시스템 : 소프트웨어, 하드웨어, 네트워크

인프라 설계

* 네트워크 설계
* 인프라 설계 (클라우드 활용)

소프트웨어 설계

* 데이터베이스 설계
* 미들웨어의 활용
* 애플리케이션 설계

아키텍처 설계

* 소프트웨어 아키텍처
  + 컴포넌트 : 시스템의 기본 구조를 구성하는 요소
  + 컴포넌트들과 환경 간의 관계
  + 컴포넌트가 연동하기 위해 필요한 설계 개념에 따른 시스템의 기본 구조
* 목적
  + 유지보수성 향상
  + 적합성 향상

마이그레이션 설계

* 개발한 시스템이 프로덕션 환경에서 작동하도록 배치하고 데이터 등을 준비하기 위한 설계
* 가장 신경 써야 하는 부분 : 데이터베이스에 저장된 데이터를 마이그레이션하는 것

운영 설계

* 마이그레이션이 완료된 시스템을 어떻게 운영할 것인지, 장애가 발생할 때 어떻게 대처할 것인지를 설계

설계는 누가? : 프로그래머, 프로젝트 리더, 시스템 개발자 혹은 아키텍트가 역할을 겸함

개발 프로세스 : 시스템 개발의 작업절차, 산출물, 개발 프로젝트 운영 방법을 정의

작업의 입력과 출력

기본적인 개발 흐름

* 업무 흐름 분석
* 유스케이스 추출
* 이터레이션 계획 -> 유스케이스 분석 -> 설계 -> 개발
* 통합 테스트
* 릴리스

설계의 목적

* 요구사항 정의 내용을 시스템에서 어떻게 구현할 것인지 검토
* 요구사항 정의에서 명확하게 드러나지 않은 시스템 기능 검토
* 프로젝트 이해관계자 간 정보 공유
* 시스템 품질 향상
* 유지보수를 위해 설계 정보 기록

좋은 설계 : 목적, 정보의 연관성, 정보의 상세 내용 기술한 것 (요구사항을 명확한 분리 필요)

설계서의 목적

* 시스템 설계 품질 향상
* 내용 공유, 작업 분담 가능
* 인수인계 하기 쉬움

설계서 종류 : 관리용 설계서 / 개요 설계서 / 개발용 설계서

내부설계 : 입력과 출력 사이에서 이루어지는 내부 처리를 설계

외부설계 : 시스템이 제공해야 하는 기능을 구체적으로 설계

+ 사용자나 외부 시스템에 제공하는 기능 / 인터페이스가 해당

+ 가장 중요한 목적은 시스템의 기능을 명세하는 것

* 유스케이스 분석 (설계 과정에서도 유스케이스 알고 있어야 함)
  + 포함 최소 내용 – 이름 / 주요 액터 / 주요 시나리오 / 확장 시나리오
  + 시나리오에 등장하지 않는 액터도 존재
  + 유스케이스는 주요 액터 관점에서!
  + 사전 조건, 사후조건
  + 트리거 : 유스케이스를 실행하는 이벤트
  + 비즈니스 규칙
* 개념 모델링
  + 유스케이스 -> 시스템 동적 행위 / 개념 모델 -> 시스템 정적 구조
  + 아키텍처 : 행위를 구현하기 위한 구조 / 개념 모델 -> 데이터 구조
  + 개념 모델로 표현 : 이름, 연관성, 연관성의 다중성 정리하기
* 용어집
  + 개념 모델링 중 다양한 개념, 속성 등장
  + 때때로 난해한 업계 용어를 개념 이름으로 채택하는 경우 있음
* 화면 설계 : 시스템 담당자가 기술적 관점에서 설계 / UI 설계 정책 수립
  + 화면 항목 패턴
  + 화면 전환도 작성 (화면 전환 다이어그램 작성)
  + 화면 목록 작성 / 화면 목업 작성 / 화면 입력 검사 명세서 작성
* 외부 시스템 인터페이스 : 외부 시스템과 데이터를 교환하는 부분
  + 동기식 : 요청과 처리 동시 수행 / 비동기식 : 요청된 타이밍 보다 늦게 처리
  + 외부 시스템과의 연동
* Batch 설계 : 일정 시간이 되었을 때 또는 수동으로 실행되는 프로그램
  + 실행 타이밍, 실행 제어 / 작업 제어, 트랜잭션, 복구
  + 가장 어려운 점 : 성능과 예외 처리
* 데이터베이스 논리 설계 : 테이블, 컬럼, 키 정의 등 수행 (ERD)
  + 정규화 : 데이터 중복을 최소화하고, 데이터 일관성을 유지하는 과정
  + 1차 : 테이블 내 모든 컬럼 원자값
  + 2차 : 부분 함수 종속성 제거
  + 3차 : 이행적 종속성 제거
  + 인조 키 / 자연 키 – 기본키 결정하기
  + 관계를 외래키로 설정하기
  + 다 대 다의 관계를 관계 테이블로 만들기
  + 상속은 관계형으로 구현 (추상 클래스와 구상클래스 별로 테이블 만들기)
* 비기능 요구사항 정의
  + 사용자가 기능을 이용할 때 시스템의 품질과 성능을 규정함
  + 운영의 문제
  + 기능성 / 신뢰성 / 사용성 / 효율성 / 유지보수성 / 이식성 / 기능성
* 시스템 인프라 설계 및 배치 설계
  + 시스템 인프라 설계 : 시스템을 구현하기 위해 네트워크와 하드웨어 구성
  + 배치 설계 : 시스템 어떻게 배치 -> 개발 이전에 수행해야 함

소프트웨어 아키텍처

* 아키텍처 : 비즈니스 요구사항을 만족하는 시스템 구축을 위해 전체 시스템 구조를 정의한 문서
* 컴포넌트, 컴포넌트 간의 관계, 컴포넌트가 다루는 정보를 정의
* 토가프 아키텍처 프레임워크
  + 표준화된 방법론
  + 포괄적 범위
  + 중복주자 및 복잡성 감소
  + 지속적인 거버넌스
* 아키텍트 : 아키텍처를 그리는 사람
  + EA / AA / TA / SA / DA
* 레퍼런스(샘플) 아키텍처
  + Common Architecture : 업무 도메인에 종속적이지 않은 범용 아키텍처
  + Industry Architecture : 특정 업무 도메인에 종속적임
  + Enterprise Architecture : 특정 회사 환경, 요구사항 종합적으로 반영
* 비즈니스 아키텍처 : 조직 차원의 비즈니스 측면을 전체적으로 정의, 관리하는 체계
  + 서비스에 대한 소개
  + 시스템에 대한 간략한 전체 구조
  + 사용자에 대한 도메인 모델
  + 주요 기능에 대한 흐름 (주요 기능 정의 : 핵심 기능 간략하게 이해 쉽게 작성)
  + 시장 현황과 차별화 전략
  + 비즈니스 로드맵과 일정
* 비즈니스 도메인 모델 (간략하게 요약하는 것이 좋음)
  + 시스템 사용자와 사용자 간의 관계 정의
  + 시스템에 정의된 Asset간의 관계 정의
  + 주요 업무 프로세스 정의
  + 상태 전의

시스템 아키텍처

: 아키텍처 설계 원칙, 애플리케이션 아키텍처, 테크니컬 아키텍처,

솔루션 아키텍처, 데이터 아키텍처

아키텍처 설계 시 주의 사항 : 소통을 하기 위해 아키텍처 문서를 만드는 것!

애플리케이션 아키텍처

* 컴포넌트 : 특정 기능, 역할 수행을 위해 논리/물리적으로 구분된 소프트웨어 단위
* 컴포넌트간의 관계 : 정적 관계 / 동적 관계
* 호출 순서
* 통신 인터페이스
* 정적 아키텍처 : 애플리케이션을 구성하는 컴포넌트들 정의 / 계층별 정의 상세화
  + 컴포넌트 간의 구조적 관계를 어떻게 구상할 것인가를 정의
  + OLAP
* 동적 아키텍처 : app이 실제로 실행 될 때 어떤 방식으로 요청이나 데이터가 흐르는지, 어떤 순서로 컴포넌트 들이 동작하는지 등을 정의
* 인터페이스 정의서
* 상세 아키텍처 : 구현 상세 설계, 예외 처리 및 에러 전략, 배포/운영 전략
* 테크니컬 아키텍처
  + 하드웨어 아키텍처
  + 솔루션 아키텍처
  + 서버 아키텍처
  + 네트워크 아키텍처
  + 스토리지 아키텍처
  + 랙 디자인 아키텍처
  + 데이터 아키텍처
  + 글로벌 Roll-Out 아키텍처

아키텍처 패턴 : 시스템 전체 구조와 구성요소 간의 관계 정의

* 계층화 패턴
* 클라이언트 서버 패턴
* 마스터 슬레이브 패턴
* 파이프 필터 패턴
* 브로커 패턴
* 피어 투 피어 패턴
* 이벤트 버스 패턴
* MVC 패턴
* 블랙보드 패턴
* 인터프리터 패턴

앤터프라이즈 시스템의 발전

* IT 시스템의 패러다임의 변화
* 시스템 통합에 대한 요구
* 비즈니스 대응 속도 강조

SOA : 서비스 지향 아키텍처

* 비즈니스 기능을 재사용 가능한 ‘서비스’ 단위로 캡슐화하고
* 표준화된 인터페이스를 통해 연동하도록 설계한 소프트웨어 아키텍처
* 각 서비스는 느슨하게 결합 / 필요시 유연하게 조합

서비스 : 표준 인터페이스를 통해 기업의 업무를 표현한 느슨 결합, 상호 조합 가능한 소프트웨어 컴포넌트

서비스 구성 : 비즈니스 적 의미를 가지는 기능을 모아놓은 소프트웨어 컴포넌트

* Task-centric Service : 특정 과업을 수행하는데 중점을 둔 서비스
* Data-centric Service : 특정 데이터나 도메인 모델에 대한 접근 제공 서비스
* Intermediary Service : Routing 등
* Process-centric Service : 비즈니스 서비스를 조합, 업무 프로세스 구현
* Application Service : Technical 기능을 구현한 서비스
* Public Enterprise Service : 외부 시스템으로 서비스 제공하고자 할 때 정의

SOA 단계적 발전 구조

* Fundamental SOA : 기존 시스템 서비스화 해, 각 시스템 통합하는 단계
* 문제점 : 시스템 크기 증가 -> 연결이 매우 복잡
* Networked SOA : 서비스 허브의 도입을 통해 서비스의 중앙 통제력, 유연성 강화
* Process-oriented SOA : BPM을 통해 비즈니스 프로세스를 모델링하고 실행
* 서비스를 순서/ 분기/ 병렬로 조합하여 업무 프로세스 자동화 (모니터링, 최적화 쉬움)
* BPM : 프로세스 + 정의 / 실행 / 모니터링 / 최적화
* SOA 구현 시 고려사항 : 서비스화, 인터페이스 표준, 트랜잭션 처리, 트랜잭션 처리 방안, 보안, 모니터링, 로깅

IT 운영의 3대 과제 : 장애, 성능 저하, 변경 지연 -> 해결하려면 복잡성 계속 증가

복잡도 : 시스템 구조 관점에서 낮은 복잡도를 유지하도록 설계하는 것이 해답

* 어떤 컴포넌트 한 줄이라도 변경을 한다면, 전체를 다시 배포

모델링 기술 흐름

* 객체 지향 분석 설계 (객체)
* 컴포넌트 기반 개발 (컴포넌트)
* 서비스 지향 아키텍처 ( + 웹서비스 & 컴포넌트/서비스)
* 마이크로 서비스 아키텍처

( + 마이크로서비스 컨테이너 & 마이크로서비스/컴포넌트)

모노리틱 아키텍처 : 하나의 서버에 모든 비즈니스 로직이 들어가 있는 형태

* 장점 : 기술 단일화 / 관리 용이성
* 여러 개 기술 혼용 사용 어려움 / 배포 및 재가동 시간 오래 걸림 / 수정이 용이 X

MSA : 시스템을 독립된 서비스들로 나눠서, 서비스들의 조합으로 비즈니스 기능 제공

* 서비스 별로 다른 기술 스택 사용 가능

팀 모델 : 분산형 거버넌스 / 팀 간 조율이 핵심 / 프로그램 매니저 / chief architect

서비스 오케스트레이션 계층의 활용

* 클라이언트에서 직접 서비스를 조합하는 방식
* 별도의 조합 계층을 넣는 방식

API 게이트웨이 : client와 Api 서버 앞에서 프록시처럼 위치해 다양한 기능 수행

* 패턴 1 : 인증, 인가의 단일화
* 패턴 2 : 멀티 앤드포인트와 멀티 프로토콜 제공
* 패턴 3 : MEP (메시지 교환 패턴) 변화
* 패턴 4 : 메시지 기반 라우팅
* 패턴 5 : 공통 기능 처리
* 패턴 6 : 다중 API 게이트웨이 패턴
* 패턴 7 : API 호출용 엔드포인트와 스트리밍용 엔드포인트 분리
* 패턴 8 : 비기능 요소 제거