3차과제 / 한아름

기존의 개발 생명 주기는?

주먹구구식 개발 모델(Build-Fix Model)
- 요구사항 분석, 설계 단계 없이 일단 개발에 들어간 후 만족할 때까지 수정작업 수행

폭포수 모델(Waterfall Model)
- 순차적으로 소프트웨어를 개발하는 전형적인 개발 모델
- 대부분의 소프트웨어 개발 프로젝트의 기본적 모델이며 가장 많이 사용되는 모델
- 소프트웨어 개발의 전 과정을 나누어 체계적이고 순차적으로 접근하는 방법

원형 모델(Prototyping Model)
- 폭포수 모델의 단점을 보완한 모델
- 점진적으로 시스템을 개발해 나가는 접근 방법
- 원형(Prototype)을 만들어 고객과 사용자가 함께 평가한 후 개발될 소프트웨어의 요구사항을 정제하여 보다 완전한 요구사항 명세서를 완성함

나선형 모델(Spiral Model)
- 폭포수 모형과 원형 모형의 장점을 수용하고 위험 분석(Risk analysis)을 추가한 점증적 개발 모델
- 프로젝트 수행 시 발생하는 위험을 관리하고 최소화 하려는 것이 목적

UP(Unified Process)
- 소프트웨어 개발이 각각 생명주기를 가지는 여러 번의 반복(iteration)을 거쳐 수행되는 모델

XP(eXtreme Programming)
- 요구사항 변경으로 인한 비용이 개발 기간에 상관없이 일정하게 유지되도록 것을 주목적으로 함
- 고객, 관리자, 프로그래머에 대한 역할 및 권한과 4가지 가치를 중시함

**4가지 가치: 의사소통, 단순성, 피드백, 용기

S/W 개발 환경의 변화

- 결과물의 배포시기가 중요해짐
- 개발 생산성 저하 기존 방법론의 한계
- 문서 및 절차 위주의 방법론 (변화에 대응이 어려움)
- 개발자의 개발능력의 차이 불안정 Waterfall model의 문제점
- 불명확하고 변화하는 사용자의 요구사항
- 개발된 모듈들의 통합의 어려움
- S/W 품질의 하락(충분한 테스트가 어려움)

최근 트렌드가 되고 있는 개발 방법론에는 무엇이 있을까?

Lean Software Development

- 토요타의 제조에 관한 혁신적인 사고방식을 애자일 프로세스를 기반으로 하여 소프트웨어 분야에 맞게 수정한 방법론

- 최근의 다른 방법론들이 고객과의 협업과 빠른 개발에 집중했다면 린은 고객 관점에서의 낭비를 제거하고 고객의 가치를 높이는 방법에 집중

- 린은 구체적인 방법론 이라기 보다 원칙에 가까운 내용들이 많음 (ex: 배움을 증폭하라, 가능한 늦게 결정하라, 팀에 권한을 위임하라, 통합성을 구축하라, 전체를 보라 등)

Scrum

- 최근 스타트업이 사용하는 개발 프로세스 중에서 가장 인기가 높은 개발 방법론

- 스크럼은 어떤 조직에서도 사용할 수 있으며 자기 주도적인 조직을 만들어가는데 도움이 됨

- 스크럼은 많은 애자일 방법론 중에서 가장 체계적이면서 관리 주도적

- 스크럼은 업무 시간에 제한을 둠으로써 생산성을 끌어내는 것이 핵심

- 특정 언어나 방법론에 의존적이지 않음

- 팀의 창조성, 역동성, 자율성을 강조

- 작은 목표를 주기적으로 전달, 진행을 모니터링 하고 결과에 따라 계획을 조절

- 비즈니스 목표가 변경되는 것과 개발팀이 민첩하게 대응하여 완료할 수 있는 것들의 밸런스를 조정

Kanban

- 칸반은 프로세스 시각화로 프로세스를 최적화할 수 있게 도와주는 도구이자 연속적 흐름 처리 방식

- 칸반은 한 번에 진행되는 일의 수를 제한함으로써 지속적인 배포에 초점을 맞춤

- 스크럼이 구체적이고 체계화되어 있는 면이 있다면 칸반은 자유도가 아주 높은 개발 방법론

- 동시에 개발이 진행 될 수 있는 아이템의 수를 제한하는 것이 칸반의 핵심 기능

- 기존에 사용하던 방법론에서 옮겨가기가 용이, 관리 포인트가 적어 이해하기 쉬운 장점이 있음

참고 : 만화로 이해하는 칸반 : http://postgame.tistory.com/455

방법론의 트렌드가 바뀌는 이유는?

- 빠르게 변화하는 기술 트렌드에 맞춰 기존의 개발 방법론으로는 고객의 니즈에 맞출 수 없게 됨
- 올바른 커뮤니케이션의 부재
- 팀의 잘못된 목표 (무조건 빨리!, 구현이 어려워!, 요구사항이 너무 많아!... 등등)

형상관리(버전관리)란 무엇이고, 형상관리(버전관리) 방법에는 어떤 것 들이 있는가?

형상관리의 정의

- 소프트웨어 개발 생명주기 전반에 걸쳐 정의되는 형상 항목과 그에 관련되어 생성되는 형상물을 종합하고 변경 과정을 체계적으로 관리하기 위한 일련의 개발관리 활동

- 소프트웨어 개발과정에서 만들어지는 각종 산출물들을 체계적으로 관리함으로써 소프트웨어에 가시성과 추적 가능성을 부여하여 소프트웨어 관리를 강화하고 품질보증을 도모하기 위한 관리기법

쉽게 정리 해 보자면, 소프트웨어 개발 및 유지 보수 과정에 있어 변경되거나 축적되는 결과물들을 관리하는 것이 형상관리!

형상관리의 목적

- 소프트웨어 생산성, 안전성, 품질확보

- 소프트웨어 개발 및 유지보수의 어려움 해결

버전관리의 정의

- 소스 관리, 소스 코드 관리란 동일한 정보에 대한 여러 버전을 관리하는 것

- 형상 항목이 과거부터 현재에 이르기까지 변경이 진행됨에 따라 각 변경에 대해서 버전을 부여함으로써 히스토리를 관리하는 것

형상관리 방법

형상 식별(Configuration Identification)

- 형상 관리의 대상을 식별하고 베이스라인의 기준을 정하는 활동

형상 통제(Configuration Control)

- 형상에 대한 변경 요청이 있을 경우, 변경 여부와 변경활동을 통제하는 것

형상 상태 보고(Configuration Status Accounting)

- 베이스라인의 현재 상태 및 변경 항목들이 제대로 반영되는지 여부를 보고하는 절차

- 베이스라인으로 설정된 형상 항목의 구조와 변경 상태를 기록하여 보고함으로써, 형상 항목의 개발 상태에 대한 가시성을 제공

형상 감사(Configuration Audit)

- 형상 항목이 요구사항에 맞도록 잘 변경되었는지 확인하는 것

- 형상 관리 계획서대로 형상 관리가 진행되고 있는지, 요구사항 문서대로 제품이 제작 되었는지 감사하는 활동

버전관리 방법

- 로컬 버전 관리 시스템, 중앙집중식 버전 관리 시스템, 분산버전 관리 시스템으로 나뉘어지는 버전 관리 시스템의 사용

- 저장소 구조의 차이에 따라 개발자 개개인이 공유된 저장소를 가지는 형태의 분산 모델 유형과, 서버에 저장소를 두고 각자 복사본을 가지고 작업하는 형태의 클라이언트-서버 모델 유형

- 소스 공개 유형에 따라 오픈소스 제품과 상용 제품으로 구분

- 서로의 작업이 충돌하는 것에 대한 인식의 차이에 따라 낙관적 잠금 방식을 채택한 제품과, 비관적 잠금 방식을 채택한 제품으로 구별

- 다양한 버전 관리 시스템 : Gnu Arch, ClearCase, Visual Source Safe, CVS(Concurrent Version System) SVN(SubVersion) , Git 등

과제를 하며 느낀 점

일주일 동안 수많은 개념들을 머릿속에 넣다 보니 정신을 차릴 수가 없다. 각각의 개념과 설명들은 어렵게 기술되어 있어서 한글임에도 불구하고 도무지 이해가 되질 않았는데 비록 시간은 많이 부족했지만 과제를 수행하며 쓰이는 상황, 각 방법론의 장단점, 소프트웨어 개발 산업이 흘러온 방향 등에 대해 흐름을 잡을 수 있는 계기가 되었다.아직도 명확하지 않고 두루뭉실한 내용들이 더 많이 남아있지만 튼튼하게 뼈대를 구성할 수 있게끔 강사님께서 큰 그림을 잘 그려주셨으니 그 틀을 채워나가야 하는 것은 이제 나의 몫이라고 생각된다. 얼핏 봐서는 간단하고 쉽게 끝낼 수 있는 과제같지만 어떻게 생각하느냐에 따라, 얼마나 관심을 갖는지에 따라 내가 얻을 수 있는 것들이 끊임 없이 나와서 느낀 점을 작성하고 있는 지금도 빠르게 흘러가는 과제 마감 시간이 아쉽기만 하다.

소프트웨어를 개발하는 데 있어서 가장 어려운 것 중에 하나가 무엇을 만드는지 결정하는 일이라고 한다. 그동안의 공부 방식과 업무 처리 방식에 있어 나는 무조건 주먹구구식으로 체계없이 진행 해 왔던 것 같다. 무엇을 만들지 결정하고 진행한다는 것은 어렵고 중대한 사항인데 진중하지 못했고 성급했음을 이번 주 수업에서, 과제를 통해서 많이 느꼈다.

그렇기에 방법론을 조사하면서 최근의 트렌드라고 해서 무조건 내 업무 과정 속에 포함시키면 효율성이 극대화 되는 부분이 아님을 또한 깨달았다. 방법론을 적용하기 전에 어떤 부분에서 도움이 필요한지 고민해보고, 팀원들과 조율해보고, 생각하는 시간과 적용 과정에서 팀의 방향성과 목표에 맞게 끔 수정하고 보완할 줄도 알아야 함을 알게 되었다.

어떤 일이든 급하게 진행하면 문제가 생긴다. 업무의 효율성을 극대화 시키려면 다양한 시행착오와 오랜 시간이 걸릴 수 있음을 미리 인지하고 함께 익히는 과정 또함 필요함을 하지만 무조건 방법론적인 틀에 맞춰 생각할 필요는 없음을 생각해 본다.

그런 점에서 프로그래머라는 직업은 끊임없이 공부하고 생각하고 상상해야 하는 다소 빡쎄지만(?) 살아있는 직업 같아 참 매력적임을 이번 주 내내 수업을 들으며, 과제를 하며 생각하게 되었다.