1 소프트웨어 아키텍쳐 구축 절차

01 소프트웨어 아키텍쳐 구축 절차

1 요구 사항 분석

- 소프트웨어 개발의 요구 사항 분석 단계와 같음
- 품질 속성과 같은 비기능적인 요구 사항에 더 많은 관심을 둠

  - 요구 사항 취득, 식별, 명세, 분류, 검증
  - 기능적/비기능적 요구 사항 분류 및 명세

01 소프트웨어 아키텍쳐 구축 절차

2 아키텍쳐 분석

01 소프트웨어 아키텍쳐 구축 절차

3 아키텍쳐 설계

① 관점 정의

- 이해 관계자 파악, 이해 관계자 별 관점 정의

② 아키텍쳐 스타일 선택

- Pipe-filter, Mvc, Layer 등의 스타일 혼용 적용 가능

01 소프트웨어 아키텍쳐 구축 절차

3 아키텍쳐 설계

③ 후보 아키텍쳐 도출

- 배경도(Context Diagram) 및
  각 관점별 다이어그램 작성, 아키텍쳐 명세서 기술

01 소프트웨어 아키텍쳐 구축 절차

4 검증 및 승인

① 아키텍쳐 평가

- 아키텍쳐 요구 사항 만족도, 적합성, 품질 속성간 절충 관계 등 평가

② 아키텍쳐 상세화(반복)

- 설계 방법 도출, 설계 패턴 고려

01 소프트웨어 아키텍쳐 구축 절차

4 검증 및 승인

③ 아키텍쳐 승인

- 이해 관계자들이 최종 승인

2 소프트웨어 아키텍쳐 스타일

02 소프트웨어 아키텍쳐 스타일

1 아키텍쳐 스타일에 따라

- 구조, 규칙, 요소, 기법 등이 결정
- 소프트웨어 특성, 전체 구조, 개발 방법을 알 수 있음

02 소프트웨어 아키텍쳐 스타일

2 좋은 소프트웨어 아키텍쳐 설계

- 소프트웨어에 적합한 아키텍쳐 스타일을 선택하고 적용하고 통합하는 것

02 소프트웨어 아키텍쳐 스타일

3 아키텍쳐 스타일을 사용한 설계의 장점

① 개발 기간 단축, 고품질의 소프트웨어 생산

- 많은 시행착오를 줄임으로써 개발 기간을 상당히 많이 단축할 수 있고, 품질 좋은 소프트웨어를 만들 수 있음

② 수월한 의사소통

- 아키텍쳐에 익숙한 개발자끼리 공통된 아키텍쳐를 공유함으로써 자연스럽게 의사소통이 가능함

02 소프트웨어 아키텍쳐 스타일

3 아키텍쳐 스타일을 사용한 설계의 장점

③ 용이한 유지보수

- 개발에 참여하지 않은 사람이 유지보수를 하더라도 비즈니스 로직만 잘 이해하고 있으면 유지보수의 어려움을 많이 줄일 수 있음

④ 검증된 아키텍쳐

- 코딩할 때 이미 개발해놓은 라이브러리 함수나 상용화 된 컴포넌트들을 사용하면 안정적으로 개발 할 수 있음

02 소프트웨어 아키텍쳐 스타일

3 아키텍쳐 스타일을 사용한 설계의 장점

⑤ 구축 전 시스템 특성에 대한 시뮬레이션 가능

- 이미 알려진 아키텍쳐 스타일을 이용하면 시스템을 개발하기 전에 시스템 특성에 대해 시뮬레이션을 해볼 수 있음

⑥ 기존 시스템에 대한 빠른 이해

- 기존 시스템이 어떤 아키텍쳐 스타일을 이용하여 개발되었는지를 알면 그 시스템을 더 빨리 이해할 수 있음

02 소프트웨어 아키텍쳐 스타일

4 아키텍쳐 스타일의 기능

- 소프트웨어 시스템의 구조를 체계적으로 구성하기 위해 기본 스키마를 제시
- 미리 정의된 서브시스템 제공
- 각 아키텍쳐 패턴 간의 책임 명시
- 패턴 간의 관계를 조직화하는 규칙, 가이드라인 제시
- 문제를 소프트웨어 모듈 단위로 분해하는 방법 제시
- 분해한 소프트웨어 모듈 단위가 상호작용하는 방법 제시

3 소프트웨어 아키텍쳐 모델

03 소프트웨어 아키텍쳐 모델

아키텍쳐 모델의 분류

- 기능과 분할 배치에 따라
  - 데이터 중심형 모델
  - 클라이언트-서버 모델
  - 계층 모델
  - MVC(Model/View/Controller) 모델
- 제어관계에 따라
  - 데이터 흐름 모델

03 소프트웨어 아키텍쳐 모델

1 Repository Model(데이터 중심형 모델)

- 특징 : 주요 데이터가 repository에서 중앙 관리

- 구성 : Repository와 여기에 접근하는 서브시스템
  - Repository - 공동으로 활용하는 데이터 보관
  - 서브시스템 - Repository에 접근하여 정보를 저장, 검색, 변경하는 역할

- 대량의 데이터를 공유하는 은행 업무 시스템에 매우 유용한 모델

03 소프트웨어 아키텍쳐 모델

1 Repository Model(데이터 중심형 모델)

장점
- 데이터가 한군데에 모여 있기 때문에
  데이터를 모순되지 않고 일관성 있게 관리 가능
- 새로운 서브시스템의 추가 용이

단점
- Repository의 병목 현상 발생 가능
- 서브시스템과 Repository 사이의 강한 결합

↳ Repository 변경 시 서브시스템에 영향을 줌

03 소프트웨어 아키텍쳐 모델

2 Client-server 모델

- 네트워크를 이용하는 분산 시스템 형태
- 데이터와 처리 기능을 클라이언트와 서버에 분할하여 사용
- 분산 아키텍쳐에 유용
  - 서버 - 클라이언트(서브시스템)에 서비스 제공
  - 클라이언트 - 서버가 제공하는 서비스를 요청(호출)하는 서브시스템

03 소프트웨어 아키텍쳐 모델

3 Layering 모델(계층 모델)

- 기능을 몇 개의 계층으로 나누어 배치
- 구성 : 하위 계층은 서버, 상위 계층은 클라이언트 역할
- 계층 모델 형태로 설계할 때는 상호작용하는 계층 간의 프로토콜을 정의해야 함
- 계층 모델은 계층 간의 역할 분담을 명확히 하여 각 계층을 필요에 따라 쉽게 변경할 수 있음
- 연결된 계층의 인터페이스만 문제없이 만들면 특정 계층을 쉽게 재사용할 수 있음

03 소프트웨어 아키텍쳐 모델

4 Model/View/Controller 모델

- 중앙 데이터 구조
- 같은 모델의 서브시스템에 대하여
  여러 뷰 서브시스템을 필요로 하는 시스템에 적합
- 세 개의 서브시스템으로 분리하는 이유
  : 변경에 대한 영향을 덜 미치도록 하기 위함

  ↳ 즉 UI부분이 자주 변경되더라도
  모델 서브시스템에는 영향을 주지 않기 위함

03 소프트웨어 아키텍쳐 모델

4 Model/View/Controller 모델

특징

- 각각의 서브시스템이 독립적임
- 뷰는 모델의 데이터를 직접 변경할 수 없고, 오직 모델이 제공하는 데이터를 가져올 수만 있음
- 그러나 모델은 뷰에 대한 정보를 알 수 없음
- 따라서 모델은 여러 개의 뷰가 어떻게 처리되는지도 알 필요 없음

03 소프트웨어 아키텍쳐 모델

4 Model/View/Controller 모델

① Model 서브시스템

- 뷰/제어 서브시스템과 독립되어 모든 데이터 상태와 로직을 처리
- 특정 입 · 출력 방식에 영향을 받지 않고, 무언가의 호출에 응답만 함

② View 서브시스템

- 사용자와 직접 대화가 이루어지는 부분으로 데이터를 사용자에게 보여주는 역할

03 소프트웨어 아키텍쳐 모델

4 Model/View/Controller 모델

③ Controller 서브시스템

- 뷰를 통한 사용자의 요청을 적절한 모델 쪽으로 넘겨주고, 모델로부터 받은 응답을 다시 뷰를 통해 사용자에게 돌려주는 역할

03 소프트웨어 아키텍쳐 모델

4 Model/View/Controller 모델

장점
- 관심의 분리
- 데이터를 화면에 표현(뷰)하는 디자인과 로직(모델)을 분리함으로써 느슨한 결합 가능
- 구조 변경 요청 시 수정 용이

단점
- 기본 기능 설계로 인한 클래스 수의 증가로 복잡도 증가
- 속도가 중요한 프로젝트에 부적합

03 소프트웨어 아키텍쳐 모델

5 Pipe and filter 모델(데이터 흐름 모델)

- Filter
  - Data Stream을 한 개 이상 입력 받아 처리(변환)한 후 Data Stream 하나를 출력

- Pipe
  - Filter를 거쳐 생성된 Data Stream 하나를 다른 Filter의 입력에 연결

03 소프트웨어 아키텍쳐 모델

5 Pipe and filter 모델(데이터 흐름 모델)

특징

- 이 모델은 필터에 해당되는 서브시스템이 하나의 데이터를 입력으로 받아 처리한 후 그 결과를 다음 서브시스템으로 넘겨주는 과정을 반복함
- 일반적으로 데이터를 변환하는 시스템에서 주로 사용하며, 전체적인 변환 작업은 독립적인 단계로 나누어질 수 있음

03 소프트웨어 아키텍쳐 모델

5 Pipe and filter 모델(데이터 흐름 모델)

적용분야

- 이 모델은 이미지 프로세싱 시스템, 컴파일러의 순차적인 변환 처리기, 유닉스의 셸(Shell) 등 파이프와 필터를 조합하여 만드는 아키텍쳐에 적합하고, 사용자의 개입 없이 데이터의 흐름이 전환되는 경우에 사용됨

장점

- 필터 또는 파이프 단위로 나누어 개발할 수 있기 때문에 동시 개발이 가능함