None - Fade - Slide - Convex - Concave - Zoom

소프트웨어 공학적 설계

소프트웨어 공학적 설계

WHAT IS SOFTWARE?

WHAT?

WHAT?

PRODUCT

WHAT IS ENGINEERING?

공업적인 생산에 응용하여
생산력과 생산품의
성능을 향상·발전시키기 위한
과학 기술의 체계적인 학문.
-Wikipedia-

생산품의 성능을 향상

그럼 생산품의 성능이 뭔데?

그럼 생산품의 성능이 뭔데?

1. 복잡도가 낮은
2. 개발 비용, 중복이 최소화된
3. 개발 기간이 짧은
4. 대규모 프로젝트에도 적용가능한
5. 신뢰성이 보장되는
6. 표준화 되어 작업 효율을 보장하는

즉, 소프트웨어 공학은 품질 좋은 소프트웨어를 최소 비용으로 계획된 일정에 맞추어 개발하는 것이다.

어떻게?

- 프로세스와 방법론
- 프로젝트 계획과 관리
- 요구 분석
- 요구 모델링
- 설계
- 아키텍처 설계와 패턴
- UI 설계

- 코딩
- 테스트
- 유지보수
- 품질 보증

어떻게?

- 프로세스와 방법론
- 프로젝트 계획과 관리
- 품질 보증

프로세스와 방법론

프로세스와 방법론

프로세스란? WHAT
방법론이란? HOW

프로세스

프로세스

제품 생명주기 모델


map


map


map


map


map

방법론


map

에자일...?

UML

UML

Unified Modeling Language

UML 왜 쓰는건데?

UML 왜 쓰는건데?

소프트웨어가 비가시적인 영역이기 때문.

UML 왜 쓰는건데?

소프트웨어가 비가시적인 영역이기 때문.
비 개발 인력, 혹은 개발인력 간의 의사소통의 수단.

- 클래스 다이어그램
- 객체 다이어그램
- 유스케이스 다이어그램
- 상태 다이어그램
- 시퀀스 다이어그램
- 활동 다이어그램
- 통신 다이어그램
- 컴포넌트 다이어그램

- 배포 다이어그램
- 복합체 다이어그램
- 구조 다이어그램
- 교류 개요 다이어그램
- 타이밍 다이어그램
- 패키지 다이어그램


map

USE CASE DIAGRAM

SEQUENCE DIAGRAM

다이어그램이 종류가 많은 이유?

다이어그램이 종류가 많은 이유?

모든 참여자를 만족시키기 위해서이다.

요구 공학


map

우리는 무엇이 필요한가?

소프트 웨어 시스템이 풀어야 할 문제를 이해하기 위함.


map

SOFTWARE ARCHITECTURE

SOFTWARE ARCHITECTURE

설계를 통해 반복적인 문제를 해결하기 위한 구조

- 계층형
- MVC / MVVC
- ClientServer
- Pipe & Filter
- PublishSubscribe
- Peer-To-Peer
- 칠판형
- Repository

- Broker


map

MVC PATTERN

DESIGN PATTERN

DESIGN PATTERN

이름

DESIGN PATTERN

동일하게 발생되는 문제 상황에서 사용되는 공통된 해결책에 이름을 정의한 것

DESIGN PATTERN

경험의 공유

DESIGN PATTERN

"바퀴를 다시 발명하지 마라!"

DESIGN PATTERN

이미 만들어진 것을 다시 만들 필요가 없다.

DESIGN PATTERN

일반화 된 문제상황에 대한 해결책이기 때문에
같은 디자인 패턴도 여러 알고리즘으로 구현할 수 있다.

- Creational Patterns
- Structual patterns
- Behavioral patterns


map


map

CREATIONAL PATTERNS

클래스의 인스턴스를 만드는 패턴

STRUCTUAL PATTERNS

클래스, 인스턴스의 관계(interface)를 확실히 하는 것.

BEHAVIORAL PATTERNS

여러 알고리즘이나 기능들이 어떻게
흐르는지
어떤 순서로 소통하는지에 대한 정의

DESIGN PATTERN 왜 써야해?

DESIGN PATTERN 왜 써야해?

효율적이고 좋은 코드를 짜기 위해서.

효율적이고 좋은 코드

- 명확하고 단순한 코드
- 각 모듈과 객체는 최소한의 기능단위
- 재사용성이 높은 코드
- 유지보수가 적은 코드
- 리소스 낭비가 없거나 최소화된 코드

효율적이고 좋은 코드

객체간 응집도는 높이고 결합도는 낮게
요구사항 변경 시, 코드 변경을 최소화하는 방향으로
SOLID 원칙.

코딩

리팩토링, 클린코드
단위 테스트

테스트

테스팅 트로피

- End-to-End 테스트: 사용자 입장에서 전체 애플리케이션이 잘 동작하는 지 테스트하는 것
- Integration(통합) 테스트: 실제 DB, 브라우저 없이 큰 규모의 기능이나 하나의 페이지가 잘 작동하는 지 테스트하는 것

- Unit(단위) 테스트: 기능의 개별적인 단위나 하나의 컴포넌트를 테스트함
- Static 테스트: 구문 오류, 나쁜 코드 스타일, 잘못된 API 사용 등을 잡아줌

유지보수

- 형상관리, 버전관리
- 역공학과 리 엔지니어링
- 유지보수 작업 방법과 지원도구

품질 보증

품질 요구사항을 잘 충족했는지
웹 표준이나 접근성 기준은 충족했는지

품질 보증

- 기능성 - 적절, 정밀, 상호운용, 보안, 호환
- 신뢰성 - 성숙, 고장허용, 회복
- 사용성 - 이해성, 학습성, 운용성
- 효율성 - 시간효율, 자원효율

- 유지보수성 - 분석성, 변경성, 안정성, 시험성
- 이식성 - 적용성, 설치성, 대체성

소프트웨어 공학은 품질 좋은 소프트웨어를
최소 비용으로 계획된 일정에 맞추어 개발하기 위한 노력