**Software Engineering Q/A Sheet (#2)**

date: 9/13 number: 2018312280 name: 이상수

**Questions from Prof.**

1. Compare sociotechnical systems to conventional technical computer-based systems.

sociotechnical system은 technical system과 operational process와 사람 등 여러 요소가 통합된 시스템으로 조직의 방침과 룰에 따라 운영됩니다.

technical computer-based system은 하드웨어와 소프트웨어가 포함되었지만 사용자와 운영관리 과정은 포함되지 않은 system입니다.

1. Explain the difference between a hardware failure and a software failure. Note that a hardware failure is represented by a bathtub curve.

하드웨어의 문제는 그 문제가 일어나는 빈도와 심각성은 시간에 따라 어느정도 예측이 가능합니다. 예를 들어 bathtub curve는 하드웨어가 처음 만들어졌을 때는 이런저런 초기 문제들이 발생하고, 이를 보완해가며 문제를 줄여 나가지만, 결국 시간이 더 많이 지나면 자체적으로 낡고 더 이상 보완할 수 없는 문제들이 발생하며 다시 문제 발생 빈도가 늘어나게 되는 현상을 말합니다.

이에 반해 소프트웨어는 감가상각 같은 요소가 적어서 하드웨어보다 시간에 따라 자연스럽게 문제가 늘어나지는 않는다. 대신 그 문제 발생의 빈도를 예측하기가 어렵다. 소프트웨어를 사용할수록 문제를 보완해서 문제 빈도가 줄어들 것 같지만, 사람이 하는 작업이기에 오히려 문제가 늘어날 때도 있습니다.

failure graph가 중요하다.

하드웨어는 bathtub curve

소프트웨어는 예측 어려움

그 이유 : 소프트웨어는 안정화가 된다. 그런데 요구사항의 변화, 안정화를 위한 개발이 오히려 문제를 일으키는 등으로 failure 커브가 감소하다가 나이키 모양처럼 다시 올라간다.

regretion 테스트 : 개발자들이 개발을 할 때 문제를 일으키지 않았나 매번 실행하는 테스트. 이를 자동화하는 것도 좋다.

1. Describe the propagation of failures between hardware, software, and operators.

하드웨어, 소프트웨어, 사용자의 문제들은 서로 영향을 줍니다.

하드웨어의 문제가 소프트웨어의 문제로 이어지고, 소프트웨어의 문제가 사용자의 문제를 유발하면 이 사용자가 소프트웨어에서 또 문제를 일으키고 이는 다시 하드웨어에도 문제를 유발할 수 있는 등, 서로 영향을 지속적으로 주게 됩니다.

예를 들어 소프트웨어를 과하게 돌려서, 발열이 과하게 일어나 하드웨어에 문제를 일으키고, 이를 고쳐보려고 무리한 시도를 유저가 하면 이게 다시 하드웨어 문제로 이어져서, 하드웨어의 성능이 저하되고 이는 소프트웨어 성능의 저하로 이어집니다.

소프트웨어 자체만으로도 완벽하기 어려운데, 하드웨어, 휴먼 에러까지 겹치면 더 불안정해진다.

1. Explain the entire system engineering process including conceptual design, procurement, development, and operation.

Conceptual design은 개발 전 시스템에 필요한 아이디어들을 모으는 단계입니다. 이 단계에서 사용자들의 요구사항, 주요 관계자들의 요구사항들을 모아서 기술적이지 않은 시스템의 목적과 구성방안을 정하는 단계입니다.

시스템 컨셉파악, 문제와 요구사항 파악, 시스템 초기 개발 계획, 시스템의 타당성 판단, 시스템의 구조 제작 등의 단계가 포함됩니다.

System procurement는 시스템의 needs를 정하는 단계로, 이 단계 이전에 시스템이 어떤일까지 해야할지, 시스템 제작 예산과 시간, 그리고 구체적이 요구사항을 파악해두어야합니다. 이를 통해 시스템을 만들지말지, 후원자와 시스템 타입을 정합니다. 이 때 decision driver인 예산, 외부 경쟁자, 조직내의 다른 시스템 등도 고려해야합니다. 이들을 기준으로 이제 시스템의 어느 부분을 외부에서 구해올지, 외주를 맡길지, 아니면 공급자를 통해 받을지를 결정합니다.

이 단계에서는 요구사항의 변경이 일어나거나, 계약을 성립해야하는 등의 문제가 발생할 수 있습니다.

Development 단계에서는 주로 plan-driven방식으로 진행이 되는데 주된 이유는 소프트웨어, 하드웨어 등 다양한 요소들이 함께 제작에 관여하는 만큼 그 다양한 요소들이 같이 발맞춰 개발되어야지 서로 맞추기 쉽고 문제를 처리하기도 쉽기 때문입니다. 이 단계에서는 다양한 엔지니어들이 함께 일을 할 수밖에 없기에 의사소통, 서로 다른 관점에 의한 문제 등을 해결해야 합니다.

요구사항 확인, 구조 디자인, 요구사항 분배, 서브시스템 개발, 서브시스템 통합, 통합 시스템 테스트, 시스템 배포의 단계를 따릅니다.

System operation 단계는 시스템이 배포되었고 이제 사용자들이 사용하는 단계입니다. 이 때는 사용자가 사용법을 익혀야 하고, 문제 발생시 고치기 쉽게 시스템은 유연하게 관리되어야 합니다. 이 때는 사용자가 문제를 발생시킬 수도 있는데, 그렇기에 사용자를 제대로 교육해야 하고 시스템으로서는 그런 문제가 발생했을 때 쉽게 시스템에 문제를 일으키지 못하도록 여러 차례 예방수단을 마련해 두어야 합니다.

1. List everything you think is necessary for parallel development of large systems.

가장 중요한 것은 엔지니어들 간의 의사소통입니다.

conceptual design과 procurement에서 개발 전 사전 계획을 잘 짜 야합니다.

개발단계에서 plan-driven방식에 따라 계획에 잘 따라는 것도 중요합니다.

각 서브시스템의 관계와 의존성을 잘 문서화해야 한다.

개발 리소스의 load를 잘 분배해야 한다.

정기적으로 리뷰하고 관리한다.

interface 정보가 중요하다

architecture 도 중요하다.

이 두가지를 여러 팀 들에게 제대로 알려줘야 한다.

1. Explain the system development process from requirement engineering to system deployment.

requirement engineering은 conceptual design에서 정해진 요구사항을 확인하고 분석하는 단계입니다.

Architectural design은 시스템의 구조를 구상하고 그 구성요소들과 요소들 간에 관계를 파악하는 단계입니다.

Requirement partitioning은 각 서브시스템에게 맞는 요구사항을 분배하는 과정입니다.

= 요구사항은 뭉퉁그려진 덩어리로 받는다. 이를 우리가 필요한 방식으로 여러 파트로 나누어 서브시스템별로 분배하는 과정이다.

subsystem engineering은 소프트웨어의 구성요소들을 개발하는 과정입니다.

system integration은 시스템 구성요소들을 통합하여 통합 시스템을 만드는 과정입니다.

system testing은 이렇게 통합된 시스템을 테스트하여 문제를 확인하는 과정입니다.

system deployment는 시스템을 사용자들에게 배포하거나 다른 시스템에 데이터를 배포하여 시스템들 간에 연계를 활성화하는 단계입니다.

1. Describe the types of procurement decisions and why they are important, particularly affecting system functions such as stability.

off-the-shelf system : conceptual design이후 승인된 application을 평가해서 제작할 시스템에 가장 필요한 시스템을 골라서 구매해서 사용하는 과정으로 수정이나 변경이 어렵습니다. (사오는데 커스텀이 불가능한 제품)

configurable systems : conceptual design이후 시장조사를 통해 시스템 후보목록을 보고 앞서 받은 요구사항들을 조금 더 상세하게 확인하여 가장 적절한 시스템 제공자를 골라 제공자와 계약을 합니다. 이 때 대신 요구사항을 수정해야 할 수도 있습니다. 이 방식은 이후에 수정을 하기 용이합니다. (사오는 건데, 커스텀이 가능한 제품)

custom systems : conceptual design이후 요구사항을 상세히 확인하고 문제를 해결해줄 사람을 요청해서 제공자를 골라 계약을 하거나 대신 요구사항을 수정합니다.

stability와 관련해서는 off-the-shelf방식의 경우 검증된 어플을 활용하기에 stability가 높습니다. configurable방식은 기초 기반이 존재하기에 이는 stable하지만, 요구하는 쪽에서 customize를 한다면 그 방식에 따라 stability가 달라질 것입니다. 마지막으로 custom system은 말 그대로 직접 구상한 일을 요청하는 것이기에 system제작에 구상을 얼마나 잘 했는가, 제작을 얼마나 잘하는가에 따라 stability가 완전히 달라질 것입니다.

1. Discuss how the various activities of the system development process affect system dependability.

시스템 요구사항을 받고 구조를 설계해서 subsystems들에 요구사항을 분배할 때 계획을 어떻게 하느냐에 따라 system dependability가 바뀔 것입니다. 또한 subsystem을 만들 때도 subsystem들이 다른 시스템에 어떤 영향을 주는지 잘 파악해야합니다. 서브시스템을 합치고 테스팅할 때는 이 dependability를 고려해서 테스트해서 차후에 시스템이 어떠한 영향을 줄지 예측하고 대비할 수 있어야합니다.

각 시스템별로 보안을 위한 subsystem을 개발해서 전체적 보안을 높인다. 서브시스템 개발시 문제도 사이사이에 테스트를 통해 보안한다.

1. Discuss one or more examples of various barriers to protecting against human errors leading to system failure.

대표적으로 try catch등으로 exception을 핸들링하는 것이 있습니다. 그리고 입력변수 타입 확인, 예상치 못한 동작이 발생했을 때 프로그램을 강제 종료하는 경우도 이에 해당한다 볼 수 있습니다.

사용법 문서화, 서버에서 에러처리를 하고 이를 모니터링 할 수 있는 환경, 회사 보안측면에서는 사전보안단계, 직원 보안 교육 등, 사용자 권한 제한

1. Explain why the evolution of large systems is expensive.

변화를 어떻게 줘야 할지 기술적, 사업적 관점에서 둘 다 확인해야 합니다. sub-system이 많을 것이고 이 각각이 변화하면 어떤 영향을 줄지 파악하기 어렵습니다. 기존 디자인의 목적을 나중에 찾아보면 찾거나 파악할 수 없을 수도 있습니다. 변화가 시스템 구조상 문제를 일으켜서 시스템 전체를 망가뜨릴 수 있습니다. 그리고 엔지니어가 검토를 하고, 수정 변경할지 결정하고, 수정을 하는 모든 과정이 전부 비용이다.

Questions from your ownself

1. 다양한 엔지니어들이 함께 작업할 때, 서로 전문으로 하는 분야가 다른 만큼 의사소통이 어려울 텐데, 이 때 서로 이해하기 어려운 기술들을 어떻게 효과적으로 공유하며 작업할 수 있을까?

각 분야 전문가들이 다른 분야에 대해서도 배경지식이 있어야한다. 각 분야의 대표자들이 활발하게 소통하여 서로 이해하지 못하는 부분은 명확하게 설명을 해주어야한다.

2. 각 분야의 failure가 다른 부분에 영향을 미치는 사례는 뭐가 있을까?

대표적으로 게임개발에서 기획부서가 기획을 한 방식이 개발에 있어서 어려운 문제를 일으켜서 개발의 failure를 불러올 수 있다.

팀 질문 :

서브시스템으로 개발하는 것으로 일어날 수 있는 문제점?

서브시스템들의 개발은 결국 각각 독립적으로 이루어지는데 이럴 때 필요한 의사소통, 분배 등이 전부 오버헤드가 될 수 있다. 시스템이 잘게 나눠지면 시스템 간에 소통을 통해 전체 시스템의 동작이 이루어지니까 이게 성능을 저하시킬 수 있다.

어느정도 규모가 큰 프로젝트가 되면 서브시스템을 안 나눌래야 안 나눌수가 없다.

개발자 수가 적을 경우 또는 프로젝트 규모가 작을 경우 서브 시스템을 덜 나눌 수 있다.

서브시스템이란 프로젝트를 나누고 나눠서 최소 단위의 unit(함수)까지 내려갈 수 있다.

서브시스템이란 독립성으로 구분된다. 여러 단위로 나눠질 때 혼자서 독립적으로 존재할 수 있는 시스템을 서브시스템이라고 한다. 독립적으로 존재 못하면 component이다.

예를 들어 다중보안 시스템에서 음성인식, 홍채인식, 지문인식을 넣었다고 할 때. 이러한 인식들이 서브시스템이다. 음성인식안에 음절단위로 분석하는 것은 독립적이지 못하기에 component이다.