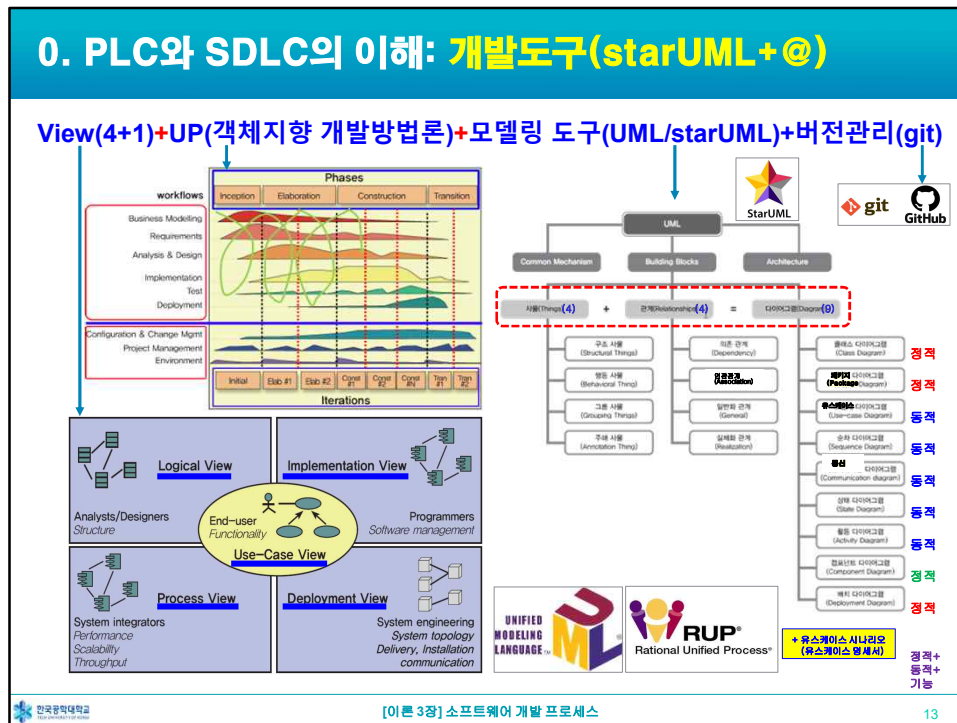












이론교재

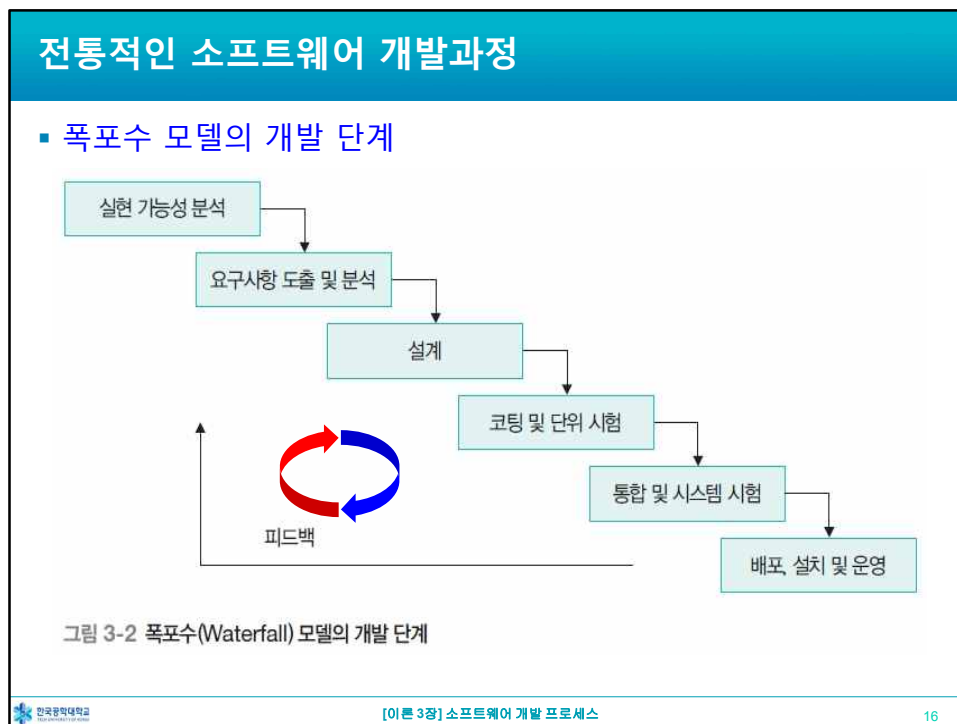
3장

소프트웨어 개발
프로세스+@
(PLC+SDLC)

0. PLC와 SDLC의 이해

1. 실현 가능성 분석
2. 전통적인 소프트웨어 프로세스
3. 애자일 및 XP 프로세스
4. 소프트웨어 프로세스 개선
5. 과제/진도/Q&A

[이론 3장] 소프트웨어 개발 프로세스



실현 가능성: 실현 가능성 분석

■ 실현 가능성 분석

- 사용자의 요구사항에 따라 소프트웨어 시스템을 개발하고자 할 때 비용, 일정, 기술적 수준 등이 충분히 가능한가를 살펴보는 것
- 따라서 개발 비용^{Cost}, 이득^{Benefits}, 대안^{Alternatives} 평가

■ 개발 비용

- 개발 인건비 뿐만 아니라 백업용 하드웨어 비용, 개발 및 운영 소프트웨어 도구 비용, 유지보수 비용, 문서화 소요 비용, 개발자 교육비, 시스템 교체 비용 등 모든 비용

■ 이득

- 새로운 시스템이 제공하는 비즈니스 프로세스의 개선된 영역은?
- 운영의 효율성과 정확성은 증진되는가?
- 의사결정을 위한 정보 제공을 적시에 제공할 수 있는가?

■ 대안

- 비용과 이득 산정 시 다양한 측면을 고려하여 Trade-off 분석
- 예) 고급 기술자를 고용하는 경우와 기존 직원을 교육하는 경우를 각각 고려해 분석

실현 가능성: 실현 가능성 분석


■ 실현 가능성 분석을 위해 고려할 측면

- 경제적 측면
 - 비용 대 효과 분석을 수행하여 프로젝트가 비용 측면에서 정당하게 평가받을 수 있는가?
- 기술적 측면
 - 소프트웨어 개발 프로젝트에서 필요한 이론이나 기술에 대해 제약 사항이 없는가?
 - 현재 가용한 기술 수준으로 목표 소프트웨어를 구현하기에 충분한가?
- 스케줄 측면
 - 소프트웨어 개발 프로젝트를 가용한 인력과 자원으로 주어진 기간 내에 완료할 수 있는가?
- 운영적 측면
 - 개발된 소프트웨어가 사용자에게 전달되었을 때, 데이터 입력이 가능한가?
 - 소프트웨어 운영에 대한 역할이나 책임을 수행하기에 시스템 관리자가 주저함이 없는가?

실현 가능성: 실현 가능성 분석

■ 실현 가능성 분석을 위해 고려할 측면

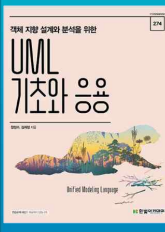

- 동기적 측면
 - 개발된 소프트웨어를 실제로 사용할 사용자가 그 필요성을 인정하고 있는가?
 - 소프트웨어가 사용자에게 배포되었을 때, 지체 없이 정확하게 업무에 적용할 수 있는가?
- 법적·윤리적 측면
 - 개발된 소프트웨어 기능을 사용하는 과정에서 개인 정보 유출과 같은 법적 분쟁의 소지가 발생할 수 있는가?
 - 사회적인 문제를 유발할 수 있는 기능이 포함되어 있는가?



이론교재


3장

소프트웨어 개발
프로세스+@
(PLC+SDLC)

0. PLC와 SDLC의 이해

1. 실현 가능성 분석
2. 전통적인 소프트웨어 프로세스
3. 애자일 및 XP 프로세스
4. 소프트웨어 프로세스 개선
5. 과제/진도/Q&A



한국과학기술대학교

[이론 3장] 소프트웨어 개발 프로세스

20

소프트웨어 프로세스의 필요성

■ 소프트웨어 프로세스(SW Process)

- 소프트웨어 개발과정을 체계적으로 계획·관리할 의도 아래 서로 관련 있는 활동을 단계로 분리하여 정의한 것

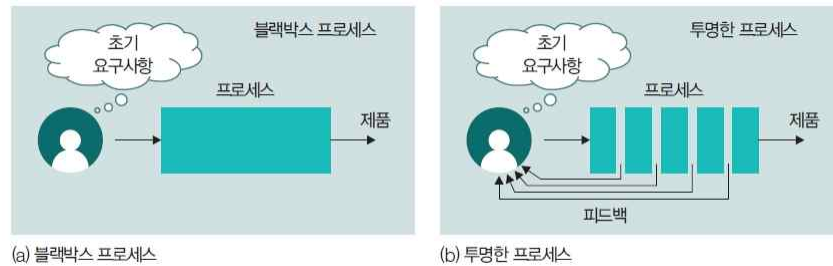


그림 3-3 소프트웨어 개발 프로세스

■ 소프트웨어 프로세스 필요성

- 요구사항 만족도 향상 -> 수정 및 재개발 절감(비용 절감) -> 품질 관리 가능 -> 프로젝트 성공율을 높여 줌

전통적인 프로세스 모델: 폭포수 모델(Waterfall Model)

■ 폭포수 모델(Waterfall Model)

- 폭포의 물이 아래로 떨어지는 것처럼 순차적인 단계로 소프트웨어를 개발하는 것
- 각 단계마다 정해진 소프트웨어 문서를 작성
 - 작성된 문서는 사용자 확인 과정을 거쳐 다음 단계의 입력으로 사용됨
- 폭포수 모델의 단점
 - 개발과정에서 요구되는 변경을 수용하기가 어려움
 - 입력으로 사용되는 이전 단계의 결과물에 오류가 잔존하는 경우, 다음 단계에 양향을 줌
- 폭포수 모델의 변형
 - 피드백이 가능한 폭포수 모델
 - 각 단계를 중첩하여 진행하는 폭포수 모델(점진적 모델) 등

전통적인 프로세스 모델: 점진적 모델

■ 점진적 모델의 전략

- 개발 및 배포: 개발자는 실사용자가 원하는 것을 개발하여 제공
- 측정 및 모니터링: 배포된 부분에 대한 사용자 유용성을 평가
- 조정 및 수정: 모니터링 결과를 설계와 구현에 반영

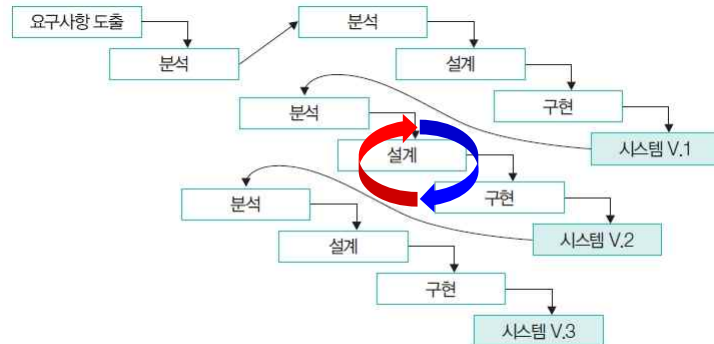


그림 3-4 점진적 모델에 의한 소프트웨어 개발 프로세스

전통적인 프로세스 모델

■ 점진적 모델의 이점

- 새로운 제품 개발 시 사용자의 요구사항이나 변경을 반영할 시간 가능
- 변경을 쉽게 수용가능
- 점진적·단계적 개발을 통해 사용자에게 사용 가능한 소프트웨어 시스템을 신속하게 전달 가능

■ 점진적 모델의 문제점

- 각 빌드에서 테스트와 통합이 반복적으로 수행되어 오버헤드 발생
- 사용자 피드백 반영하면 이미 개발된 시스템 구조화가 무너질 수 있음
- 일부 기능만 있는 소프트웨어의 신속한 배포가 오히려 전체 시스템을 기대한 사용자에게 실망을 줄 수 있음

전통적인 프로세스 모델: 프로토타입 모델(Prototype Model)

■ 프로토타입 모델(Prototype Model)

- 점진적 모델의 한 유형으로, 개발 및 배포, 사용자 만족도 측정, 조정 및 수정을 통한 시스템 개발 전략을 기반으로 소프트웨어 시스템 개발
- 기능 메뉴(혹은 버튼)만 포함하는 사용자 인터페이스의 원형을 보여주거나 사용자에게 중요하다고 판단되는 핵심 모듈만 우선적으로 개발하여 사용자에게 제공하고 이를 통해 사용자의 요구사항을 만족했는지 여부를 판단하고, 이를 바탕으로 최종 시스템을 구현
- 즉, 요구사항을 검증하기 위해 프로토타입 개발 → 폭포수 모델에 따라 전체 시스템을 개발

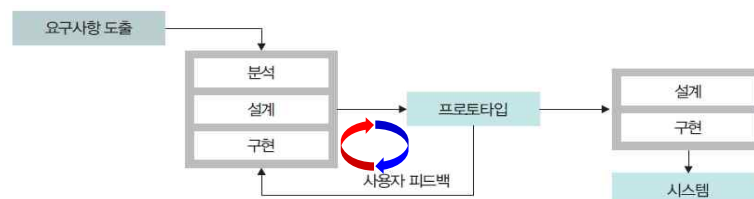


그림 3-5 프로토타입 모델에 의한 소프트웨어 개발 프로세스

전통적인 프로세스 모델

■ 프로토타입 모델의 이점

- 사용자 요구사항을 검증하여 개발 비용과 시간 단축 가능
- 프로토타입을 통해 사용자와 개발자 혹은 개발자 간 의사소통이 이루어져 상호 동일한 개념 확보 가능
- 소프트웨어를 개발하는 동안 보다 빠른 시점에서 오류 탐지 가능

■ 프로토타입 모델의 문제점

- 사용자 검증 과정 이후에 변경이 발생하는 부분을 고려하지 못함

전통적인 프로세스 모델: 나선형 모델(Spiral Model)

■ 나선형 모델(Spiral Model)

- 소프트웨어 시스템 개발 시 위험을 최소화하기 위해 점진적으로 전체 시스템으로 개발해나가는 모델로 소프트웨어 개발과정이 반복적이고 점진적으로 진행되는 나선 모양
- 목표 설정→위험 분석→구현 및 테스트→평가 및 다음 단계 수립의 활동 반복

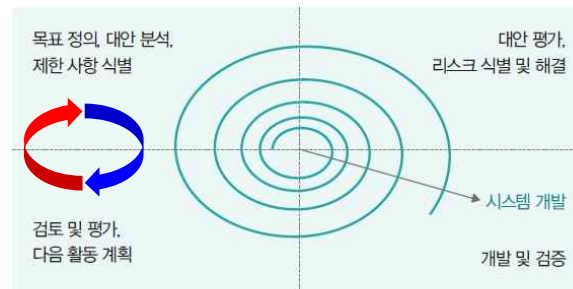


그림 3-6 나선형 모델에 의한 소프트웨어 개발 프로세스

전통적인 프로세스 모델

■ 나선형 모델의 장점

- 체계적인 위험 관리로 위험성이 큰 프로젝트를 수행하기에 적합
- 사용자 요구사항을 더욱 상세히 적용 가능
- 변경되는 요구사항을 반영하기 쉬움
- 최종 개발된 소프트웨어 시스템에 대한 사용자 만족도와 품질 향상

■ 나선형 모델의 문제점

- 프로젝트 기간이 길어짐
- 반복되는 사이클이 많아질수록 프로젝트 관리가 어려움
- 위험 분석에 따른 비용이 발생하고 위험 관리 위한 전문적 지식 요구

전통적인 프로세스 모델: V 모델(V Model)

■ V 모델(V Model)

- 폭포수 모델의 확장한 **Verification and Validation 모델**
- V 모양의 왼쪽은 아래 방향으로 내려가면서 폭포수 모델에 따른 SW 개발 과정이 진행되고, 코딩 단계 이후에는 다시 오른쪽으로 올라가면서 테스트(시험) 과정이 단계별로 수행됨

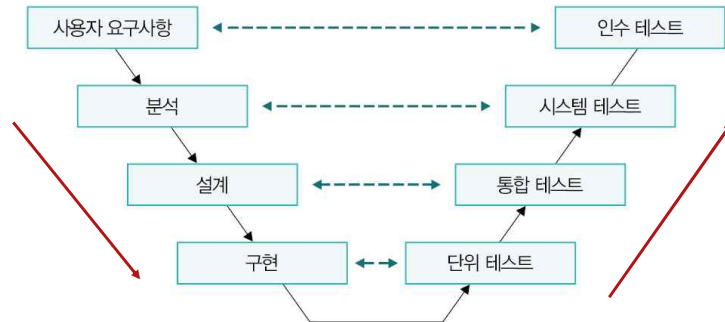


그림 3-7 V 모델에 의한 소프트웨어 개발 프로세스

전통적인 프로세스 모델

■ V 모델의 장점

- 소프트웨어 개발 결과물에 대한 단계적인 검증과 확인 과정을 거침으로써, 개발 소프트웨어에 대한 오류를 줄일 수 있음
- 요구사항이 정확히 이해되는 작은 시스템 개발 시 매우 유용

■ V 모델의 단점

- 폭포수 모델을 적용함으로써 개발 활동에 대한 반복이 허용되지 않기 때문에 변경을 다루기 쉽지 않음

전통적인 프로세스 모델: 변환 모델(Transformation Model)

■ 변환 모델(Transformation Model)

- 정형 명세 Formal Specification를 기반으로 하는 소프트웨어 개발 프로세스
- 일반적으로 소프트웨어 요구사항을 제트Z, 패트리 넷Petri Nets, SDL Specification and Description Language, 상태 차트State Chart, 시간 논리Temporal logic 등의 정형 기법을 이용하여 명세한 후에, 자동화된 도구를 사용해 직접 코드로 변환하여 소프트웨어 개발

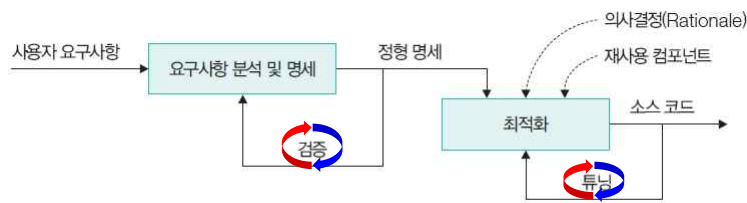



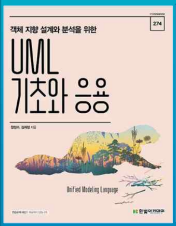

그림 3-8 변환 모델에 의한 소프트웨어 개발 프로세스



이론교재

3장

소프트웨어 개발
프로세스+@
(PLC+SDLC)

0. PLC와 SDLC의 이해

1. 실현 가능성 분석
2. 전통적인 소프트웨어 프로세스
3. 애자일 및 XP 프로세스
4. 소프트웨어 프로세스 개선
5. 과제/진도/Q&A

애자일 개발 프로세스

■ 애자일Agile 방법론

- 기민한, 좋은 것을 빠르고 낭비 없게 만드는 개발을 가능하게 해주는 방법론 전체
- 대표적인 애자일 방법론들
 - 동적 시스템 개발 방법Dynamic System Development Method, 데인 포크너Dane Faulkner
 - 적응형 소프트웨어 개발Adaptive Software Development, 짐 하이스미스Jim Highsmith
 - 크리스털 패밀리Crystal Family, 앨리스테어 쿡번Alistair Cockburn
 - 익스트림 프로그래밍XP, 칸트 벡Kent Beck.에릭 콤마Eric Gamma
 - 스크럼Scrum, 켄 스와버Ken Schwaber.제프 서더랜드Jeff Sutherland
 - 린 소프트웨어 개발Lean Software Development, 메리 팝펜딕Mary Poppendieck, 톰 팝펜딕Tom Poppendieck
 - 기능 주도 개발Feature-Driven Development, 피터 코드Peter Coad.제프 드루카Jeff DeLuca
 - 애자일 유피Agile Unified Process, 스콧 앰블러Scott Ambler

신속한 소프트웨어 개발

■ 애자일의 적용 원리

- 고객 만족을 가능한 한 빨리 이끌어내고, 유연한 소프트웨어를 지속적으로 사용자에게 제공
- 개발과정에서 늦은 시점일지라도 요구사항 변경을 적극 수용
- 실행 가능한 소프트웨어를 1~2주일에 한 번씩 사용자에게 배포
- 사용자와 개발자 간 친밀한 미팅을 거의 매일 수행
- 프로젝트는 신뢰할 수 있고 동기가 부여된 개인을 중심으로 진행
- 의사소통을 위해 가능하면 한 곳에 모여 대화
- 실행 가능한 소프트웨어를 기준으로 프로젝트 진척률 결정
- 일정한 개발 속도를 유지하며, 지속 가능하도록 소프트웨어 개발
- 우수한 기술의 도입과 좋은 설계에 지속적인 관심
- 단순화는 필수!. 작업을 가장 단순화 형태로 분리하여 고민하고 해결
- 최상의 아키텍처, 정제된 요구사항, 좋은 설계는 잘 구성된 팀을 통해 개발
- 팀은 더 효과적인 방법의 적용, 적절한 조정 과정을 정기적으로 수행

애자일: XP 프로세스

■ XP(eXtream Programming) 반복개발방법 실천사항(Practices)

표 3-1 XP 프로세스의 10가지 실천 사항

실천 사항	가이드라인
증분 계획 (Incremental Planning)	사용자 요구사항을 스토리 카드에 기록하며, 배포 시 들어갈 스토리는 사용 가능한 시간과 상대적 우선순위에 따라 결정한다. 개발자는 이러한 스토리를 개발의 단위 태스크로 분리한다.
작은 배포 (Small Release)	비즈니스 가치를 제공할 수 있는 최소한의 유용한 기능을 먼저 개발하여 배포한다. 소프트웨어 시스템 배포의 경우, 첫 번째로 배포된 소프트웨어에 기능을 점진적으로 추가하면서 배포한다.
단순 설계 (Simple Design)	추후 나올 가능성이 있는 요구사항이 아닌 현재의 요구사항을 충족시키도록 설계를 수행한다.
테스트 우선 개발 (Test-first Development)	기능 자체가 구현되기 전에 새로운 기능에 대한 테스트를 수행하기 위해 자동화된 단위 테스트 프레임워크를 사용한다.
재구조화 (Refactoring)	모든 개발자는 코드 개선이 가능한 부분을 발견하는 즉시 코드를 재구조화해야 한다. 이를 통해 코드를 간단하고 쉽게 유지보수할 수 있다.

애자일: XP 프로세스

■ XP(eXtream Programming) 반복개발방법 실천사항(Practices)

짝 프로그래밍 (Pair Programming)	개발자는 쌍으로 작업하여 서로의 작업을 확인하고, 항상 훌륭한 작업을 수행할 수 있도록 지원한다.
공동 소유권 (Collective Ownership)	개발자들은 시스템의 전 영역에 개발 작업을 수행함으로써, 경험과 전문 지식이 상호 연관되어 적용될 수 있다. 이로 인하여 모든 개발자가 모든 코드를 책임을 지며, 누구나 아무거나 변경할 수 있도록 한다.
지속적 통합 (Continuous Integration)	단위 태스크가 완료되자마자 이는 전체 시스템에 통합한다. 통합한 후, 시스템에 대한 모든 단위 테스트가 통과되어야 한다.
지속 가능한 개발 속도 (Sustainable Pace)	초과 근무가 많아지면 종종 코드 품질이 저하되고 중간 생산성이 감소될 수 있으므로 허용되지 않는다.
현장 고객 (On-site Customer)	고객/사용자 대표는 XP 개발 팀이 항상 접촉할 수 있어야 한다. XP 프로세스에서 고객은 개발 팀의 구성원이며, 고객은 소프트웨어 구현을 위해 시스템 요구사항을 팀에 제공해야 한다.

애자일: XP 프로세스

■ XP의 개발 사이클

• 엔지니어링 사이클

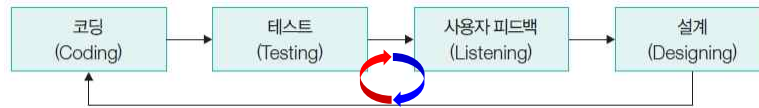


그림 3-10 XP 프로세스의 소프트웨어 엔지니어링 사이클

• 배포 사이클



그림 3-11 XP 프로세스의 소프트웨어 배포 사이클

애자일: 스크럼 프로세스

■ 스크럼(Scrum) 프로세스: 반복적 개발의 관리 관점 강조

■ 스크럼 프로세스의 용어

- 스프린트: 반복적인 개발 주기(계획 회의부터 제품 리뷰가 진행되는 날짜까지의 기간이 1 스프린트)
- 제품 백로그: 개발할 제품에 대한 요구사항 목록
- 스프린트 백로그: 스프린트 목표에 도달하기 위해 필요한 작업 목록
- 제품 증분: 스프린트 결과로 나오는 실행 가능한 제품
- 번다운 차트: 스프린트 백로그에 남아 있는 작업 목록을 보여주는 차트
- 번업 차트: 배포에 필요한 진행 사항(진척도)을 보여주는 차트
- 제품 책임자: 제품 백로그를 정의하여 우선순위를 정하는 책임자
- 스크럼 마스터: 프로젝트 관리자

애자일: 스크럼 프로세스

■ 스크럼 프레임워크

- 솔루션에 포함할 기능 및 개선점에 대한 우선순위 부여
- 개발 주기는 30일 정도로 하며, 개발 주기마다 실제 동작할 수 있는 결과 제공
- 개발 주기마다 적용할 기능이나 개선에 대한 목록 제공
- 날마다 15분 정도 회의
- 항상 팀 단위로 생각
- 원활한 의사소통을 할 수 있도록, 구분 없는 열린 공간 유지

애자일: 스크럼 프로세스

■ 스크럼 프로세스

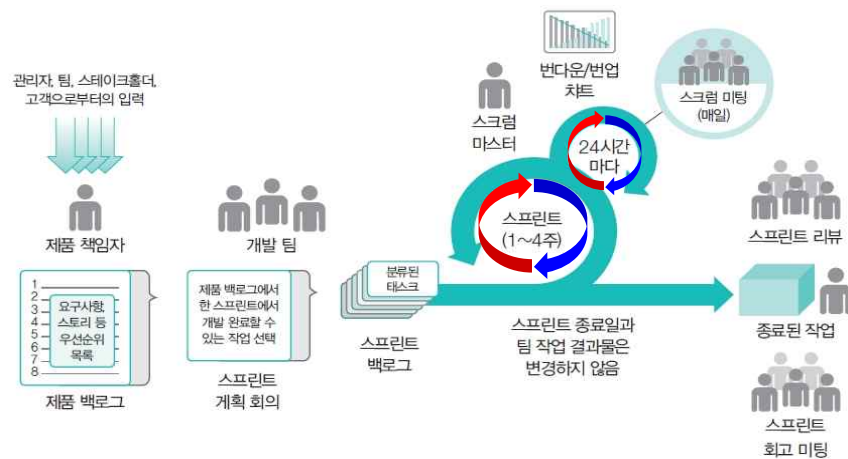


그림 3-12 스크럼 프로세스의 소프트웨어 개발 프로세스

(출처: neon rain interactive, <https://www.neonrain.com/agile-scrum-web-development>)

[실습 3-1] 스크럼 이해하기

실습 3-1

스크럼 이해하기

구글 Google에서는 소프트웨어 개발을 위하여 스크럼 프로세스에 의한 개발 활동을 진행하고 있다. 구글의 소프트웨어 엔지니어들은 자신에게 부여된 태스크에 대하여 자유롭게 편하게 소프트웨어 개발 작업을 수행한다. 그리고 매일 아침 토스트와 사과 주스를 먹으며, 스크럼 미팅 Daily Scrum Meeting을 진행한다. 다음 물음에 대하여 자신의 생각을 정리해봅시다.

- (1) 매일 스크럼 미팅을 수행하는 목적이 무엇일까?
- (2) 개발자들은 스크럼 미팅에서 어떠한 이야기를 할까?

Chaos와 DevOps

■ Chaos Engineering

- 다양한 서비스 운영 환경에 대응하고 새로운 기능을 중단 없이 제공하기 위해서, 기존 서비스에 임의의 결함을 주입하고 이들이 어떠한 행동을 보이는지 동적으로 검사하면서, 시스템 운영에 대응하는 방법을 찾는 것
- 예측 불가능한 상황에서 나타나는 시스템 동작의 규칙성을 찾아내고 이에 대응하는 활동



그림 3-13 넷플릭스의 Chaos Engineering 소개 논문

Chaos와 DevOps

■ DevOps의 정의

- Wikipedia의 정의: DevOps는 소프트웨어 개발자와 운영자 간의 소통과 협업, 통합을 강조하는 소프트웨어 개발 방법으로, 개발과 운영의 상호 의존성을 명확히 함으로써, 제품 혹은 서비스의 신속한 개발을 목표로 함
- IBM의 정의: 소프트웨어 개발 팀과 운영 팀 간의 조정을 위하여 제시된 프로세스
- O'reilly의 정의: 시스템의 운영 인프라가 'Platform as a Service' 형태로 정의되어 응용 프로그램의 일부가 되고 있으며, 운영은 사라지는 것이 아니라 개발 활동의 일부로 고려해야 함. 즉, 운영 인프라는 사라지지 않고 운영자가 개발 팀에 합류해야 하는 소프트웨어 개발 방법의 변화임
- IEEE 표준의 정의: 소프트웨어 개발과 관련된 이해관계자들이 소프트웨어 시스템 또는 서비스를 명세·개발·운영하고 시스템 전체 수명주기에 걸쳐 지속적인 개선을 도모하기 위하여 의사소통과 협업을 증진하기 위한 원리와 실천 사항들의 모임

Chaos와 DevOps

■ DevOps의 핵심 원리

- 자동화
 - DevOps는 개발 환경에서 나타나는 소프트웨어 동작이 배포 및 운영 환경에서도 동일하게 보장될 수 있도록 개발과정에서도 운영 환경을 고려
 - 운영 환경의 오류를 신속하고 일관성 있게 해결하기 위하여 자동화된 환경을 전체 수명주기에 구축
- 반복
 - 개발과 운영을 순환 구조로 연결
 - 유지보수라는 기존 개념보다는 운영 과정에서 모니터링된 요구사항을 시스템에 반영해 개발하는 반복 개발 형식으로 진행
- 자기 서비스
 - DevOps 조직은 협업 및 자동화 환경 구축을 통해 개발자와 운영자가 서로 방해하지 않고 독립적으로 일할 수 있도록 지원
 - 예를 들어, 개발자는 소프트웨어 시스템에 대한 운영 환경을 자체적으로 구축하여 신속한 개발 및 테스트를 수행
 - 운영자에게 개발 시스템 테스트를 위해 실환경을 요청하지 않아도 됨

Chaos와 DevOps

■ DevOps의 핵심 원리

- 지속적 개선
 - DevOps에서 지속적인 개선은 사용자 피드백을 소프트웨어 및 운영 환경을 개선하기 위해 활용하는 것
 - 피드백은 소프트웨어의 기능 향상, 기능 추가 및 삭제 등 근거가 될 뿐만 아니라 소프트웨어 성능이나 서비스 수준 만족을 위한 개선 근거
- 협업
 - 성공적인 DevOps는 신속하고 신뢰성있는 소프트웨어의 개발과 배포를 보장해야 함
 - 이를 위해 필요한 사안에 대하여 개발 팀과 운영 팀 간의 성공적이고 지속적인 협업 능력 요구
- 지속적 테스트
 - 개발자는 개발 혹은 수정된 코드에 대해 단위 테스트를 수행하고, 이 코드를 서버에 전달하여 통합 테스트 수행
 - 품질 보증 팀은 운영 환경과 동일한 환경에서 시스템 테스트를 수행하고, 운영 팀은 인수 테스트, 안정성 테스트 등을 수행
 - 이러한 테스트들은 지속적인 통합과 배포 과정에서 반복적으로 발생

Chaos와 DevOps

■ DevOps의 핵심 원리

- 지속적 통합과 지속적 배포
 - 다수 팀이 큰 규모의 소프트웨어를 개발할 때, 각 팀은 개발한 결과물은 서버로 전달
 - 서버에서는 팀에서 전달한 소프트웨어 컴포넌트를 주기적으로, 가능하면 매일 통합
 - 올바르게 통합된 컴포넌트는 품질 보증의 단계(테스트 단계 포함)를 거쳐, 생산 단계로 이동되며, 이후 사용자에게 배포
 - 통합이 주기적으로 이루어지듯 배포도 문제가 없는 한 지속적 수행

Chaos와 DevOps

■ DevOps 프로세스

- 소프트웨어 요구사항을 작은 단위의 개발 태스크로 분할하고, 이들에 대한 개발 및 배포 계획을 수립(Plan)한 후, 이 계획에 따라 단위 소프트웨어 컴포넌트를 개발(Code)
- 개발된 코드들은 상호 통합되어 배포 가능한 기능으로 구성(Build)되며, 이는 테스트(Test) 과정을 거쳐 Ops로 릴리즈(Release)

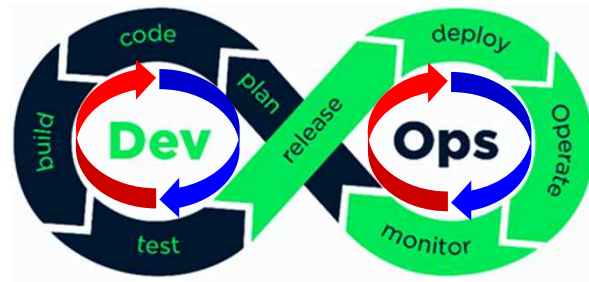



그림 3-14 DevOps(데브옵스) 프로세스

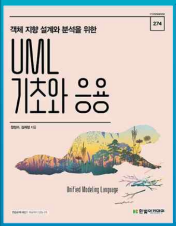



UNIFIED
MODELING
LANGUAGE™

이론교재

3장

소프트웨어 개발
프로세스+@
(PLC+SDLC)

0. PLC와 SDLC의 이해

1. 실현 가능성 분석

2. 전통적인 소프트웨어 프로세스

3. 애자일 및 XP 프로세스

4. 소프트웨어 프로세스 개선

5. 과제/진도/Q&A

한국과학기술대학교
KUSTECH

[이론 3장] 소프트웨어 개발 프로세스

48

CMM과 CMMI

■ CMM(Capability Maturity Model)

- 1986년 미 국방부가 방산 업체의 소프트웨어 개발 활동에 대한 데이터를 수집하고 이를 체계화하여 개발한 시스템 개발 프로세스의 성숙 모델
- 성숙도 Maturity: 임의로 수행해온 개발 활동을 공식적인 개발 단계, 단계별 활동에 대한 측정 및 평가, 프로세스 최적화 등의 관점에서 얼마나 잘하고 있는지 나타내는 척도
- 모델 CMM은 기존 소프트웨어 개발 프로세스 개선을 목적으로 함

■ CMMI(Capability Maturity Model Integration)

- 다양한 CMM 모델의 형식·용어·측정 방법의 차이점 존재
- 통합 적용의 어려움과 같은 단점을 해결하기 위한 시도 발생 -> CMMI 탄생

CMMI 성숙도 수준

- '무엇을 하는지'의 프로세스 기반 활동 영역과 '얼마나 잘 하는지'를 나타내는 능력도 범위로 표현

- 5개 단계의 성숙도 수준 정의

표 3-2 개발 CMMI의 수준별 프로세스 영역

수준	관심사	프로세스 영역
5. 최적화 수준 (Optimizing)	지속적인 프로세스 개선	CAR - Causal Analysis and Resolution OPM - Organizational Performance Management
4. 정량적 관리 수준 (Quantitatively managed)	정량적 자료 관리	OPP - Organizational Process Performance QPM - Quantitative Project Management
3. 정의 수준 (Defined)	표준화된 프로세스	DAR - Decision Analysis and Resolution IPM - Integrated Project Management OPD - Organizational Process Definition OPF - Organizational Process Focus OT - Organizational Training PI - Product Integration RID - Requirements Development RSKM - Risk Management TS - Technical Solution VAL - Validation VER - Verification
2. 관리 수준 (Managed)	기본적인 프로젝트 관리	CM - Configuration Management MA - Measurement and Analysis PMC - Project Monitoring and Control PP - Project Planning PPQA - Process and Product Quality Assurance REQM - Requirements Management SAM - Supplier Agreement Management
1. 수행 수준 (Performed)	초기 소프트웨어 개발	해당 없음

CMMI 구성 요소

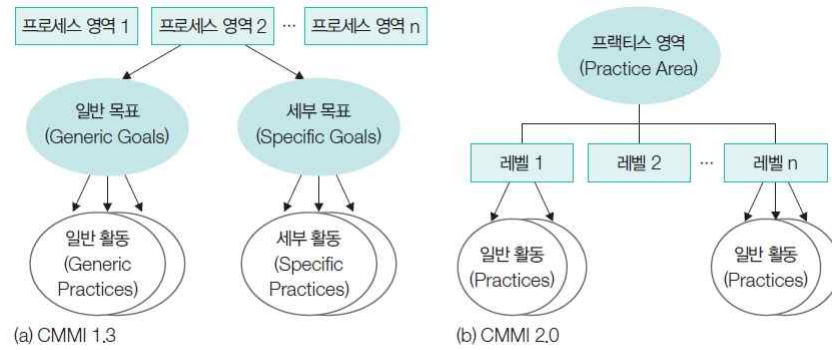
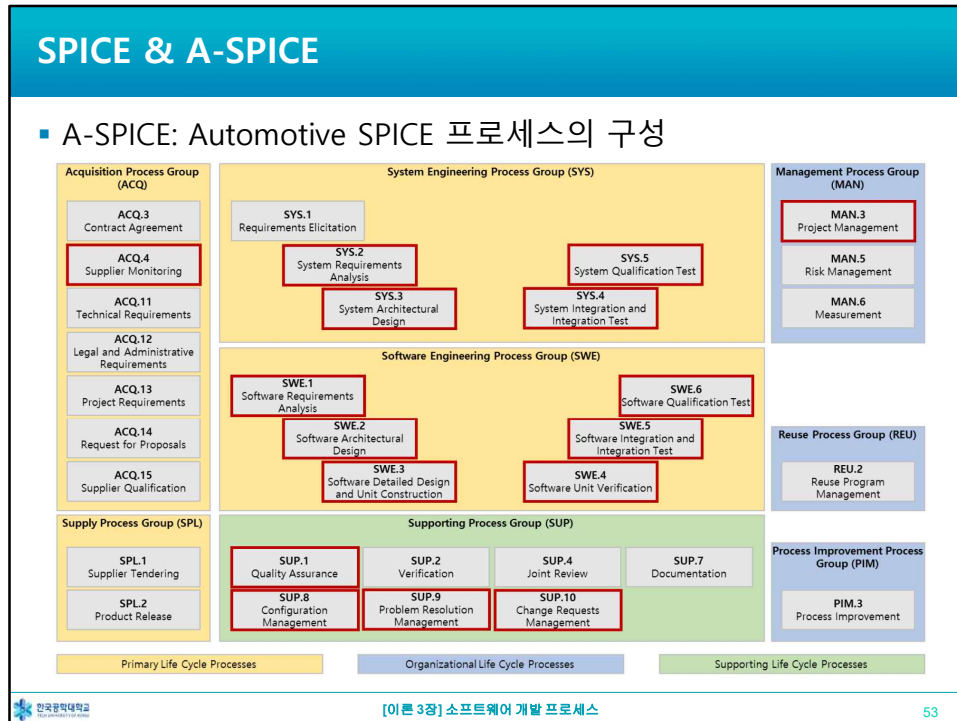


그림 3-15 CMMI 프로세스 영역의 구성 요소

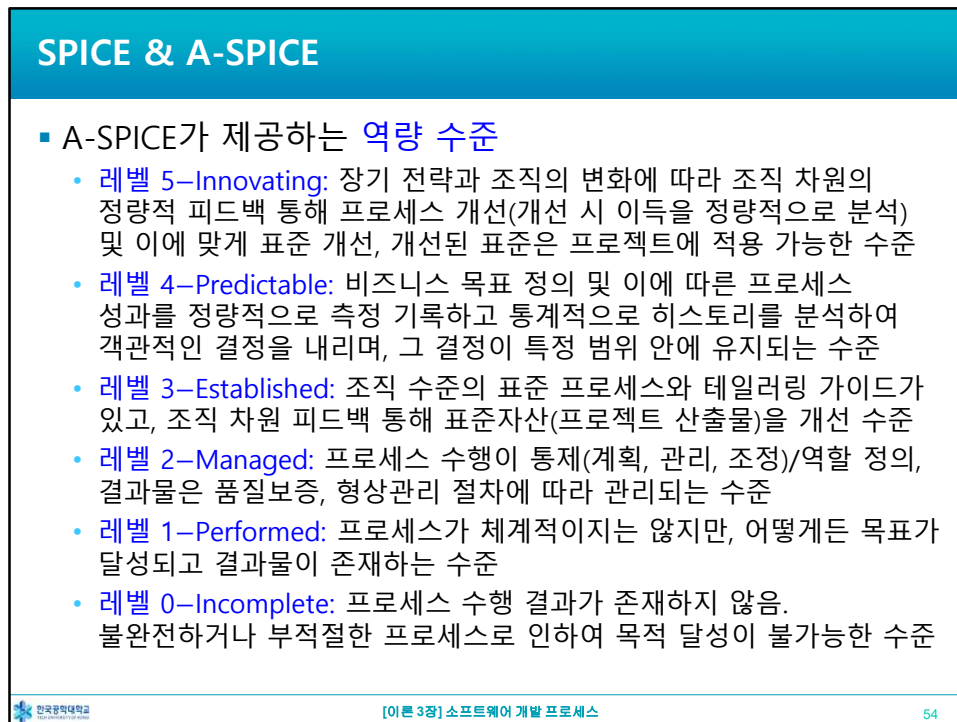
SPICE & A-SPICE

■ SPICE(Software Process Improvement and Capability dEtermination)

- 소프트웨어 개발 프로세스의 개선을 목적으로 제정된 ISO 15504 표준의 별칭
- 6개 단계의 성숙도 수준 정의
 - 레벨 5-최적화 프로세스
 - 레벨 4-예측 가능한 프로세스
 - 레벨 3-구축된 프로세스
 - 레벨 2-관리되는 프로세스
 - 레벨 1-적용하는 프로세스
 - 레벨 0-불완전한 프로세스
- 15504 표준에서 정의하는 프로세스 속성들: (1) 프로세스 성능, (2) 성과 관리, (3) 산출물 관리, (4) 프로세스 정의, (5) 프로세스 배포, (6) 프로세스 측정, (7) 프로세스 제어, (8) 프로세스 혁신, (9) 프로세스 최적화



53



54

[실습 3-2] A-SPICE로 생각해보기

실습 3-2

A-SPICE 생각해보기(자동차 소프트웨어의 개발이 중요한가?)

자동차가 오동작한다면 어떠한 일이 발생할까? 고속도로 주행 중에 핸들 조작이 안 된다면 이는 상상만으로도 끔찍하다. 최근 자동차에 탑재되는 소프트웨어를 개발하는 국내 기업은 물론 국제적인 자동차 회사에서도 자동차 개발 과정을 체계화함으로써, 고장 없는 소프트웨어 개발에 많은 관심과 투자가 진행되고 있다. 이에 따라 자동차와 관련된 소프트웨어 개발 기업들에서는 A-SPICE 인증을 추진하고 있다. 이와 관련하여 다음 물음을 생각해보고 답을 적어봅시다.

- (1) A-SPICE 인증을 받으려면 적지 않은 자금을 투자해야 한다. 그럼에도 불구하고 A-SPICE 인증을 받으려는 이유는 무엇인가?
- (2) A-SPICE 레벨이 향상된다는 것은 무엇을 의미하는가? 구체적인 예를 들어 설명하라.

식스 시그마(6-SIGMA, 6-σ)

■ 식스 시그마(6-SIGMA, 6-σ)

- 프로세스 개선을 위한 공학적 기술 및 지원 도구를 제공하는 **프로세스 관점의 접근 방법론**
- 1986년 미국 모토로라의 엔지니어, 빌 스미스(Bill Smith) 제시
- 결함 원인의 식별 및 제거, 기존의 제조 및 비즈니스 프로세스의 변경 최소화, 공학적 활동의 품질을 개선하는 것이 목표
- **식스 시그마 도입을 위한 개선 목표들**
 - 공정 주기 단축
 - 결함 및 문제 감소
 - 비용 감소
 - 고객 만족도 향상
 - 이익 증가

기타 프로세스 표준

- **ISO 12207:** Systems and software engineering-Software life cycle processes
 - **합의 프로세스:** 구매자와 공급자 간의 계약 수립과 관련된 프로세스를 포함하며, 세부 프로세스로 Acquisition 프로세스와 Supply 프로세스가 있음
 - **조직 프로젝트 지원 프로세스:** 조직에서 시스템 개발 및 관련 프로젝트를 시작, 통제 및 관리, 지원하기 위해 사용될 수 있는 프로세스 영역. 수명주기 모델 관리 프로세스, 인프라구조 관리 프로세스, 포트폴리오 관리 프로세스, 품질 관리 프로세스 등으로 구성
 - **기술 관리프로세스:** 프로젝트 계획, 프로젝트 평가, 의사결정, 리스크 관리, 형상 관리, 품질 보증과 같은 활동을 정의하는 프로세스들로 구성
 - **기술 프로세스:** 소프트웨어 개발과 직접적으로 관련된 프로세스. 요구사항 정의, 아키텍처 정의, 시스템 분석, 구현, 통합, 검증, 운영 유지 등의 프로세스를 포함

기타 프로세스 표준

- **ISO 29110:** Systems and Software Life Cycle Profiles and Guidelines for Very Small Entities(VSEs)
 - Planning, Execution, Evaluation, Closure의 활동으로 구성되는 **프로젝트 관리 프로세스**와 Initiation, Analysis, Design, Construction, Integration and test, Delivery 활동으로 구성되는 **구현 프로세스 영역으로 구분**
 - Entry 테스트, Basic 태스크, Intermediate 태스크, Advanced 태스크로 구분해 **선택적으로 태스크를 수행할 수 있도록 함**
 - 프로젝트 관리 프로세스(117개), 구현 프로세스(164개)의 태스크 정의
 - 외부에서 제품을 구매하는 작은 기업을 위해 **획득 관리 프로세스와 이관 프로세스를 보조적으로 제시**

기타 프로세스 표준

- **ISO 15288:** Systems and software engineering-System life cycle processes
 - **합의 프로세스:** ISO 12207과 동일하게 획득 프로세스와 공급 프로세스로 구성
 - **조직 프로세스:** 조직 환경 관리 프로세스, 투자 관리 프로세스, 시스템 수명주기 프로세스 관리 프로세스, 자원관리 프로세스, 품질 관리 프로세스 등으로 구성
 - **프로젝트 프로세스:** 프로젝트 계획, 평가, 관리, 의사결정, 위험 관리, 형상 관리 등과 같은 프로세스로 구성
 - **기술 프로세스:** 요구사항 정의, 요구사항 분석, 아키텍처 설계, 구현, 통합, 검사 및 검증, 운영, 유지보수 등의 프로세스로 구성

[이론 3장] 소프트웨어 프로세스 요약

- 소프트웨어 프로세스 **필요성**
 - 요구사항 만족도 향상 -> 수정 및 재개발 절감(비용 절감) -> 품질 관리 가능 -> 프로젝트 성공율을 향상
- **전통적인** 소프트웨어 프로세스 **모델**
 - 폭포수 모델, 점진적 모델, 프로토타입 모델, 나선형 모델, V 모델 등
 - 각 프로세스 모델의 장단점
- **새로운** 소프트웨어 프로세스 **모델: 애자일과 DevOps**
 - **애자일**
 - **XP 프로세스:** 소프트웨어 개발의 반복적인 개발 활동 중심(10가지 실천사항)
 - **스크럼 프로세스:** 반복 개발의 관리적 접근 방법
 - **DevOps:** 신속한 개발 및 배포를 위해 **개발과 운영의 통합**
- 소프트웨어 프로세스 **개선**
 - 프로세스 개선 모델: CMM, CMMI, SPICE, A-SPICE, 6 시그마 등
 - 프로세스 표준: ISO 12207, ISO 29110, ISO 15288

[프로젝트 III] 프로세스 모델 선정하기

프로젝트 III


프로세스 모델 선정하기

1장에서 정의한 프로젝트 정의서와 2장에서 식별한 핵심 품질 요소를 중심으로 대상 소프트웨어를 개발하기 위한 프로세스 모델을 선택 혹은 정의합니다. 이 결정 사항은 추후 프로젝트 관리 계획서 작성에 반영됩니다.

(1) 개발 대상 시스템에 대한 특성 정리하기

- 대상 시스템과 유사한 시스템이 기존에 존재하는가?
- 대상 시스템의 핵심 기능에 신규성과 복잡성이 높은가?
- 대상 시스템의 규모는 대략 어느 정도인가?
- 개발 과정에서 새로운 기술이 적용되어야 하는가?

(2) 적용 가능한 프로세스 모델 선정하기

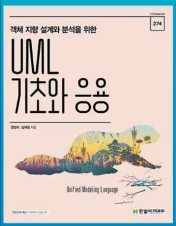



UNIFIED
MODELING
LANGUAGE™

이론교재

3장

소프트웨어 개발
프로세스+@
(PLC+SDLC)

0. PLC와 SDLC의 이해


1. 실현 가능성 분석

2. 전통적인 소프트웨어 프로세스

3. 애자일 및 XP 프로세스

4. 소프트웨어 프로세스 개선

5. 과제/진도/Q&A



한국과학기술대학교

[이론 3장] 소프트웨어 개발 프로세스

62

5. 과제/진도: [과제#2-1] 팀 편성/역할 분장+프로젝트 주제 도출

■ [과제#2-1] 팀 편성/역할 분장(R&R)+프로젝트 주제 도출(2점, 팀별 제출)

- 작성/제출 방법: 제시한 작성양식(샘플)을 팀별로 최적화해서 작성(팀별, HWP/WORD 사용), 작성 후 pdf 파일로 변환해서 LMS(e-class)에서 제출기한 이내에 업로드
- 내용/분량: 제한 없음
- 제출 파일명
 - [과제#2-1]-게임공학과-2분반X조-팀편성+프로젝트 주제도출-1차-20240925
- 제출기한: 2024.09.25.(수요일), 24시 이전까지

5. 과제/진도: 3주차/전체

주차	강의 내용	수업 유형	학습 활동
1	공통0장 강의안내+이론1장 SE개요+GSE 과목 사전 설문지 작성	대면수업(이론/실습)	대면수업/실습, 과제 해결
2	이론2장 SW 품질+실습1장 UML 이해+12장 starUML 모델링도구 설치 및 사용법+피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
3	이론3장 SW 개발프로세스+실습2장 UML 구성요소/뷰+프로젝트 팀편성/주제 도출/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
4	이론4장 SW 개발방법론(DevOps+UP)+실습3장 유스케이스 다이어그램+문제기술서(SOP) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
5	국경일(개천절) 휴강(15주차 보장)	국경일 휴강	국경일 휴강
6	이론5장 프로젝트 관리+실습4장 클래스 다이어그램+프로젝트정의서(PC) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
7	이론6장 SW 비용산정+실습5장 순차 다이어그램+프로젝트관리계획서(PMP) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
8	이론7장 요구사항 도출+실습6장 통신 다이어그램+요구사항정의서(SRD)/중간발표(PT+PMR) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
9	중간고사(필기+개인)+프로젝트 중간발표(PT+PMR+팀별)/피드백	대면수업(시험/발표)	서술형 필기시험/구두발표
10	이론8장 객체지향 분석+실습7장 활동 다이어그램+요구사항추적표(RTM) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
11	이론9장 모듈화 설계+실습8장 상태 다이어그램+1. 요구사항명세서(SRS) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
12	이론10장 설계 패턴+이론11장 객체지향 설계+실습9장 컴포넌트 다이어그램+설계기술서(SDD) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
13	이론12장 인스펙션+이론13장 코딩+실습10장 배치 다이어그램+구현계획서(SIP) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
14	이론14장 화이트박스 테스트+이론15장 블랙박스 테스트+실습 11장 패키지 다이어그램+시험계획서(STP)/시험설계서(STD) 작성/피드백	대면수업(이론/실습)	대면수업/실습, 과제 해결
15	이론16장 SW 개발 적용 기술+실습12장 깃과 깃허브 활용 방법+구현결과서(SIR)/시험결과서(STR)/최종발표(PT+PCR) 작성/피드백	대면수업(이론/실습) (5주차 보장)	대면수업/실습, 과제 해결 (5주차 보장)
16	기말고사(L&L+개인)+프로젝트 최종발표(PT+PCR+팀별)/피드백+최종 종료보고서(PCR) 제출	대면수업(시험/발표)	서술형 필기시험/구두발표

5. 과제/진도: 3주차(결과)-4주차(계획)

3주차 강의 진행 결과

주차	주요학습내용	학습성과 학습목표	수업운영방법	학습준비사항	교재, 참고도서 (page)
3주차	<ul style="list-style-type: none"> 이론 3장 SW 개발 프로세스 실습 2장 UML 구성요소/뷰 팀 편성/구축+프로젝트주제 선정(주제 선정/개요 작성) 	<ol style="list-style-type: none"> SW 개발프로세스의 이해 UML 구성요소와 뷰의 이해 팀 편성 및 구축방법, 주제선정 방법/개요 작성 이해 	<ul style="list-style-type: none"> 대면강의+실습 [과제#2-1] 팀 편성/역할 분장+프로젝트 주제 도출 	교재 준비(이론, 실습) 및 이론 3장/실습 2장 읽어 보기	강의계획서+이론/실습 교재/참고도서+강의자료
4주차	<ul style="list-style-type: none"> 이론 4장 소프트웨어개발 방법론 (DevOps+UP 방법론) 실습 3장 유스케이스 다이어그램 프로젝트 주제에 대한 문제기술서 (SOP) 작성(1) 	<ol style="list-style-type: none"> 소프트웨어개발 방법론(DevOps+UP 방법론) 이해 유스케이스 다이어그램 작성방법 이해 문제기술서(SOP) 작성방법 이해(1) 	<ul style="list-style-type: none"> 대면강의+실습, [과제#3-1] 문제기술서(SOP) 작성 제출1 	교재 준비(이론, 실습) 및 이론 4장/실습 3장 읽어 보기	강의계획서+이론/실습 교재/참고도서+강의자료

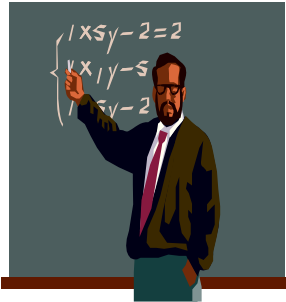
5. 과제/진도: [진도] 4주차 계획

- [강의계획서] 내용을 잘(정확히) 숙지하고, 매주 강의 진도 확인하기
- [과제#2-1] 팀 편성/역할 분장+프로젝트 주제 도출
- [강의교재] 이론 4장+@, 실습 3장 읽어보기

📌 4주차: 이론4장+@ SW 개발방법론(DevOps+UP)+실습3장
유스케이스 다이어그램+문제기술서(SOP) 작성/
피드백

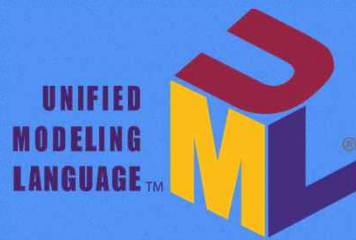
❑ 궁금하면 질문하자!

- **LMS의 질의응답, 댓글, 쪽지 및 메일 기능을 적극 활용하기 바랍니다!**



학습(學習)은 배우고 익히라는 것이다.
배우기만 힘쓰면 스스로 할 수 없는 사람이 된다!
항상 배우고 익혀야 한다!

Q & A



[GSE 3주차-1] 수고했습니다!
다음 4주차에도 반갑게 만납시다~~

본 수업에 사용된 일부 자료, 영상물 등은 강의 내용을 보충하기 위해 교육 목적으로 활용하였습니다. 본 강의 자료 및 영상물의 불법적 이용, 무단 전재 및 재배포는 법적으로 금지되어 있으니, 학생 여러분은 학습 이외 용도로 사용을 주의하기 바랍니다.