Intro: Why and what do we study under the term "software architecture"?

AAP: Introduction

- sofrware system 은 꾸준히 신경쓰지 않으면 카오스가 됩니다.

Encapsulation and abstractions

- encapsulation = simplifying behavior + hiding data
- eucapsulate 한다는 것은 code 를 통해 구현하려는 task 를 잘 identify 하고 well-defined 된 object, function 에 해당 task 를 할당하는 것입니다. 이런 object, function 을 abstraction 이라고 합니다.

Layering

- encapsulation, abstraction 을 신경쓰면서 동시에 object, function 끼리의 interaction 도 신경써야합니다.
- 이들을 다루기 위한 layer architecture 중 하나는 three-layered architecture 가 있습니다.
    - Presentation Layer -> Business Logic -> Database Layer

Dependency Inversion Principle

- DIP 정의
    - high-level module 은 low-level module 에 의존하지 않는다. 둘 다 abstraction 에 의존한다.
    - abstraction 은 detail 에 의존하지 않는다. 대신 detail 는 abstraction 에 의존한다.
- 여기서 '의존한다' 의 의미는 단순히 import, call 의 의미라기보다 know about, need 의 의미입니다.
- 먼저 첫번째 정의는 high, low-level 간의 변화과 최대한 독립적으로 이루어져야한다는 의미입니다.
    - hige-level module 은 business 변화에 따라 상대적으로 변동이 많습니다. 이에 대해 빠르게 대응해야합니다. low-level module 은 high-level 에 영향없이 변동이 가능해야합니다.
- 두번째 정의는 책의 뒷부분에서 설명합니다.

DDIA: chapter1 Reliable, Scalable, and Maintainable Applications

- database, queue, cache 등 다양한 tool 을 data system 이라는 관점에서 바라봐야하는 이유가 무엇일까요?
- 이유는 이들의 경계가 무너지고 있고 더 세분화된 task 에 따라 최적화된 tool 이 더 다양해지고 있기 때문입니다.

Reliability

- reliable software: continuing to work correctly, even when things go wrong
    - 사용자가 예상한대로 작동
    - 사용자의 실수나 예상과 다른 방법으로 사용해도 어느 정도 fault-tolerant(resilient) 함
    - 의도대로 사용하면 예상한 performance 를 냄
    - 허가되지 않은 접근이나 사용을 막아냄
- fault 는 failure 랑 다릅니다.
    - fault: one component of the system deviating from its spec
    - failure: system stops providing the required service
- fault 의 종류
    - hardware fault

    - software error
    - human error

Scalability

- scalability: a system's ability to cope with increased load
- application 마다 필요한 scalable architecture 는 다른 형태를 보입니다. 하지만 일반적인 형태가 있기는 합니다.
    - scaling up: vertical scaling, 더 성능이 좋은 machine 사용
    - scaling out: horizontal scaling, 작지만 더 많은 수의 machine 으로 분산

Describing Load

- scalability 를 고려하기 위해서는 현재 load 에 대해 파악하는게 중요합니다.
- 파악하는 수치를 load parameters 라고 합니다.
    - 예를 들어, 웹서버에서 초당 request 수

Describing Performance

- load 가 증가할 때 살펴봐야 하는 점
    - system resource 의 변화, 이에 따른 system 이 받는 영향
    - performance 를 유지하기 위해 얼마나 더 resource 가 필요한지
- performance 관련 지표
    - throughput: 초당 처리되는 record 수
    - response time: client 가 request 쏘고 reponse 받는데 걸린 시간

Maintainability

- legacy 를 만들지 않고 maintenance 동안 불필요한 작업을 최소화하가 위해 고려해야할 점은 다음과 같습니다.
    - operability
    - simplicity
    - evolvablility

Operability

- 좋은 operation team 이 해야할 일들
    - system 상태 모니터링, 문제가 생기면 빠르게 다시 시작
    - software, security 등을 최신화
    - 문제를 미리 알 수 있는 능력
    - system 간의 관계, 영향도 파악
    - 등등

simplicity

- accidental complexity 를 최소화하여 simple 한 system 을 구축해야 합니다.
- 이를 위해 좋은 방법은 abstraction 입니다.

Evolvability

- system 의 requirements 는 계속해서 변화합니다. 이에 잘 적응하고 대응할 수 있어야 합니다.

- 따라서 이는 simplicity, abstraction 과 연관이 깊습니다.

FSA: chapter 1 introduction

- software architect 를 향한 명확한 길이 없을까요?
    - industry 에서 software architecture 에 대하 좋은 정의가 없습니다.
    - software architecture 가 포함하는 범위가 너무 넓습니다.
    - software architecture 는 계속해서 진화하고 변화합니다.

define software architecture

- software architecture 구성 요소
    - structure
    - architecture characteristics (define the success of a system)
        - 예를 들어, scalability 같은 것들
    - architecture decision (define the rules for how a system should be constructed)
    - design principle (guideline rather than rule)

Expectations of an Architect

- Make architecture decisions
    - 기술적인 선택을 확정하기보다 가이드합니다.
- Continually analyze the architecture
    - architecture 를 계속 분석하고 개선점을 제안합니다.
- Keep current with latest trends
- Ensure compliance with decisions
    - ensuring compliance: 정해진 arhchitecture rule, guide 에 맞춰서 개발이 진행되는지 확인하는 것
- Diverse exposure and experience
    - 하나만 깊이 알기보다 다양한 기술, 프레임워크, 플랫폼 등을 파악하는 것이 좋습니다.
- Have business domain knowledge
- Possess interpersonal skills
- Understand and navigate politics