embarcadero®

소프트웨어 개발 라이프 사이클(SDLC)에서의 팀 협업

도입

소프트웨어를 개발할 때 다른 사람의 협조와 협업은 늘 어느정도 필요했지만 지금과 같지는 않았다. 개발자 혼자서 아주 간단한 애플리케이션이나 유틸리티를 만들더라도 유지관리, 확장, 수정, 오류 해소를 하는 누군가는 있어야한다.

팀 협업이 견고할 때 생기는 장점은 분명하다. 어느 한 사람에게 문제가 발생하면 다른 사람이 들어와서 맡아 주기가 쉽다. 고객의 니즈와 요구사항의 변화가 빨라지면서 이제는 소프트웨어를 매우 짧은 시간 안에 개발, 출시해야 할 뿐만 아니라 품질 또한 좋아야 한다. 이럴 수록 팀 협업은 반드시 필요하다.

소프트웨어를 개발할 때 당면하게되는 이런 현실의 어려움을 이겨내려면, 생산성 높은 협업 환경을 구축할 수 있는 새로운 문화를 형성하고 협업 도구를 활용해야 한다. 코드를 안전하게 공유 및 수정할 수 있고, 조직 구성원 모두가 직급과 관계없이 의사소통에 참여할 수 있고, 현대식 소프트웨어 개발 라이프 사이클 (SDLC)에 방해가 되는 불필요한 규칙들을 제거할 수 있는 방법론이 필요하다. (예: 애자일, 데브옵스, 신속 반복 프로세스 모델)

프로젝트 매니지먼트 인스티튜트(Project Management Institute)에 따르면, 프로젝트 세 개 중 하나는 의사소통 부족으로 인해 실패한다. 프로젝트 실패가 회사의 손익까지 바꾸게 된다면, 결국 사업 성공과 실패를 가르는 요인이 될 수도 있다. 경영진이 개발자를 위해 열린 의사소통과 정보 공유를 보장하고, 관련 협업 도구에 투자하면 의사소통 문제를 현격하게 낮출 수 있다.

협업 환경

팀 협업을 중시하는 소프트웨어 개발 환경에서는 팀 구성원 모두가 팀의 목표를 알고, 그 목표를 중심으로 함께 일한다. 설계, 개발, 테스트, 운영 단계을 따로 구분짓지 않고, 프로젝트 구성원들 모두가 프로젝트의 모든 단계에 참여한다. 올바로 수행되면 다음과 같은 결과를 얻게 된다:

- 의사소통이 더 좋아진다.
- 요구사항 변경에 신속하게 적응하는 능력이 생긴다.
- 문제점을 더 쉽게 파악한다.
- 한계를 더 잘 이해한다.
- 개선 가능한 부분을 더 잘 파악하고 개선한다.
- 출시가 더 빨라진다.
- 결과물의 품질이 더 높아진다.
- 지원하기가 더 쉬워져서 지원 비용이 더 낮아진다.

SDLC 모델
(Software Development Lifecycle Models)

협업에 대해 더 자세히 살펴보기 전에, 협업 향상을 위해 고안된 모델 몇 가지를 살펴보자. SDLC란 설계, 개발, 테스트, 배포 등 소프트웨어 개발이 시작되고 끝나는 과정 전체를 말한다. 대부분 정해진 양식에 따라 SDLC를 명시적으로 정의한다. 그렇지 않은 경우에도 결국 SDLC는 다음과 같은 계층으로 나뉜다.

01 계획 (Planning)
02 정의 (Defining)
03 설계 (Designing)
04 구축 (Building)
05 테스트 (Testing)
06 배포 (Deployment)

매우 많은 SDLC 모델이 있지만, 그 중에서 가장 잘 알려진 방법론 몇 개만 뽑아서 비교해보자.

폭포식 (WATERFALL)

윈스턴 로이스(Winston Royce)가 자신의 논문에서 '**순차적인 선형 라이프 모델 (*linear sequential life-model*)**은 업무 수행이 잘 되지 않는 회사와 방법론'이라고 설명하기 위해 폭포수 모델(WATERFALL) 이라는 용어를 사용한 것에서 비롯되었다. 로이스 자신은 폭포수 모델이 수많은 문제를 일으킨다고 지적했지만, 안타깝게도 폭포수 모델 (Waterfall model)은 사전 정의가 먼저 완성되고 다음 단계로 넘어가는 방식을 표현하는 용어로 사용되고 있다. 폭포식이라고 표현된 이유는 프로젝트 내의 각 작업이 단계별로 차근차근 흘러내려가고, 앞선 단계를 끝내면서 완성한 결과물을 다음 단계를 담당하는 그룹에게 넘기는 방식으로 진행되기 때문이다.

폭포식은 철저하게 문서 집중식 프로세스이다. 가장 먼저 요구사항 문서를 만든다. 이것을 개발팀에게 넘겨주면, 개발팀은 프로젝트를 구축하고 구현을 완료한다. 완성된 프로그램은 품질 보증팀(QA)에게 전달되어 테스트 과정을 거친다. 마지막으로 모든 것이 확실하게 검증되었다고 판단되면 배포를 하는데, 배포 후 운영팀에서 유지보수 하는 것은 차후에 생각할 문제로 간주한다. 폭포식에서는 팀 별로 각각 고립된 상태에서 업무 수행을 한다. 만약 테스트 중에 문제가 발생한다면, 프로젝트는 개발팀에게 돌아가고 개발 단계가 다시 시작된다.

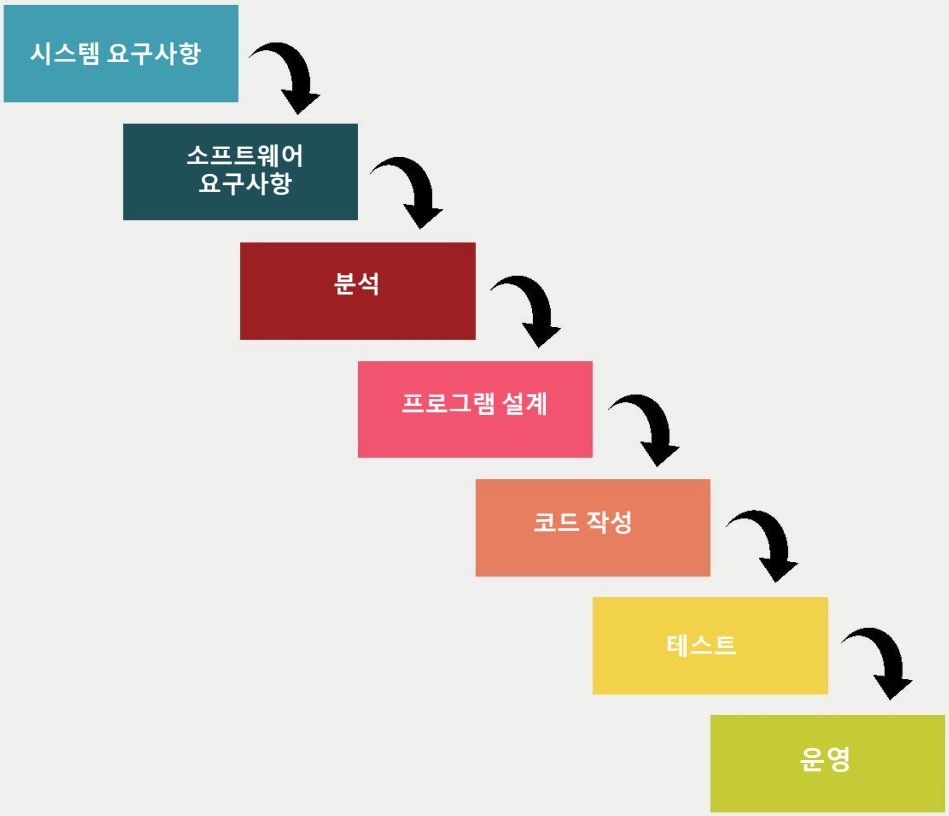


그림 2. 폭포식 모델의 파이프라인 단계

[1] Winston W. Royce (1970). "Managing the Development of Large Software Systems" in: Technical Papers of Western Electronic Show and Convention (WesCon) August 25–28, 1970, Los Angeles, USA.

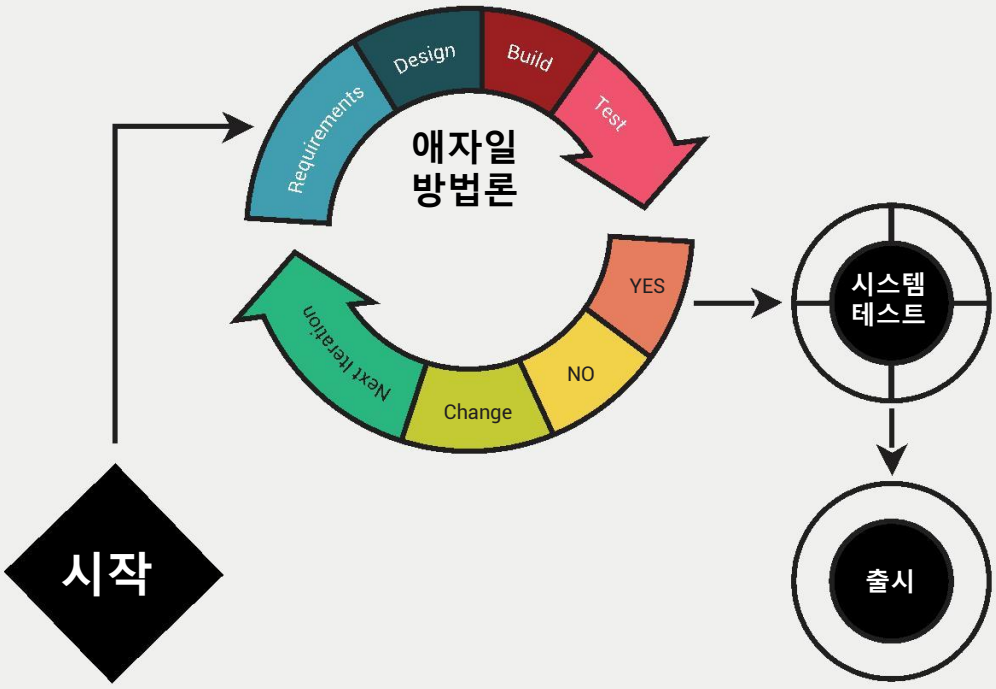
각 그룹 간에는 거의 협업을 하지 않는다. 전통적인 소프트웨어 개발 주기가 대부분 이 모델을 따른다. 대체로 계층 구조가 명확한 대형 조직일수록 폭포식 방법론을 좋아하는데, 그 이유는 요구사항부터 설계, 코딩, 테스트, QA 완료까지 전 과정에서 프로젝트 진척도와 책임 소재를 파악하기 쉽기 때문이다. 폭포수 모델은 요구사항을 모두 취합하고 나서야 프로젝트가 진행되는 (그후에는 시스템이 배포될 때까지 요구사항을 거의 변경하지 않는다) 순차적 프로세스이므로 변경 사항을 반영하려면 시간이 매우 오래 걸린다. 따라서 배포된 소프트웨어가 정작 비즈니스의 니즈를 충족하지 못하는 상황이 발생하기도 한다. 일반적으로 구상에서 최종 출시 까지는 최소 여섯달 이상 걸린다. 요구사항 단계가 완료되고 나면 더이상 요구사항을 재검토하지 못한다는 점이 이 폭포수 모델의 가장 치명적인 단점이다.

요즘은 규모와 관계없이 모든 조직에서 협업형 방법론을 적용하여, 고립을 최소화하고, 소프트웨어를 더 빠르게 출시하고, 요구사항이나 기술 변경을 빠르게 반영하려고 노력한다. 대표적인 협업형 방법론으로는 다음 세가지가 있다.

협업형 방법론 1: 애자일 (AGILE)

애자일은 각 프로젝트는 분리하여 관리하고, 주어진 요구사항을 충족할 수 있는 알맞은 방법을 프로젝트에 적용하는 것을 전제로 한다. 애자일에서는 설계자, 개발자, 테스트 엔지니어가 한 팀을 이루고, 프로젝트 설계부터 완성까지 함께 작업한다. 팀 구성원은 다른 구성원이 무엇을 하는지 알아야 한다. 따라서 프로젝트에서 어느 한사람이 빠지면 다른 구성원이 신속하게 해당 업무를 맡을 수 있다.

프로젝트는 스프린트(Sprint)라고 하는 작은 작업 단위로 쪼개진다. 각 스프린트는 할일 목록, 단기 목표, 완료일이 매우 명확하게 정해지고, 팀 구성원 모두가 동시에 함께 진행한다. 작업을 스프린트로 작게 나누면, 요구사항이 변경된다고 해서 몇 달이나 걸려서 작성한 코드를 버리는 일이 생기지 않는다. 스프린트는 작업 내용이 시작 단계부터 명확하게 정해져야 진행될 수 있다. 따라서 프로젝트 범위가 슬그머니 커지는 경우를 방지할 수 있다. 스프린트 중에 문제가 생기면, 그 문제는 다음 스프린트로 미루고, 계속 개발을 진행한다. 오직 정해진 스프린트의 목표만을 완수하면 된다. 정해진 목표 이상을 하지는 않는다.



애자일 방법론은 프로젝트 진행 중에 "테스트할 수 있는 작동하는 소프트웨어" 작성을 목표로 제시하기 때문에, 각자의 임무와 상호 작용에 집중하는 응집력 높은 팀을 만들 수 있다. 애자일은 고객 요구사항과 소프트웨어 요구사항 사이 그리고 개발팀과 테스트팀 사이의 커뮤니케이션 문제를 해결하고자 고안되었다. 고객 또는 "고객의 소리"를 반영하는 누군가가 프로세스에 참여하고 소프트웨어가 요구사항을 충족하는 지를 확인한다. 애자일은 개별 조각을 만들고 반복하는 방식이므로, 손쉽게 되돌릴 수 있고, 이슈를 다시 반영한다고 해서 설계/개발 프로세스가 지연되지 않는다. 그 결과, 애자일 방법론을 도입하면 변경사항을 신속하게 반영하고 짧은 시간 안에 높은 품질을 실현할 수 있다.

협업형 방법론 2: 린 (LEAN)

린 프레임워크는 반복형 접근법을 소프트웨어 개발에 반영한다. 린은 일본의 토요타 (토요타 생산 시스템)에서 처음 개발되었는데, 제한된 리소스만으로 고품질을 실현하기 위해 조직을 수평적으로 만들었다. 프로젝트에서 각 부분은 서로 다른 스트림 (또는 칸반이라고도 한다)으로 나누어진다. 각 스트림 즉 칸반(Kanban) 안에서는 작업은 각자 나누어지지만 동시에 일제히 진행된다. 각 스트림별로 작업 흐름을 관리하는데, 워크 인 프로세스 (work-in-progress, WIP) 한계를 설정하여 팀에게 과도한 부담을 주지 않도록 한다. 린은 공식적인 품질 방법 대부분이 그렇듯이 Dr. W. Edwards Deming 과 J. M. Juran 의 영향을 받았다. 따라서 프로세스를 통계적으로 관리함으로써 결함을 방지하는 것을 강조한다. 폭포식과 반대로 린은 설계에서 출시까지 소요되는 시간이 현격하게 단축된다. 린 생산은 저스트 인 타임 (Just-In-Time, JIT) 낭비 줄이기 방법론을 사용한다. 이 방법론은 필요할 때 정확한 수량을 보유하고 있어야 하므로 리소스를 세심하게 관리해야 한다.

린이 폭포식과 다른 점은 개개인이 해당 스트림 안의 작업 양상을 인지할 수 있다는 점이다. 칸반 안에 참여하는 사람들은 테스트 단계로 가기 전 품질을 향상을 확인하는 과정에서 문제를 찾게되면 액티비티를 멈출 수도 있다. 테스트는 이 프로세스에서 빠질 수 없는 중요한 부분이라고 여겨진다.

린은 (더 큰 그룹에서는 공유하기 쉽지 않은) 특별한 지식을 가진 개인들을 작은 그룹으로 분리할 수 있다는 장점을 제외하면, 각 스트림 내에서는 애자일과 같은 특징을 가진다. 이 방법에서는 각 개인이 자신들이 맡은 일을 각각 완료할 수 있으며, 다른 스프린트가 완료될 때를 기다리지 않아도 된다.

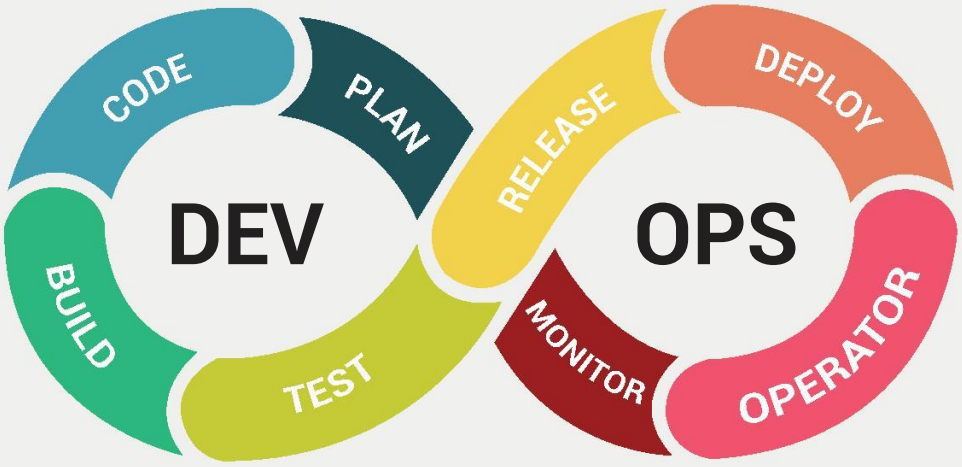
칸반은 그룹 안에서 합심하여 서로 협업 잘하는 그룹 안에서 개개인이 짝을 이룰 수 있다는 장점도 있다. 예를 들어 서로 함께 잘하는 내성적인 개인은 외향적인 사람들과 다른 스트림에 들어갈 수 있다.

협업형 방법론 3: 데브옵스 (DEVOPS)

애자일이 제품 요구사항과 설계팀 사이 그리고 개발자와 테스트 엔지니어 간의 커뮤니케이션 향상에 초점을 맞추고 있다면, 데브옵스는 개발/테스트 팀과 운영/IT 인프라 팀 사이의 간격을 좁히는 것에 초점을 둔다. 데브옵스는 IT부서 내에서 프로세스를 시작부터 끝까지 관리할 수 있도록 특별히 고안되었다. 애자일과 마찬가지로, 데브옵스 역시 변경 적응에 초점을 두면서도, 소프트웨어를 끊임없이 테스트하고 배포하는 것에도 관심을 가진다. 데브옵스에는 명확하게 정의된 프레임워크가 없지만 그 대신 협업 자체를 최우선으로 한다.

데브옵스는 가치 스트림 맵핑이라는 개념을 사용한다. 병목을 식별하고 품질을 향상하고 소프트웨어 개발 프로세스 전반에 걸쳐 효율성을 향상하려고 한다. 개발팀과 운영팀 간의 충돌뿐 아니라 각 단계에 팀들 간에 발생하는 충돌을 줄이는 것이 목적이다. 이 프로세스는 끝없이 반복된다;

다시 말해서 병목 하나를 해소하고 나면, 그 다음 제약 사항을 개선한다. 어느 시점이든 이슈가 발생하면 QA 와 개발팀에게 다시 넘긴다.



이 세가지 방법들 즉 애자일, 린, 데브옵스의 구성 요소들은 필요하면 서로 바꿔서 사용될 수도 있다는 점을 아는 것이 중요하다.

”이상적인 협업”은 어떤 모습일까?

팀 관계를 둘씩 나누어 보면, 어떻게 하면 팀 사이에 좋은 협업 관계를 찾고 발전시킬 수 있는지를 알 수 있다.

디자이너와 개발자

설계자와 개발자 사이에 이상적인 협업은 상대방이 정보를 만들고 관리하는 방법을 깊이 이해하는 것부터 시작된다.
그렇게 되면 사용성과 기능 요구사항을 충족하고, 기술적으로 구현가능한 설계를 개발자에게 제공할 수 있으므로, 재작업 위험이 적다.

설계자는 와이어프레임 도구(화면 청사진 제작 도구)를 사용하고, 개발자는 전달받은 화면 UI 와 그래픽 요소를 손쉽게 개발 도구로 가져올 수 있도록 한다. 설계자들은 템플릿을 사용하여 화면 요소, 기능, 제약 등을 정의하여 개발자가 작업할 수 있는 모델을 제공한다. 개발자는 가장 적합한 언어를 선택하여 설계자의 의도에 맞게 구현한다.

개발자와 테스트 엔지니어

데브옵스(DevOps) 모델에서 개발자와 테스트 엔지니어는 품질보증 (QA)을 함께 책임진다. 지속적 반복 테스트를 하면 개발 주기 동안 충돌이 줄어들고, 초기에 이슈를 찾아서 해소하며, 지속적 배포(Continuous delivery, CD)를 실현할 수 있게 된다.

유닛 테스트를 품질 테스트 도구를 사용하여 자동화하면 개발팀과 테스트팀 사이의 협업이 향상되고 조직은 지속적 프로세스에 더 가까이 가게 된다. 자동화할 수 없는 수작업 테스트에 대해서는 잘 개발된 테스팅 프레임워크를 통해 전체 작업 흐름을 향상하고 병목이 될 만한 요소를 제거할 수 있다. 이 테스트 프레임워크는 종종 개발자의 직접적인 참여가 필요하다 (일명, "테스트 중인 소프트웨어 개발 엔지니어" (SDET, Software Development Engineer in Test) 라고 한다.)

알맞은 유스 케이스를 알맞은 시간에 활용하는 테스팅 프레임워크와 테스트 자동화는 의사소통이 좋아야만 만들 수 있다. 테스트 엔지니어들은 테스트 소프트웨어의 목적이 무엇인지를 충분히 이해할 수 있도록 도울 수 있고 도와야 한다. 그래야 더 좋은 테스트 코드가 만들어진다. 개발자와 테스트 엔지니어 양쪽의 지식이 모두 담긴 테스트 환경을 통해 일찍 테스트를 하면, 대개의 경우 품질 보증 (QA) 단계에 갔을 때, 품질은 이미 현격하게 더 좋은 상태이다.

개발(DEV)/테스트(TEST)와 운영(OPS)

개발팀과 운영팀이 함께 일하면 시스템 사용과 제한이 반영된 더 좋은 유닛 테스트를 만들 수 있다. 개발자에게 유닛 테스트는 익숙하지만, 운영팀 대부분은 세상 밖의 이야기라고 생각하기도 한다. 운영팀이 개발 프로세스에 참여하면 보다 쉽게 빌드 자동화에 적응할 수 있다.

개발자와 테스트 엔지니어 사이의 의사소통을 향상하고 자동화할 때 사용되는 프로세스를 개발/테스트 팀과 운영팀 사이에도 그대로 적용하는 것이 좋다. 만약 도구를 통해서 목적에 대해 의사소통을 분명하게 하고 테스트 프로세스를 넘어서서 할 수 있다면 의사소통 프로세스의 대부분이 자동화될 수 있다.

개발자는 시스템의 제한요소가 무엇인지, 어떤 구성이 필요한 지를 알게되고, 운영팀은 최적화된 진행 성과를 위해 무슨 리소스를 할장하고 제공해야하는지를 알게 된다. 이 프로세스 자체는 반복되기 때문에 당연히 모니터링되고 시간이 갈수록 조정된다. 하지만, 의사소통이 견고하면 이 모든 것이 더 쉬워진다.

더 나은 협업을 위한 도구

아래 도구들은 모든 수준에서 커뮤니케이션과 협업을 크게 향상시킨다.

프로젝트 관리 도구

- 지라(JIRA) – 이슈 추적 및 프로젝트 관리 도구로 애자일 환경에 적합
- 아사나(ASANA) – 작업 관리 플랫폼
- 슬랙(Slack) – 채팅 및 협업 도구. 의사소통을 위한 팀별 주제별 등 다양한 채널을 만들고, 코드 공유, 문제 해결, 현황 업데이트 등등을 비동기 방식으로 소통할 수 있다.

디자이너/개발자 도구

- FMX스텐실 (FMX Stencils) – 디자이너와 개발자 사이의 의사소통과 협업을 향상시킬 수 있는 도구이다. 디자이너와 개발자가 동일한 언어와 같은 화면 요소를 사용할 수 있어, 의사소통 오류는 줄어들고 프로토타입과 앱 개발을 촉진할 수 있다.
- GUI 템플릿 (GUI Templates) – RAD스튜디오에서 앱 화면 UI에 테마를 일관성 있게 반영한다.
- 프리미엄 스타일 (Premium Styles) – RAD스튜디오에서 서로 다른 장비에 배포할 때 각 장비 상황에 맞추면서도 디자인 일관성을 유지한다.
- 장비에서 실시간 미리보기 (Live On-Device Previews) – RAD스튜디오에서 디자인/개발되고 있는 내용에 대한 피드백 공유 주기가 짧아진다.

개발/운영 (Dev/Ops) 도구

- DUnitX 와 DUnit – RAD Studio 용 유닛 테스트 자동화 도구는 개발 / 테스트 / 운영 간의 협업을 지원한다.
- 어셈블라 (Assembla) – 엔터프라이즈 급 소스 코드 및 리포지토리를 안전하게 관리할 수 있다.
- 래노렉스 (Ranorex) 자동화 된 GUI 테스트는 프로그램 품질을 향상시키고 개발 팀과 테스트 팀 간의 마찰을 줄인다.
- RAD스튜디오 (RAD Studio) – 엠바카데로 사의 RAD스튜디오는 C++과 델파이 개발자들을 위한 공통된 통합 개발 환경(IDE) 이다.

토의 사항 / 당면 과제

협업형 SDLC 방법론을 도입할 때 가장 큰 장애물은 기술적 도구 구현이 아니라 효과적으로 팀과 조직 전반의 문화를 바꾸는 것이다. 개개인은 성격, 학습 능력, 작업 스타일이 제각각이고 익숙한 프로세스에서 나오기를 거부하기도 할 것이다. 따라서 이슈와 문제가 정직하고 신속하게 제기되로 나면 바로 해소될 수 있도록 환경을 안전하게 만들기 위한 배려가 필요하다.

협업을 바로 시작할 수 있도록, 많은 조직들은 개별 사무실이나 칸막이를 없애고 개방된 작업 환경이 되는 사무실 배치 변경부터 시작한다. 성격이 비슷한 사람들은 함께 있게 되면 이런 환경에 잘 적응한다. 내성적인 사람들은 자신에게 주어진 과업에 집중할 때에는 다른 사람들과 함께 일하는 데 문제가 없다. 하지만, 외향적인 사람들은 잘 배려된 환경이 있어야 팀 안에서 다른 사람들과 의견이 맞지 않을 수 있는 상황에서 어려움을 겪지 않는다. 외향적인 사람들은 쉽게 산만해지고 내성적인 사람들을 방해하는 경우도 있다. 충돌이 커지면 문제가 된다. 작업 흐름을 변경하는 것 자체가 바로 커뮤니케이션 문제일 수도 있다. 해결책 하나는 린(Lean)/칸반(Kanban) 접근법을 사용하여 같은 마음으로 진행하도록 사람들을 배치하는 것이다.

애자일과 데브옵스에 적응하는 시간이 매우 느리게 보일 수 있다. 처음에는 팀 생산성이 떨어지는데 일단 새 도구, 새 프로세스를 편하게 사용하게되고 문제 해결을 이끌 수 있을 정도로 능력이 향상되고나면 극복된다. 변화는 어색하기 마련이다. 사람들이 새로운 협업형 작업 흐름에 적응하는데는 시간이 필요하다. 첫 몇달 정도가 지나면 향상되는 것이 눈에 보이기 시작하고, 생산성은 기존 체계에서는 생각하지 못한 수준으로 높아진다.

과거의 모델과 비교하면, 비용 대비 효과 분석이 강력하게 추진되면 어떤 조직이든 협업형 작업흐름으로 가게 된다. 기존 체계 하의 프로젝트 실패의 위험을 고려한다면 더욱 분명한 사실이다. 고객 만족도를 향상시킬 수 있고, 시장 출시 속도가 향상되며, 작업 환경이 상당히 좋아진다는 점에서 올바른 선택임이 분명하다.

효율성과 보안성이 높은 엠바카데로의 협업 기술

엠바카데로 제품 - C++빌더, RAD 스튜디오, 델파이 - 는 엔터프라이즈 소프트웨어 개발팀이 효율적으로 협업하면서 멋진 소프트웨어를 개발하고 배포할 수 있도록 한다.

엠바카데로의 FMX 스텐실, FMX GUI 템플릿, 프리미엄 스타일은 디자이너와 개발자들이 함께 긴밀하게 일할 수 있도록 한다. 디자인 작업과 앱의 최종 UI 화면사이에 오류 발생 여지가 거의 없다.

RAD 스튜디오에서는 동일 프로젝트에서 C++과 델파이를 모두 사용할 수 있다. 따라서, 개발자는 선호하는 언어를 사용하면서, 두 언어에서 제공되는 다양한 라이브러리를 활용하며 더 큰 프로젝트에 반영하고 동일한 도구 안에서 컴파일할 수 있다.
RAD 스튜디오는 코드 변경을 추적하고 관리하는 여러가지 도구를 제공한다. 비교 및 병합 뷰어는 코드 내에서 달라진 곳을 빠르게 찾아내므로 개발자들은 동료 개발자가 추가한 코드가 무엇인지 식별하고 이해할 수 있다. 주요 소스 버전 관리 시스템 - 서브버전(Subversion), 깃(Git), 머큐리얼(Mercurial) - 들이 RAD 스튜디오와 통합될 수 있다.

따라서 테스트팀과 운영팀은 개발팀에서 반영한 코드를 더욱 신뢰할 수 있다.

RAD 스튜디오에는 MSBuild 또는 CMake 를 사용하여 지속적 빌드 구성을 빠르게 통합하기에 적합하도록 명령줄 컴파일러가 제공된다. 또한 허드슨(Hudson), 젠킨스(Jenkins), 래노렉스(Ranorex) GUI 테스팅 등 외부 도구 그리고 DUnitX/DUnit 에서 제공하는 내장 유닛 테스트 도구를 통해 지속적 통합 (Continuous Integration, CI)을 견고하게 구축할 수 있다. **델파이와 C++빌더로 개발된 네이티브 애플리케이션은 자바 가상머신 (Java VM), 닷넷(.NET) 또는 다른 특정 언어의 런타임에 대한 의존성의 아예 없거나 거의 없는 경우가 대부분이다. 설치 모델은 "X-Copy 배포"에 가깝기 때문에 개발팀과 배포/설치팀 간의 협업이 훨씬 더 단순해진다.**

협업형 SDLC 방법론을 처음이든 아니면 회사에서 데브옵스(DevOps)를 충분히 도입하고 있든, 엠바카데로 개발 도구는 스마트한 통합팀이 더 좋은 소프트웨어를 설계, 코딩, 배포할 수 있는 기반을 제공한다.

엠바카데로에 대하여: 엠바카데로는 애플리케이션 개발자들이 생산성과 소프트웨어 출시 관련 문제를 잘 해소할 수 있도록 다양한 도구를 제공합니다. 엠바카데로 기술을 통해, 개발자들은 애플리케이션 설계, 개발, 배포를 할 수 있습니다. 시간과 비용, 어려움은 줄이고 단일 소스코드 베이스만으로 주요 플랫폼 모두에 네이티브 애플리케이션을 배포할 수 있습니다.

엠바카데로 제품 홈페이지:
https://www.devgear.co.kr/products

embarcadero®

An Idera, Inc. Company