## 2019311801 이균서

## A1

-Techical computer-based systems: operators 와 operational processes 가 일반적으로 시스템의 일부로 여겨지지 않는 하드웨어와 소프트웨어를 포함하는 시스템.

- Example: 책을 쓰기 위한 word processor
- Socio-technical systems: technial system 뿐 아니라 이를 사용하고 이와 상호작용하는 operational processes 와 사람을 포함하는 시스템. Socio-technical 시스템은 organizational policies 와 rules 에 의해 운영된다. 비결정적인 요소가 더 부각된다.
- Example: 책을 생산하기 위한 출판 시스템

## A2

하드웨어 실패는 시스템의 하드웨어 구성 요소에서 발생하는 물리적 실패이다. 예측 가능하며 욕조 곡선으로 표시되어, 구성 요소가 처음 사용될 때 실패율이 높고, 안정 상태로 떨어지며, 구성 요소가 노화됨에 따라 다시 증가한다. 반면 소프트웨어 실패는 물리적 마모와는 관련이 없으며 소프트웨어의 오류로 인해 발생한다. 요구 사항이 변하다 보니, 안정화 되려고 하면 다시 failure 가 올라갈 수 있다. (나이키 로고 모양으로)

## A3

하나의 구성 요소 (하드웨어, 소프트웨어 또는 operators) 에서의 실패는 다른 구성 요소에서의 실패로 이어질 수 있다. 예를 들어, 소프트웨어 오류로 인해 하드웨어 구성 요소가 실패할 수 있으며, 이로 인해 운영자가 오류를 범할 수 있다. 반대로 운영자의 오류로 인해 소프트웨어 오류가 발생하면 하드웨어 실패로 이어질 수 있다. 서로에게 영향을 줄 수 있다. 예시: ui/ux 가 잘못돼서, software, hardware 에 안 좋은 영향을 끼칠 수 있다. operator error 로 이어질 수 있다. 예시: 서버 장비의 문제가 software 문제로 이어질 수 있다. 또 이것이 사용자의 문제로 이어질 수 있다. 예시: 컴퓨터 사양에 넘어서는 프로그램을 돌리면, 발열이 나고, 이는 사용자가

software 는 deterministic 하다. 그렇게 해도 일정 시점이 지나면, 다른 동작을 할 수 있다.

## A4

Conceptual Design: 시스템의 목적, 필요성 및 사용자가 시스템에서 기대할 수 있는 고수준 기능을 설정한다. 조달 업무를 하기 위해서 최소한으로 필요한 requirement 및 정보를 파악하는 과정.

Procurement: 고수준 시스템 요구 사항이 정의되며, 기능 분배에 대한 결정이 이루어지고 시스템 구성 요소가 구매된다. (내가 개발하는 것도 조달, 사오는 것도 조달, 외주를 주는 것도 조달)

Development: 시스템은 세부 요구 사항 정의, 구현, 테스트, 운영 프로세스 정의 및 사용자 교육 설계와 함께 개발된다.

Operation: 시스템이 배포되어 사용된다. 새로운 요구 사항이 나타나면 변경이 이루어지며, 결국 시스템이 폐기된다.

## A5

큰 시스템에서 어떻게 병렬적으로 개발할 것인가? 로 해석

예: 10 개의 팀이 나눠져 있다. 병렬 개발, 동시 개발, 병행 개발. 일을 어떻게 나눠 주면 될까? "10 명의 팀장을 불러서, 개발해."

- 아키텍처 정보와 subsystem 간의 interface 정보를 줘야 함
- subsystem 간의 소통이 중요함

---

- api 명세
- 의존성 분리
- mocking test
- CI/CD 파이프 라인
- 개발 리소스의 로드 밸런싱 -> 우선 순위를 고려해서 개발 인력 자원을 분배하나, 순간 순간에 맞게 개발 리소스를 로드 밸런싱해야 함.
- 품질 관리: 코드 품질을 모니터링하고 개선하기 위한 정기적인 코드 리뷰와 품질 관리 프로세스를 수립해야 한다.

## A6

- 요구 사항 엔지니어링: 고수준 및 비즈니스 요구 사항을 정제, 분석 및 문서화한다. requirement partioning, requirement grouping
- 아키텍처 설계: 시스템의 전체 아키텍처를 설정하고, 구성 요소와 그들 간의 관계를 식별한다.
- 서브시스템 엔지니어링: 소프트웨어 구성 요소 개발, off-the-shelf 하드웨어 및 소프트웨어 구성, 운영 프로세스 정의.
- 시스템 통합: 새로운 시스템을 만들기 위해 시스템 요소를 결합한다.
- 시스템 테스트: 문제를 식별하기 위해 전체 시스템을 테스트한다.
- 시스템 배포: 시스템을 사용자에게 제공하고, 기존 시스템에서 데이터를 전송하며, 다른 시스템과의 통신을 설정한다.

## A7

### type

- COTS(commercial off the shelf): 기성품. 수정이 어렵고 원하는 기능이 다 없을 수 있다. 그대로 쓴다. stability 가 높을 것이다.
- configurable systems: COTS 중에서는 각자에 맞게 tailoring 을 할 수 있는 것.
- custom system: 조직 내에서 새로 외주를 주는 것은 stability 가 다를 것이다.

조달 단계에서 내린 결정은 시스템 엔지니어링 프로세스의 후속 단계에 중요하다. 부실한 조달 결정은 시스템의 지연된 배송과 같은 문제로 이어질 수 있으며, 운영 환경에 적합하지 않은 시스템의 개발로 이어질 수 있다. 잘못된 시스템이나 공급 업체를 선택하면 시스템 및 소프트웨어 엔지니어링의 기술 프로세스가 더 복잡해진다.

## A8

dependablity:

conceptual design, procure, development, operation activity 에서 모두 고려해야 하는 것이다. high dependability 를 달성하는 건 당연히 어렵다.

## A9

human error 가 반드시 생긴다. 그 이유는 뭘까? 교재에서는 사람을 다그치거나 시스템을 다그친다.

시스템을 보완을 해야한다.

어떻게?

사용자가 어떤 시도를 했을 때 그것이 system failure 로 가지 못하게 해야 한다. Swiss cheese 구멍 하나가 쭉 이어지면 망한다.

#### 사용자 관점

- 사용법에 대한 명세를 넣는다.
- 서버 단에서 error 처리를 하고, 이를 모니터링하는 시스템을 구축하는 것.
- 안드로이드 OS 에서 루트 권한 취득 시도를 막았다.

#### 개발적인 관점에서

- branch
- PR
- merge
- CI/CD

## A10

대규모 시스템은 긴 수명을 가지고 있으며 변화하는 요구 사항을 충족시키기 위해 진화해야 한다. 진화는 본질적으로 비용이 많이 든다. 변경에 대한 기술 및 비즈니스 분석의 필요성, 예기치 않은 문제를 초래할 수 있는 서브시스템 간의 상호 작용, 원래의 설계 결정에 대한 근거의 부재, 변경이 이루어짐에 따라 시스템 구조가 손상되는 것 등 때문이다.

large system 은 조금 바꿔도 비용이 많이 발생한다. 1 개의 subsystem 이 딴 subsystem 에 영향을 줄 수 있는지 파악을 해야 한다. 수정/변경에 대한 accept 를 할지 말지도 결정해야하고. 이것이 현재 기술에서 가능할 것인가. => 뭐 하나 만만하게 결정할 수 없다 (?)

특히 cost distribution 에서 life system 이 긴 system 의 경우 dev 보다 유지보수 비용이 몇십배는 더 든다.

## Q1 from myself

휴먼 에러가 해당 조직을 망가뜨리는 사례와 이를 예방할 수 있는 시스템을 어떻게 구축할 수 있는지.

## A

파생 상품 거래 시스템에서 직원이 값을 잘못 올려서, 해당 회사가 파산한 경우가 있다. 이 때 너무 큰 수치가 input 값으로 들어가는지 check 를 하는 시스템을 구축했다면 이 위기를 면할 수 있었다.

## Q2 from myself

서브시스템 기반 개발이 좋다는 트렌드가 깔려 있는데, 오히려 서브 시스템 개발로 인한 문제점은 뭐가 있을지.

## A

서브 시스템 개발자 간의 커뮤니케이션 cost 가 더 많이 들 것이다. 하나의 서브 시스템이 다른 서브 시스템의 개발이 완성돼야 해당 서브시스템이 동작을 할 수 있다. 너무 잘게 쪼갠다면 성능의 오버 헤드가 발생할 수 있다.

### 서브시스템과 컴포넌트의 차이

서비시스템: 하나 똑 떼서 독립적인 시스템으로 볼 수 있는 것 (도커 컨테이너) 컴포넌트: 독립성이 없고, 의존적인 것