Section 2. 소프트웨어 개발 방법론

구조적 방법론 vs 객체 지향 개발 방법론

1. 소프트웨어 개발 방법론 종류

(1) 구조적 방법론

절차지향 객체지향

하향식 상향식

- 절차지향 소프트웨어 개발 방법론
- 제한된 구조에서 코드 생성 및 순차적 실행
- 구조적 방법론 기본 개발 과정

과정	설명
요구사항 분석	- 고객의 요구사항을 끌어내어 명세화 하는 과정
구조적 분석	- 고객이 원하는 기능/환경/데이터를 종합하여 데이터 흐름도 작성
구조적 설계	- 모듈 중심 설계 과정
구조적 프로그래밍	- 순차, 선택, 반복의 논리 구조 구성으로 프로그램 작성

- 구조적 방법론 구성요소 : 데이터 흐름도(DFD), 자료사전(DD), 상태전이도(STD), 소단위 명세서(Minispec)

(2) 정보공학 방법론

- 기업의 주요 부분을 계획, 분석, 설계, 구축에 정형화된 기법들을 상호 연관성 있게 통합, 적용하는 데이터 중심 방법론
- 빠른 결과물 확인이 가능하며 단순 S/W 개발이 아닌 기업의 경영전략에 초점을 둔다.
- 정보공학 방법론 기본 개발 과정

과정	설명
정보전략계획 수립단계	- 현행 업무프로세스와 시스템을 분석하고 미래 아키텍처와 전략계획을 수립
업무영역 분석단계	- 기업의 업무 현황을 분석해서 개념 수준의 데이터와 프로세스를 설계하는 업무분석 단계 - 데이터 모델링 : ERD - 프로세스 모델링 : 프로세스 계층도(PHD), 프로세스 의존도(PDD), 자료흐름도(DFD)
시스템 설계단계	- 실질적으로 시스템을 설계하는 단계 - 논리적 ER 다이어그램으로 데이터를 설계하고 분할 다이어그램, 액션 다이어그램, 의존 다이어그램을 사용해 프로세스를 설계
시스템 구축단계	- 설계명세서를 토대로 데이터베이스를 생성하고, 소스코드를 구현한다.

(3) 객체지향 개발 방법론

- 현실세계의 개체(Entity)를 속성(Attribute)과 메서드(Method)형태로 표현
- 객체, 클래스 간의 관계를 식별하여 설계 모델로 변환하는 방법론
- 분석과 설계, 구현의 전 과정을 객체 중심으로 개발
- 전체 프로세스 방향성 유지와 상속에 의한 재사용성 향상
- 특징 : 캡슐화, 정보은닉, 상속, 다형성, 추상화

(4) CBD(Component Based Development) 분석 방법론

- 재사용 가능한 컴포넌트의 개발 또는 상용 컴포넌트를 조합해 어플리케이션 개발
- 새로운 기능 추가가 쉬운 확장성
- 생산성 및 품질이 향상
- 시스템 유지보수 비용 최소화

(5) 애자일 방법론

- 기존 방법론들이 절차를 중시한 나머지 변화에 빠른 대응을 할 수 없는 단점 개선을 위해 등장
- 애자일 방법론 종류 : XP(eXtreme Programming), SCRUM, FDD, Crystal 방법론 등
- 애자일 선언문

> 공정과 도구보다 개인과 상호작용을
> 포괄적인 문서보다 작동하는 소프트웨어를
> 계약 협상보다 고객과의 협력을
> 계획을 따르기보다 변화에 대응하기를
>
> 왼쪽에 있는 것들도 가치가 있지만,
> 우리는 오른쪽에 있는 것들에 더 높은 가치를 둔다.

(6) 개발 방법론 선택 기준

- 프로젝트 특성 및 규모
- 프로젝트 참여자의 수준
- 가용 자원의 정도(인력, 장비, 시간, 비용)
- 요구사항의 명확도
- 위험도

2. 소프트웨어 개발 모델

(1) 폭포수 모델(Waterfall Model)

- 계획, 분석, 설계, 구현, 테스트, 운영 등 전 과정을 순차적으로 접근하는 개발모델 선행순차 모델
- 각 단계의 검증 후에 다음 단계를 진행한다.
- 각 단계가 순차적으로 진행되며, 병행되거나 거슬러 반복 진행되지 않는다.
- 가장 오래된 모형으로 적용 경험과 성공사례가 많다.
- 요구사항의 변경이 어렵다.
- 단계별 정의가 분명하고, 단계별 산출물이 명확하다.

(2) 프로토타이핑 모델(Prototyping Model)

- 고객이 요구한 주요 기능을 프로토타입으로 구현하여 완성해가는 모델
- 개발자가 구축할 소프트웨어의 모델을 사전에 만들어 요구사항을 효과적으로 유도하고 수집한다.
- 프로토타이핑에 의해 만들어진 프로토타입은 폐기될 수 있고, 재사용될 수도 있다.

- 순서
  - 계획수립 → 프로토타입 개발 → 사용자 평가 → 구현 → 인수
- 장점
  - 사용자의 요구사항을 충실히 반영할 수 있다.
  - 비교적 빠른 기간 안에 사용자가 평가할 수 있는 결과물이 만들어진다.
  - 오류를 초기에 발견할 수 있다.
  - 변경이 용이하다.
- 단점
  - 최종적으로 시간과 비용이 훨씬 많이 들 수 있다.
  - 사용자가 실제 제품과 혼동할 수 있다.
  - 문서작성이 소홀해질 수 있다.
  - 프로토타입 폐기에 따른 비용이 든다.

(3) 나선형 모델(Spiral Model)

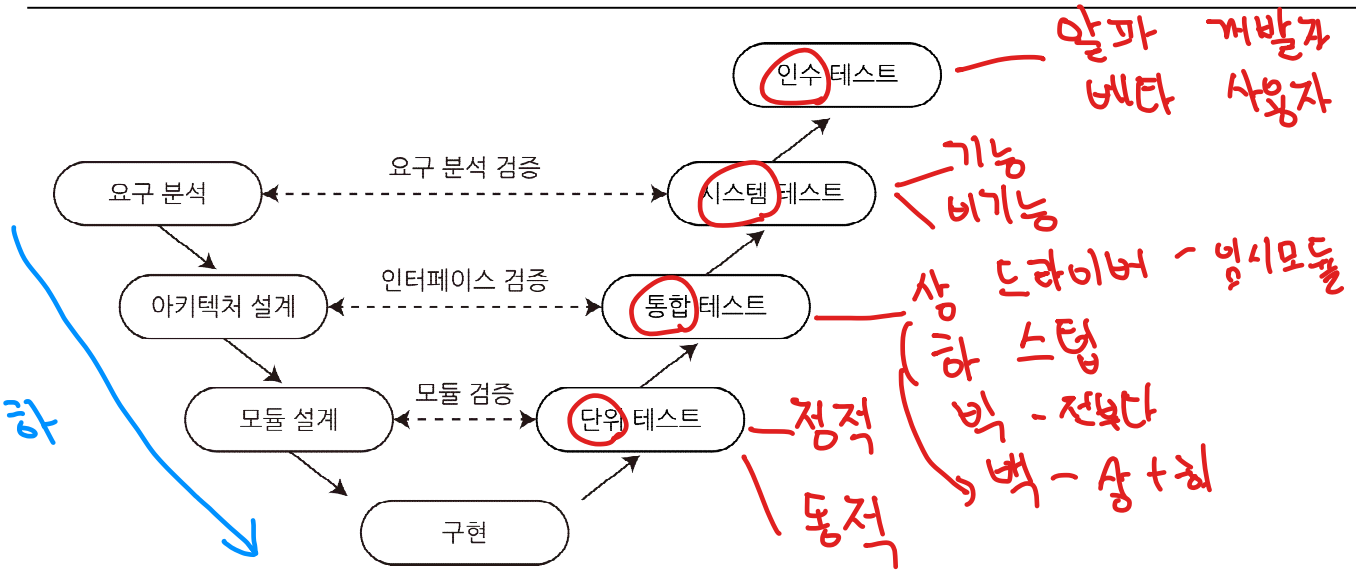
- 폭포수 모델과 프로토타이핑 모델의 장점을 수용하고, 위험 분석을 추가한 점증적 개발 모델
- 프로젝트 수행 시 발생하는 위험을 관리하고 최소화하려는 것이 목적
- 대규모 프로젝트 및 위험 부담이 큰 시스템 개발에 적합
- 순서
  - 계획수립(planning) → 위험분석(Risk Analysis) → 공학적 개발(Development) → 평가(Evaluation)
- 장점
  - 위험분석 과정으로 위험성이 큰 프로젝트를 수행할 수 있다.
  - 고객의 요구사항을 보다 더 상세히 적용할 수 있다.
- 단점
  - 시간과 비용이 많이 들 수 있다.
  - 반복 단계가 길어질수록 프로젝트 관리가 어렵다.

(4) RAD(Rapid Application Development) 모델

- 매우 짧은 개발 주기를 강조하는 점진적 소프트웨어 개발 방식
- 강력한 소프트웨어 개발 도구를 이용하여 매우 짧은 주기로 개발을 진행하는 순차적 소프트웨어 개발 프로세스
- CASE(Computer Adied Software Engineering) 도구를 이용해 시스템을 개발
- 개발 기간이 60일~90일 정도로 짧다.
- 기술적으로 위험이 적고 빠른 개발이 요구될 때 사용이 적합

(5) V 모형

- 폭포수 모델에 시스템 검증과 테스트 작업을 강조
- 높은 신뢰성이 요구되는 분야에 적합함



(6) 4세대 기법(4th Generation Techniques)

- CASE등의 자동화도구를 이용하여 요구사항 명세로부터 원시코드를 자동으로 생성

3. 애자일(Agile) 방법론

(1) 애자일 방법론의 개념

- 애자일 방법론은 소프트웨어 개발 방법에 있어서 아무런 계획이 없는 개발 방법과, 계획이 지나치게 많은 개발 방법들 사이에서 타협점을 찾고자 하는 방법론
- 애자일 개발 방법론이란 어느 특정 개발 방법론을 가리키는 말은 아니고 애자일 개발을 가능하게 해주는 다양한 방법론 전체를 일컫는 말
- 경량(Lightweight) 프로세스라고도 함
- 프로젝트를 시작한 후 끊임없이 개선 노력

(2) 애자일 프로세스의 등장배경

- 기존 소프트웨어 개발 방법론의 문제점을 개선하기 위해서 등장
- 기존 소프트웨어 개발 방법론의 주요 문제점
  - 계약과 계획준수를 중요시하는 갑과 을의 문화가 지배적
  - 문서를 중시함
  - 프로세스나 툴 적용을 중시함
  - 성과가 나쁠 때 계획 또는 통제의 실패로 인식함

(3) 애자일 선언문

- 공정과 도구보다 개인과 상호작용을
- 포괄적인 문서보다 작동하는 소프트웨어를
- 계약 협상보다 고객과의 협력을
- 계획을 따르기보다 변화에 대응하기를
- 우리는 왼쪽 항목의 가치를 인정하면서도 오른쪽 항목을 더 중요하게 여긴다.

(4) 애자일 특징

- 고객과 개발자의 지속적인 소통을 통하여 변화하는 요구사항을 신속하게 수용
- 개발자 개인의 가치보다는 팀의 목적을 우선시 하며 고객의 의견을 가장 우선
- 팀원들과의 주기적인 회의와 제품을 시연함으로써 소프트웨어를 점검
- 작업 계획을 짧게 세우고, 반복적으로 수행함으로 변화에 유연하게 대처

(5) 애자일 방법론 종류

① XP(eXtream Programming)

- 특징
  - 문서보다는 코드를 중시하고, 5가지 핵심가치와 12개 실천 항목이 존재
  - 개발을 세분화 하여 1~3주의 반복으로 개발을 진행
- XP 5가지 핵심가치
  - 용기 : 고객의 요구사항 변화에 능동적인 대처
  - 존중 : 개발자의 역량을 존중하고 충분한 권한과 권리를 부여
  - 의사소통 : 개발자, 관리자, 고객 간의 원활한 의사소통
  - 피드백 : 의사소통에 따른 즉각적인 피드백
  - 단순성 : 부가적 기능, 사용되지 않는 구조와 알고리즘 배제
- 12가지 실천사항

실천사항	설명
짝 프로그래밍 (Pair Programming)	하나의 작업을 2명의 프로그래머가 코딩·리뷰 공동 수행
계획 세우기 (Planning Game)	게임처럼 선수와 규칙, 목표를 두고 기획 수행
테스트 기반 개발 (Test Driven Development)	선 단위 테스트 후 실제 코드 작성
고객 상주 (Whole Team)	개발 효율을 위해 고객을 프로젝트 팀원으로 상주
지속적인 통합 (Continuous Integration)	상시 빌드 및 배포가 가능한 상태로 유지
코드 개선 (Design Improvement)	코드 개선 작업 수행(가시성, 성능 등), 불필요한 기능 제거 및 리팩토링
작은 릴리즈 (Small Releases)	짧고 잦은 릴리즈로 고객이 변경사항을 볼 수 있게 함 (잦은 피드백)
코딩 표준 (Coding Standards)	표준화된 관례에 따라 코드 작성
공동 코드 소유 (Collective Code Ownership)	시스템에 있는 소스코드는 팀의 모든 프로그래머가 언제라도 수정 가능
간단한 디자인 (Simple Design)	가능한 가장 간결한 디자인 상태 유지
시스템 메타포어 (System Metaphor)	최종적으로 개발 되어야 할 시스템의 구조를 조망
작업시간 준수 (Sustainable Pace)	일주일에 40 시간 이상 작업 금지, 2주 연속 오버타임 금지

② 스크럼 (SCRUM)

- 소프트웨어에 포함될 기능·개선점에 대한 우선 순위를 부여
- 개발 주기는 30일 정도(스프린트)로 조절하고 개발 주기마다 실제 동작할 수 있는 결과를 제공
- 개발 주기마다 적용할 기능이나 개선에 대한 목록을 작성
- 날마다 15분 정도의 회의
- 항상 팀 단위로 생각한다.

③ 그 외 애자일 방법론

- 크리스털 패밀리(Crystal)
  - 프로젝트의 규모와 영향의 크기에 따라서 여러 종류의 방법론을 제공
- Feature-Driven Development (FDD)
  - feature마다 2주 정도의 반복 개발을 실시
  - 기능 주도 개발
- Adaptive Software Development, ASD
  - 합동 애플리케이션 개발을 사용하는 방법론