1. software 소프트웨어

컴퓨터 프로그램 + 중간 결과물 (문서: 중간결과물로 도출되는 모든 문서)

소프트웨어 전문가가 구축한 후 장기적으로 지원하는 제품

2. software product 소프트웨어 최종 결과물

- Generic: 특정 고객에 한정된 것이 아닌 일반적으로 누구든 사용 가능 (대부분)

- Custom: 사용자 맞춤형 프로그램 (고객의 사양에 따라 고객을 위해 개발)

3. software 특징 (개발이 어려운 이유)

- invisibility 무형의 사물 보이지 않음

- complexity 일정한 s.w 개발 시 복잡도가 증가함 (고유한 복잡성은 줄일 수 있으나 낮추는데 한계가 있음)

- Uncertainty of “input” 고객이 개발 시 요구하는 변동사항이 불명확하다. 타이밍도 모름.

- changeability 변경 가능성 (고객에게 납품하자마자 진화함 – 업데이트 개념)

- 100프로 자동화 불가

- 소통능력 요구 (통신 요구사항)

- 유지 보수 필요 (관리 문제)

4. software engineering(SE)은 효과적인 소프트웨어를 생성하기 위해 복잡성의 모든 원천을 제거하는 것임.

5. software의 정의

- 품질 좋은 sw를 생성함에 있어 경제적인 생산(클라이언트 매출 성공)을 위한 공학, 과학, 수학 원리 및 방법의 규율적 적용 방법

- 소프트웨어의 개발, 운영, 유지 보수, 폐기까지 전 범위를 아우르는 체계적인 접근법(IEEE)

다양한 버전의 sw를 다양한 사람들이 함께 만들어 나가는 것

- 설계, 유지 보수, 대규모 sw의 설계 구축 및 유지 관리

6. software 수명 주기

: 시스템 공학 -> 요구사항 분석 -> 프로젝트 계획 -> 상위설계　->　하위설계 -> 실행 -> 유지보수 + 품질 보증(각 단계가 잘 수행되고 있는지 감사)

7. software programming 소프트 웨어 프로그래밍 (혼자)

- 개발자 한명

- 작은 사이즈 어플리케이션

- 개발 기간 짧음

- 한명 혹은 거의 없는 stakeholders (sw가 처음 계획되고 만들어지기 시작해서 끝날 때까지 전 과정에 연관이 된 사람들)

- 설계자 = 개발자 = 관리자 = 테스터 = 고객 = 사용자

- 범용적이지 않고 독특하며 유일무이한 시스템

- Built from scratch 밑그림 아무것도 없는 상태에서 시작

8. software engineering 소프트웨어 공학 (팀)

- 개발자 여러명 팀 단위 (다양한 역할을 가진 개발자 팀)

- 복잡한 시스템

- 개발기한 뿐만 아니라 sw활용기간도 김 (무기한 수명)

- 수많은 stakeholders (Numerous)

- 설계자 개발자 관리자 테스터 고객 사용자

- 큰 사이즈의 시스템 (families)

- sw 재사용하여 bulid 업데이트 방식

9. software engineering의 경제 및 관리 측면

- 향상된 R.O.I (Return On Investment) 투자 수익

- 소프트 웨어 생산 = 관리 + 개발 + 유지보수

- 유지보수 비용: 전체(성공) 개발 비용의 60-80%이다.

- 빠른 개발이 항상 바람직한 것은 아님

- 잘못 설계/실행된 sw는 시스템 비용 및 지연에 중요한 비용 요인

10. 4C in SE (소프트웨어 공학 분야에서 연구하고 관리할 4가지 요소)

- Complexity (복잡도)

- Change (변화)

- Cost (비용)

- Communication (의사소통) : 관련자들과 효과적인 소통

11. Complexity(복잡도)

: 소프트웨어가 복잡해지는 이유는 4차 산업혁명

12. Change(변화)

sw는 계속 진화함 따라서 변화는 불가피

sw 개발 중에 어떤 변화가 있었나?

- 요구사항

- 설계/코드

- 사람

13. reliability(신뢰성) : 소프트웨어 품질 목표 중 주어진 시간동안 주어진 기능을 오류 없이 수행하는 정도

14. Cost(비용)

- sw 비즈니스는 무엇을 포함하는가 ?

- sw development (sw 개발)

- sw maintenance (sw 유지관리)

- sw acquisition (sw 획득)

- sw operation (sw 작동)

15. sw 비지니스와 관련된 가장 주된 비용 요소는 무엇인가?

- 과거: 컴퓨터 성능이 가장 높은 비용

- 현재: 소프트웨어 엔지니어의 인건비

16. SE의 목표

양질의 SW시스템을 예산 범위 내에서 적시에 개발하는 것 vs SW에서 수익을 창출하는 것

17. Communication(의사소통)

sw 개발자는 누구와 소통해야 하는가?

- 다른 역할뿐만 아니라 동일한 역할을 가진 다른 소프트웨어 개발자

- stakeholders

18. sw 개발 중 모호함없이 소통을 추진하는 방법

- 자연어와 비교

19. 요약

<SE의 정의>

- (IEEE) : sw개발, 유지보수 및 운영에 체계적이고 규율적이며 정량화 할 수 있는 접근법의 허용

- (NRC Canada) : 프로젝트에 종사하는 사람이 두명 이상일 때 수행하는 작업

- sw 엔지니어가 수행하는 작업

<SE의 목표>

- 예산 범위 내에서 사용자의 요구 사항을 충족하는 고품질 소프트웨어 생산

- 품질 시스템 및 서비스 생산

- 결국 sw를 통해 사업 이익을 얻음

20. Process

- 일이 처리되는 과정이나 공정

- 주어진 일을 해결하기 위한 목적으로 그 순서가 정해져 수행되는 일련의 절차

- 공정효과 : 목적달성

21. sw 프로세스 모델이란?

sw개발과 진화에 관련된 단계 순서를 결정하고 한 단계를 다음 단계로 진행하기 위한 전환 기준을 설정하는 것

- 현재 단계에 대한 완료 기준과 다음 단계에 대한 선택 기준 및 입장이 포함

- 다음에 뭘 할지 언제까지 계속할지에 대한 질문 해결

22. process model의 2가지 효과

- engineer에게 프로젝트 내에서 다양한 기술 활동을 수행해야 하는 순서에 대한 지침 제공

- resource 추정, 마침표 정의, 진행상황 모니터링 등을 가능하게 하는 개발 및 유지관리 framework(틀)을 제공

23. process modle이 왜 중요한가?

- 업계가 시장 출시 기간을 단축하고 생산 비용을 절감하기 위해 서로 다른 프로젝트와 생산성에 걸쳐 성능의 균일성과 같은 본질적 특성에 관심을 갖기 때문

- 하지만 process는 제품의 품질(Quality)에 결정적인 영향을 미치기 때문에 중요

- 즉, process를 제어함으로서 필요한 제품의 품질을 더 잘 제어할 수 있음

Software processes are important that they have a crucial influence on the ( Quality ) of software products, that is, by controlling processes, we can achieve better control of the required ( Quality ) of products.

- process 품질은 제품의 품질로 이어짐

sw development process 소프트웨어 개발 프로세스

: 작업(task)순서의 집합 + 제약조건(일정,예산,자원)을 포함하는 일련의 활동(activity)

작업(task): sw를 개발할 때 일을 수행하는 작은 단위

좁은 의미: 개발할 때 필요한 절차, 과정, 구조

(사용자의 요구사항을 sw시스템으로 구현하기 위한 일련의 활동)

넓은 의미: 절차, 구조, 방법, 도구, 참여자까지 모두 포함

(sw개발 목적을 이루는데 필요한 통합적 수단)

process의 목적

이전에 얻은 노하우를 전달 받음으로써 시행착오 감소 + 빠르게 적응

sw 개발 시 가이드(Guide) 역할

software process model 정의

SE 프로젝트에서 수행되는 활동의 순서와 이러한 활동의 상대적 순서에 대한 설명

소프트웨어 개발 생명주기(SDLC: Software Development Life Cycle)

sw를 어떻게 개발할 것인가에 대한 전체적인 흐름을 체계화한 개념

개발 계획 수립부터 최종 폐기 때까지의 전 과정을 다룸

순차적인 단계로 이루어짐

software process model 목적

- 양질의 소프트웨어 제품을 만들기 위해서

소프트웨어 개발의 전 과정을 하나의 프로세스로 정의

주어진 예산과 자원으로 개발하고 관리하는 방법을 구체적으로 정의

고품질의 소프트웨어 제품 생산을 목적으로 함

software process model 역할

프로젝트에 대한 전체적인 기본 골격(틀)을 세워줌 (Building basic framework)

일정 계획을 수립할 수 있음 (Scheduling)

개발 비용 산정 뿐 아니라 여러 자원을 산정하고 분배가능 (Cost estimation)

참여자 간에 의사소통 기준 정할 수 있음 (Communication)

용어의 표준화를 가능하게 할 수 있음 (Standardization of terminology)

개발 진행 상황을 명확히 파악 할 수 있음 (clear ubderstanding of development prgress)

각 단계별로 생성되는 문서를 포함한 산출물을 활용하여 검토할 수 있게 해줌

CASE tool

CASE : Computer-Aided Software Enginnering 컴퓨터 자원 소프트웨어 공학

소프트웨어 공학의 여러 작업들을 자동화하는 도구

소프트웨어 시스템의 문서화 및 명세화를 위한 그래픽 기능을 제공

자료흐름, 비즈니스 프로세스 등의 다이어 그램을 쉽게 작성

software process model 종류

- Bulid-and-Fix

- Waterfall

- Prototyping

- Incremental & Iterative

- Spiral

- V model

- Agile

<Bulid-and-Fix(Code-and-Fix)>

Bulid-and-Fix(Code-and-Fix) model

요구분석이나 설계단계가 없음

공식적인 가이드 라인이나 프로세스가 없는 개발 방식 요구분석 명세서나 설계 단계 없이 간단한 기능만을 정리하여 개발하는 형태

일단 코드부터 작성하여 제품 만든 후 요구 분서, 설계, 유지보수에 대해 생각

Bulid-and-Fix(Code-and-Fix) model의 사용

개발자 한 명이 단시간에 마칠 수 있는 경우에 적합

대학 수업의 한 학기용 프로젝트 정도

Bulid-and-Fix(Code-and-Fix) model의 단점

정해진 개발 순서나 각 단계별로 문서화된 산출물이 없어 관리 및 유지보수가 어렵다

좋은 sw architecture 만들 수 없음

일을 효과적으로 나눠 개발할 수 없음, 프로젝트 진행 상황 파악 어려움(문서 없음)

계속적 수정으로 인해 프로그램의 구조가 나빠져 수정이 매우 어려워짐

< Waterfall model >

Waterfall model : Documents 요구사항 변화 적은 프로젝트 적합

: 요구 -> 설계 -> 실행 -> 검증 -> 유지

1970년도 대중화

표준 산업 관행

문서 지향 (문서화된 산출물!!!)

한 단계의 출력이 다음 단계의 입력

많은 변형 존재

① Requirment analysis and specification 요구사항 분석 및 사양

어플리케이션에 필요한 품질 식별 (Functional and Nonfunctional)

무엇을 해야하는지 명시 (어떻게 해야하는지가 아님)

고객과 설계자 모두 사용

설계된 문서 : 문제에 대한 정의, 대체 솔루션 및 예상되는 이점, 각각 필요한 리소스 및 비용, 제공 날짜

=> SRS 도출

② Design and specification 설계 및 사양

문제에 대한 해결책 제안

시스템을 모듈로 분해

모듈간의 관계를 지정

각 모듈을 설계

High-level(Preliminary, architectual), Low-level(detailed), User interface 3가지로 나눌 수 있음.

=> SDS 도출

③ Coding and testing (코딩 및 테스트)

설계를 코드로 이상적으로 변환

Unit(module) testing 단위(모듈) 테스트

integration testing 통합 테스트

system testing 시스템 테스트

Acceptance testing 합격 테스트

=> program source code 프로그램 소스코드 도출

④ Delivery and maintenance (전달 및 유지보수)

가장 많은 비용과 노력이 투입됨

유지보수 유형

1) Corrective 수정: 20% 시스템을 운영하면서 발견한 오류 수정 활동

2) Adaptive 적응: 20% 환경변화(HW,OS등)를 반영하기 위한 활동

3) Perfective 완벽: 60% 새로운 기능 추가, 성능 개선을 위한 확장 활동

4) Preventive 예방: 미래의 유지보수를 용이하게 하기 위한 변경 활동

요구사항 분석은 심각한 문제의 원인이다.

많은 오류는 시스템이 전달될 때까지 제거되지 않는다.

제품의 변경사항을 통합하는 것이 어렵다.

⑤ Documentation 문서 중심

⑥ Verification 검증

응용 프로그램의 품질을 모니터링

각 단계가 끝날 때 수행

방법 (Review, Walk-through, Inspection)

Inspection : 체계적으로 정의된 절차를 기반으로 결함을 발견하기 위해 훈련된 엔지니어에 의해 수행되는 산출물의 동료 검토(peer review)를 의미

⑦ Management 관리

process 조정

정책 정의

프로세스에 영향을 미치는 모든 resource 제거

Waterfall model 특징

Linerar(순차적인): 개발은 분석에서 코딩까지 순차적으로 진행

Rigid(엄격한): 각 단계의 결과는 다음 단계로 진행하기 전에 중지

Monolithic(획일적이고 자유성이 떨어지는): 모든 계획은 단일 납품일을 지향

Waterfall model 장점

관리가 용이함 (Ease of management)

체계적인 문서화

요구사항의 변화가 적은 프로젝트에 적합

Waterfall model 단점

이전 단계를 완료해야 각 단계 수행 가능

각 단계의 결과물이 완벽한 수준으로 작성되어야 다음 단계에 오류를 넘겨주지 않는다

사용자가 중간에 가시적인 결과를 볼 수 없어 답답함

< Prototyping >

prototyping

: 대량 생산에 앞서 미리 제작해보는 원형 또는 시제품, 제작물의 모형

sw개발에서의 prototype: 정식 절차에 따라 완전한 소프트웨어를 만들기 전에 사용자의 요구를 받아 일단 모형을 만듦. 이 모형을 사용자와 의사소통하는 도구로 활용

prototype(만들어진 모형)을 통해 사용자의 요구사항 만족 여부 판단 후 최종 시스템 구현

프로토타입 개발 과정 + waterfall

prototype process

사용자의 요구사항을 반복적으로 반영하여 최종 프로토타입 생성

최종 프로토타입을 통해 결정된 사용자의 요구를 가지고 다시 처음부터 본격적으로 제품 개발 (최종 프로토타입은 버림)

① Requirement analysis (요구사항 정의 및 분석)

- 1차 개략적인 요구사항 정의 후 반복하면서 최종 프로토타입 개발

② Prototype design (프로토타입 설계)

사용자와 대화할 수 있는 수준의 설계

입출력 화면을 통한 사용자 인터페이스 중심 설계

③ Prototype development (프로토타입 개발)

완전히 동작하는 완제품 개발하는 것이 아님

입력 화면을 통한 사용자의 요구 항목 확인

출력 결과를 통해 사용자가 원하는 것인지 확인

④ Prototype evaluation (사용자에 의한 프로토타입 평가)

프로토타입 평가 -> 추가 및 수정 요구 -> 프로토타입 개발 -> 프로토타입 평가 ...

⑤ Implementation (실행)

최종 프로토타입 개발

Prototype 장점

반복된 요구사항 정의를 통해 사용자 요구가 충분히 반영된 요구 분석 명세서 작성

오류를 일찍 발견하는데 큰 도움

초기 프로토타입 사용을 통한 새로운 요구사항 발견

프로토타입 사용을 통합 완성품의 예측 가능

Prototype 단점

반복적 개발을 통한 투입 인력 및 비용 산정의 어려움

프로토타입 과정에 대한 통제 및 관리의 어려움

중간 산출물 생성의 어려움, 문서 관리의 어려움

불명확한 개발 범위로 인한 개발 종료 및 목표의 불확실성

< Incremental & Iterarive model >

Incremental (점진적으로 증가하는 모델)

: 애매모호한(정해져있지 않은) 아이디어를 현실화, 구체화함. 갖춰져있지 않은 상태를 정제된 상태로 (전체 파트 다 그리고 점점 세부화)

Iterative model (반복하는 모델)

: 반복하는 모델 (작은 결과물에서 반복해서 큰 결과물로), 브랜드 측에 결과물을 제공

delivery에 초점 (전체 밑그림 구조화 후 특정한 파트를 완벽하게 반복적으로)

Incremental & Iterarive model 특징

단계적으로 진행하는 방식 (Stepwise development)

waterfall model에서 소개되어 있는 규율, 원칙을 각각 each bulid 단계에서 유지함 (Incremental에 가까움)

하나의 단계 stage에서 waterfall 방식 적용 (각 단계마다 waterfall 적용)

각 단계가 끝나면 delivery하여 고객에게 검사와 리뷰 (승인)

Incremental & Iterarive model 장점

사용자에게 새제품에 적용할 시간을 제공

작업이 반복적으로 진행되면서 사용자에게 피드백 (사용자 변경사항, 추가 요구사항 등)을 받을 수 있음.

변경사항 수용이 쉬움

phased delivery: 비용이 많이 들지 않음

Incremental & Iterarive model 문제

추가적인 작업물이 기존의 소프트웨어 구조에 반드시 통합되어야 함

설계 신경써서 해야함 (장점이 될 수도)

자칫하면 bulid&fix model처럼 될 수 있음

< Spiral model >

Spiral model 나선형 모델 – 위험 분석

planning 계획 -> risk analysis 위험 분석 -> engineering 개발 -> customer evaluation 고객 평가

Risk 위험 요소 예

번번히 변경되는 요구사항

팀원들의 경험 부족

결속력이 떨어지는 팀워크

프로젝트 관리 부족

개발자의 이직

발주사의 재정적 어려움

예상을 빗나간 투입 입력

개발 기간의 부족

개발비의 초과

① Planning abjectives (계획 및 요구 분석 단계)

사용자의 개발 의도 파악, 프로젝트의 명확한 목표 수립

제약 조건의 대안을 고려, 기능/비기능 요구사항 정의 및 분석

② Risk analysis (위험 분석 단계)

소프트웨어 개발을 진행함에 있어 방해될 수 있는 모든 요소

③ Development (개발 단계)

n차 프로토타입 개발

④ Evaluation (사용자 평가 단계)

프로토타입 평가 -> 추가 및 수정 요구 -> 프로토타입 개발

Sprial model 장점

사전 위험 분석을 통한 돌출 위험 요소 감소 (프로젝트 중단 확률 감소)

사용자 평가에 의한 개발 방식 (요구가 충분히 반영된 제품, 사용자의 불만 감소)

Sprial model 단점

반복적 개발에 의한 프로젝트 기간 연장의 가능성

반복 회수의 증가에 따른 프로젝트 관리의 어려움

위험 관리의 중요

< V model >

V model (V 모델) - 작은 시스템에 유용 (요구사항 명확)

waterfall model + Extended testing phases (테스트 단계 추가 확장한 모델)

각 개발 단계를 검증하는 데 초점

테스트 단계 : 요구분석, 설계, 디자인, 실행 모든 단계에서 테스트

테스트 계획은 테스트 수행 직전에 수립하는 것이 아니라 각 단계 종료 시점에서 결정

V model 장점

결과물에 대한 단계적인 검증과 확인을 통해 개발 소프트웨어에 대한 오류를 줄일 수 있음

요구사항이 정확히 이해되는 작은 시스템 개발 시 매우 유용

V model 단점

waterfall model을 적용함으로써 개발 활동에 대한 반복이 허용되지 않기 때문에 변경을 다루기 쉽지 않음.

--------------------------- traditional process model -------------------------

< Agile >

Aglie (날렵한, 민첩한)

traditional process model보다 가벼운 개발 프로세스

Build-and-Fix와 상세하고 구체적인 계획을 수립하여 진행하는 개발 방식들 사이에서 타협점을 찾고자 하는 방법론

고객의 요구에 민첩하게 대응하고 그때그때 주어지는 문제를 낭비없이 풀어나가는 방법론

Aglie 선언문 (기본 가치)

정형화된 process 도구 중심이 아닌 개개인과의 상호 소통 중시

문서 중심이 아닌 실행 가능한 소프트웨어 중시

계약과 협상 중심이 아닌 고객과의 협력 중시

계획 중심이 아닌 변화에 대한 민첩한 대응 중시

Aglie software development 특징

가치 있는 소프트웨어를 빨리 지속적으로 제공하여 고객 만족도 증가

요구사항 변화에 빨리 대응

작동중인 소프트웨어가 자주 제공됨 (몇 개월이 아니라 몇 주)

비즈니스 담당자와 개발자 간의 긴밀한 협력

대면 미팅이 가장 좋은 커뮤니케이션

지속 가능한 개발, 일정한 속도 유지 가능

자체 구성 팀에서 최고의 아키텍처, 요구사항 및 설계 제공

정기적으로 팀은 어떻게 하면 더 효과적으로 될 수 있는지 반성하고 그에 따라 조정

Agile software development 종류

Extreme Programming(XP)

Scrum

Dynamic systems Develop Method (DSDM)

Lean

Kanban

Scrumban

< Extreme Programming(XP) >

Extreme Programming(XP) : 팀워크 중요

프로그램 자체를 Extreme하게 극한으로 올려서 작업 (문서x)

프로그램 중심으로 모든 것을 처리

팀워크를 강조 (관리자, 고객, 개발자는 모두 동등한 파트너)

소통, 단순성, 피드백, 존중, 용기 5가지 방법으로 개선

Extreme Programming(XP) 실천사항

Pair programming 짝 프로그래밍: 하나의 컴퓨터에 개발자 2명 (쌍으로)

small release 작은 배포: 최소한의 기능만 먼저 개발하여 배포

Incremental planning 증분 계획: 계획 계속 확장하여 붙여나감

< Scrum >

Scrum

복잡한 제품을 개발, 제공 및 유지하기 위한 민첩한 framework

개발에 대한 실천사항 보다 반복적인 개발을 관리하는데 초점

Scrum main artifact 주된 산출물

1) Product backlog: 제품 기능 우선순위 정리

: 소프트웨어 product에서 사용자에게 제공하는 기능이 어떤 기능이 있는지 정리

2) Sprint backlog: 하나의 sprint동안 개발할 목록

: 각각의 sprint들이 어느정도 시간이 걸리고 언제부터 시작해서 끝나는지를 적어둔 목록

sprint: 하나의 workig software 작업하는데 필요한(반복해야 하는 기간) 1~4주

3) Burndown chart: 개발을 완료하기까지 남은 작업량 보여주는 그래프

Scrum 특징

하나의 sprint(1~4주 단위의 반복 개발 기간)마다 주된 산출물 관리하며 진행

하나의 sprint마다 일일 스크럼 (매일 15분~30분간 진행상황 공유 회의)

하나의 sprint가 끝나면 결과물 가지고 고객 확인과 피드백

< Danamic Systems Development Method (DSDM) >

DSDM

sw 전체 솔루션 개발하기 위해 들어간 시간의 20% 이내에 전체 솔루션의 80%정도가 개발

ex) sw 기능 100개 중요한 기능 80개 개발 (우선순위 높은 순서): 전체 100개 개발하는 데 걸리는 시간에 20%만 들이면 100개의 기능 중 80%는 개발이 됨 (시간 100% 쓸 필요 x)

ex) 1년 중 12개월의 20% = 약 2개월이면 충분 (핵심 기능 80개)

80% (Must have + Should have)

MoSCow rules 무엇을 먼저 개발할지 우선순위

M: Must have requirements 반드시 수행

S: Should have if all possible 가능하면 반드시

C: Could have but not critical 있으면 좋지만 중요하진 않음

W: Won’t have this time, but potentially later 딱히 만들 필욘 없지만 담에 기회되면

< Lean >

Lean

: sw process 기반 아님 (HW 개발에 있어서 유용)

빠른 피드백을 통한 제품 개발과 실험, 정확한 성과 측정을 통해 고객이 원하는 방향 집중

만들기-측정-학습의 반복을 통한 지속적인 혁신

Lean 핵심

Eliminating Waste 낭비되는 요소들을 다 식별하여 제거 (쓸데없는 작업 x)

실수에 대한 학습 확대, 확산

결정할 때 가장 늦게 신중하게

Delivery 할 때는 가장 빨리

팀의 성과를 극대화

Bulid의 품질 자체를 개발 업체 내에 확보

전체에 대해서도 최적화

------------------------------------Agile------------------------------------

< DevOps >

DevOps 등장

-> Dev(개발자): 새로운 개발, 변화, 빠른 개발 vs Ops(운영자): 안정적, 장애 x, 빠른 대응

개발과 운영팀에 트러블이 생기므로 처음부터 개발할 때 sw를 함께 고려, 협업

Agile은 시스템 규모가 커지고 복잡해지면서 안정적인 운영을 필요, 빠른 개발 방식에도 소비자들의 불만 태클이 걸림

1) 운영자 측 문제: 빠른 개발은 종종 장애를 일으킴, 새로운 기술 변화 부담, 사용자의 빠른 피드백 반영 필요

2) 개발자 측 문제: 비대해진 기존 시스템 파악하기 어려움, 운영자 도움 부족한 상황 속 새로운 기술 구현 문제

DevOps

Development(개발)와 Operation(운영)이 합쳐진 용어

개발과 운영활동에 대한 협업과 활동의 자동화

개발 시스템의 Bulid와 Operation의 반복 과정을 거침

Dev: paln (비용 산정, 설계 계획) -> code (소스 코드 구현) -> bulid (모델을 통합 하나의 sw) -> test (테스팅) / -> Ops: release (최종 산출물을 sw운영팀이 쓰기 위해 github에 업로드) -> deploy (운영을 하기 위한 서버코드 올림) -> Operate -> moniter (모니터링 특정한 상황에서 문제 없는지 문제 발생 시 개발팀으로)

DevOps 특징

지속적 통합과 지속적 배포 (Continuos Integration(CI) / Continuos Deployment(CD))

다수 팀이 큰 규모 sw개발 할 때 결과물 서버로 전달 서버는 전달 받은 결과물 가능하면 매일 통합

올바르게 통합된 컴포넌트는 품질 보증의 단계를 거쳐 생산 후 사용자 배포

통합 주기적으로 이루어지듯 배포도 문제가 없는 한 지속적으로 진행

자동화 (Automation): 오류 신속하고 일관성 있게 해결하기 위해 자동화된 환경을 전체 수명주기에 구축

DevOps 변형

DevSecOps, DevDataOps, MLOps

요약

- 모든 프로젝트에 대해 작동하는 절대적인 솔루션은 없다.

프로젝트에 적합한 모델을 선택할 수 있는 전문 지식 필요

UML : 설계에서 중요한 부분

설계 산출물을 내기 위해 수단적인 언어 필요

통합된 설계 언어 (설계를 위한 Language)

소스 코드보단 그림 형태로

UML 스펙: 객체관리그룹 (OMG: Object Management Group)에서 여러 표기법 통합

객체지향 시스템 개발 분야

통합된 객체지향 모델링 언어를 정의

- 현재 UML의 최신 스펙은 2017년 12월에 출시된 2.5.1 버전

UML용도

쉽게 이해될 수 있도록 sw를 표현해보자

시스템을 만들기 전에 모델을 만드는 것과 같은 역할

시스템 만드는 데도 어휘, 규칙 마련하여 시스템을 개념적, 물리적으로 표현하는 모델 필요

객체지향적인 분석과 설계를 위한 표준으로 인정받는 모델링 언어 UML필요

개발방법론에 관계없이 적용 가능

UML 특징

UML은 가시화 언어이다

소프트웨어 개념 모델을 시각적인 그래픽 형태로 작성

표기법에 있어서는 각 symbol에 명확한 정의 존재

개발자들 사이에 오류 없는 원활한 의사소통 가능

UML은 구축 언어이다

다양한 프로그래밍 언어로 표현 가능 (모든 Language 이용 가능)

명세화된 설계 모델은 프로그램 소스 코드로 변환하여 구축 가능

구축되어 있는 소스를 UML로 역변환하여 분석하는 역공학 가능

Modeling (요구사항 정의, 분석, 설계)

시스템을 구축할 때 개발자가 고민하고 결정하는 모든 활동

Model: Modeling 활동의 결과 (분석 모델, 설계 모델)

Modeling Language

Model을 표현할 때 사용되는 언어

UML은 이러한 목적으로 사용될 수 있게 정의된 모델링 언어 중 하나

Modeling vs Programming

1) Modeling

- 목적: 구축할 시스템의 모습 정의 (설계 단계)

- 세부 수행 활동: 요구사항 정의, 분석, 설계

- 결과물 : Model

표기법: 모델링 언어

2) Programming

목적: 시스템의 실제 구현 (구현 단계)

세부 수행 활동: 소스코드, 편집, 컴파일, 테스팅, 디버깅

결과물: 소스코드를 포함한 구현된 시스템

표기법: 프로그래밍 언어

Modeling 개념

소프트 웨어 개발 : 모델링 (요구사항 정의, 분석, 설계) + 구현

UML(modeling language)은 모델링 + 구현 및 테스트까지 포함

각 단계별로 사용되는 다이어그램 존재

UML 다이어그램

UML 다이어그램: 구조 다이어그램 (structure) + 행위 다이어그램 (behavior)

구조 다이어그램

소프트웨어 시스템 자체가 정적인 상태에서 어떤식의 구조를 갖는지?

class 다이어그램

행위 다이어그램

실제로 소프트웨어가 동적인 runtime 상태에서 (동작 후)

Use case 다이어그램

Sequence diagram 순서 다이어그램

interction diagram 상호작용 다이어그램

행위 다이어그램

소통 다이어그램

상호작용 overview 다이어그램

타이밍 다이어그램

참여자간의 시간적 순서에 따른 상호작용 표현

객체들간의 동작 표시

state machine diagram

sw 시스템이 특정 시점에서 가지는 상태 표현

행위 다이어그램

Use case

외부에서 본 시스템의 view, behavior

sw system이 어떤 행위를 하냐 (어떤 기능을 사용자에게 제공하냐)

시스템에 무슨 서비스 있는지 사용자 관점으로 본 것 (기능 중심)

개발 대상이 되는 시스템이 제공하는 개별적인 기능

시스템의 범위에 해당되어 개발될 시스템의 단위기능

시스템과 외부 액터 사이의 관계

Use case 작성 목적

시스템의 범위를 정하는데 도움 됨

개발 과정을 계획하는데도 사용

요구를 개발하고 검증하는데 사용

테스트 케이스 (하나하나의 기능을 테스팅 하기 위한 기준)을 정의하는 데 기초

사용자 매뉴얼 구성하는데 사용됨

Use case diagram : 시간의 흐름도 아님

Use case들의 모음을 그림으로

system이 어떤 기능을 제공하고 그 기능을 누가 사용하는지 범위가 어디까지인지를 표현

Use case diagram 구성요소

1) System 2) Actor 3) Use case 4) Association

System

만들고자 하는 시스템의 범위

use case를 포함한 직사각형 틀로 표현

시스템이나 모델의 명칭을 사각형 안쪽 상단에 기술

서브 시스템일 경우 <<subsystem>> 이라 기술

Actor

사람 혹은 시스템

sw system을 사용하는 사람

sw system 제공하는 기능을 이용하는 매개체

시스템과 상호작용하는 시스템 외부의 존재

시스템으로부터 서비스를 받을 필요가 있는 외부 요소

외부에 있고 상호작용 하는 “사람”또는 “다른 시스템”

원과 선을 조합하여 사람모양

그 아래에 Actor명 표시

Actor명은 Actor의 역할로 정함

시스템 주된 사용자 누구?

시스템과 상호작용하는 다른시스템 있나? ( 내부가 아니라 외부에 존재 )

시스템 출력에 관심이 있는 사람이나 다른 시스템은 무엇인가 ?

Use case

Actor에게 서비스를 제공하기 위하여 시스템이 수행하는 중요 기능

시스템의 요구사항 보여줌

타원으로 표시 안쪽이나 아래쪽에 use case명 기술

Use case 이름은 “~한다”와 같이 동사로 기술

각 use case가 개발될 기능 하나와 연결될 수 있도록

<주의사항>

단일 use case가 여러 Actor에 의하여 구동될 때

작업이 동일하면 (사용자와 관리자가 주문취소 했을 때 내부 흐름이 동일할 경우에) 같은 사용 사례 (use case)

이벤트 흐름이 다르면 (내부 흐름이 다르면) 다른 사용 사례 (use case)

Association

Actor와 use case 사이의 관계

Actor와 use case 사이의 실선으로 표시

1) Association (연관관계)

액터와 유스케이스 상호작용

실선으로 표시

2) Dependency (의존관계)

① include (포함관계)

유스케이스와 유스케이스

반드시 함께 실행되어야 하는 경우

포함하는 유스케이스 (큰 개념) ----> 포함되는 유스케이스 (작은 개념)

점선 표시 후 <<include>>라고 표기

② extend (확장관계)

유스케이스와 유스케이스

포함 반드시는 아님 경우에 따라 수행하기도 하는 경우

선택적 실행

옵션 느낌

확장 대상 유스케이스 (글을 등록) <---- 확장 기능 유스케이스 (파일 첨부)

3) Generalization (일반화 관계)

액터와 액터

액터와 유스케이스

액터들이 유스케이스와 중복하여 관계가 나타나면 액터들 통합하여 일반화

Use case diagram 작성 순서

시스템 그리기 2. 액터 식별 3. 유스케이스 식별 4. 관계 정의

Use case 작성 팁

소프트웨어가 제공할 기능에 집중

사용자 관점으로 작성

목표 지향적으로 작성

시스템 흐름도와는 다름 (시간의 흐름이 아님)

읽기 쉽게 작성

StarUML

UML이라고 하는 설계 Language를 수월하게 활용해 줄 수 있도록 하기위한 tool

상업적 도구에 준하는 기능을 갖춘 소프트웨어 모델링 도구

요약

< Actor >

시스템과 상호작용을 하는 시스템 외부의 존재이다

시스템과 상호작용 해야한다

시스템 관점에서 바라본 사용자 역할을 뜻해야 한다

시스템의 사용자와 외부 연동되는 시스템을 정확하게 파악할 수 있어야 한다

액터 이름만으로 해당 액터의 역할을 명확하게 이해할 수 있어야 한다.

< Use case >

개발 대상이 되는 시스템이 제공하는 개별적인 기능을 뜻한다

표현된 기능은 시스템의 사용자가 이용한다

기능과 이를 이용하는 액터 사이의 연관관계를 명시적으로 표현한다

시스템의 전체 기능적 요구사항은 표현된 use case로 구성된다

use case는 사용자가 인지할 수 있는 하나의 기능 단위이다

구체적이어야 한다

하나의 독립적인 기능을 구성하는 다양한 세부 상황은 하나의 유스케이스로 표현되어야 함

모든 활성화 액터에게 동일한 기능을 제공해야한다

트랜잭션 성격을 가져야 한다

use case 이름으로부터 해당 use case가 나타내는 시스템의 기능을 명확하게 이해할 수 있어야 한다

< Association >

Actor와 use case 간의 연관 관계는 둘 간의 상호작용을 뜻한다

Association은 반드시 시스템이 제공하는 기능이어야 한다

Association의 방향은 제어 흐름을 뜻해야 한다

Use case diagram 모델링 절차

1) 시스템 상황 확인 (Requirement 분석)

2) Actor 식별

3) Use case 식별

4) Association 식별 및 Use case diagram 작성

5) 유스케이스 명세서 작성 (Use case description)

Use case description 유스케이스 명세서

Use case 보완하기 위한 산출물

각각의 Use case에 대해 해당 Use case가 어떻게 수행되는지를 표현하는 수단

Use case 명세서 항목

1) use case 이름: 가장 간결한 명세로 use case를 통해 제공되는 시스템의 기능을 명확한 동사구 형태로 표현 ex) Log-in

2) summary: 모든 이해관계자는 가장 먼저 use case 명세서의 summary 항목을 통해서 use case를 이해함

유스케이스가 뜻하는 시스템의 기능을 간결하고 명확하게 기술

3) Actor: use case가 동작할 때 필요한 주변 Actor를 이해할 수 있음. Primary Actor와 필요시 Secondary Actor 구분하여 기록

- Primary Actor : 기능 메인 수행

- Secondary Actor: 유스케이스로 나온 결과를 받아서, 사람 아닌 경우 많음 (시스템)

4) pre-condition (선행 조건): use case 시작 시 만족되어야 할 조건, 만족되지 않으면 유스케이스 시작되지 않음

5) post-condition (후행 조건): use case 종료 시 만족해야 하는 조건, 유스케이스의 정상 동작 여부에 대한 최소한의 판단 기준

6) Scenario: use case의 관련 Actor와 System rksdml 상호작용에 대한 구체적인 정의

① 기본 시나리오 (Main succes scenario): 가장 일반적이고 정상적인 하나의 상황

② 확장 시나리오 (Extension scenario): 기본 시나리오가 아닌 모든 다른 시나리오, 일반적이지 않은 특수한 상황 혹은 비정상적인 상황

기본 시나리오로부터 분기

분기된 후 종료되는 경우도 있고 다시 기본 시나리오로 분기되는 경우도 있음

시나리오는 Actor와 System간의 구체적인 상호작용을 명시적으로 정의해야한다.

7) Related information: 성능, 신뢰도, 보안 등 use case와 관련된 비기능적 요구사항 (NFR – Non-Functional Requirements) 기술

요약

< Summary >

유스케이스가 나타내는 전체적인 기능이 명확히 기술되어야 한다

유스케이스와 상호작용 하는 Actor를 기술한다

유스케이스의 일부 기능만을 뜻해서는 안된다

주요 시나리오(기본 및 확장)가 언급되어야 한다

관련 Actor가 언급되어야 한다

Summary에서 시스템 내부의 기능과 Actor와의 상호작용을 상세하게 기술할 필요는 없다

< Actor >

관련 Actor는 use case diagram과 일관되어야 함

< pre-condition >

유스케이스가 정상적으로 수행되기 위해 가정하고 있는 상황을 표현해야 함

선행 조건은 유스케이스의 수행 시작을 위하여 항상 만족 되어야 함

선행 조건은 Actor와 시스템 상태에 대한 제약으로 표현됨

use case의 선행조건은 사용자 인터페이스에 반영됨

< Post-condition >

유스케이스 수행 결과를 후행조건을 통하여 파악할 수 있음

후행 조건은 유스케이스의 수행 완료 후에 만족이 되어야 하는 조건

후행 조건은 입력과 시스템 상태의 변화에 대한 조건으로 기술

< Scenario >

유스케이스와 관련된 모든 Actor와의 모든 상호작용 기술해야함

시나리오는 명확하고 이해가 용이한 문장 스타일로 기술

개발자의 기술적인 용어를 사용하지 않고 도매인의 용어를 사용

각 스텝의 주어는 System 또는 Actor를 이용하여 능동태의 문장으로 기술

한 스텝에는 System 또는 하나의 Actor에 의한 기능/행위를 기술

System과 Actor와의 입/출력이 명확하게 기술

각 스텝은 Actor와 System간의 입/출력 및 시스템의 기능의 궁극적인 목적을 기술

Actor가 인식할 수 없는 시스템 내부의 동작과 Actor와의 입/출력 방법을 상세하게 기술하지 않음

기본 시나리오와 주요 확장 시나리오 모두 기술해야 함

< NFRs >

검증이 가능하도록 명확하고 구체적으로 기술해야 함.