20160108 서은정
개발생명주기, 개발방법론, 형상관리(버전관리)

개발 생명 주기 / 소프트웨어 생명 주기
소프트웨어 생명 주기(Software Life Cycle)는 소프트웨어 개발 방법론의 바탕이되는 것으로 소프트웨어의 개발, 운용, 유지보수 등의 과정을 각 단계별로 나눈 것이다.

소프트웨어 생명주기의 역할
- 프로젝트 비용 산정과 개발 계획을 수립할 수 있는 기본 골격이 된다.
- 프로젝트 진행 방향을 명확하게 파악하게 한다.
- 용어 및 기술의 표준화를 가능하게 한다.
- 프로젝트 관리를 용이하게 한다.
- 여러 소프트웨어 간에 상호 일관성을 유지하게 한다.

=> 한 마디로, 일을 효율적으로 하기 위한 방법론!

왜 이 방법을 사용해야 할까? 사용하지 않으면 어떻기에??
프로젝트에 개발 생명 주기를 적용했을 때와 그렇지 않을 때 얻게 되는 장점과 단점을 살펴보면 그 필요를 느낄 수 있지 않을까.

개발 생명 주기 없는 작업의 장점 : 시간, 명확한 목표
각 개발자들은 자신이 맡을 부분을 선택하고 작업 후 한번에 합친다. 그리고 자신에게 뭐가 필요한지 잘 알고있기 때문에 질문하러 돌아다닐 필요가 없다. -> 각자 알아서 시간을 잘 지킴.
개발자들은 주어진 문제와 그에 맞는 솔루션을 제공하는데 집중하지만 개발 생명 주기에는 개발 목표를 흐리는 많은 사람들이 참여한다. -> 어떤 문제가 주어지면 그것만 해결하면 된다. 다른 이슈가 끼어들지 않음

개발 생명 주기 없는 작업의 단점 : 문서 없음
개발 생명 주기가 없는 작업이 부적절한 이유 중 하나는 문서의 부족이다. 개발 생명 주기 없이도 소프트웨어는 개발되지만 프로그램의 개발을 지원하는 문서는 하나도 없다.(개발 상황이나 작업 진행상황에 대한 문서가 하나도 없다는 걸 의미하는 듯)
-> 개발을 하는 과정에서 문서화를 잘 한다면 나중에(제품을 출시한 이후) 문제가 생겨도 문서를 보고 유지보수를 할 수 있다. 문서가 없다면 유지보수에 많은 시간이 걸릴 것. 그리고 문서는 제품의 수명을 연장해주는 효과도 있다. 각 단계와 과정마다 문서화가 잘 되어있다면 이는 다른 프로그램을 개발하는 데도 활용될 수 있다.

개발 생명 주기를 적용한 작업의 장점 : 디테일한 계획
개발자는 무엇을 하고, 하지 않을지 명확히 구분지을 수 있게 된다. 해결책을 명확히 알고있기 때문에 자세한 개발 계획이 나올 수 있다. 개발자들은 한번에 여러 문제를 해결할 프로그램을 만들 수 있다. 개발 착수 전에 모든 것이

계획되어 있기 때문에 목표는 명확해지고 제 시간에 구현할 수 있다. 계획을 벗어나는 예외가 발생할 수도 있지만 좋은 프로젝트 매니저는 예외를 잘 처리할 것이다. -> 정확도 높은 예측이 가능하다

개발 생명 주기를 적용한 작업의 단점 : 예상과 결과의 불일치, 경직성

단점을 찾는 것은 어렵다. 하지만 누구나 생각할 수 있는 단점 중 하나는 예상물과 결과물의 불일치다. 개발 도중 기획에선 생각지 못한 일들이 발생할 수 있다. 이 경우 클라이언트에게, 특히 서드 파티 개발자에게 귀감을 주지만 제품이 출시된 이후라면 그리 좋은 상황이 아닐 것이다.

또 다른 단점은 창의적이기 보다 경직된 소프트웨어를 만들어낼 수 있다는 것이다. 개발 생명 주기에서도 개발자들이 창의성을 발휘하긴 하지만 그것은 정해진 요구 안에서만 가능한 것이다. 만약 개발의 경계나 한계가 없다면 개발자들은 더 많은 것을 해낼 것이다.

(참고 링크:http://www.learn.geekinterview.com/it/sdlc/sdlc-necessity.html)

=> 개발 생명 주기를 옹호하는 사람의 편향적인 의견이 아닌가 싶지만,

1. 명확한 목표로 예상한 기한 내에 개발이 가능하다.
2. 꼼꼼한 문서 작업으로 출시 이후에도 유지/보수가 가능하다.

결국 프로그래밍을 비즈니스 영역에 최적화 시키기 위한 방법론이 아닌가 싶다. '효율성'을 위한 '표준화'

개발 생명 주기 모델

폭포수 모델

- 고전적 라이프사이클 패러다임
- 분석, 설계, 개발 구현, 시험, 유지보수를 순차적으로 접근하는 방법

프로토타이핑 모델

- 개발 대상인 시스템 주요기능을 초기에 운영모델로 개발하는 것
- 점진적 개발방법(폭포수 모델 단점 보안)
- 일회용, 진화용 시제품

나선형 개발 모델

- 폭포수 모델변형으로, 소프트웨어를 구조적 관점에서 하향식 계층구조의수준별 증분을 개발하여 이를 통합하는 방식
- 시스템이 가지는 여러 구성요소의 핵심부분을 개발한 후 각 구성요소를 개선 발전시켜 나가는 방법

RAD기법 모델

- 사용자 주도로 요구사항 정의, 분석, 설계
- Code Generator(코드 자동화)에 의한 신속한 시스템 개발 기법

4세대 모형

- CASE 및 자동화 도구를 이용하여 요구사항 명세로부터 실행코드를 자동으로 생성할 수 있게 해주는 기법

소프트웨어 개발 방법론

소프트웨어 개발 생명주기 내의 각 단계에서의 수행 방법과 활동들을 구체적으로 정의한 것.

구조적 방법론, 정보공학 방법론, 객체지향 방법론, 컴포넌트 기반 방법론(여기 까지 폭포수 모델을 따름)/ 애자일 방법론 등이 있다. 동일한 생명주기를 따르더라도 어떤 개발 방법론을 사용하느냐에 따라 개발 방법, 결과물, 내용 등이 달라질 수 있다.

애자일에 대하여

"폭포수로 대변되는 전통적인 계획중심방법론은 엄격하고 체계적인 요구사항 분석 및 계획, 정량적인 평가, 품질관리, 문서화, 훈련된 프로세스, 주요 마일스톤에 따른 중앙 집중 관리 및 순차적인 절차를 따른다. 따라서 상대적으로 요구사항의 변화가 적은 대규모의 복잡한, 혹은 고도의 확실성을 요구하는 프로젝트에 적합하다.

반면, 애자일방법론의 기본 철학은 프로젝트 초기부터 고객의 적극적인 참여를 유도하고, 단기간의 반복/점증적인 제품 출시를 통해 시장의 반응을 빠르게 적용하며, 전문성이 높은 소규모의 팀을 꾸려 효과적인 대화와 협력을 통해 생산성을 극대화하는 것이다. 이처럼 변화무쌍한 시장과 고객의 요구 변화에 유연하게 대응함으로써 고객의 만족과 비즈니스의 성공을 이끌어내는 데에 그 핵심이 있다. (중략) 애자일은 프로젝트 가시성이 높아 현업에서 부담으로 작용할 수도 있다. 하지만 이를 사용해본 개발팀의 과반수 이상이 지속적인 사용을 원하는 이유는, 사람이 기본적인 개발 주체인 소프트웨어 프로젝트에 엄격하고 한정된 프로세스를 적용하기보다는 변화의 가능성을 열어두고 환영하는 애자일 철학이 비즈니스 성공과 더불어 개인 능력의 향상에도 큰 도움을 주기 때문이다."

(http://www.sw-eng.kr/member/customer/Webzine/BoardView.do?boardId=00000000000000021960)

+ 애자일의 원칙

1 .우리의 최우선 순위는, 가치 있는 소프트웨어를 일찍 그리고 지속적으로 전달해서 고객을 만족시키는 것이다.

2. 작동하는 소프트웨어를 자주 전달하라. 2-4주 간격으로 하되 더 짧은 기간을 선호하라.

3. 작동하고 있는 소프트웨어를 보여주는 것이 진척을 확인할 수 있는 길이다.

4. 비록 개발의 후반부일지라도 요구사항 변경을 환영 하라. 애자일 프로세스들은 변화를 활용해 고객이 경쟁력 에 도움이 되게 한다.

5. 비즈니스 쪽의 사람들과 개발자들은 프로젝트 전체에 걸쳐 날마다 함께 일해야 한다.

6. 동기가 부여된 개인들 중심으로 프로젝트를 구성하라. 그들이 필요로 하는 환경과 지원을 주고 그들이 일을 끝내리라고 신뢰하라.

7. 개발팀으로, 또 개발팀 내부에서 정보를 전하는 가 장 효율적이고 효과적인 방법은 얼굴을 보며 할 수 있는 대화이다.

8. 최고의 아키텍처, 요구사항, 설계는 스스로 움직이 는 조직적인 팀에서 탄생한다.

9. 기술적 탁월성과 좋은 설계에 대한 지속적인 결과 물의 전달이 민첩함을 높인다.

10. 애자일 프로세스들은 지속 가능한 개발을 장려한다. 스폰서, 개발자, 사용자는 일정한 속도를 계속 유지 할 수 있어야 한다.

11. 단순함, 안 해도 되는 일은 최대한 안 하게 하는 기교, 이것이 핵심이다.
12. 팀은 정기적으로 더 효과적으로 일할 수 있는 방법을 숙고하고 그에 따라 자신의 행동을 조율하고 수정한다.

=> 애자일을 개발 '방법'론이라고 하지만 '가치'와 '이념'에 더 가까운 것 같다. 가장 최우선의 목표는 고객, 사용자를 향해 있고 사용자를 만족시키는 것, 그들을 만족시키는 결과물을 지속적으로 내는 것을 중요시 한다. 그리고 이 목표는 팀원들의 커뮤니케이션과 적극성, 자발성을 기반으로 한다. 마치 사회주의 국가에서 민주주의 국가로 넘어간듯한 기분이다.
프로젝트를 진행함에 있어 커뮤니케이션이 중요함을 깨닫게 된 것도 있겠지만, 요구 사항의 다양화와 빠르게 변하는 시장상황도 애자일을 적용하게 만든 것 같다.

애자일 방법론1 : 스크럼
- 작은 목표를 주기적으로 전달하는 것(Baby Step)
- 진행을 모니터링 하고 결과에 따라 계획을 조절하는 것
- 비즈니스 목표가 변경되는 것과 개발팀이 민첩하게 대응하여 완료할 수 있는 것들의 밸런스를 조정하는 것

애자일 방법론2 : 스프린트
- 1~6주 범위 내에서 특정 기간을 선정
- 일반적으로 2주 or 4주를 가장 많이 사용함
- 스프린트 기간 동안에는 분석, 설계, 코딩 그리고 테스트와 배포까지의 모든 과정을 포함함 — Schwaber et al, Agile Software Development with Scrum, Prentice-Hall, 2002 (pp 50)

애자일 방법론3 : XP(eXtreme Programming)
1999년 켄트 백의 저서인 'Extreme Programming Explained - Embrace Change' 에서 처음 발표
10-12 개의 실천법(Practices)들로 구성

(이 외에도 많다고 한다. 참고: http://www.slideshare.net/hoonsbara/ss-43231586)

형상관리(버전관리)
매 번 변경되는 작업 결과물들을 기능적, 물리적으로 구별지어 문서화 한다. 문서화 된 결과물들에 변경이 일어날 경우 변경과 변경 후의 상태도 기록하고 관리한다.
-> 왜??? 요구사항의 변화가 많고 이러한 변화가 관련 작업자에게 제대로 통보되지 않는다. 그리고 많은 개발자들이 동일한 결과물에 대해 개별적으로 이중 작업을 한다. 다양한 사본이 존재하게 되어 작업에 대혼란이 온다.
-> 결국 프로젝트의 생명 주기 동안 제품을 잘 관리하기 위해서 + 위험 리스크를 줄이기 위해 사용

여기까지 밖에 못했습니다 ㅠㅠㅠㅠㅠ