개발생명주기

기본 개념은 소프트웨어 자체를 하나의 생명체로 간주하는 것입니다. 탄생에서 사망까지의 변환과정을 말하며 각 단계가 존재합니다.

타당성조사 > 요구분석 > 설계 > 개발 > 시험 > 유지보수 > 폐기

그리고 이 과정의 형태별로 여러 모델이 존재합니다.

-폭포수모델

고전적인 고전적 라이프 사이클 패러다임으로 분석, 설계, 개발/구현, 시험, 유지보수를 검토 및 승인을 거쳐 순차적 하향적으로 접근/개발하는 생명주기 모델입니다. 기술적 위험이 낮고 유사프로젝트 경험이 있는 경우 사용하며, 요구사항이 명확하고 산출물이 고정적일 경우 사용합니다.

낮은 복잡성과, 관리의 용이함, 풍부한 사례, 전체과정을 이해하는 것이 쉽다는 장점이 있습니다. 반대로 소프트웨어의 거대화, 요구사항의 구체적 정의 어려움, 후반문제에 대한 대처 어려움, 단계별 피드백이 발생하는 단점이 있습니다.

-원형모델

고객과의 원활한 의사소통을 통한 개발모델이며 폭포수 모델의 단점을 보완한 모델입니다. 사용자의 요구사항을 충분히 분석할 목적으로 시스템의 핵심적인 기능을 먼저 만들어 평가한 후 구현하는 점진적 개발 방법이기도 합니다.

요구사항 도출이 용이하고 시스템의 이해도와 의사소통이 향상되는 장점이 있습니다. 하지만 폐기 시 비용부담이 있고, 최종결과물로 오해를 받기도 하며, 일회용이 될 수 있다는 단점이 있습니다.

-나선형모델

폭포수 모델과 원형모델의 장점에 위험분석을 추가하여 만든 모델입니다. 시스템을 개발하며 생기는 위험을 최소화하기 위해 점진적으로 완벽한 시스템으로 개발합니다. 위험 최소화가 주목적입니다. 이는 대규모 개발 프로젝트나 국책 사업에 적합합니다.

1. 계획 수립 : 목표/제약조건을 설정
2. 위험분석 : 우선순위 기능 선택
3. 개발/구축 : 선택된 기능 개발
4. 평가 : 개발경과 평가

5. 의사소통 : 고객의 요구사항/인터뷰 계획 수정

위와같은 단계를 거치며 그 장점으로 정확한 요구를 파악하여 위험부담을 감소시키면서 품질을 확보합니다. 테스트와 피드백이 용이합니다. 또한 대규모 시스템에서 반복적 개발 및 테스트를 통해 강인성을 향상시킵니다. 그리고 한 사이클에 추가하지 못한 기능을 다음 단계에 추가가 가능합니다.

다만 관리가 복잡하고 많은 시간이 소요되며, 새로운 모형으로써 사례가 많지 않아 충분한 검증이 미흡하기도 합니다. 이때문에 관리 및 위험분석이 매우 중요합니다.

-반복적모델

사용자의 요구사항 일부분 혹은 제품의 일부분을 반복적으로 개발하여 최종 제품으로 완성하는 모델입니다. 두가지 방법이 존재합니다.

첫번째는 증분형 개발 모델로 폭포수 모델을 변형하여 소프트웨어를 구조적 관점에서 하향식 계층구조의 수준별로 분류 개발/통합하는 방식입니다. 고객과 의사소통이 원활히 일어나며 이로인해 고객의 시스템 적응과 이용성이 확대됩니다. 다만 그만큼 요구가 많아지거나 변화가 자주발생할 수 있어 구조화가 힘들어질 위험성도 있습니다.

두번째는 진화형 개발 모델입니다. 시스템의 구성요소 중 핵심 부분을 개발한 후 각 구성요소를 개선 및 발전시켜 나가는 방법입니다.

-RAD기법 모델

2~3개월의 짧은 개발 주기동안 소프트웨어를 만들기위한 순차적인 프로세스 모델로 코드 제네레이터에 의한 신속한 개발이 가능하고 폭포수 응용 형태로 컴포넌트 기반이 지원됩니다.

고객과 비지니스모델을 작성/검토를 반복하여 분석하는 단계를 지나 설계에 들어갑니다. 또한 고객과 원형모델로 개발/수정/보완으 반복하여 설계하고 마지막으로 관련 기술을 이용해 시스템을 구축하고 운영합니다. (Computer - Aided Software Engineering 사용)

이를통해 요구사항 이해와 범위가 명확할 시 신속하고 완전한 기능을 구현할 수 있습니다. 하지만 기술적 위험이 높을경우엔 부적합하며, 적절한 모듈화와 책임감 있는 구성원이 필요합니다.

-V모델

폭포수모델에서 시스템검증과 테트스, 작업을 강조한 모델입니다. 모듈의 상세설계를 단위 테스트 과정에서 검증하고 시스템 설계는 통합테스트 단계에 사용자의 요구는 시스템 테스트

과정에서 검증합니다.Validation과 Verification 단계로 구성됩니다. 감추어진 반복과 재작업을 드러내어 작업결과 검증에 초첨을 두며 신뢰성이 높이 요구되는 분야에 적용됩니다. 반복이 없어 변경을 다루기가 쉽지 않지만 오류를 줄일 수있는 장점이 있습니다.

개발방법론의 최근 트렌드

우선 최근 트렌드는 애자일 프로세스라 할수 있습니다. 이는 소프트웨어 개발 방법에 있어서 아무런 계획이 없는 개발 방법과 계획이 지나치게 많은 개발 방법들 사이에서 타협점을 찾는 방법론입니다. 계획이 없으면 향후 예측이 힘들고 효율적이지 못하고, 반대로 계획이 너무 많으면 형식적인 절차를 따르는데 필요한 시간과 비용이 많이 들며, 전체적인 개발 흐름을 느리게합니다.

그러므로 애자일프로세스 최소한의 문서화(문서화를 하지 않는 것은 아닙니다)를 지향하며 실질적인 코딩을 통한 방법론이 되겠습니다. 일정한 주기를 가지고 끊임없이 프로토타입을 만들어내며 그때 그때 필요한 요구를 수정하여 하나의 커다란 소프트웨어를 개발해 나가는 적응형방식이라 설명할 수 있겠습니다.

'애자일 = 기민한, 좋은 것을 빠르고 낭비없이 만드는것)'이라는 뜻대로 특정 개발 방법론이 아니라 기민한개발을 가능하게 하는 다양한 방법론 전체를 일컫는 말이므로 트렌드니까 무턱대고 따라하기엔 위험한 방식입니다.

배경을 잠시 짚고 넘어가자면, 애자일 프로세스는 소프트웨어 개발이 다른 공학 프로세스와 큰 차이가 있음을 인지하는 데에서부터 시작됐습니다. 90년대 후반까지 S/W 공학과 개발 방법론은 장기간에 걸쳐 많은 사람들을 투입하고 충분한 비용을 투입하여 진행하는 다른 공학의 프로세스와 비슷한 맥락에서 진행됐습니다.

하지만 S/W는 유동적이고 개방적이어서 요구사항의 변경에 따른 작업량을 예측하기 힘들었습니다. 그래서 고전적인 기법으로는 대처하기 힘들게 되었습니다. 이에 기술적 해결책으로 객체지향이 등장했고 이로써 적절하게 대처할 수 있게 됐습니다. 그리고 이를 위한 적절한 개발 프로세스가 필요하게 되었고 수많은 애자일 개발 프로세스가 이러한 필요에 따라 만들어졌습니다. 따라서 애자일 개발 프로세스의 상당수는 객체지향 기술을 기반으로 합니다.

애자일 개발 프로세스는 제한된 시간과 비용 안에서 정보는 불완전하고 예측은 불가능하다는 전제를 가집니다. 그리고 그 전제 아래에서 합리적인 답을 찾아내는 프로세스입니다.

애자일 개발 방법론에는 각자 다른 특징과 적용범위가 있으며, 서로 조합도 가능합니다. 이를 통해 독자적 방법을 만들어 낼 수도 있습니다.

-익스트림 프로그래밍

애자일 개발 프로세스의 대표자로 애자일 개발 프로세스 보급에 큰 역할을 하였스니다. 고객과 함께 2주 정도의 반복개발을 하고, 테스트와 우선 개발을 특징으로 하는 명시적인 기술과 방법을 가집니다.

-스크럼

30일마다 동작 가능한 제품을 제공하는 스프린트를 중심으로 합니다. 매일 정해진 시간에 정해진 장소에서 짧은 시간의 개발을 하는 팀을 위한, 프로젝트 관리 중심의 방법론입니다.

-크리스털 패밀리

프로젝트의 규모와 영향의 크기에 따라서 여러종류의 방법론을 제공합니다. 그중에서 가장 소규모 팀에 적용하는 크리스털 클리어는 익스트림 프로그래밍 만큼 엄격하지도 않고 효율도 높지 않지만, 프로젝트에 적용하기 쉬운 방법론 입니다.

-FDD(Feature-Driven Development)

feature(구분지어진 각각의 서비스들 혹은 기능단위)마다 2주정도의 반복 개발을 실시합니다. UML을 이용한 설계 기법과도 밀접한 관련을 가집니다.

*UML이란 소프트웨어 개발 과정에서 산출되는 산출물들을 명시, 개발, 문서화하기 위한 모델링 언어입니다. 다이어그램을 제시함으로써 소프트웨어 개발과정의 산출물들을 시각화하여 주고, 개발자들과 고객 또는 개발자들 간의 의사소통을 원활하게 할 수 있도록 합니다. UML은 시스템을 모델링 할 수 있는 다양한 도구들을 제공하기 때문에, 도메인을 모델링하기가 훨씬 용이할 뿐만 아니라 모델링한 결과를 쉽게 파악할 수 있게 됩니다. 또한 산업계 표준으로 채택되었기 때문에 UML을 적용한 시스템은 신뢰성 있는 시스템으로 평가받을 수 있습니다.

-ASD(Adaptive Software Development)

소프트웨어 개발을 혼란 자체로 규정하고, 혼란을 대전제로 그에 적응할 수 있는 소프트웨어 방법을 제시하기 위해 만들어진 방법론입니다. 내용적으로는 다른 방법론들과 유사하지만, 합동 애플리케이션 개발(사용자나 고객이 설계에 참가하는 방식)을 사용하고있는 것이 조금 다른점 입니다.

-익스트림 모델링

익스트림 모델링은 UML을 이용한 모델링 중심 방법론입니다. 다만, 여타 모델링 바업들과는

달리, 언제나 실행할 수 있고 검증할 수 있는 모델을 작성하는 공정을 반복해서, 최종적으로는 모델로부터 자동적으로 제품을 생성하게 합니다.

형상(버전)관리란?

SW개발 및 유지보수 과정에서 발생하는 소스코드, 문서, 인터페이스 등 각종 결과물에 대해 형상을 만들과, 이들 형상에 대한 변경을 체계적으로 관리, 제어하기 위한 활동입니다. 단순히 말하자면 프로젝트를 진행하면서 생성하는 소스코드를 CVS나 SVN 또는 GIT과 같은 버전관리 시스템을 이용하는 것을 말합니다. 다수의 개발자가 프로젝트에서 동일한 기능을 동시에 개발한다고 할 때, 작성된 소스코드와 변경사항을 확인하고, 수정하는 협업과정을 도와주는 시스템이라고 할 수 있습니다.

형상관리 시스템을 사용하는 이유는 프로젝트를 진행하다 보면, 잘못 만들어진 소스코드를 다시 이전 상태로 되돌려야 할 때도 있고, 변경된 이력을 확인해야 할 때가 있습니다. 그리고 여러 개발자가 동시에 같은 소스코드를 개발하면서 발생하는 충돌에 대한 처리도 필요합니다. 이러한 일들을 관리해주는 형상관리 시스템이 없다면 문제가 생겼을 때 책임을 추궁하기가 힘들어 지고 시간을 허비하게 니다.

형상관리 방법

일반적인 사람들이 형상관리를하는 방법은 파일이 저장된 순서대로 숫자나 시각을 붙입니다. 이러한 방법은 실수를 하기 쉬운데, 잘못 덮어쓰거나 무엇이 변경되었는지 일일히 확인하기가 힘듭니다. 이런 문제들을 해결하기 위해 프로그래머들은 간단한 데이터베이스에 파일의 변경사항을 추가로 기록하는 로컬버전 관리 시스템을 만들었습니다.

하지만이것으로 끝이 아니였습니다. 시스템 외부에 있는 개발자들과 함께 작업하는 것은 불가능 했기에 중앙집중식 버전관리시스템을 만들었습니다. 버전이 관리되는 모든 파일을 저장하는 하나의 서버와, 이 중앙 서버에서 파일들을 가져오는 다수의 클라이언트를 상정합니다. 가장 대표적이고 오래된 버전관리 방식입니다. 로컬방식과 비교해보면 누구나 다른 사람들이 무엇을 하고 있는지 알 수 있고, 관리자는 누가 무엇을 할 수 있는지 꼼꼼하게 관리할 수 있게됩니다.

허나 여기에도 단점은 있었습니다. 중앙 서버가 잘못되면 모든것이 잘못된다는 점입니다. 서버가 다운될 경우 서버가 다시 복구될때까지 다른 사람과의 협업도, 진행중이던 작업을 버전관리하는 것도 불가능해집니다. 중앙 데이터베이스가 저장된 하드디스크에 오류가 발생하고

백업도 없다면, 사람들이 각자 자신의 컴퓨터에 가지고있던 스냅샷 외에는 그동안 쌓인 프로젝트의 이력을 모두 잃게 됩니다.

이를 해결하기위해 분산버전관리시스템이 개발 됐습니다. 여기선 클라이언트가 파일들의 마지막 스냅샷을 가져오는 대신 저장소를 통째로 복제합니다. 따라서 서버에 문제가 생겨도 어느 클라이언트든 복제된 저장소를 다시 서버로 복사하면 서버가 복구됩니다. 체크아웃을 할 때마다 백업이 일어나는 셈입니다.

게다가 다수의 원격 저장소를 갖는 것이 가능해졌기 때문에 동시에 여러 그룹과 여러방법으로 함께 잡업할 수 있게 됐습니다. 이로인해 계층 모델등 중아집중 시스템에서는 할 수없는 다양한 작업 방식들을 사용해볼 수 있습니다.