## 소프트웨어 생명 주기

컴퓨터 소프트웨어의 개발 단계의 총체로써, 초기 개발 단계를 시작으로 마지막 출시가 되고 나서까지의 모든 과정을 말합니다. 이 과정에는 소프트웨어 업데이트 버전, 버그 수정 등의 개선 내용을 포함합니다.

이 개발의 단계를 보면

프리 알파 -> 알파 -> 베타 -> 오픈베타 or 클로즈 베타 -> 출시 후보로 나뉘는데, 프리알파(Pre-AlPha)는 테스트하기를 하기 전에 프로젝트를 진행하는 동안에 수행되는 모든 활동을 말합니다. 이 활동에서는 소비자의 요구사항을 분석하고 소프트웨어를 설계, 개발하며 테스트 등을 합니다.
알파(Alpha)는 소프트웨어의 생명 주기 중 한 단계로, 소프트웨어 테스트를 시작하는 첫 단계를 말합니다. 한 마디로 생명 주기 과정 중 최초로 테스트를 하는 단계라고 말할 수 있습니다. 그만큼 불안정하며 충돌이나 데이터의 손실을 일으킬 수 있습니다.
베타(Beta)는 소프트웨어가 제 기능을 완성할 때부터 일반적으로 이 단계가 시작됩니다. 베타 단계의 소프트웨어는 아직 온전히 완성된 상황이 아니어서 수 많은 버그가 존재합니다.
오픈 및 클로즈 베타(Open or Closed Beta)는 완성된 소프트웨어를 소비자에게 혹은 타겟층에 출시 함으로써 테스트를 실시하는 것인데, 클로즈드 베타는 몇몇의 제한된 그룹으로 출시하는 반면에 오픈 베타는 보다 많은 그룹을 대상으로 테스트를 하는 것이라고 볼 수 있습니다. 이 테스트는 이전 단계보다 훨씬 다양한 사용자에게 테스트를 하는 만큼 오류가 많아지고 수 많은 수정과 업데이트가 일어나는 과정으로 볼 수 있습니다.
출시 후보(Release Candidate)는 수 많은 업데이트와 버그 수정을 통해 최종 단계에 있는 소프트웨어로써 치명적인 오류가 나지 않는다고 판단될 시 출시할 계획을 갖고 있는 소프트웨어가 이 단계에 속합니다.
GA(General Availability)는 상업화 활동을 통해 매체를 통해 시장에서 이용 됨을 말합니다.

## 소프트웨어 개발방법론

과거 개발 프로세스들은 정해진 과정과 세밀한 계획을 바탕으로 개발을 진행시키는 방법들이 대부분이었고, 이것은 빠른 이해를 도우며 사용하기도 쉽고 바람직한 개발 과정으로 오랫동안 여겨져 왔으나, 최근 소프트웨어의 몸집이 커지면서 개발 진행도중 여러 가지 예상 이외의 오류와 부작용으로 계획에 차질이 생기게 되고 이를 보완하기 위해 시간 싸움을 시작하게 되면서 개발자들의 철야, 출시일 지연, 지연으로 인한 스트레스와 클라이언트의 비난으로 인한 개발자들의 에너지 소진 등등 문제점들이 생기게 되었습니다.
이러한 문제점들이 생겨나기 시작하면서 최근에는 아무런 계획 없이 진행하는 개발과 지나친 계

획과 함께 진행되는 개발에서 타협점을 찾은 애자일 개발방법론이 대두되고 있습니다.
기존의 공학적인 프로세스와는 엄청난 차이가 있음을 깨달은 후부터 소프트웨어 산업의 위기를 극복해 내기 위해 제작된 방법인데 애자일 개발방법론은 기본적으로 제한된 시간과 비용 속에서는 전제 안에서 합리적인 답을 내도록 고안되었습니다.

**애자일 개발 프로세스**

익스트림 프로그래밍 – 애자일 개발 프로세스의 대표적인 방법으로 최근 애자일 개발 프로세스 보급에 큰 역할을 하였습니다. 이 방법은 고객과 함께 2주 정도 반복 개발을 하고, 테스트와 우선 개발을 특징으로 하는 명시적인 방법을 사용합니다.

스크럼 - 30일마다 동작 가능한 제품을 제공하는 방법을 사용하며 매일 정해진 시간에 정해진 장소에서 짧은 시간의 개발을 하는 팀을 위한 프로젝트 관리 중심의 방법론입니다.

애자일 개발 프로세스들은 각자 다른 특징들과 소프트웨어 규모마다 적용하는 범위가 다르며, 각각 다른 프로세스마다 조합해서 개발하는 것도 가능합니다. 특히 애자일 방법론은 위의 정해진 프로세스의 과정을 곧이곧대로 맞추는 것이 아니라 개발 환경과 여건에 따라서 독자적인 프로세스를 만들어 사용함으로써 큰 효과를 올리고 있습니다.

## 형상관리

형상관리란 다른 말로 소스 버전 관리라고 말하기도 하는데, SCM이라는 단어로 표현하기도 합니다. SW개발 및 유지보수 과정에서 발생하는 소스코드, 인터페이스 등등 결과물에 대해 형상을 만들고, 이들에 대한 변경을 체계적으로 관리, 제어하기 위한 활동을 말합니다.
단순히 정리하자면 프로젝트를 진행하면서 생성되는 소스코드들을 CVS, GIT과 같은 버전관리 시스템을 이용해서 관리하는 것을 말합니다.
이러한 버전 관리 시스템이 필요한 이유는 프로그램을 만들다 보면, 잘 못된 코딩으로 인해 코딩을 하기 이전 상태로 되돌리거나, 변경된 이력들을 확인 할 필요가 있기 때문입니다. 그리고 여러 명의 개발자들이 동시에 같은 소스코드를 개발할 때 발생하는 여러 가지 충돌을 해결하는데 필요하기도 합니다. 시대가 지남에 따라서 소프트웨어의 규모가 점점 커지며 이런 버전관리 시스템이 없다면 잘 못된 소스코드 개발을 누가 했는지 알아낼 길이 없으며, 실수를 쉽게 되돌릴 수 있는 방법도 없게 됩니다. 이러한 문제들을 사전에 방지하기 위해서 버전 관리 시스템이 필요한데 버전 관리 시스템은 상용(IBM Rational ClearCase, Perforce, PTC Integrity)과 비 상용(Subversion, CVS, Git)이 있습니다.