**Software Engineering Q/A Sheet (#3)**

date: 9/20 number: 2018312280 name: 이상수

**Questions from Prof.**

1. Explain the strengths and weaknesses of the Waterfall model.

waterfall model 은 가장 기본적이고 일반적인 방법이다.

장점 : Plan-driven 형식이기에 서브시스템들의 병렬적 개발을 주도하기 좋고, 관리가 쉬워 큰 시스템을 위한 프로젝트에 적합합니다.

requirement가 명확하다. 프로세스 가시성이 좋아서 전체적인 흐름 파악이 좋다.(waterfall model은 문서화를 강제 시 한다)

소프트웨어의 가시성 : 문서화를 뜻한다.

앞에 있는 문서화와 함께, architecture design을 미리 해서 interface에 대한 디자인을 초기에 하기 때문에 대규모 프로젝트에 적합한다.

단점 : 각 단계를 유동적으로 변화하거나 이동하기가 어렵기에 변화에 잘 대응하지 못합니다. 이렇기에 requirements가 처음부터 잘 정의되어야 하는데, 이것은 현실적으로 어려운 일입니다.

1. Explain the strengths and weaknesses of the incremental model.

장점:

고객 요구사항이 변화해도 이를 적용하는데 비용이 적습니다.

이미 완료된 개발작업에 대해서 고객의 피드백을 적용하기 좋습니다.

고객에게 좀 더 빠르게 배송, 배포할 수 있어 고객이 소프트웨어 사용으로 얻는 이익이 증가합니다.

단점:

프로세스가 가시화하기 어렵습니다. 시스템이 빠르게 개발될수록 이를 문서화해서 유지 관리하기가 어렵습니다.

시스템을 수정이나, 추가를 할수록 구조가 불안정해질 가능성이 높습니다.

* 문서화를 할 수는 있지만, 자주 수정을 하게 되는데 매번 문서화를 하면 그 비용이 커지기 때문이다.

1. Describe the concepts of refactoring and find examples of them.

refactoring은 외적인 기능을 바꾸지 않고, 내부 구조를 고치는 행위. (refactoring은 매우 중요한 내용이다. 이를 면접에서도 물어본다더라)

변수 종류를 통일하고, format을 통합하고, 함수의 기능을 문서화하고, 함수나 변수 중에 불필요한 내용을 수정하고 최적화를 하고 주석을 추가하는 일. 함수 분할을 통한 최적화

예시로는 소프트웨어를 refactoring한다고 하면 코드의 구조를 재 설계하거나, 재 배치하고 이전에 내용에서 추가된 내용을 적절한 구조에 맞게 적용을 하는 것입니다.

데이터 베이스 구조의 최적화도 refactoring이라고 한다.

1. Compare the waterfall model with the incremental model for the workflow.

waterfall model은 작업 순서가 정해져 있고, 대부분의 경우 한번 완료된 작업으로는 되돌아가지 않습니다.

Requirement definition, System and software design, implementation and unit testing, integration and system testing, operation and maintenance의 단계로 진행됨.

incremental model은 requirement가 충분히 정의되지 못할 경우에 유용한 방법으로 그나마 있는 requirement를 기준으로 점진적으로 개발하는 방식.

작업 순서가 있기는 하지만 해당 작업 단계들을 좀 더 자유롭게 이동하며 최종 출시를 하기 전까지 concurrent activity 인 specification, development, validation 단계를 오고 가며 개발할 수 있습니다.

1. Describe four fundamental process activities.

Specification : 시스템이 수행해야 할 목표 정의

Design and implementation : 시스템의 구조와 시스템의 적용 방식 정의

Validation : 시스템이 고객의 요구를 수행하는지 평가

Evolution : 고객의 요구 변경에 따른 시스템의 수정

1. Describe the requirement engineering process with key activities and their work products.

Requirement engineering process는 Software specification에서의 작업입니다.

Feasibility study :

기술적, 제정적으로 실현 가능한가?

Requirements elicitation and analysis :

stakeholder들이 시스템에 무엇을 원하고 기대하는지를 파악하고 분석합니다.

Requirements specification :

requirements를 자세하게 정의한다.

Requirements validation :

requirement의 validity를 테스트하여 고객의 요구를 만족하는지 테스트하고 이를 이용해 최종적 요구사항에 대한 문서를 작성합니다. 이 때 이 최종 문서가 잘못 작성되면 요구사항에 대한 분석을 잘못 한 거다.

(여기에는 미래에 고객이 필요할 요구사항도 예측할 수 있어야 한다. 이를 고객에게 설득할 수도 있어야 한다.)

1. Describe the differences between component, system and acceptance tests.

component test는 개별 component들을 개별적으로 테스트하는 과정입니다. component는 함수, 오브젝트, 또는 이들의 집합 (unit의 집합)

* component들에 정의된 입력과 출력을 테스트하는 과정
* 이러한 component테스트를 하나씩 모아서 하는 걸 interface testing

system test는 총 시스템을 테스트합니다.

-emergent property, safety, reliability 등의 테스트를 할 수 있는 테스트가 이 subsystem이 다 모인system test에서만 가능하다.

delivery는 그냥 고객에게 던져주는 것, 고객의 다른 시스템들과 연동해서 사용 가능하게 해주는 것은 deployment.

acceptance test는 고객데이터로 고객의 요구에 맞는지 테스트

-고객의 상황에 맞춰서 고객에 환경과 데이터에서 테스트를 해보고 돈을 받는다.

-그냥 시장에 푸는건, 알파테스트, 베타테스트이다. (알파는 실험실내에서 사용자들 모아놓고 하는거) (베타는 사람들한테 풀어서 테스트하는거)

1. On slide 28, discuss when to plan each test with why.

acceptance test는 requirement specification과 system specification사이에 plan합니다. 이는 이 작업들 사이에 실행해야지 고객 요구사항에 대해 가장 많이 논하고 알 수 있는 시점이기에 이 때 plan하는 게 가장 적합한 시기입니다.

system integration test는 system specification, system design 사이에 plan합니다.

sub-system integration test는 system design과 detail design사이에 plan합니다.

Module and unit code and test는 테스트 가능한 작은 단위 코드나 모듈을 만들 때마다 테스트 가능합니다. 이는 작은 단위로 테스트하는 과정이기에 만들 때 바로바로 테스트가 용이하기 때문입니다.

이렇게 test할 때 plan을 미리 하는 이유는, 테스트에 필요한 데이터가 생성되는 타이밍들이다. 이는 초기에 어떻게 테스트할것이라고 계획해뒀다는 증빙 자료도 된다.

테스트가 1,2년 후에 이루어질수도있는데 그 때 계획하면 뭐해야하는지 다 까먹어서 답도없다.

1. Change is inevitable in every large software project. There are two related approaches you can use to reduce rework costs. First, explain the concept of rework and describe two related approaches.

rework는 말 그대로 이전에 만들어 둔 결과를 다시 만들거나 수정하는 일을 뜻합니다. 이는 상황에 따라 많은 비용을 투자해야할 수 있기에 최소한으로 하는 것이 좋습니다.

change avoidance는 처음부터 rework를 할 일이 적도록 프로토타입을 만들어 개발 전에 고객들에게 핵심 요소를 설명하고, 고객들이 요구사항을 좀 더 명확하게 하도록 돕는 방법입니다.

Change tolerance는 incremental development처럼 변화에 적응이 빠른 개발 방법을 사용하는 것입니다. 점진적 개발을 하면 중간에 고객의 요구사항이 바뀌거나 하는 일이 생겨도 수정하기가 용이합니다.

1. On side 38, discuss the benefits of prototyping. and explain the reasons for them.

프로토타입의 장점

프로토타입 : interface만 가진 시스템. 겉 보기에만 입력 출력만 보여주어 결과물의 전체 동작이 어떻게 진행될지 보여주는 것.

시스템 활용도 증가 : 시스템을 미리 프로토타입으로 보여줌으로 서 이를 활용할 방법들을 더 많은 고객들이 알게 되고, 이에 대해 더 다양한 고객들이 유치될 수 있습니다.

고객의 진짜 요구에 맞춰질 가능성 증가 : 고객에게 제품 개발 전 미리 요구사항을 좀 더 명확하게 얻어낼 수 있기에 고객이 진짜 원하는 요구를 미리 전달해줄 가능성이 높아집니다.

디자인 퀄리티 증가 : 디자인 단계 이전에 프로토타입을 만들어 봄으로서, 기존에 디자인에서 문제가 되었던 내용이나 프로토타입을 실행해본 결과 생겨난 예상치 못한 문제점 등을 파악해 전체적인 디자인을 이후 향상시킬 수 있습니다.

-추가로 사용자가 느끼는 불편함을 개발자나 기획자는 모를 수 있기에 프로토타입으로 이를 미리 파악할 수 있다.

유지보수성 증가 : 프로토타입을 만들어 테스트하고 요구사항들을 미리 명확하게 함으로서 디자인이 좋아졌기에 그 만큼 나중에 수정 보완 해야할 때 더 좋은 디자인으로 만들어둘 수 있고, 요구사항을 미리 잘 맞춘만큼 수정 보완할 일이 줄어듧니다.

개발 비용 감소 : 만약 프로토타입이 없었다면 고객의 요구가 명확하지 못했을 것이고, 이러면 개발 도중에 요구사항이 수정되거나 할 경우 비용이 클 수 있는데 이를 방지합니다.

usability : 사람이 시각적으로 느끼는 것, 사람이 사용하는 정도를 평가하는 지표

-디자인이 중요하다

1. Compare throw-away prototyping with exploratory prototyping.

throw-away :

프로토타입을 기반으로 기능적인 부분 이외의 요구사항을 맞춰 조정해서 만들기는 불가능에 가깝다.

프로토타입은 문서화를 흔히 하지 않는다.

프로토타입은 얼마든지 쉽게 바뀌기에 성능이 좋지 않다.

프로토타입은 흔히 기존 조직의 기준에 맞지 않을 것이다.

이렇기에 프로토타입은 사용한 후에 버리는 방식이 throw-away prototyping입니다.

exploratory prototyping은 프로토타입을 만들고 이를 확장해가면서 최종 결과를 만드는 방식입니다. 이는 처음부터 제대로 된 문서화를 하거나 디자인을 하지 않았기에 나중에 복잡한 문제가 발생하는 등 비 효율적입니다.

1. With incremental delivery, customers can feel more confident about the system. Guess why.

중요한 increment를 먼저 만들어서 배포하니 이를 매번 incrementally하게 만들고, 합치고, 배포하고, 테스트하게 되어 사용자들이 이에 대해 신뢰도가 높아진다.

1. Complete testing is impossible in practice. Discuss cost-effective testing strategies in development environments where time and cost are insufficient. –

- Hint1: Take advantage of available statistics.

= Hint2: If we don't have enough time and budget, we need to make choice and concentration. This is always the case in real development situations. There is a very interestion investigation in this regard. According to an IBM study, a company tried to eliminate 65% of the defects in a system, but the reliability improvement of the system that the customer felt was only 3%.

우선 가능하다면 프로토타입을 만들어서 테스트를 해보고 고객에게 소개를 하는 것이 최대한 이후에 생길 추가 비용을 감소시키는데 큰 도움을 줄 것입니다. 개발 모델은 incremental development model이 좋아 보입니다. 개발 시간, 비용이 부족한 만큼 이를 최대한 늘릴 방법이 필요합니다. 그렇기에 개발 과정에서 고객과 소통하며, 개발 제품도 순차적으로 배포하여 개발 기간을 늘릴 수 있습니다. 한 번에 개발해서 배포하려고 하면 개발 기간도 길어지고, 고객도 그 기간동안 제품이 늦게 나온다고 생각할 확률이 높아질 것입니다. 그러니 개발을 점진적으로 하면서 배포 가능한 수준마다 하나씩 배포하면 고객은 마치 개발 시간이 짧게 걸린 듯한 느낌을 받을 것입니다. 또한 사용자에게 중요한 내용을 먼저 개발해 배포하면 사용자들이 신뢰도가 높아질 것이다. -통계적으로 많이 쓰는 기능을 먼저 테스트하는게 좋은 방법이다.

* 어떤 데이터를 사람들이 많이쓰는지를 파악해서 그 사람들이 많이 쓰는 부분에 문제점을 주로 고쳐야한다. 어떤 쪽에 시스템을 많이 쓰는지를 주로 테스트를해서 그 부분을 중심적으로 개발을 해야한다.

Questions from your ownself

1. 고객에게 프로토타입을 제공해주기 어려운 시스템은 뭐가있을까

답변 : 온라인 다인 상호작용 제품. 대표적으로 대형 온라인 게임의 경우 개발이 많이 진행되지 않은 상태에서 프로토타입을 만들면, 프로토타입에서 실제 제품 출시할 때의 온라인 서비스, 실시간 상호작용, 그래픽의 퀄리티 등을 전부 유저들에게 소개하기 어려울 것입니다.

팀 질문 :

프로토타입이 표현할수 있는 가시성과 정보에는 한계가 있다. 예를 들어 대규모의 프로젝트는 프로토타입과 최종결과물은 많이 다를수 밖에 없는데, 이러한 간격을 어떻게 채우고 고객한테 설명할 것인가 ?

1. 비슷한 제품을 예시로 설명.

2. 이미 있는 고객 데이터를 이용해서 프로토타입을 대신해서 고객 요구사항을 예측한다.

3. 시뮬레이션 시스템으로 하드웨어 적인 부분들은 대체해서 설명한다.

4. 실제 작동하는 프로토타입을 만들기 어렵다면, 프로그램 아닌 작동 시나리오 영상이나 이미지로 고객의 이해를 도울 수 있다.

복잡한 시스템, 하드웨어 종속 시스템, 실시간 시스템, 의료/의약품 관련 소프트웨어 등이 프로토타입을 만들기 어렵다.

-교수님 답변 : 프로토타입은 많은 시스템을 보여주려는게 아니라 최소한의 UI만 보여주는 수준이라 가능한 선 안에서 보여주면 된다.