[소프트웨어 공학]

Part1. 소프트웨어 공학 개요

Ch01. 서론

[소프트웨어 공학]

▶소프트웨어

-기반 시설이 컴퓨터 기반 시스템으로 제어

-많은제품에 컴퓨터와 제어 SW를 탑재

-제조, 유통, 금융 등 분야의 컴퓨터화

-소프트웨어는 추상적이며 무형.

물리적 제약이 없으므로 시스템이 되려 극도로 복잡해질 수 있음

▶다양한 종류의 소프트웨어 시스템

-단순한 임베디드 시스템부터 복잡한 정보 시스템까지다양한 소프트웨어 시스템 존재

-소프트웨어 시스템 종류에 따라 다른 접근법 필요

-소프트웨어 공학에는 보편적 표기법, 방법, 기법 존재 X

[소프트웨어 프로젝트 실패 요인]

⇒ 시스템 복잡도 증가

-더 크고 복잡한 시스템 구축 가능해짐에 따라 계속 '요구사항이 변함'

-시스템 더 신속하게 공급해야 함. 더 크고 복잡한 시스템이 필요함

-이전에 불가능하게 여겨지던 새로운 기능을 가져야 함

⇒ 소프트웨어 공학 방법론 사용의 실패

-소프트웨어 공학 방법과 기술 사용 없이도 프로그램 작성 쉬움

-많은 기업들이 제품과 서비스 진화시키면서 동시에 SW 개발함

-일상 업무에 SW 공학 방법을 사용 X

-결과적으로 SW는 더 비싸지고, 안정성, 신뢰성 부족하게 됨

[소프트웨어 공학의 역사]

-1968년 소프트웨어 위기(software crisis), 70-80년대 본격적 개발

-구조적 프로그래밍, 정보 은닉, 객체지향 개발, 표준 표기법과 도구

1.1 전문적 소프트웨어 개발

소프트웨어란 ?
: 컴퓨터 프로그램 + 관련 문서들까지 포함

좋은 소프트웨어의 특성 : 유지보수성 (maintainability) / 수용성(acceptability)
확실성(dependability) / 사용성(usability) / 보안성(security)

소프트웨어 공학 ?
초기 구상 단계부터 운영. 유지보수까지 포함하는 SW 생산의 모든 관점과 연관된 공학 분야

소프트웨어 공학 기본 활동 ***
명세(specification)/ 개발(development) / 검증(validation) / 진화(evolution)

소프트웨어 공학과 컴퓨터 과학의 차이
-컴퓨터 과학 : 이론/기본
-소프트웨어 공학 : 유용한 SW 개발하고 인도하는 '실무적 내용'과 관련

소프트웨어 공학이 직면한 주요 도전 과제
-증가하는 다양성, 인도기간 단축 요구, 신뢰할 수 있는 SW 개발에 대응하는 것

소프트웨어 공학 비용 : 개발 비용 60% + 테스팅 비용 40%
- 맞춤 소프트웨어의 경우 진화비용 > 개발 비용 넘어서기도 함

[소프트웨어 제품]

1. 일반 제품
   -수평적(horizontal) 제품 : 응용분야에 관계없이 사용되는 애플리케이션
   -수직적(vertical) 제품 : 특정 응용분야를 위한 애플리케이션
2. 맞춤식 소프트웨어

[좋은 소프트웨어의 필수적 특성]

⇒ 수용성 (acceptability)
-SW는 해당 사용자에 의해 수용 가능해야 함
-이해 쉽고 사용 쉽고 기존 시스템과 호환성 있어야 함

⇒ 확실성 (dependability)과 보안성(security)
-신뢰성. 보안성, 안정성을 포함하는 여러 특성을 의미
-확실성 있는 SW는 시스템 장애 시에도 물리적/경제적 손실 야기 X
-악의적 사용자가 시스템에 접근하거나 피해 줄 수 없도록 해야 함

⇒ 효율성 (effciency)
-SW는 메모리. CPU 등 시스템 자원 낭비해서는 안된다.

⇒ 유지보수성(maintatinability)
-소프트웨어는 고객 변경 요구에 맞도록 진화. 변화하는 비즈니스 환경에 맞게 작성

[소프트웨어 공학]

▶소프트웨어 공학

-초기 시스템 명세 작성부터 시스템 사용.유지보수되는 것까지 소프트웨어 제품화와 관련된 모든 관점을 다루는 공학 분야

▶소프트웨어 생산.개발의 모든 관점

[소프트웨어 공학의 중요성]

-개인과 사회가 SW 시스템에 더 많이 의존한다.

-전문적 SW 개발을 위해 소프트웨어 공학 기법 사용하는 것이 보통 비용 적게 든다.(장기적)

[소프트웨어 프로세스 기본 활동] : 공통적인 활동 (*****)

소프트웨어 명세화(specification)	(요구사항 분석) 고객과 개발자가 소프트웨어의 기능.운영 제약조건을 정의
소프트웨어 개발(development)	(설계. 구현) 소프트웨어를 설계하고 프로그램 작성
소프트웨어 검증(validation)	(테스팅) 소프트웨어가 고객이 요구하는 것이 맞는지 확인
소프트웨어 진화(evolution)	(유지보수) 고객/시장 요구에 따라 SW 수정

Ch02. 소프트웨어 프로세스

[소프트웨어 프로세스]
-소프트웨어 시스템을 개발하는데 관련된 활동의 집합
-다양한 유형의 시스템 개발에 적용 가능한 '보편적' 소프트웨어 공학 기법 존재 X

[기본적인 소프트웨어 공학 활동]
-소프트웨어 명세화(specification) : 요구사항 분석
-소프트웨어 개발(development) : 설계/구현
-소프트웨어 검증(validation) : 테스팅
-소프트웨어 진화(evolution) : 유지 보수

[프로세스 설명에 포함되어야 하는 것]
- WHAT(제품/산출물), WHO(역할), WHEN(순서/절차), HOW(기법/노하우/방법론)
  1. **제품(product), 산출물(deliverable/artifect)**
  - 프로세스 활동의 결과물
  - 명세화, 설계, 구현, 테스팅
  2. **역할(role)**
  - 프로세스에 참여하는 사람들의 책임
  - 프로젝트 관리자, 형상(버전) 관리자, 프로그래머
  3. **사전/사후 조건(pre/post condition)**
  - 프로세스 활동이 이루어지거나 제품 만들어지는 전후 만족되어야 하는 조건

[소프트웨어 프로세스 종류]
-대부분의 실제 프로세스는 계획주도와 애자일 접근법을 둘 다 포함함
▶['계획주도' 프로세스] : plan-driven
-모든 프로세스 활동을 미리 계획하여 계획 대비 실적을 측정
-안정성 O
▶['애자일' 프로세스] : agile
-계획을 점증적으로 세우고 고객 요구 반영하여 프로세스 간단히 변경
-유연성 O/ 쉽게 변경 O

2.1 소프트웨어 프로세스 모델들

[소프트웨어 프로세스 모델]

-실제 대부분의 대형 시스템 개발 프로세스는 이들 모델의 요소를 포함

폭포수 모델 (waterfal)	-계획주도 프로세스 -명세화/개발/검증/진화를 개별적 프로세스 단계로 구분함
점증적 개발 (incremental)	-계획주도 or 애자일 -명세화/개발/검증 활동이 '중첩'. 반복됨 -시스템은 이전 버전과 비교하여 기능을 추가하면서 일련의 증분(incremental)으로 개발함
통합 및 설정 (integration and configuration)	-계획주도 or 애자일 -'재사용' 할 수 있는 기존의 부품 설정 통합하여 개발

▶ [폭포수 모델]

-폭포수 모델은 **'단계'로 구분**됨

요구사항 분석 및 정의	-시스템의 서비스, 제약조건, 목표를 설정하여 '문서화' (시스템 명세서 작성)
시스템 및 소프트웨어 설계	-시스템 아키텍처(전체 구조) 수립, -소프트웨어 구성요소 추상화와 이들 간의 관계
구현 및 단위 테스팅	-소프트웨어 설계를 프로그램으로 실체화 -단위 테스팅에서 각 단위가 명세에 맞는지 확인
통합 및 시스템 테스팅	-프로그램 단위를 통합하여 완전히 시스템으로서 테스트하고 요구사항 충족 여부 확인
운영 및 유지보수	- 시스템 설치되고 사용됨 - 오류수정, 구현을 개선, 새로운 요구사항 반영

<폭포수 모델의 특징>

-원칙적으로 각 단계의 결과로 하나 이상의 산출물(문서) 나와야 함

-이전 단계 끝나기 전까지 다음 단계 시작X

<실제 소프트웨어 개발은 각 단계가 중첩됨>

-한 단계(구현/설계)에서 다르 단계(설게/요구사항)로의 피드백이 일어남

-이전 단계의 문서를 수정하게 되면 프로세스 지연 발생

-추가 변경 반영하지 않도록 명세서 확정

-변화하는 고객 요구사항에 대응 어려움

<폭포수 모델 적용 가능 분야>

-변경 적고, 요구사항 이해도가 높으며 요구사항 변경 제한된 분야

  -임베디드 시스템 :하드웨어와 연동해야 하므로 구현 시까지 의사 결정 못 미룸

  -중대한 시스템 : 명세 설계에 대한 분석, 검토가 필요함

  -대규모 소프트웨어 시스템: 여러 회사, 장소에서 개발하므로 완성된 명세서가 필요

<폭포수 모델 적합 X 예>

-자유로운 팀 커뮤니케이션 가능. 요구사항 빠르게 변화

-> 이 경우는 반복적 개발/애자일 방법이 더 적합

▶ **[점증적 개발] : incremental development**

-초기구현 -> 사용자 피드백 받아서-> 여러 버전을 거쳐 진화하는 방식으로 요구한 최종 시스템을 개발하는 것

-명세/개발/검증 명확 구분X, 각 단계가 중첩되어 있음 O

-요구사항 쉽게 변하는 시스템 개발에 적합. 변경에 대처 쉬움

-시스템의 각 증가분(버전)은 고객이 필요로 하는 기능 일부를 포함

-초기증가분은 가장 중요하거나 긴급하게 요구되는 기능을 포함시켜놓음

<장점>

-요구사항 변경 구현 비용 줄어듬

-이미 개발된 내용에 대해 고객 피드백 받기 쉬움

-모든 기능 개발 전에 유용한 소프트웨어를 고객에게 전달하여 설치, 사용 가능하게 함

<문제점>

-프로세스가 가시적이지 않음

-새로운 증가분 추가됨에 따라 시스템 구조 저하되는 경향 있음

-대규모 시스템 개발에는 부적합

2.2 프로세스 활동

[프로세스 활동]

-실제 소프트웨어 프로세스는 명세, 설계, 구현, 테스팅 등의 활동이 중첩되어 있음

-기본 프로세스 활동 : 명세화/개발/검증/진화

-개발 프로세스마다 기본 활동이 다른 방식으로 구성됨

  -폭포수 모델은 순차적으로 구성

  -점증적 모델에서는 중첩되어 있음

[1. 소프트웨어 명세화]

▶[요구 공학] (requirements engineering) : 요구사항 도출해내는 과정

-시스템이 제공해야 하는 서비스를 이해하고 정의하며 시스템의 운영과 개발에 대한 제약 조건을 찾아내는 프로세스

-요구공학 이전에 **타당성 조사(할까 말까) 나 마케팅 조사 수행할 수 있음

▶[요구 공학 프로세스 주요활동]

요구사항 도출과 분석 (requirements elicitation and analysis)	-기존 시스템 관찰, 사용자와 토의, 업무 분석 등을 통해 요구사항 도출
요구사항 명세화 (requirements specification)	-요구사항 문서 작성(사용자 요구사항, 시스템 요구사항)
요구사항 검증 (requirements validation)	-요구사항의 현실성/일관성/완정성 검사

[2. 소프트웨어 설계 및 구현]

▶소프트웨어 설계/구현

-시스템 명세로부터 **실행 가능한 시스템으로 변환하는 프로세스** (What 으로부터 How)

[설계 프로세스 활동] : Design activities

▶아키텍처 설계

-시스템의 전체적 구조

-주요 컴포넌트(서브 시스템)와 이들 간의 관계를 식별

▶데이터베이스 설계

-시스템 데이터 구조 설계. 기존 데이터베이스 사용하는 경우를 고려

▶인터페이스 설계

-컴포넌트 간 인터페이스 정의

-인터페이스 명세 있으면 컴포넌트 개별적 설계 가능

▶컴포넌트 선택 및 설계

-재사용 가능 컴포넌트 찾음

-새로운 컴포넌트를 설계

[시스템 설계]

-설계와 구현 중첩되는 경우 多

-프로그래밍은 개별적 활동

-테스팅과 디버깅

: 개발한 코드는 단위 테스팅 필요

: 테스팅으로 결함의 존재 확인

: 디버깅은 결함의 위치 찾아 수정 작업

[3. 소프트웨어 검증]

▶[검증 및 확인] : V&V (verification & validation)

-시스템이 명세 준수하는지(검증), 시스템이 고객 기대 충족하는지(확인)

-테스팅은 주요한 검증 기법

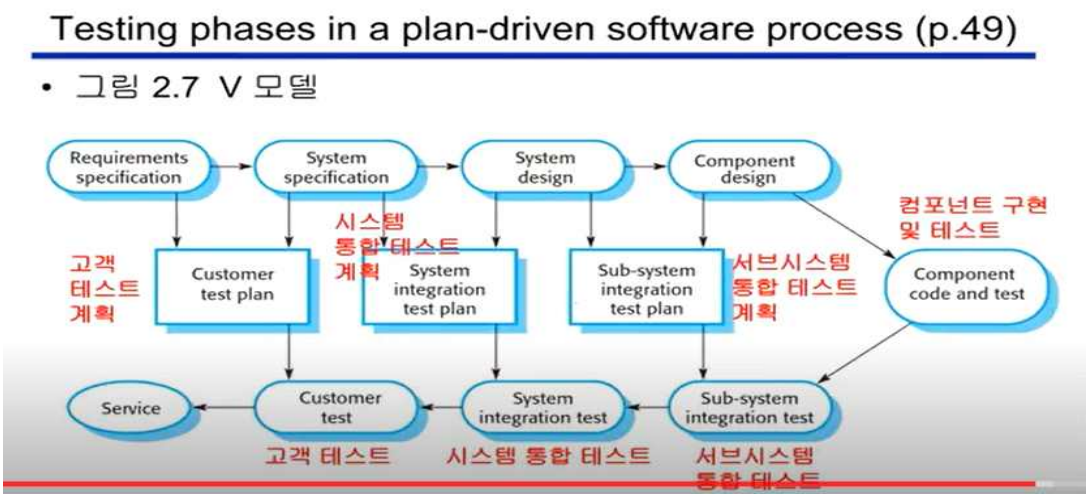
▶[테스팅 프로세스 단계]

1. 컴포넌트 테스팅(=단위 테스팅. Unit 테스팅)
   -개별 컴포넌트(함수,객체 등)를 테스팅
2. 시스템 테스팅
   -컴포넌트를 통합하여 시스템 구성하고 이를 테스팅
   -컴포넌트 간 예기치 못한 상호작용과 인터페이스 문제 등을 찾음
   -기능적/비기능적 요구사항 만족 여부, 창발적 시스템 속성을 테스트
3. 고객 테스팅
   -고객의 실제 데이터를 이용하여 테스트

[V 모델]

-테스트 계획이 테스팅과 개발 활동 사이를 어떻게 연결하는지 보여줌

-소프트웨어 검증 활동이 폭포수 모델의 각 단계에 대응되는 것을 알 수 O



[4. 소프트웨어 진화] : Software evolution

[소프트웨어 진화]

▶소프트웨어의 유연성(flexibility)

-크고 복잡한 시스템이 만들어짐

-소프트웨어 변경은 하드웨어 변경에 비해 쉬움

▶요구사항이 변화하므로 소프트웨어도 진화하고 변경되어야 함

▶개발과 유지보수의 구분이 줄어들고 있음

-완전히 새로 만든 소프트웨어 시스템은 거의 X

-요구사항 변화를 지속적으로 반영하면서 소프트웨어가 진화

2.3 변경 처리

-소프트웨어 프로젝트에서 변경은 불가피

-재작업은 개발 비용을 증가시키므로 재작업 비용 줄이는 것 필요

<변경 처리/시스템 요구사항 변화를 위한 2가지 방법>
1. 시스템 프로토타이핑
2. 점증적 인도

[1. 프로토타이핑]

▶[프로토타입] : prototype

-특수 목적을 가진 소프트웨어 시스템의 초기 버전 (시연용)

-아이디어 시연, 설계 옵션 시도, 문제점과 해결책 찾음

▶[프로토타입 용도]

⇒ **요구 공학 프로세스**

-요구사항 도출과 검증에 도움

-프로토타입 사용해보며 아이디어 얻음

⇒ **설계 프로세스**

-설계 문제에 대한 해결책 찾음. 설계의 타당성 확인 위해 실험

-사용자 인터페이스 개발

[2. 점증적 인도]

-점증적 인도 : 개발을 마친 증가분 일부를 고객에게 전달함으로써 고객의 업무 환경에서 사용할 수 있도록 배포하는 것.

---------- ----------

2.4 프로세스 개선

***프로세스 개선: 기존 프로세스 이해 후 이 프로세스를 바꾸어 제품 품질 향상시키거나 개발 시간 단축시키는 활동**

[프로세스 개선/변경 기법]

1. 프로세스 성숙도 접근법
2. 애자일 접근법

[프로세스 성숙도 모델 수준]

1. 초기(initial)
2. 관리(managed)
3. 정의(definde)
4. 정량적 관리(quantitatively managed)
5. 최적화(optimizing)

Ch03. 애자일 소프트웨어 개발

▶신속한 개발과 인도(delivery) 필요
-다수 비즈니스 시스템에서 신속한 개발과 인도가 중요함
-변화하는 비즈니스 환경에 따라 안정적인 요구사항 얻기 힘듬
-시스템 경험 후 요구사항 명확히 알게 되지만 요구사항은 계속 바뀜

▶계획 주도 프로세스 :신속한 소프트웨어 개발에 부적합
-이 프로세스는 안정성 중심 시스템. 완벽한 분석 필요한 경우 적합

▶애자일 기법의 등장
-익스트림 프로그래밍 (Extreme Programming) : XP 개발
-스크럼(Scrum) : 관리

[애자일 기법의 특징]
-명세, 설계, 구현이 중첩되며 설계문서 최소화
-시스템을 연속적인 증분으로 구현.
-최종 고객 사용자 + 이해당사자가 증분의 명세화와 평가에 참여
-보통 2-3주마다 새로운 증분을 인도
-테스트 자동화 도구, 형상관리 도구, UI 생성 도구 등 여러 도구를 활용
-(고객) 커뮤니케이션 활성화, 문서 최소화

[계획주도 VS 애자일 비교]
▶계획주도 개발
-프로세스 단계별로 해당하는 산출물 생성하도록 각 단계를 나눔.
-각 단계의 산출물이 다음 단계에서 사용됨
-활동 별로 반복 이뤄짐
▶애자일 개발
-**설계와 구현**을 프로세스 중심 활동으로 두고 요구사항, 테스팅 등을 설계와 구현에 포함시킴
-요구사항과 설계를 따로 다루지 않고 함께 발전시킴

3.1 애자일 기법

-계획주도 접근법은 요구사항/설계 등의 '문서화'에 중점
-대규모로 오래 지속되는 소프트웨어 시스템 개발에 적합
-중소 규모 비즈니스 시스템을 개발하기에 오버헤드 큼
-요구사항 바뀌면 명세, 설계, 프로그램을 함께 변경해야 함

[애자일 기법]

-1990년대 말 등장
-개발팀이 설계, 문서 작업보다는 소프트웨어 자체에 더 집중 O
-요구사항 자주 변경되는 애플리케이션 개발에 적합
-반복적 개발
-고객이 작동하는 소프트웨어 빨리 받아 요구사항 도출에 기여

[애자일 선언]

-프로세스의 도구 < 개인과 상호작용
-문서 < 작동 소프트웨어
-계약 협상 < 고객과의 협업
-계획 < 변화에 대응

[애자일 기법 원칙들]

고객 참여
변화 수용
점증적 인도
단순성 유지
프로세스가 아닌 '사람'

[애자일 기법 적용]

-소프트웨어 회사가 중소규모의 제품 개발 시.
-고객이 개발 프로세스에 참여 의사O 참여 가능O
-외부 이해당사자나 규제 없는 조직 내에서 이뤄지는 맞춤형 개발
-같은 장소에 있는 팀으로 구성 (밀접/신속 의사소통 O)

3.2 애자일 개발 기법

[Extreme programming] XP : '개발' 기법

-극단적으로 반복적인 접근법

-하루에 여러 개 버전. 2주마다 증분을 인도

-요구사항을 사용자 스토리라는 **'시나리오'로 표현.** 사용자 스토리를 태스크로 나누어 표현

-프로그래머는 '짝'으로 개발하고 **코드 작성 전 테스트를 먼저 작성**

-새로운 코드를 시스템에 통합하려면 **모든 테스트를 통과**해야 함

(기존 테스트 자동화 + 추가 테스트)

-시스템 자주 배포함. 시스템 배포 간격이 짧음

[XP 실무]

공동 소유권	모든 개발자는 모든 코드에 책임 O
연속적 통합	작업 끝나자마자 시스템에 통합. 통합 후 모든 테스트 통과해야함
점증적 계획	요구사항은 스토리 카드(시나리오)에 기록. 개발자는 스토리를 개발 태스트로 나눔
고객 참여	고객은 개발 팀의 구성원. 요구사항 도출 책임 가짐
짝 프로그래밍	개발자는 짝을 지어 일하고 서로의 작업 검토
리팩토링	기능은 그대로. 코드의 단순화(리팩토링)
단순한 설계	현재의 요구사항만을 생각하며 설계 수행 (미래 반영X 고려X)
소규모 릴리스	최소한의 기능 먼저 개발 초기 릴리스에 점증적으로 기능 추가하며 자주 시스템 릴리스함
유지할 수 있는 속도	과도한 초과근무 X. 되려 코드 품질과 생선성 떨어뜨림
테스트 주도 개발	자동화된 테스트 프레임워크 이용하여 새 기능 구현 전 테스트 먼저 작성함

[애자일. XP 원칙]

1, '점증적 개발'은 빈번하고 작은 릴리스를 통해 지원됨
2. '고객참여'는 고객이 개발팀에 지속적으로 관여함을 의미.
3. '프로세스가 아닌 사람'은 짝 프로그래밍, 공동 소유권, 초과근무 지양하는 지속 가능 개발
4. '변화 수용'은 정기적 릴리스, 테스트 우선 개발, 리팩토링, 새로운 기능의 연속적 통합을 통해 지원됨
5. '단순성 유지'는 코드 품질 개선하는 계속적인 **리팩토링과 미래의 변경을 불필요하게 반영하지 않는 단순한 설계를 통해 지원됨

[실무적 XP 기법]

▶대부분의 조직에서 XP는 너무 극단적. 실무 적용 어려움

-실무 원칙을 골라서 사용함

-Scrum 관리 중심 기법과 함께 사용

▶주요 실무 원칙

-테스트 우선(주도) 개발 : TDD(Test Driven Development)

-리팩토링(refactoring)

-점증적 개발

-사용자 스토리

-짝 프로그래밍

[1. 사용자 스토리]

***사용자 스토리 :시스템 사용자가 경험할 수 있는 일종의 사용 시나리오**

▶XP의 요구사항 관리

-요구사항 변경 지원을 위해 별도의 요구공학 활동 두지 않음

-시스템 사용자가 경험할 수 있는 일종의 사용 시나리오(사용자 스토리) 만들어 관리

-고객은 개발팀과 스토리 카드를 작성. 스토리 카드를 '과업'(task)로 나누어 개발

▶사용자 스토리의 장단점

-요구사항 문서나 유스케이스 보다 쉽게 이해 가능

-요구사항 완전성 문제

: 스토리들이 시스템 주요 요구사항 전체를 포함하였는지

: 개별 스토리에 누락된 내용은 없는지

[2. 리팩토링]

▶전통적인 소프트웨어 공학

-변경을 고려한 설계를 하라

▶XP

-변경이 없거나 예상치 못한 변경 요청 일어나기도 하므로 변경 고려한 설계 = 시간낭비

-변경은 필연적 발생 + 점증적 개발 -> 소프트웨어 구조 망가짐

-지속적 리팩토링으로 소프트웨어 구조와 가독성 개선 -> 구조적 악화 막고 변경 처리 쉬움

▶리팩토링의 예시

***리팩토링 : 개선 가능 부분 찾아서 즉시 구현하는 것 의미**

-중복 코드 제거위한 클래스 구조 변경

-속성/메소드 이름 적절 변경

-비슷한 코드 반복 사용을 메소드 호출로 묶어서 대체

[3. 테스트 우선 개발]

▶XP의 테스트

-명세 없이 테스팅하는 문제

-테스팅을 자동화하고 변경 후에 모든 테스트 통과해야 진행

▶XP 테스팅 특징

-테스트 우선 개발 -> 코딩 -> 테스트

-시나리오 이용하여 점증적 테스트 개발

-테스트 개발과 검증에 사용자 참여

-테스트 자동화 프레임워크 사용

-스토리 카드 -> 태스크 -> 태스크 별 테스트 작성

-고객은 다음 릴리스에서 구현할 스토리에 대한 인수 테스트 개발을 도움

▶[테스트 주도 개발(TDD)]

-코드 작성 전 테스트 먼저 작성

▶테스트 주도 개발 특징

-테스트 작성하려면 기능, 인터페이스에 대해 알아야 하므로 요구사항 명확히 할 수 있음

-테스트 지연 문제 방지

-자동화된 테스트 프레임워크 사용 필수 (ex. JUnit)

: 테스트 쉽게 작성할 수 있게 함

: 테스팅을 자동으로 실행할 수 있으며 테스팅 결과가 명세에 맞는지 확인 가능

▶[테스트 자동화]

-테스트 자동화는 TDD에 필수적

⇒ **[테스팅 컴포넌트]**

-독립 실행 가능

-테스트 입력과 결과 확인 가능

-JUnit 등 자동화 프레임워크 이용

⇒ **[테스트 커버리지] coverage** (***)

-완전성 판단할 수 없음

[4. 짝 프로그래밍]

▶짝 프로그래밍

-프로그래머가 짝을 지어 코드 같이 개발. (짝 바뀔 수 O)

▶장점

-시스템에 대한 공동 소유권, 책임

-다른 사람이 같이 코드 보기 때문에 비공식적 '인스펙션'(코드 리뷰) 프로세스 진행됨

-리팩토링 장려,지지

▶짝 프로그래밍의 실무 적용

-경험 많은 개발자 + 경험 부족한 개발자

-학생 개발자의 경우, 짝 프로그래밍 생산성 낮지 않다는 연구

-경험 많은 개발자의 경우, 생산성이 떨어진다는 연구

-지식 공유가 중요

3.3 애자일 프로젝트 관리

▸ **계획주도 접근법**
-무엇이 언제 인도될 것인지, 누가 산출물 만들 것인지 계획 가지고 있음
-관리 쉬움
▸ **애자일 기법**
-프로젝트 진척도가 가시적 X
-비공식적 계획과 프로젝트 관리
-큰 조직에 부적합

[스크럼] : Scrum

▶ 스크럼: 애자일 프로젝트 조직화, 외부 가시화 제공 위한 프레임워크
-반복적 개발 관리에 중점을 둔 방법
-특정 애자일 실무 원칙 요구 X

▶ 용어

스크럼 마스터	프로젝트 관리자
제품 ***백로그(backlog)**	할 일 목록 To do list
***스크럼**	개발 팀 일일 회의 (짧은 회의)
***스프린트(sprint)**	반복

[스프린트 주기]

▸ **2-4주 범위의 고정 길이**
-완성 못한 작업 처리 목적으로 기간 늘리지 X 않음
-마치지 못한 작업은 '제품 백로그'(전체 할 일)에 돌려놓음

▸ **제품 백로그 항목 우선순위 매겨서 해당 스프린트에 작업할 '스프린트 백로그'를 선정**
-'스프린트 백로그'(이번에 할 일)

▸ **매일 스크럼 진행**
-진척도 점검. 우선순위 조정하는 짧은 미팅
-진척 사항, 문제점, 계획 등 정보 공유

▸ **'스크럼 보드'를 두어 정보 공유함**
-스프린트 백로그, 진행상황, 완료된 작업 등 표시
-누구든지 변경 가능

▸ **스프린트 끝날 때 리뷰 미팅 실시**

Ch04. 요구공학

[요구사항] : requirements

-시스템이 제공하는 **서비스(기능적 요구사항)**와 서비스에 대한 **제약조건(비기능적 요구사항)**

[요구공학] : requirements engineering

- 요구사항 도출, 분석, 문서화, 점검 프로세스

[요구사항의 유형]

▶사용자 요구사항

-사용자 대상 고수준 추상적 요구사항

-자연어와 다이어그램 활용하여 사용자 위해 작성

▶시스템 요구사항

-시스템이 제공해야 할 내용을 '상세하게' 기술한 요구사항

-시스템의 기능, 서비스, 제약조건을 구조화한 문서로 작성

> ▸시스템 이해당사자
> - 시스템의 영향 받는 사람
> - 사용자, 시스템관리자, 시스템 소유자, 외부 이해당사자 등

[타당성 조사] (***) | Feasibility studies

-요구공학 프로세스 초기의 짧은 기간 동안 진행

-타당성 조사 목적

-시스템이 조직 전체 목표에 기여?

-시스템을 기한 내에 주어진 예산으로 현재의 기술 이용해서 구현 가능?

-시스템을 사용되고 있는 다른 시스템과 통합 가능?

-시스템이 기술적/재정적으로 타당한지 평가하여 프로젝트 진행 여부 결정

4.1 기능적/비기능적 요구사항

▶[기능적 요구사항]

-시스템이 제공해야 하는 '서비스'

-시스템이 특정 입력 상황에 어떻게 반응/행동하는지

-시스템이 무엇을 해야하고 무엇을 하지 말아야 하는지 명시적 기술

▶[비기능적 요구사항]

-시스템이 제공하는 (서비스)기능에 대한 '제약조건'

-개발 프로세스, 표준에 대한 제약조건

-개별 특징이나 서비스 < 시스템 전체에 적용되는 경우 多(창발성)

<도메인 요구사항> p,100

-응용 도메인에서 주어진 새로운 요구사항 or 기존 요구사항에 대한 제약조건

-소프트웨어 엔지니어는 도메인을 잘 이해 못할 가능성 O

[1. 기능적 요구사항]

▸시스템이 무엇을 해야 하는지 나타냄

▸시스템의 기능(functionality)나 서비스를 기술

▸**[기능적 '사용자' 요구사항]**

-시스템이 무엇을 해야 하는지 고수준으로 기술

▸**[기능적 '시스템' 요구사항]**

-시스템 서비스 자세하게 기술

▸EX) Mentcare 시스템의 기능적 요구사항 예시

-사용자는 예약 내역 검색 가능

-각각의 클리닉에 대해 당일 방문 환자 리스트 생성 가능

-8자리 직원 번호로 직원 구분 가능

▶요구사항 명세의 부정확성

: 모호한 요구사항은 사용자, 개발자가 서로 다르해석 가능해서 분쟁 가능

▶요구사항 완전성

: 필요로 하는 모든 서비스와 정보가 정의되어야 함

▶요구사항의 일관성

: 충돌, 모순되는 요구사항 없음

▶이상적 요구사항 : 완전성 + 일관성

-현실적으로 불가능

[2. 비기능적 요구사항]

▸**시스템 제공 특정 서비스(기능)가 아닌 요구사항을 의미**

-시스템에 대한 명세나 제약조건

-신뢰성, 응답시간, 메모리 사용량 등 '창발적'(emergent) 시스템 속성과 관련된 제약

-I/O 장치 성능, 다른 시스템과 인터페이스에 사용되는 데이터 표현 등 시스템 구현과 관련된 제약

▸**비기능적 요구사항 만족되지 않는 것이** (기능적 만족X 보다) **심각할 수 O**

▸**비기능적 요구사항은 특정 컴포넌트 < 시스템 전체 아키텍처에 영향을 받음**

[비기능적 요구사항 분류]

▶제품 요구사항

-소프트웨어의 실행시간 동작에 대한 제약조건

-성능(속도, 용량), 신뢰성(고장률), 보안, 사용성 등

▶조직 요구사항

-고객과 개발자 조직의 정책, 절차로부터 오는 요구사항

-운영 프로세스, 개발 프로세스, 개발환경, 프로세스 표준, 운영환경 등

▶외부 요구사항

-시스템과 개발 프로세스 외부로부터 오는 광범위한 요구사항

-규제, 법률, 윤리 등

[요구사항 목표]

▶비기능적 요구사항 문제는 이해 당사자들이 일반적인 목표(goal)로 요구사항 제시함

- 사용 편함, 고장 시 시스템 회복능력, 사용자 입력에 대한 빠른 반응 등

▶목표

-사용자의 일반적 의도

-개발자에게 사용자 의도 전달하는데 도움 됨

▶확인 가능한 비기능적 요구사항

-객관적으로 테스트할 수 있는 척도로 표현된 문장

-정량적인 명세로 표현

[비기능적 요구사항 명세 척도]

속성	척도
속도 Speed	초당 처리 트랜잭션 수 사용자/사건 응답 시간 스크린 리프레시 시간
크기 Size	메가바이트 ROM 칩 개수
사용 편의성 Ease of use	교육 시간 도움말 프레임 수
신뢰성 Reliability	평균 고장 시간(mean time to failure) 고장 발생 비율 가용성(availability)
견고성 Robustness	고장 후 재가동 시간 고장을 일으키는 사건의 비 고장에 의한 데이터 손실 확률
이식성 Portability	타겟 시스템에 종속된 코드 비율 타겟 시스템의 수

4.2 요구공학 프로세스

[요구공학 프로세스 활동]

-실무에서는 요구공학 활동들 중첩되어 진행됨

-요구사항 도출(elicitation)

-요구사항 분석(analysis)

-요구사항 명세화(specification)

-요구사항 검증(validation)

4.3 요구사항 도출

[요구사항 도출 프로세스]

-이해당사자들의 업무를 이해하고 시스템을 업무에 활용하는 방식 알아냄

-이해당사자들로부터 응용 도메인, 시스템이 제공해야 하는 서비스, 요구되는 시스템 성능, 하드웨어 제약 등을 알아냄

[요구사항 도출 프로세스 활동]

- 요구사항 발견 및 이해
- 요구사항 분류 및 구성
- 요구사항 우선순위 지정 및 협상
- 요구사항 문서화

[관점] Viewpoints

-관점 : 공통점을 가진 이해당사자 그룹으로부터 요구사항을 수집하고 구성하는 방식

-요구사항 분석을 위해 이해당사자 정보를 조직화

: 이해당사자 그룹이 하나의 관점을 가졌다고 간주하고

해당 그룹이 가진 관점으로부터 요구사항을 수집

-도메인 요구사항이나 타 시스템 관련 제약조건을 나타내는 관점을 포함시킬 수 O

-상이한 이해당사자들은 요구사항의 중요도와 우선순위를 다르게 생각

[1. 요구사항 도출 기법]

▶제안된 시스템에 대한 정보를 찾는 방법

-다양한 유형의 이해당사자, 기존 시스템에 대한 지식, 문서

▶요구사항 도출을 위한 주요 기법

1) 인터뷰

2) 관찰 or 문화기술적 연구

---

(1) 인터뷰

▶[인터뷰 유형]

-폐쇄적 인터뷰: 미리 정의된 질문 목록 있음

-개방적 인터뷰

▶[인터뷰의 문제점]

-도메인 전문 용어는 비전문가가 이해하기 어려움

-전문가는 어떤 도메인 지식은 언급하지 않음

(2) 문화기술적 연구

▶[문화기술적 연구]

-운영(동작) 프로세스를 이해하고, 이와 같은 프로세스를 지원하는 소프트웨어의 요구사항을 얻기 위해 사용하는 관찰 기법

▶문화기술적 연구가 효과적인 경우

-실제 일하는 방식으로부터 얻을 수 있는 요구사항

-협력과 다르사람의 활동을 인식하는 것으로부터 얻을 수 있는 요구사항

▶문화기술적 연구는 프로토타입 개발과 결합될 수 O

▶혁신과 문화기술적 연구

-Nokia는 문화기술적 연구를 통해 새 제품 요구사항을 도출

-Apple은 기존 사용법을 무시

[2. 스토리와 시나리오]

▶[스토리와 시나리오]

-특정 작업을 위해 어떻게 시스템을 사용하는지 기술

-무엇을 하는지, 사용하는 정보는 무엇인지, 어떤 시스템을 이용하는지

▶시나리오

-사용자 상호작용이 이루어지는 일정 구간에 대한 사례를 구조적으로 설명

▶시나리오 구성

-시나리오가 시작될 때 시스템과 사용자가 기대하는 것에 대한 설명

-정상적인 사건 흐름에 대한 설명

-비정상적인 사건 흐름에 대한 설명

-동시에 진행할 수 있는 다른 활동에 대한 정보

-시나리오가 끝날 때 시스템 상태에 대한 설명

4.4 요구사항 명세

[요구사항 명세]

▸ **'이상적으로'** 요구사항은 시스템의 외부 행동과 운영상의 제약조건을 기술해야 하며
(시스템의 설계와 구현에 대한 사항 다루지 않아야 함)

▸ **'현실적으로'** 요구사항에서 설계 정보 완전 제외하는 것은 **불가능**

-초기 아키텍처 설계가 요구사항 구성하는데 도움이 됨
-대부분의 경우 시스템은 기존 시스템과 상호작용 필요
-비기능적 요구사항 만족시키기 위해 특정 아키텍처 사용해야 하는 경우

[1. 자연어 명세]

[자연어 명세 지침]

-표준 형식 만들어 사용
-필수적인 사항과 바람직한 사항 구분하기 위해 일관성 있는 표현 사용
-중요 부분 글자 강조
-요구사항의 독자가 기술적이고 소프트웨어공학 용어를 이해한다고 생각하지 말 것
-가능하다면 사용자 요구사항에 대한 이유를 기록하여 변경 시 활용
-왜 포함되었는지
-누가 제안했는지 (요구사항 출처)

[2. 구조적 명세]

[구조적 명세]

-표준 서식 만들어 사용
-표에 구조화된 양식(템플릿) 맞춰서 정보 기입

[3. 유스케이스]

[유스케이스 모델]

-시스템 행동을 모델링

[구성요소]

-유스케이스(use case) : 시스템이 구행하는 기능 (기능적 요구사항)
-액터(actor) : 시스템과 직접 상호작용하는 모든 것 (시스템 or 사람)

<관계 표현>

유스케이스 <-> 액터 관계	연관 (association)
액터 간의 관계	일반화(generalization)
유스케이스 간의 관계	-일반화(generalization) -포함(include) -확장(extend)

[4. 소프트웨어 요구사항 문서화] //그냥 넘김

4.5 요구사항 검증

[요구사항 검증]

-고객이 원하는 시스템을 제대로 정의하고 있는지 점검

-요구사항 문제 수정 비용이 >> 설계, 구현 오류 수정 비용보다 매우 크다

[요구사항 검증 기법]

-요구사항 검토(review)

-프로토타이핑(protyping)

-테스트 케이스(test case) 생성

[요구사항 점검 내용]

1) 유효성(validity) : 사용자 실제 요구 반영 여부 점검
2) 일관성(consistency) : 요구사항 상호 충돌, 모순 여부 점검
3) 완전성(completeness) : 사용자가 의도한 모든 기능, 제약조건 포함 여부 점검
4) 현실성(realism) : 기존 지식과 기술 사용하여 주어진 예산으로 구현 가능한지 점검
5) 검증가능성(verifiability)
   : 시스템이 각 요구사항 만족시키는지 보여줄 테스트 작성 가능한지 점검

4.6 요구사항 변경

[요구사항 변화 주요 원인]

-시스템의 비즈니스와 기술 환경 계속 변화함

-개발 비용 지불하는 사람이 시스템 사용자가 아님

-대규모 시스템은 서로 다른 요구사항, 우선순위를 가진 다양한 이해당사자 집단이 관여

[공식적 요구사항 관리 프로세스가 필요]

-요구사항 변경으로 인한 영향을 평가

-개별 요구사항 추적하고 연관된 요구사항들 간 관계를 관리

-*애자일 프로세스 : 개발 과정에서 변경되는 요구사항 처리하므로 공식적 변경 프로세스X

[요구사항 추적성] : 추적가능성 (Traceability)

▶추적가능성 유지해야 한다. (*******)

-요구사항 출처, 요구사항, 시스템 설계 간 관계를 추적 가능해야 함

-변경 이유와 출처 관리

-제안된 변경이 시스템의 어떤 부분에 영향 주는지 분석

-한 변경으로 인한 파장이 시스템 전체로 어떻게 퍼지는지 추적

Ch05. 시스템 모델링

[시스템 모델링]

-시스템의 추상 모델을 개발하는 프로세스

-각 모델은 시스템의 서로 다르 뷰 or 관점을 나타냄

[모델의 용도]

-요구 공학 프로세스 :상세 요구사항 도출 위해

-설계 프로세스 : 시스템 설명 위해

-구현 후 : 시스템 구조, 동작을 문서화하기 위해

[시스템 모델은 시스템의 완전한 표현 X]

-이해하기 쉽게 자세한 부분은 모델에서 의도적으로 제외 (생략)

-모델은 시스템의 다른 표현이 아닌 시스템의 추상화임

[시스템 관점]

1) 외부(external) 관점

- 시스템의 컨텍스트 or 환경을 모델링

2) 상호작용(interaction) 관점

- 시스템과 그 환경 사이의 상호작용(시나리오) or 시스템 컴포넌트 간 상호작용을 모델링

3) 구조(structural) 관점

- 시스템의 구성이나 시스템에 의해 처리되는 데이터 구조를 모델링

4) 동작(behavioral) 관점

- 시스템의 동적인 행동과 시스템이 이벤트에 어떻게 반응하는지 모델링

[그래픽 모델 사용 방법]

▸ **시스템에 대해 토론, 아이디어 도출 용도 (아이디어 스케치용)**

- 모델은 불완전할 수 있으며 표기법을 약식으로 사용 가능
- 주로 애자일 기법에서 사용 多

▸ **기존 시스템 문서화 용도**

-모델이 완전할 필요는 X.

-모델이 정확해야 함 O

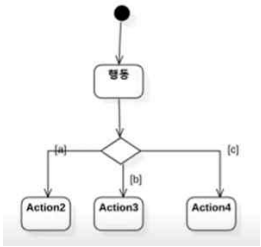
▸ **시스템 구현을 생성하는 모델 기반 프로세스의 입력**

-소스 코드 생성하는데 사용. 모델이 완전하고 정확해야 함

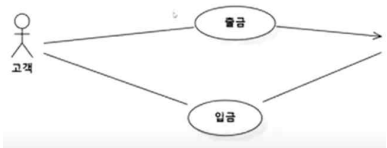
[UML 다이어그램 유형] (5가지)

액티비티(activity) 다이어그램	프로세스나 데이터 처리 관련 액티비티 흐름 (행동 단위 순서도)
유스케이스(use case) 다이어그램	시스템과 환경 간 상호작용
시퀀스(sequence) 다이어그램	액터와 시스템 간, 시스템 컴포넌트 간 상호작용 시나리오를 표현
클래스(class) 다이어그램	시스템 객체 클래스들과 클래스들 간의 관계
상태(state) 다이어그램	시스템 내/외부 이벤트에 대한 반응 (상태 단위 순서도)

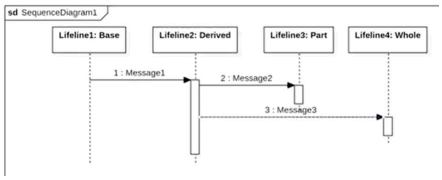
액티비티 다이어그램



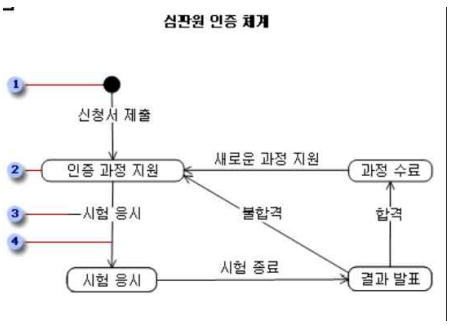
유스케이스 다이어그램



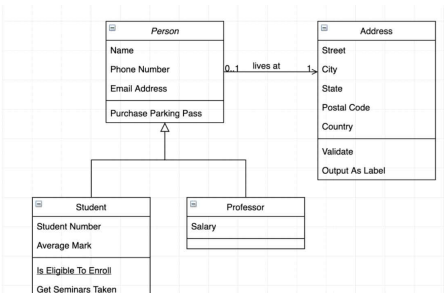
시퀀스 다이어그램



상태 다이어그램



클래스 다이어그램



5.1 컨텍스트 모델

[컨텍스트 모델]

-시스템 운영 환경 기술

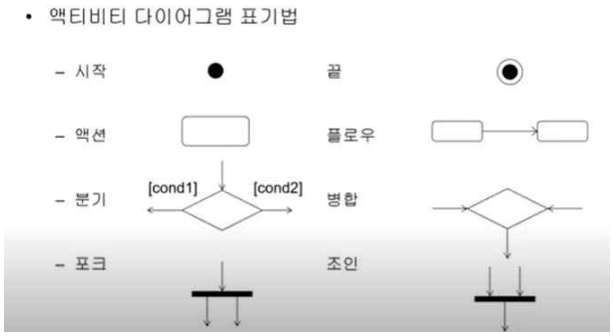
-시스템의 경계와 시스템과 연동되는 외부 시스템을 보여줌

[비즈니스 프로세스 모델]

-전반적인 업무 흐름 보여줌

-컨텍스트 모델과 함께 사용O

-UML 액티비티 다이어그램으로 표현



5.2 상호 작용 모델

[상호 작용 모델의 용도]

-사용자와 시스템 간 상호작용 모델 : 사용자 요구사항 확인에 도움

-시스템 간 상호작용 모델 : 시스템 간 커뮤니케이션 문제 발생 가능성 해결

-컴포넌트 간 상호작용 모델 :시스템 구조가 성능과 확실성 제공할 수 있는지 이해에 도움

[상호 작용 모델 종류]

유스케이스 다이어그램	시스템과 사용자/ 다른 시스템 간 상호작용
시퀀스 다이어그램	시스템 컴포넌트 간 상호작용

[1. 유스케이스 모델링]

[유스케이스]

-시스템 제공 기능

-공통적인 사용자 목표와 관련된 시나리오 집합

[액터]

-시스템과 직접 상호작용하는 모든 것

[연관]

-액터가 유스케이스를 수행

[2. 시퀀스 다이어그램]

5.3 구조 모델

[구조 모델]

-시스템을 구성하는 컴포넌트들과 그들 간의 관계를 보여줌

-시스템 설계 구조를 보여주는 정적 모델

-시스템 실행 시 구성을 보여주는 동적 모델

[1. 클래스 다이어그램]

-소프트웨어 시스템 구성 객체 클래스들의 정적 구조 보여줌

[클래스]

-한 종류의 시스템 객체 단위를 일반적으로 정의

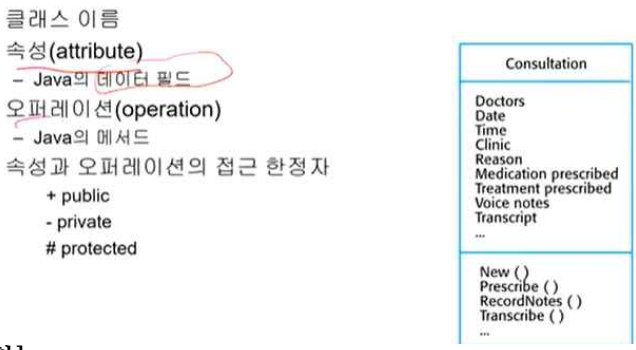
-같은 속성, 메소드, 관계, 의미 가지는 객체 집합에 대한 설명

-객체는 클래스의 인스턴스

[연관]

-클래스 객체 간의 관계

-연결은 연관의 인스턴스



[2. 일반화]

[일반화] generalization

-상속(inheritance)

-Java의 extends

[3. 집합]

[집합] aggregation

-한 객체(전체)가 다른 객체(부분)으로 구성됨

-복합 관계는 집합 관계의 특수한 경우

5.4 동작 모델

[동작 모델]

-시스템이 실행될 때의 동적 행동을 보여줌

-자극(stimulus)에 대한 시스템 반응을 모델링

[2가지 종류의 자극]

-데이터 : 데이터가 오면 시스템에 의해 처리

-이벤트 : 이벤트 발생하면 이에 대해 반응

▶[데이터 흐름도]

-데이터가 처리되는 과정을 보여줌

-액티비티 다이어그램을 이용

▶[상태 다이어그램]

-내부/외부 이벤트에 대한 시스템 반응을 보여줌

[1. 데이터 주도 모델링]

[데이터 주도 모델] : 입력 -> 처리 -> 출력

[데이터 흐름도] : DFD (Data Flow Diagram)

-자료흐름(객체)와 처리(액티비티)로 구성

[2. 이벤트 주도 모델링]

[이벤트 주도 모델] : 시스템이 외/내부 이벤트(자극)에 반응하는 방식

[이벤트 주도 모델의 구성]

-시스템은 유한 개의 상태(state) 가짐

-이벤트에 의해 시스템 상태 전이됨

[상태 다이어그램]

- 각 상태 별 설명 존재해야함

Ch06. 아키텍처 설계

[아키텍처 설계]

-시스템 전체 구조 설계

-시스템 주요 구조 컴포넌트(subsystem)들과 상호작용하는 컴포넌트 간의 관계

[아키텍처 변경]

-아키텍처 변경 비용 많이 듬

-애자일 개발 초기 단계는 아키텍처 설계에 초점 맞춰야 함

-컴포넌트 리팩토링은 비교적 쉽지만 아키텍처의 점진적 개발은 바람직 X

[아키텍처 설계와 요구공학 프로세스의 중첩]

-이상적으로는 요구사항 명세에 설계 포함 X

-요구공학 프로세스의 일부로서 추상 시스템 아키텍처가 제시되어야 함

[아키텍처 요구사항]

-시스템의 개별 컴포넌트가 '기능적' 요구사항을 구현

-시스템의 아키텍처는 '비기능적' 요구사항에 영향을 줌

[아키텍처 명시적 설계와 문서화의 장점]

-상위 수준 시스템 표현으로 이해당사자 간 의사소통에 도움

-시스템의 중대한 비기능적 요구사항(성능, 신뢰성, 유지보수성) 만족시킬 수 있는지 분석

-비슷한 요구사항 가진 시스템 아키텍처 재사용

6.1 아키텍처 설계 결정

[아키텍처와 비기능적 특성 간 관계]

1) 성능(performance)

-컴포넌트 간 통신을 줄임, 시스템 중복, 부하 분산

2) 보안성(security)

-중요한 자산을 가장 안쪽 계층에 두는 계층 구조 사용

3) 안전성(safety)

-안전 관련 작업을 소수의 컴포넌트에 배치하여 안전 검증과 대응 간단히 함

4) 가용성(availability)

-중복 컴포넌트 배치, 시스템 중단 없이 컴포넌트 교체 및 갱신

5) 유지보수성(maintainability)

-변경이 용이한 독립적 컴포넌트 사용

6.2 아키텍처 뷰

[4+1 뷰 모델] : 시스템 아키텍처를 여러 개의 뷰로 바라봤다.

1) 논리적 뷰

-시스템의 논리적 구조 보여줌. 클래스 다이어그램 등

2) 프로세스 뷰

- 시스템의 프로세스/쓰레드의 런타임 상호작용을 보여줌
- 성능, 가용성 등 비기능적 시스템 특성과 관련

3) 개발 뷰

-소프트웨어가 개발을 위해 어떻게 분해되는지 보여줌

4) 물리적뷰

-시스템 하드웨어와 소프트웨어 컴포넌트들의 배치 보여줌

5) 유스케이스 뷰

-시스템의 행동(유스케이스 시나리오)를 액터 관점에서 보여줌

6.3 아키텍처 패턴

[패턴]

-자주 발생하는 문제에 대한 해법
-지식 공유, 재사용 목적
-Gamma 등 객체 클래스 설계 패턴 널리 알려짐
-이름, 설명, 문제, 해법 등으로 구성

[아키텍처 패턴]

-아키텍처 스타일이라는 이름으로 제안됨
-해당 영역에서 성공적이고 바람직한 시스템 구조 사례를 추상화

ex. 모델 뷰 제어기 (MVC: Model-View-Conroller)

[MVC 패턴]

▸설명

-데이터로부터 표현과 상호작용 분리시켜, 3개의 논리적 컴포넌트로 구조화

-모델: 데이터와 이에 대한 오퍼레이션 관리
-뷰: 사용자에게 데이터 표현 관리
-제어기 : 사용자 상호작용 관리하고 이를 뷰와 모델에 전달

▸언제 사용

-데이터 표현과 상호작용 방법이 여러 가지일 때 사용

▸장점

-데이터 표현과 무관하게 데이터 변경 O
-동일한 데이터를 서로 다른 방식으로 표현하는 것 지원.
-뷰의 추가/변경 쉬움

▸단점

-데이터 모델과 상호작용 단순한 경우, 불필요하게 코드만 복잡해짐

[아키텍처 패턴]

(1) 계층 아키텍처 (.. 그냥 넘어감)

▸ 설명

-각각의 기능을 '계층' 단위로 분리하여 전체 시스템을 구성
-각 계층은 상위 계층에 서비스를 제공함

▸ 언제 사용

-기존 시스템에 '새로운 기능' 구축할 때
-계층별로 분리하여 여러 팀이 각각 독립적으로 개발할 때
-다중 수준 보안 필요할 때

▸ 장점

-시스템 확실성 높이기 위해, 중복된 기능 제공할 수 O
-인터페이스 동일하면 계층 대체 가능 O

▸ 단점

-계층 명확 구분 어려움
-불필요하게 모든 계층 처리를 거치며 성능 저하될 수 있음

(2) 저장소 아키텍처

▸ 대량 데이터 사용 시스템은 '공유 데이터베이스(저장소) 중심으로 구성된다.

▸ 저장소를 중심으로 '컴포넌트(도구)'들을 배치

-공통 저장소 데이터 모델(스키마) 구조를 따라야 함
-효율적으로 대량 데이터 공유. 저장소에서 데이터를 저장/인출

▸ 장점

- 컴포넌트들이 독립적이라 다른 컴포넌트 알 필요 X
- 데이터 일관성 있게 관리 가능O

▸ 단점

-통신이 저장소에 집중되어, 저장소에 문제 발생 시 -> 시스템 전체 영향을 줌

(3) 클라이언트-서버 아키텍처

▸ 분산 네트워크 환경에서 '클라이언트-서버' 아키텍처는 주요 아키텍처

▸ 클라이언트 서버 아키텍처의 컴포넌트

-서비스 제공 서버(프로세스) 집합
-서비스 요청 클라이언트(프로세스) 집합
-클라잉언트와 서버 연결 네트워크

▸ 클라이언트와 서버 연ㅇ결

▸ 특징

-분산 아키텍처이므로 서버 추가/통합/업그레이드 쉬움
-네트워크 성능이 시스템 성능에 영향 줌. 서비스 거부 공경(Dos)에 취약

(4) 파이프 필터 아키텍처

▸ 파이프 필터 아키텍처

-입력 데이터를 기능적 변환 처리하여 출력 데이터 생성
-데이터가 변환을 통해 흘러감
-변환이 순차적/병렬적으로 실행

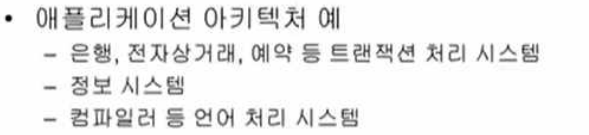
▸ 변환 재사용 쉽고, 일괄처리 시스템에 적합

6.4 애플리케이션 아키텍처

[애플리케이션 아키텍처] //그냥 읽고 넘김

-비즈니스 영역에 따라 사용되는 **애플리케이션 시스템은 공통점 있음**

-같은 유형의 시스템 개발 시 **공통적인 아키텍처 구조가 재사용**됨



[1. 트랜잭션 처리 시스템] // 대충 설명 후 넘김

[트랜잭션]

-원자성을 가지며, 트랜잭션 수행되면 트랜잭션 내 모든 작업들이 완료되어야 함

-cf. 데이터페이스 트랜잭션 ACID

-예시) 은행 계좌 이체

[트랜잭션 처리 시스템]

-데이터베이스에 있는 정보에 대한 (사용자 요청/갱신 요청) 처리 위해 설계됨

-사용자 서비스 요청을 비동기적 처리하는 대화식 시스템

[트랜잭션 처리 위한 미들웨어]

-TP(Transaction Processing) monitor

-Application Server :WAS, Aapche Tomcat, WebLogic. Jeus 등

[2, 언어 처리 시스템] //대충 설명 후 넘김

Ch07. 설계와 구현

[소프트웨어 설계와 구현]

▶SW 설계와 구현 : 실행 가능한 SW 시스템이 개발되는 소프트웨어 공학 프로세스 단계

-설계(how)는 요구사항(what)을 실현할 소프트웨어 컴포넌트들과 그들 간의 관계 식별 활동

-구현 : 설계를 프로그램으로 실체화 시키는 활동

▶설계와 구현은 필연적으로 중첩됨(interleaved)

-설계와 구현은 밀접하게 연결되어 있으며 설계 시 구현 이슈 고려함

▶개발 프로세스에 따른 차이

- ▸계획 기반 : 설계 단계가 있으며 설계가 모델링되고 문서화됨
- ▸애자일 : 설계는 대략적인 스케치만 하고 설계 결정은 프로그래머가 함

7.1 UML을 이용한 객체 지향 설계

[객체 지향 설계 프로세스]

▶객체지향 시스템

- 시스템은 상호작용하는 객체들로 구성됨
- 객체는 자신의 상태 유지하며 오퍼레이션 제공

▶객체지향 설계 프로세스

-객체 클래스들을 찾고 클래스 간의 관계 식별(역할, 책임 등)

▶객체지향 설계 프로세스의 일반적인 절차

-시스템 컨텍스트와 외부 상호작용 식별

-시스템 아키텍처 설계

-주요 객체 식별

-설계 모델 개발

-객체 인터페이스를 명시

[(1). 시스템 컨텍스트와 상호작용]

▶시스템 컨텍스트 모델

-개발하는 시스템의 환경에 있는 다른 시스템들과의 관계를 보여줌

-클래스 다이어그램과 '연관'을 이용하여 표현 가능

▶상호작용 모델

-시스템이 사용될 때 다른 시스템들과의 상호작용을 보여줌

-유스케이스 모델 이용하여 표현 가능

-각 유스케이스 내용을 기술해야 함 (구조화된 자연어 등)

[(2). 아키텍처 설계]

▶아키텍처 설계

-시스템 구성 주요 컴포넌트(서브 시스템)들과 그들 간의 상호작용을 식별

-아키텍처 설계의 일반 지식과 도메인 지식을 활용

[(3). 객체 클래스 식별]

▶객체 클래스 식별 지침

-관련 문서에서 객체와 속성은 명사. 오퍼레이션이나 서비스는 동사.

-응용 도메인의 실제 개체를 나타내는 클래스를 만든다

-시나리오 기반 분석을 사용

▶객체 클래스 식별은 반복적인 과정

-대략적 시스템 설명으로부터 클래스, 속성, 오퍼레이션 식별

-응용 도메인 지식과 시나리오 분석 이용하여 초기 객체들 정련

-요구사항 문서, 사용자 인터뷰, 기존 시스템 분석 등으로부터 정보 수집

+ 디자인 패턴 ***
결합도 낮게, 응집도 높게
(coupling) (cohesion)

[(4). 설계 모델] Design models

▶설계 프로세스에 따른 상세 수준

-애자일 프로세스

-계획 기반 프로세스

▶설계 모델 종류

-구조 모델 : 시스템 정적 구조를 객체 클래스와 클래스들 간의 관계로 표현

-동적 모델 : 실행 중에 일어나는 객체 만의 상호작용 표현

▶유용한 UML 설계 모델

-서브시스템 모델(구조) : 클래스 다이어그램

-시퀀스 모델(동적) : 시퀀스 다이어그램

-상태 기계 모델(동적) : 상태 다이어그램

[(5). 인터페이스 명세] Interface specification

▶인터페이스

-객체 또는 객체 그룹에 의해 제공되는 서비스

-인터페이스 클래스를 이용하여 서비스의 시그니처 정의

<<interface>> 스테레오 타입 이용

-오퍼레이션을 가지며 데이터는 가지지 않음

7.2 디자인 패턴

[디자인 패턴] Design patterns

▶패턴

-어떤 종류의 문제에 대한 해결책

▶디자인 패턴과 재사용

-패턴은 모범 사례와 바람직한 설계를 기술

-경험과 추상적 지식을 재사용할 수 있게 함

▶GoF 패턴 (Gang of Four)

▶패턴은 주로 객체지향 설계와 관련됨

-상속, 다형성, 인터페이스, 추상 클래스

[디자인 패턴 4가지 핵심 요소]

▶이름

-패턴을 지칭하는 의미 있는 이름

▶문제 서술

-패턴이 적용될 수 있는 경우를 설명하는 문제 영역 서술

▶해법 서술

-설계 해법의 구성요소에 대한 서술

-구체적 설계가 아닌 인스턴스화 될 수 있는 템플릿 형식

▶결과

-패턴 적용한 결과에 대한 설명

[Observer 패턴] : The Observer pattern

▶이름 : Observer

▶설명

-객체 상태 표시를 객체 자체로부터 분리하고 상태를 표시하는 대안 제공할 수 있게 함

-객체 상태가 변경되었을 때 모든 표시에 변경 반영되도록 자동으로 통지되며 표시를 갱신

▶문제 서술

- 상태 정보를 여러 방법으로 표시할 필요가 있다.
- 정보 명시할 때 모든 표시 방법이 알려져 있는 것은 아니다.
- 모든 표시는 상호작용을 지원하고 상태 변경 시 표현이 갱신되어야 한다.
- 상태 정보를 위해 하나 이상의 표시 형식이 필요하고 상태 정보를 유지하는 객체가 특정 표시 형식이 사용되는지 알 필요가 없는 경우에 사용될 수 있다.

▶해법 서술

-추상 객체 Subject Observer와 추상 객체 상속받는 ConcreteSubject, ConcreteObserver를 사용

-표시할 상태는 ConcreteSubject 객체에 유지되는데 표시(Observer) 추가. 삭제하는 오퍼레이션과 상태 변경 통지하는 오퍼레이션을 Subject로부터 상속받는다.

-ConcreteObserver 는 상태 표시 갱신하는 오퍼레이션을 Observer로부터 상속받는다.

-ConcreteObserver 는 자동으로 상태를 표시하고 상태 변경될 때마다 이를 반영한다.

▶결과

- Subject는 구체적 클래스의 상세 내역을 모르는 추상 Observer만 알기 때문에 객체 간 결합 최소화 된다. 상세 내역을 모르기 때문에 표시 성능 향상위한 최적화가 실용적이다. Subject에 대한 변경은 모든 Observer들의 갱신을 유발하지만 일부는 불필요할 수 있다.

7.3 구현 이슈

[(1). 재사용]

▶재사용(reuse)

-대부분 최신 SW는 기존 컴포넌트, 시스템을 재사용하여 구축됨

-가능한 기존 코드 많이 사용하는 것이 바람직

▶형상 관리(configuration management)

-개발 중 여러 버전 버전이 생성됨 이를 형상관리 시스템(Git) 이용하여 추적해야 함

▶호스트 타겟 개발(host-target development)

-호스트 시스템에서 SW 개발하여 타겟 시스템에서 실행

[SW 재사용 수준]

▶재사용 수준

▸추상 수준

-디자인 패턴, 아키텍처 패턴 재사용

▸객체 수준

-프로그래밍 언어 라이브러리 재사용

▸컴포넌트 수준

-컴포넌트 :관련 기능, 서비스 제공 객체들 모임

-컴포넌트 프레임워크 재사용

▸시스템 수준

-전체 애플리케이션 시스템 (COTS : commercial off-the-shell)

▶재사용 비용

-평가(access)/구매(buy)/개조(adapt) 및 설정(configure)/통합(integrate) 등

[(2). 형상 관리] : Configuration management

▶[형상 관리 활동]

▸버전 관리

-소프트웨어 컴포넌트의 서로 다르 버전 저장하고 검색

-체크아웃(check out) -> 수정 -> 체크인(check-in) -> 버전 생성

▸시스템 통합

-시스템 구성 컴포넌트들의 버전 선택하여 시스템 빌드

▸문제 추적

-사용자들이 문제점 보고할 수 있음

-개발자들은 누가 작업하는지 언제 고쳐졌는지 알 수 있음

▸릴리스 관리

-새로운 릴리스 기능 계획

-소프트웨어 배포 구성

[(3). 호스트 타겟 개발] // .. 읽고 넘김

[호스트와 타겟]

호스트 :SW 개발되는 컴퓨터, 개발 플랫폼

타겟: SW 실행되는 컴퓨터, 실행 플랫폼

일반적으로 임베디드와 모바일 시스템은 호스트와 타겟이 다름

7.4 오픈 소스 개발

[오픈 소스 개발]

▶오픈소스 개발 : 소스코드 공개되고 개발에 지원자들이 참여하는 SW 개발 접근법

▶자유 소프트웨어 재단(Free Software Foundation) 으로부터 유래.

-누구든지 참여 가능하나, 실제로는 핵심 개발 그룹이 주도

▶널리 사용되는 오픈소스 시스템은 안정적이고 버그 수정도 신속

▶대표적 오픈소스 시스템

-Linux, Apache, Eclipse

▶오픈소스 관련 사이트

[(1). 오픈소스 라이센스]

[오픈소스 라이센스 모델] : License models

▶[GPL] : GNU General Public License

-프로그램에서 GPL 소스를 (일부라도) 사용.

-변경한다면 소스 공개해야 하고 프로그램도 GPL 라이센스 따름

▶[LGPL] : GNU Lesser General Public License

-완화된 GPL로서, LGPL 라이브러리 사용하면 프로그램 소스 공개할 필요 X

-LGPL 라이브러리를 수정한 경우 소스 공개 O

▶[BSD] : Berkley Standard Distribution

-BSD 소스 변경 사용 가능하며, 변경 내역과 프로그램 소스 공개 의무 X

-저작권자 이름과 라이센스 내용 같이 배포해야 함

Ch08. **소프트웨어 테스팅**

[프로그램 테스팅] Program testing

▶[테스팅 2가지 목적]

-검증 테스팅: 프로그램이 의도대로 수행되는지 보여줌

-결함 테스팅 : 프로그램 사용 전 결함 발견

▶[프로그램 테스팅]

-인위적인 데이터 이용하여 프로그램 실행시키고 실행 결과 점검

▶테스팅은 오류 존재 밝힐 수 있지만 오류가 없음을 보일 수 없음

[검증 테스팅] : validation testing

-소프트웨어가 고객 요구사항에 맞는지 보여줌

-요구사항마다 적어도 하나의 테스트 있어야 하며 시스템이 정확하게 수행되는 것 기대

[결함 테스팅] : defect testing

-소프트웨어 결함에 의해 제대로 동작하지 않는 경우를 발견함

-테스트 케이스(test case)는 결함 드러낼 수 있도록 설계

[V&V 기술] : Verification & validation **//구분 필요**

▶소프트웨어 검증 및 확인(V&V) 프로세스

▶확인(validation)

-올바른 제품 만들고 있는가?

-소프트웨어가 고객 기대에 맞는지 보여줌

-명세 따르는 것을 넘어 고객의 요구(기대)하는 것을 제공하는지 확인하는 과정

▶검증(verification)

-제품 올바르게 만들고 있는가?

-소프트웨어가 기능적/비기능적 요구사항 명세에 맞는지 점검

[소프트웨어 인스펙션과 테스팅]

▶[소프트웨어 인스펙션]

-요구사항/설계/소스코드 분석하고 검사하여 문제 찾음

-프로그램 실행시키지 않는 정적(Static) V&V 기술

-주로 소스코드를 대상으로 하지만 소프트웨어의 어떤 표현이라도 검사 가능

▶[소프트웨어 테스팅]

-프로그램 실행시켜 동작을 관찰하는 동적(dynamic) V&V 기술

▶[인스펙션의 장점]

-테스팅 중에는 오류가 다르 오류들을 가릴 수 있다.

-인스펙션은 정적 프로세스이므로 오류 간 상호작용과는 무관

-인스펙션은 한 번에 여러 오류 발견 가능

-시스템의 불완전한 버전 검사 가능

-불완전 프로그램 테스트용 추가 코드 필요

-인스펙션은 결함 찾는 것 뿐 아니라 표준 준수, 이식성, 유지보수성 등의 품질 속성도 검토

▶인스펙션과 테스팅은 상호보완적

-인스펙션이 결함 발견하는 건 테스팅보다 효과적

-인스펙션이 테스팅 대체 불가능

-시스템 컴포넌트 간 상호작용, 타이밍, 성능

[테스트 케이스] : test case

-테스트 케이스 : 무엇을 테스트하는지 테스트 입력-예상 출력에 대한 서술

-테스트 데이터 : 시스템 테스트 위한 입력

-테스트 결과 : 테스트 데이터 입력에 따른 출력

[테스팅의 3단계]

▶개발 테스팅 (development testing)

-버그와 결함 찾기 위해 '개발 중 테스팅'

-시스템 설계자, 프로그래머가 수행

▶릴리스 테스팅 (release testing)

-요구사항 만족되는지 시스템 완성 버전 테스팅

-별도의 테스트팀이 수행

▶사용자 테스팅 (user testing)

-시스템 인수 여부 결정

-(잠재적) 사용자들이 자신의 환경에서 시스템 테스팅

[테스트 자동화] : Test automation

▶자동화된 테스팅

-테스트 실행하는 프로그램 이용

-테스트 주도 개발(TDD)를 위해서는 자동화된 테스팅이 필수임

▶회귀 테스팅(regression testing) ******

프로그램 변경 후 새 버그 생겼는지 여부 점검 위해 이전 과정에 수행한 테스트 다시 수행

8.1 개발 테스팅

[개발 테스팅의 세 단계]

▶[단위 테스팅] : unit testing

-함수, 클래스 등 프로그램의 단위(모듈)를 테스팅

-단위의 기능 테스팅에 집중

▶[컴포넌트 테스팅] : component testing

-단위들이 통합된 복합 컴포넌트를 테스팅

-컴포넌트의 인터페이스 테스팅에 집중

▶[시스템 테스팅] : system testing

-시스템 전체로서 테스팅

-컴포넌트 상호작용 테스팅에 집중

[개발 테스팅 목표]

-주로 버그 발견용. 결함 테스트

-디버깅과 중첩됨

< 참고>

[개발 테스팅 분류]

▶단위(unit) 테스팅

-모듈 기능 테스트

-테스트 드라이버, 스텁 필요

▶통합(integration) 테스팅

-모듈 간 상호작용 테스트

-모듈 통합 방법(빅뱅/하향식/상향식)

▶시스템(system) 테스팅

-시스템 기능 테스트

-요구사항 만족 여부 테스트

8.2 테스트 주도 개발

[(1). 단위 테스팅] : Unit testing

▶단위 테스팅에서는 개별적인 컴포넌트를 테스팅

-함수, 메소드, 클래스 등의 기능 테스트

▶[일반적인 지침]

-클래스의 모든 오퍼레이션 테스트

-객체가 가질 수 있는 모든 상태를 거치도록 테스트 (모든 상태 전이 순서를 테스트)

-상속된 오퍼레이션은 사용된 모든 곳에서 테스트 (클래스 상속이 테스팅의 복잡도 높임)

- JUnit 등의 테스트 자동화 프레임워크 사용

▶컴포넌트 분리 테스트 위해 '모형 객체' (mock object) 필요

[(2). 단위 테스트 케이스 선정]

▶효과적인 단위 테스트

-검증: 컴포넌트가 예상(명세) 대로 동작하는 것을 보임

-결함: 결함 찾음

▶[테스트 케이스 선정 전략]

▸[동등 분할 테스팅] ***

-공통 특성 가진 입력 그룹 식별하여 각 그룹별 테스트 케이스 선정

▸가이드라인 기반 테스팅

-프로그래머가 자주 범하는 오류 발견을 위한 테스트 케이스 선정

▸[코드 커버리지] ***

-소스 코드 중 테스트 된 비율

-문장 커버리지/ 결정 or 분기(decision or branch) 커버리지/조건 커버리지 등

[분할 테스팅] : Partition testing

▶[동등 분할(eqivalence partition)]

- 입력 데이터 또는 출력 결과를 공통 특징 가진 그룹으로 분할

▶[분할 테스팅] *****

-입력과 출력 분할을 식별 (동등 분할)

-각 분할에서 테스트 케이스 선정

-일반적으로 분할 경계와 분할 중간점을 선정