애 자 일 초 심 자 를 위 한

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

차례

01

OVERVIEW

애자일 선언문

애자일 기원 및 종류

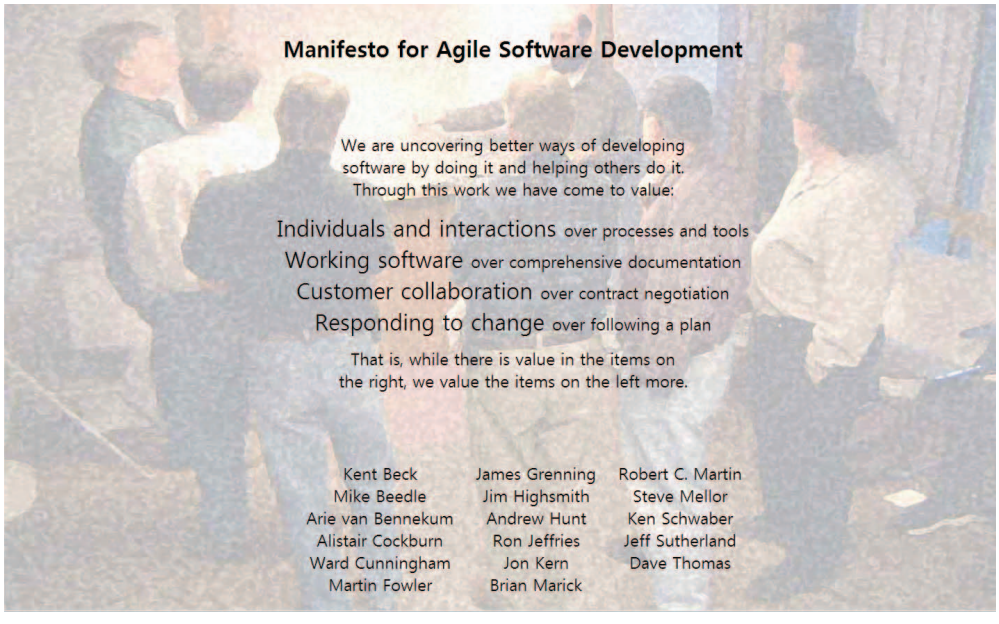
스크럼

XP

린

애자일 선언문

Manifesto for Agile Software Development



| 그림 1-1. 애자일 선언문 원문

2001년 1월, "애자일 연합(Agile Alliance)"이라는 그룹에서 '애자일 선언문'이라는 공동의 선언서를 만들어 냅니다. 이 문서는 지금까지도 애자일 소프트웨어 개발의 기초 원칙과 정신으로 이야기 되고 있는 중요한 선언서입니다. 아래는 해당 선언서를 번역한 내용입니다.

애자일 선언문

우리는, 소프트웨어를 개발하면서, 그리고 또한 다른 사람의 개발을 도와주면서 소프트웨어를 개발하는 더 나은 방법들을 찾아나가고 있다. 이 작업을 통해 다음과 같은 가치를 추구하게 되었다.

프로세스나 도구 보다는 개인과 상호 작용을,
포괄적인 문서보다는 작동하는 소프트웨어를,
계약에 대한 협상보다는 고객과의 협력을,
계획을 고수하기 보다는 변화에 대응을

더욱 가치 있게 여긴다. 이 말은, 전자도 가치가 있긴 하지만, 우리는 후자 쪽에 더 많은 가치를 둔다는 것이다.

애자일 선언문은 첫 페이지에서 위와 같은 내용을 전면에 내세우고 있지만, 한편으로는 '애자일 소프트웨어의 12가지 원칙'이라는 이름으로 좀 더 구체적인 실천원칙에 대해 이야기를 하고 있습니다. 그 내용은 다음과 같습니다.

애자일 선언문의 바탕에 깔려있는 원칙들

우리는 다음과 같은 원칙들을 따른다.

- 우리의 최고 우선 순위는 가치 있는 소프트웨어를 일찍 그리고 지속적으로 전달함으로써 고객을 만족시키는 것이다.
- 비록 개발 후반부일지라도 요구사항 변경을 환영하라. 애자일 프로세스들은 변화를 활용해 고객의 경쟁력에 도움이 되게 한다.
- 작동하는 소프트웨어를 자주 전달하라. 약 2주에서 2개월의 정도의 간격으로 전달하되, 간격이 짧을수록 좋다.
- 비즈니스 영역 사람들과 개발자들은 프로젝트 전체에 걸쳐 매일 함께 일해야 한다.
- 동기가 갖추어져 있는 개인들로 프로젝트를 구성하라. 그들에게 그들이 필요로 하는 환경과 지원을 제공하라. 그리고 그들이 일을 끝낼 수 있도록 신뢰하라.
- 어떤 다른 개발팀을 상대로, 혹은 개발팀 내에서, (서로 간의) 정보를 전달하는 가장 효율적이고 효과적인 방법은 얼굴을 보고 하는 대화이다.
- 작동하는 소프트웨어가 진척 측정의 주된 척도이다.
- 애자일 프로세스들은 지속 가능한 개발을 장려한다. 스폰서, 개발자, 그리고 사용자들은 일정한 속도를 계속 유지할 수 있어야 한다.
- 기술적 탁월함과 좋은 설계에 대한 지속적 관심이 기민함을 향상시킨다.
- 간결함-하지 않아도 되는 일의 양을 최대화하는 기술-은 필수적이다.
- 최상의 아키텍처, 요구사항, 그리고 설계는 자기조직화(self-organizing)되어 있는 팀에서 나온다.
- 정기적으로, 팀 차원에서 어떻게 하면 더 효과적일 수 있을지에 대해 되돌아보며 자신들의 행동을 이에 따르도록 조율하고 조정한다.

'애자일 선언문'은 특정한 문제 상황을 가정하고 있지는 않지만, 본문에도 나와있듯이 '산출물'과 '문서' 위주의 개발 방식으로 개발이 진행되면서도 효율적으로 진행되고 있지 못하는 수 많은 프로젝트들에 대해 새로운 접근을 제안하고 있습니다.

애자일 선언문을 읽거나 해당 내용을 출력해서 눈에 잘 보이는 곳에 붙여 놓으면 어떤 문제가 해결될까요?

선언문을 열심히 읽고 공감하는 마음에 무릎을 치며 감탄을 한다 해도 사실 문제가 바로 해결되는 건 아닙니다. 이는 마치 우리가 TV프로그램에서 '등산'이 주는 건강의 효과에 대해 감탄하며 보더라도, 자신이 직접 '등산'을 하지 않는다면 전혀 소용이 없는 것과 마찬가지입니다. 선언문은 애자일 개발로 전환하는 팀의 시작 지점이며, 일종의 등대 역할을 하게 됩니다. 현재 어떠한 방법론을 사용해 개발을 하더라도 한 번씩 되새겨보며 방향을 다시 잡는데 사용하길 권합니다.

선언문 자체에서는 주의사항이라 할만한 부분은 존재 하지 않습니다. 하지만 때때로 애자일 선언문이 '추구해야 하는 가치'만을 선(善)으로 추구하고 '~보다'라는 비교 표현의 대상으로 쓰인 가치들, 이를테면 프로세스나 도구, 계약 등에 대해서는 마치 무시하라는 것처럼 자칫 오해하는 경향이 있습니다. 하지만 실제로는 그렇지 않습니다.

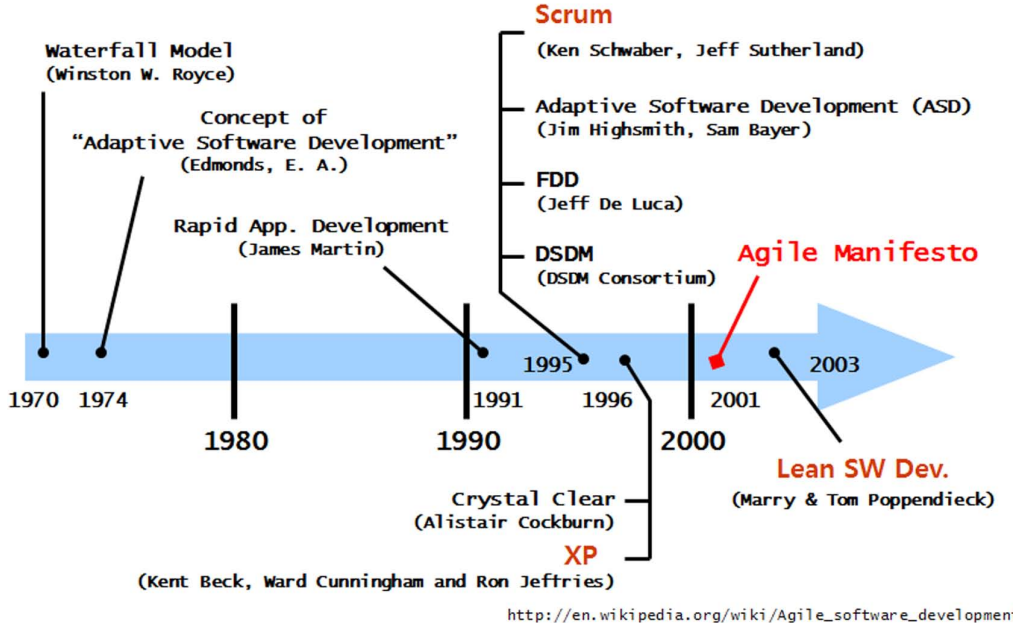
애자일을 도입하기 전에 한 페이지 짜리 선언문 뿐 아니라, '바탕에 깔린 원칙'이라고 이름 붙여진 보충 글도 꼭 읽어볼 필요가 있습니다. 애자일을 적용하는 방법에 대한 좀 더 구체적인 원칙들을 배울 수 있습니다. 또한, 애자일 선언문은 '법'이나 '규약'이 아닌 추구하고자 하는 가치임을 생각하고, 현실에 있어 최대한 따르도록 노력하되, 무비판적으로 따르는 것을 경계하도록 노력해야 합니다. 하지만 이로 인해 애자일을 적용함에 있어 적당히 '현실 타협'적인 접근을 하게 된다면 마찬가지로 오히려 장점은 퇴색시키고 단점을 부각시키게 만드는 경우가 발생하니, 주의해야 할 필요가 있습니다.

애자일 기원 및 종류소개

애자일 방법론의 기원

애자일 소프트웨어 개발에 대한 다양한 방법론들이 처음 등장하기 시작한 것은 1990년대 중반으로 기존의 무겁고 규범적인 방법론에서 탈피하여 가벼운 방법론을 지향하며 등장했습니다. 처음부터 이런 방법론들을 애자일(Agile)이라고 불렀던 것은 아닙니다. 초기에는 단순히 경량방법론(Lightweight methods)으로 통칭되었습니다만, 애자일 선언문(Agile Manifesto)을 만들면서 비로소 '애자일'이라고 이름 불리게 되었습니다. 이후부터 애자일은 소프트웨어 개발의 한가지 특징적인 개발방식으로 의미를 갖기 시작했습니다.

아래 타임라인에서 확인할 수 있듯이 애자일 소프트웨어 개발은 소프트웨어 공학이 처음 등장한 1945년에 비하면 10년 정도밖에 되지 않는 영역으로서 아직도 많은 보완과 발전이 필요한 방식입니다.



http://en.wikipedia.org/wiki/Agile_software_development

그림 1-2. 애자일 방법론의 등장 시기

애자일 방법론의 종류

애자일 선언문이 나온 이후로 다양한 애자일 방법론이 등장하게 됩니다. 하지만 그 기반에는 공통적으로 '신뢰성 높은 소프트웨어를 빨리 개발하자'는 가치를 공유하고 있습니다. 다음은 애자일 개발 방법의 이름과 그 방법론을 정의한 사람의 목록입니다.

- 스크럼(Scrum), 켄 슈와버/제프 서덜랜드
- 적응형 소프트웨어 개발 방법론(Adaptive Software Development, ASD), 짐 하이스미스
- 기능 주도 개발방법론(Feature Driven Development, FDD), 피터 코드/제프 드루카
- 동적 시스템 개발 방법론(Dynamic Systems Development Method, DSDM), 데인 포크너 외
- 크리스탈 패밀리(Crystal Family), 앨리스테어 코번
- 익스트림 프로그래밍(eXtreme Programing, XP), 켄트 벡/에릭 감마
- 린(Lean) 소프트웨어 개발방법론, 메리 포펜딕/톰 포펜딕
- 애자일 UP(Agile Unified Process, AUP), 스콧 앰블러

이 중에서 특히 세계적으로 널리 채택된 애자일 방법론은 스크럼과 익스트림 프로그래밍(XP)입니다.

초기에는 XP를 주로 사용했지만 스크럼이 점점 많은 인기를 끌면서 현재는 스크럼과 XP를 같이 사용하는 게 일반적인 추세입니다. 이 두 가지는 뒤에서 좀 더 자세히 살펴보겠습니다. 그 외, 동적 시스템 개발 방법론은 문서화가 잘 된 애자일 방법론으로서 주로 유럽 회사의 컨소시엄에서 시작한 방법론이며 린 소프트웨어 개발 방법론은 1980년대 도요타 시스템의 린 생산방식을 소프트웨어 개발에 적용하자는 운동으로 최근 스크럼과 함께 많이 사용되는 방식입니다.

애자일 방법론의 특징

사실 다양한 애자일 방법론의 특징을 한 가지로 정의하기는 어려운 일입니다. 하지만 각각의 방법론이 가지고 있는 공통된 부문만을 모아보면 '애자일 개발'이라 부르는 소프트웨어 개발방식이 가지는 특징을 찾을 수 있습니다.

애자일 소프트웨어 개발은 반복점진적(iterative and incremental) 개발을 기본 스타일로 가집니다. 그리고 이런 스타일의 개발방식을 효과적으로 진행하기 위해 자기조직화(self-organizing)나 교차기능팀(cross-functional teams)등의 기법들을 활용하고 있습니다.

이런 방법론이 제시하는 실천법은 애자일 선언문의 가치와 원칙을 실현하는 과정에서 나온 것입니다. 방법론이 지향하는 바를 제대로 이해하기 위해서는 애자일 선언문의 가치와 원칙을 먼저 숙지할 필요가 있습니다.

폭포수 방법론과 애자일 방법론간 주요 차이점

애자일 방법론과 전통 방법론인 폭포수 방법론 간의 주요 차이점을 통해서 애자일의 특징에 대해 조금 더 자세히 알아보겠습니다.

계획 중심 vs 고객 중심

폭포수 방법론은 프로젝트를 시작하기 전에 프로젝트 기간 전체에 대한 일정 계획을 수립하며, 이 계획에 따라 프로젝트를 진행합니다. 애자일 방법론은 계획을 수립하되 불확실한 프로젝트 기간 전체를 감안하여 무리하거나 현실성 없는 계획을 수립하는 것이 아니라 현재 시점에 고객에게 중요하거나 확정된 내용을 중심적으로 수립합니다. 프로젝트 상황에 따라 프로젝트 계획은 변경될 수 있다는 사실을 인정하고 진행합니다. 그래서 계획보다는 고객이 중요하게 생각하는 기능을 먼저 개발하는 것을 원칙으로 합니다.

빅뱅 릴리즈 vs 작은 릴리즈

형태적으로 보았을 때, 폭포수 방법론이 프로젝트가 종료되는 시점에 한꺼번에 모든 기능을 릴리즈한다고 한다면 애자일 방법론은 이터레이션이라는 일정 기간 단위로 작은 규모 크기의 릴리즈를 반복합니다. 이렇게 하면 고객은 요구사항이 제대로 반영되고 있는지 조기에 확인할 수 있어서 개발이 모두 끝난 다음에야 비로소 요구사항에 맞지 않다고 이야기 하고, 결국 재개발하게 되는 경우를 방지할 수 있습니다.

산출물 중심 vs 동작하는 SW 중심

폭포수 방법론에서는 계획된 단계별로 지정된 산출물이 작성되었는지 여부를 확인함으로써 프로젝트가 제대로 진행되고 있는지 확인합니다. 애자일 방법론에서는 소프트웨어가 제대로 동작하는지, 얼마나 요구사항에 맞게 개발되었는지가 중요합니다. 이러한 특징으로 인해 '애자일을 적용해서 개발을 진행하면 문서를 만들지 않아도 된다'는 오해를 받기도 합니다. 하지만 사실 문서를 만들지 않는 것이 아니라 상황에 맞는 다양한 형태로 산출물을 만들 수 있다는 좀더 유연한 의미로 받아들일 필요가 있습니다.

왜 애자일 방법론을 도입하려 하는가

많은 기업에서 애자일 방법론을 도입하려 하고 있고, 현재 하고 있습니다만, 사실 그 이유나 목적은 다양합니다. VersionOne이라는 애자일 개발 전문 컨설팅 회사에서 분기별로 진행하는 설문조사 자료를 기준으로 기업들의 애자일 도입 이유를 정리해보면 다음과 같습니다.

- 팀의 생산성을 높이고 제품을 적기에 출시하기 위해
- 개발에 들어가는 비용을 줄이기 위해
- 소프트웨어 품질을 향상시키기 위해
- 팀의 사기와 업무 만족도 향상을 위해

사실 애자일 방법론을 도입한다고 해서 위에서 언급한 내용을 반드시 달성할 수 있는 것은 아닙니다. 하지만 기존 방법론으로 해결하지 못하는 부분을 애자일을 통해 해결한 사례가 많습니다. 이 부분에 대해서는 계속해서 살펴보도록 하겠습니다.

스크럼이란

스크럼 개요

스크럼은 프로젝트 관리를 위한 애자일 방법론으로서 추정 및 조정 기반의 경험적 관리기법의 대표적인 형태입니다. 처음 시작은 1986년 타케우지 & 노나카 교수가 HBR에 기고한 “The New New Product Development Game” 이라는 기사를 그 기원으로 봅니다. 이후 1995년에 켄 슈와버와 제프 서덜랜드가 이 방법을 소프트웨어 개발에 소개하면서 스크럼이라 부르게 되었습니다.

스크럼 역할

스크럼에는 크게 3가지 역할자가 있습니다.

제품 책임자 Product Owner
: 제품 기능목록에 해당하는 제품 백로그(product backlog)를 만들고 우선순위를 조정하거나 새로운 항목을 추가하는 일을 관리합니다. 스프린트에 대한 계획을 수립할 때까지 중요한 역할을 하지만 스프린트가 시작되면 최대한 팀 운영에 관여하지 않는 걸 권장합니다.

스크럼 마스터 Scrum Master
: 스크럼의 원칙과 가치를 지키면서 스크럼 팀이 개발을 진행할 수 있도록 지원합니다. 스크럼 팀의 업무를 방해하는 요소를 제거하기 위해 노력합니다.

스크럼 팀 Scrum Team
: 보통 5~9명으로 구성되며 하나의 스프린트 기간 동안 구현해야 할 기능을 사용자스토리로 도출하고 이를 구현합니다. 스프린트 동안 구현해야 하는 기능을 완료하기 위해 노력하며 이를 위한 권한을 갖습니다.

스크럼 프로세스

스크럼의 프로세스를 구성하는 것은 스프린트, 3가지 유형의 미팅과 산출물입니다.

스프린트 Sprint
: 달력기준 1~4주 단위의 반복개발기간을 가리킵니다.
3가지 미팅
: 일일 스크럼, 스프린트 계획, 스프린트 리뷰
3가지 산출물
: 제품 백로그, 스프린트 백로그, 소멸 차트

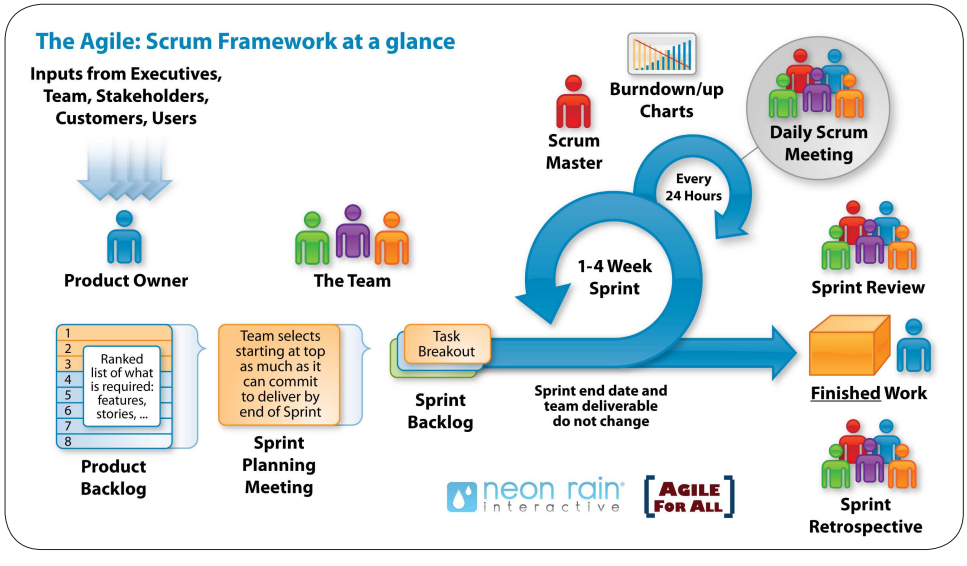


그림 1-3. 스프린트 진행 방식 요약

3가지 산출물을 좀 더 자세히 살펴보면 다음과 같습니다.

제품 백로그 Product Backlog
: 제품에 담고자 하는 기능의 우선순위를 정리한 목록, 고객을 대표하여 제품 책임자가 주로 우선순위를 결정합니다. 제품 백로그에 정의된 기능을 사용자 스토리라고 부르며 사용자 업무량에 대한 추정은 주로 스토리 포인트라 불리는 기준을 이용합니다.

스프린트 백로그 Sprint Backlog
: 하나의 스프린트 동안 개발할 목록으로 사용자 스토리와 이를 완료하기 위한 작업을 태스크로 정의합니다. 각각의 태스크의 크기는 시간 단위로 추정합니다.

소멸차트 Burndown chart
: 개발을 완료하기까지 남은 작업량을 보여주는 그래프, 각 이터레이션 별로 남아있는 작업량을 스토리 포인트라는 것으로 나타낸 것입니다.

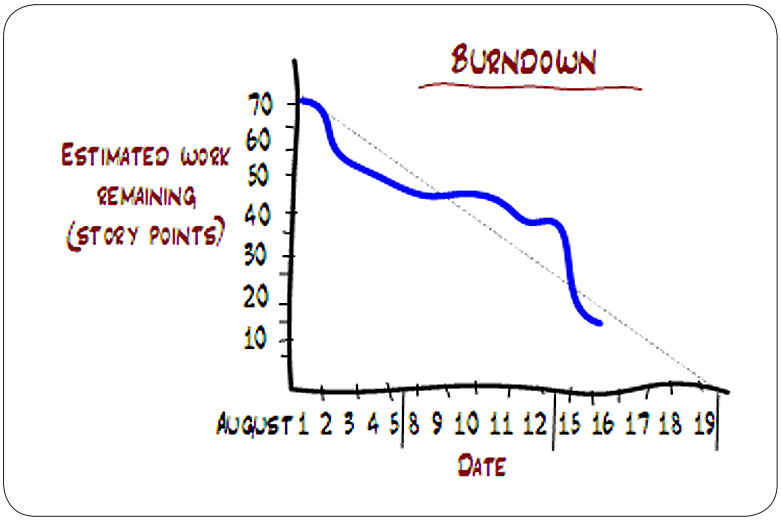


그림 1-4. 스프린트 소멸차트

3가지 미팅을 좀 더 자세히 살펴보면 다음과 같습니다.

스프린트 계획 Sprint Planning

: 각 스프린트에 대한 목표를 세우고 제품 백로그로부터 스프린트에서 진행할 항목을 선택합니다. 각 항목에 대한 담당자를 배정하고 태스크 단위로 계획을 수립합니다.

일일 스크럼 Daily Scrum

: 매일 진행하는 15분간의 프로젝트 진행상황을 공유하는 회의입니다. 모든 팀원이 참석하며 매일매일 각자가 한 일, 할 일, 문제점 등을 이야기합니다.

스프린트 리뷰 Sprint Review

: 스프린트 목표를 달성했는지 작업 진행과 결과물을 확인하는 회의입니다. 스크럼 팀은 스프린트 동안 작업한 결과를 참석자들에게 데모하고 피드백을 받습니다. 가능하면 해당 스프린트 동안 진행된 모든 작업에 대한 데모를 진행합니다. 고객이 참여하는 것이 좋습니다. 이 때, 스크럼 마스터는 스프린트 동안 잘된 점, 아쉬웠던 점, 개선할 사항 등을 찾기 위한 회고를 진행할 수 있습니다.

스크럼의 특징

투명성 Transparency

: 프로젝트가 현재 어떤 상태인지 계획대로 진행되고 있는지 어떤 문제점을 가지고 있는지 정확히 파악하는 건 어려운 일입니다. 스크럼은 스크럼 회의, 소멸차트, 스프린트 리뷰와 같은 기법을 이용해서 다른 방법론에 비해 프로젝트의 상태나 문제점을 잘 드러내줍니다.

타임박싱 Time boxing

: 스크럼을 구성하는 모든 실천법에 있어 시간은 매우 중요합니다. 일일 스크럼은 매일같이 15분이라는 짧은 시간에 진행해야 하며, 스프린트 리뷰는 매 이터레이션 마다 주기적으로 진행합니다. 스크럼 자체를 진행하는데 들어가는 시간을 엄격하게 제한함으로써 프로젝트 진행에만 집중할 수 있게 해줍니다.

커뮤니케이션 Communication

: 스크럼에서는 팀원 간 커뮤니케이션을 원활하게 하기 위해서 많은 노력을 기울입니다. 일일 스크럼에서 각 개발자들이 어떤 방해물(blocker)로 인한 문제를 겪고 있는지 공유하고, 흔히 플래닝 포커라는 방식으로 진행되곤 하는 각 사용자 스토리의 구현 난이도/시간을 토론하는 절차는 프로젝트 내 커뮤니케이션을 원활하게 해주는 스크럼의 특징 중 하나입니다.

경험주의 모델 "Inspect & Adapt" model

: 스크럼은 고유의 프로세스 모델을 가지고 있지만 많은 기법들이 프로젝트에 참여하고 있는 개개인의 경험에 기반합니다. 프로젝트마다 고유한 상황과 특징을 가지고 있기 때문에 기존의 정형화된 프로세스로는 이런 최근의 프로젝트 상황을 따라가기 어렵습니다. 그러므로 스크럼은 기본적인 구조만 같을 뿐 실제로 일을 진행하게 되면 팀마다 달라지는 것을 허용합니다.

XP 란

XP는 eXtreme Programming의 약자로 애자일 SW 개발 방식의 하나입니다. 1990년대 후반 켄트 벡(Kent Beck)을 중심으로 여러 엔지니어들이 프로젝트를 진행하며 얻었던 교훈을 기반으로 효과적이라 생각되는 개발 기법을 모아 하나의 방법론으로 정립하게 되었습니다. 보통 개발조직이 기반이 되는 중소규모 팀에 적합한 경량화된 개발방식입니다.

경우에 따라서는 다른 애자일 개발 방식과 마찬가지로 XP도 '방법론'이라 불리는 데에 있어 여러 이견이 있을 수 있으며, 특히나 XP의 경우 테스트 주도 개발(TDD), 일일빌드(Daily Build), 지속적인 통합(Continuous Integration) 등 개발 테크닉과 연관된 부분이 많기 때문에 종종 '방법론'으로써 규정짓는 것에 대해 논란이 되곤 합니다. 팀의 개발 문화가 제대로 정립되어 있지 않거나, 계획 및 관리 중심으로 프로젝트를 유지하던 팀의 경우는 XP도입 초기에 자잘한 난관이 존재하기도 합니다.

국내의 경우 작은 개발 팀 중심으로 기법의 일부를 차용해 적용하는 경우가 많았으며, 최근에는 XP만을 사용하는 경우보다는 스크럼 등의 보완적인 애자일 개발 방식과 함께 사용하는 경우가 많습니다.

XP는 기본적으로 가치(value)와 그 가치를 이루기 위한 실천법(practice), 이 두 개를 큰 축으로 놓고 원칙(principle)을 세워서 그 둘 사이에서 균형을 맞추는 것을 권장합니다.

XP의 가치

1 의사소통

보통 어떤 팀이 발전적인 방향으로 존속하는 데에 있어서 가장 중요한 점으로 '의사소통'을 꼽을 수 있습니다. 최근, 이 부분은 SW개발팀에만 국한되지 않고 팀 활동을 하는 대부분의 조직에 있어 핵심적인 요소 중 하나로 당연하게 받아들여지고 있는 부분입니다.

실제로 팀 개발을 하게 되면 상당수의 문제가 팀 내의 의사소통과 관련하여 발생하게 됩니다. 팀 원 각자가 겪는 문제에 대해 서로 인지하고, 문제가 반복하지 않도록 도우며, 또한 문제가 감추어진 상태로 진행되다가 결국 걷잡을 수 없는 규모로 악화되지 않도록 하려면 팀 내의 의사소통은 매우 중요한 요소입니다.

하지만 의사소통을 잘 해야 한다는 점을 안다고 해서 쉽게 의사소통이 가능해 지는 건 아닙니다. 이를 위해서는 우리 개개인에게 작은 무언가가 준비되어 있어야만 가능합니다. 그건 바로 의사소통 시 발생하는 두려움과 마주할 때 필요한 개개인의 '용기'입니다.

2 용기

SW개발을 하는 것은 곧잘 커다란 미지의 두려움과 마주해야 하는 일상의 연속이 되곤 합니다. 낯선 기술과 모르는 사람들, 마찬가지로 알 수 없는(고객을 포함해) 타인의 감정과 대면했을 때 많은 경우 우리는 용기를 버리고 '속임', '회피', '분노', 혹은 '비난' 등을 선택하게 됩니다. 이를 극복하고 나아가기 위해서는 '용기'가 필요합니다. 모르는 걸 모른다고 말할 수 있는 용기, 먼저 손을 내어 도움을 주고 도움을 요청할 수 있는 '용기'가 필요합니다. 어쩌면 이건 말처럼 쉬운 일이 아닐 수 있지만, '용기'를 통해 우리는 성공적인 SW개발의 필수 요소인 '피드백'이라는 중요한 가치를 얻을 수 있게 됩니다.

3 피드백

피드백은 SW개발 및 의사소통의 핵심입니다. 좋은 SW의 특징 중 하나는 개발 도중에도 지속적인 피드백을 받으며 만들어 진다는 점입니다. 그리고 좋은 팀은 의사소통과 피드백이 멈추지 않고 계속 순환되는 분위기를 유지합니다. 이는 추정도 아니고 이론도 아닙니다. 바로 현상이며 사실입니다.

SW개발이 시작되면 가급적 빠른 시일 내에, 그리고 자주, 고객으로부터 피드백을 받는 것이 좋습니다. 때로는 더 자주 더 많이 새로운 요구사항이 만들어 진다는 점 때문에 피드백의 간격을 늘이거나 횟수를 줄이려고 하는 경우가 있습니다만, 이는 결과적으로 개발팀과 고객 모두에게 만족스럽지 못한 결과를 가져옵니다.

SW 개발에 대해 피드백을 자주 받고 새로운 요구사항을 제대로 어렵지 않게 잘 반영하기 위해서는 유지해야 하는 중요한 가치가 있습니다. 그건 '단순함'이라는 가치입니다.

4 단순함

SW 개발은 점점 더 어려운 문제를 해결하기 위해 점점 더 복잡해져 가고 있습니다. 우리가 현재 SW 개발에 있어 직면하고 있는 큰 아이러니 중 하나는 복잡한 업무를 해결하기 위해 그 만큼이나 복잡한 방식으로 SW를 만들고 있다는 점입니다. SW가 비용을 줄여주고 생활을 편리하게 만들어 주는 것은 사실이지만, 복잡한 SW는 고장 나기 쉽고, 문제를 일으키기 쉽고, 결국 버려지기 쉽게 변해 버립니다.

의사소통을 극대화하고 용기를 내서 적극적으로 피드백을 받는다 해도 SW가 간결하게 유지 되지 않는다면 어려움이 해결되지 않을 겁니다.

5 존중

위 모든 가치를 추구하는 과정에서 반드시 근간에 유지되어야 하는 것은 바로 '존중'입니다. 개개인에 대해 상호간의 '존중'이 없으면 팀은 결국 와해되어 버리거나 수준 높은 SW를 다시 함께 개발하는 것이 불가능하게 됩니다. 이에 대한 켄트 벡은 다음과 같이 이야기 하였습니다.

"소프트웨어 개발에서 생산성과 인간성을 동시에 개선하려면, 팀에 속한 모든 개인의 기여를 존중해야 한다. 나도 중요한 사람이고 당신도 중요한 사람이다."

XP의 기본 실천방법들

1. 함께 앉기
개발 작업은 팀 전체가 들어가기에 충분할 정도로 크고 열린 공간에서 하라.

2. 전체 팀
프로젝트가 성공하기 위해 필요한 기술과 시야를 지닌 사람들을 전부 팀에 포함시켜라.

3. 정보를 제공하는 작업 공간
작업 공간을 작업에 대한 것들로 채워라. 프로젝트에 관심이 있는 관찰자라면 누구든지 팀이 사용하는 공간에 들어와서 15초 안에 프로젝트가 어떻게 진행되는지 대략 감을 잡을 수 있어야 한다.

4. 활기찬 작업
생산적으로 일할 수 있는 정도의 시간만, 그리고 일의 활력을 유지할 수 있는 정도의 시간만 일해라.

5. 짝 프로그래밍
제품으로 출시할 프로그램을 두 사람이 컴퓨터 한 대에 앉아 작성해라.

6. 스토리
고객에게 가치를 줄 수 있는 최소한의 기능을 단위로 해서 계획하라.

7. 일주일별 주기
한 번에 일주일 분량의 일을 계획하라.

8. 분기별 주기
한 번에 한 분기 분량의 일을 계획하라.

9. 여유
어떤 계획이든, 일정에 뒤쳐질 경우 포기할 수 있는 비교적 급하지 않은 업무를 포함시켜라.

10. 10분 빌드
10분 만에 자동으로 전체 시스템을 빌드하고 모든 테스트를 돌려라.

11. 지속적인 통합
변경한 것은 두 세 시간만에 통합하고 테스트해라.

12. 테스트 우선 프로그래밍
코드를 한 줄이라도 변경하기 전에, 자동화된 테스트를 먼저 작성하라.

13. 점진적인 설계
시스템의 설계에 매일 투자하라.

XP의 원칙

1. 인간성
소프트웨어는 인간이 개발한다.

2. 경제성
이 모든 것에 누군가는 돈을 지불한다는 경제성의 원칙을 인정한다.

3. 상호이익
모든 활동은 그 활동에 관련된 모든 사람에게 이익이 되어야 한다.

4. 자기 유사성
어떤 해결책의 구조를 다른 맥락에도 적용해 보라

5. 개선
소프트웨어 개발에서 '완벽하다'란 없다. '완벽해 지기 위해 노력한다'만 있다.

6. 다양성
팀에는 비록 갈등의 요소가 될 수 있을지라도 다양성이 필요하다.

7. 반성
좋은 팀은 실수를 숨기지 않고 오히려 실수를 드러내어 거기에서 배운다.

8. 흐름
개발의 모든 단계를 동시에 진행함으로써 가치 있는 SW를 물 흐르듯 끊임없이 제공해야 한다.

9. 기회
가끔씩은 생각을 전환해서 문제를 기회로 보는 방법을 배우자.

10. 잉여
SW 개발에서 핵심적이고 해결하기 어려운 문제는 해결방법을 여러 개 만들어 놓아야 한다.

11. 실패
성공하는데 어려움을 겪는다면 실패하라.

12. 품질
품질을 희생하는 것은 프로젝트 관리의 수단으로 삼기에 효과적이지 않다.

13. 책임 소재
어떤 일을 하겠다고 선언한 사람이 그 일의 책임도 가진다.

14. 아기걸음
단계를 작게 쪼갤 때 걸리는 부하가, 큰 변화를 시도했다가 실패해서 돌아갈 때 드는 낭비보다 훨씬 작다는 사실을 인정하자.

린Lean 소프트웨어 개발이란

린의 기원

도요타 자동차의 독특한 생산방식의 원칙과 실천법을 정리하는 데서 린(Lean)이라는 개념은 시작되었다고 합니다. 우리나라 제조/생산 업계에서는 이미 도요타 생산 방식(TPS)를 오래 전부터 벤치마킹 해왔습니다. 린에서 중요한 개념은 JIT(Just In Time) 입니다. JIT은 필요한 시점에 필요한 만큼만 생산하는 것을 의미하며 이를 통해 재고를 최소화하고 나아가 비용을 최소로 줄여나갑니다. 또 하나의 중요한 용어가 칸반(Kanban) 입니다. 칸반은 일종의 작업지시서로서 Pull 방식의 생산시스템을 구축하는데 중요한 역할을 합니다.

린 소프트웨어 개발의 시작

생산방식으로서의 린을 SW 개발에 적용하려는 시도는 여러 번 있었으나 이를 체계적으로 정리한 것은 포펜딕(톰 포펜딕, 메리 포펜딕) 부부입니다. 린의 중요한 목표인 재고 절감을 SW 개발에 대입하여 불필요한 낭비를 제거하자는 운동으로 발전한 것이 린 소프트웨어 개발입니다.

린 소프트웨어 개발의 원칙

낭비를 제거하라.
: 파레토 법칙(2:8의 법칙)에 의거하여 개발에 정말 중요한 20%에 집중하고 낭비되는 요소(가외기능, 혼란, 경계 넘어가기)를 제거합니다.

- **가외기능**: 80%의 가치를 제공하는 20%의 기능에 초점을 맞춰라. 20%의 가치를 제공하는 80%인 가외기능은 나중으로 미룬다.
- **혼란**: 요구사항 혼란을 겪는다면 스펙을 너무 일찍 결정한 것이다. 테스트하고 수정하는데 혼란을 겪는다면 테스트를 너무 늦게 하는 것이다.
- **경계 넘어가기**: 조직간의 경계는 통상 25% 이상의 비용을 증가시키는데, 버퍼를 만들고 응답시간을 늦추며 커뮤니케이션을 방해하기 때문이다.

품질을 내재화 하라.
: 개발 초기부터 각각의 SW 모듈에 대한 품질을 강조하면 마지막에 빅뱅 식 통합이나 테스트 때문에 발생하는 문제를 해결할 수 있습니다.

- **테스트 주도 개발(TDD)을 통해 코드의 실수를 방지하라**: 요구사항 문서 대신 실행 가능한 명세를 작성하라.
- **레거시 코드를 만들지 마라**: 레거시 코드는 자동화된 단위테스트와 인수테스트가 없는 코드다.
- **빅뱅 통합은 진부하다**: 지속적인 통합과 중첩된 동기화 기법을 사용하라.

지식을 창출하라.
: 표준은 개선하기 위해 존재하는 것이며 과학적 방법을 통해 누구나 변경하고 장려할 수 있도록 해야 한다고 이야기 합니다.

- **과학적 방법을 사용하라**: 가설을 세우고, 신속하게 다양한 실험을 해보고, 문서를 간결하게 작성하고, 최선의 대안을 구현할 수 있도록 답을 가르쳐라.
- **표준은 도전 받고 개선되기 위해 존재한다**: 모든 사람들이 따라 하고 잘 알려진 실천법을 표준에 포함하되, 누구든지 표준에 도전하고 변경하도록 장려하라.
- **예측 가능한 성과는 피드백에 기반한다**: 예측 가능한 조직은 미래를 추측하고 그것을 계획이라고 하지 않으며, 그 보다는 미래가 펼쳐질 때 신속하게 대응하기 위한 역량을 개발한다.

확정을 늦춰라.
: 마지막까지 변화를 수용할 수 있도록 코드를 작성합니다. 돌이킬 수 없는 결정은 마지막에 내리라고 충고합니다.

- **의존성을 깨뜨려라**: 시스템 아키텍처는 언제 어떤 기능이 추가되더라도 그것을 수용할 수 있어야 한다.
- **옵션을 유지하라**: 코드를 실험으로 생각하라. 변화를 수용할 수 있게 작성하라.
- **돌이킬 수 없는 결정은 마지막 결정의 순간에 하라**: 돌이킬 수 없는 결정을 내리기 전에 가능한 많이 학습하라.

전체를 최적화하라.
: 고객 요구에서 SW 배포까지 전체적인 흐름에 초점을 맞춰야 합니다.

- **전체 가치 흐름에 초점을 맞춰라**: 컨셉에서 현금까지, 고객 요구에서 소프트웨어 배포까지
- **완전한 제품을 인도하라**: 소프트웨어만이 아닌 완전한 제품을 개발하라. 완전한 제품은 완전한 팀에 의해 만들어진다.
- **더 높은 것을 측정하라**: 국지적인 성과측정은 부분 최적화를 야기하며 낭비를 발생시킨다. 전체 가치 사슬의 시작과 끝의 소요시간, 일명 사이클 타임(Cycle Time)을 측정하라. 또는 인도된 비즈니스 가치로 팀 성능을 측정하라.

사람을 존중하라.
: 팀은 공동 목표를 달성하기 위해 자부심, 신뢰, 칭찬을 통해 맺어진 상호간의 책임의식을 가져야 한다.

- **팀은 자부심, 책임감, 신뢰, 칭찬을 통해 번성한다**: 무엇이 팀을 만드는가? 팀원들은 공동 목표를 달성하기 위해 상호간 책임의식으로 뭉쳐있다.
- **효과적인 리더십을 제공하라**: 효과적인 팀에는 팀을 최고로 이끄는 훌륭한 리더가 존재한다.
- **파트너를 존중하라**: 파트너/외주업체와 상생적인 관계를 유지해야 한다. 그들은 우리의 종이 아니라 우리 SW를 함께 만드는 동료다.

빨리 인도하라.
: 한번에 전달되는 제품의 크기를 작게 하고 작업량을 줄여야 한다.

- 신속한 인도, 고품질, 저비용은 공존할 수 있다.
- 대기행렬 이론을 개발에 적용하라.
- **일의 양을 할 수 있는 만큼으로 제한하라**: 반복개발에 안정적이고 반복 가능한 속도를 가져라. 고객에게 인도할 수 있는 역량에 맞게 대기열의 길이를 정기적으로 제한하라.

린 소프트웨어 개발과 칸반

칸반은 생산시스템에서 일하는 작업자들이 어떤 작업을 해야 하는지 알려주는 작업지시서에 해당한다. 린 소프트웨어 개발에서 이 칸반을 활용하게 되면 다음과 같은 장점을 얻을 수가 있다.

워크플로를 가시화한다.

- 일을 작게 나누어서 카드에 하나씩 써서 벽에 붙인다.
- 각 항목이 현재 어느 단계에 있는지 알 수 있도록 워크플로를 나타내는 열에 제목을 붙인다.

작업중인 것을 제한한다.(WIP)

- 워크플로 상에 얼마나 많은 항목이 진행되고 있는지 제한을 둔다.

작업에 소요되는 시간을 측정한다.

- 한 항목을 완료하는데 걸리는 평균시간, 사이클 타임으로 부르기도 한다.
- 예측가능하고 소요시간을 최소화하기 위해서 프로세스를 최적화 한다.

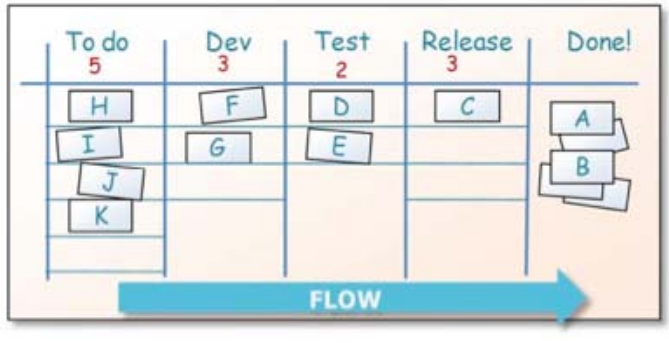
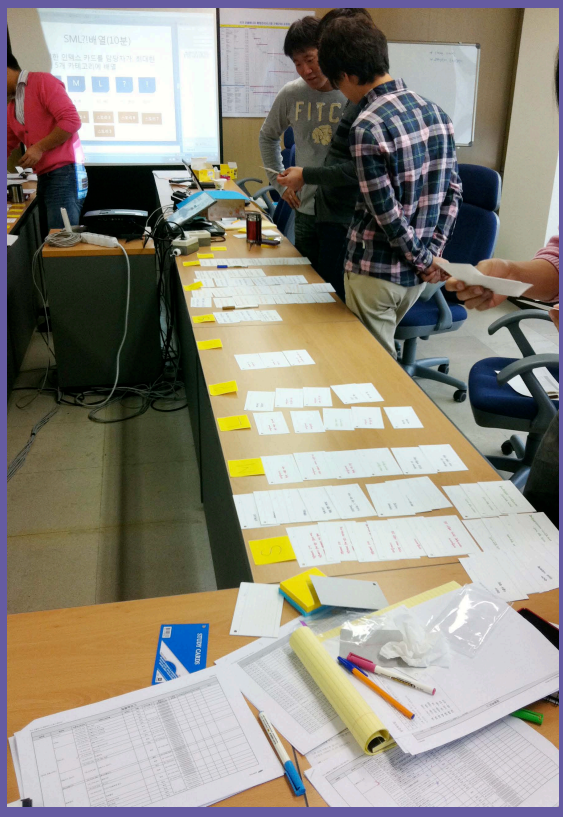


그림 1-5. 칸반 워크플로 보드 예시

02



PRACTICE I

현장고객 On-site Customer

사용자 스토리 User Story

릴리즈 계획 Release Planning

플래닝 포커 Planning Poker

추정 Estimation

스프린트 Sprint

스프린트 계획 Sprint Planning

일일 스크럼 Daily Scrum

짝 프로그래밍 Pair Programming

CI Continuous Integration

속도 Velocity

스프린트 리뷰 Sprint Review

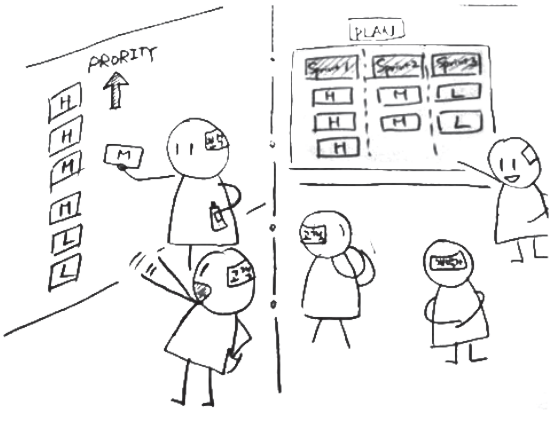
회고 Retrospective

Practice Map

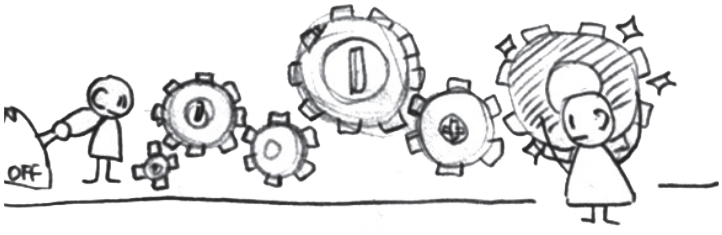
8 Daily Meeting

2 User Story

3 Release Planning



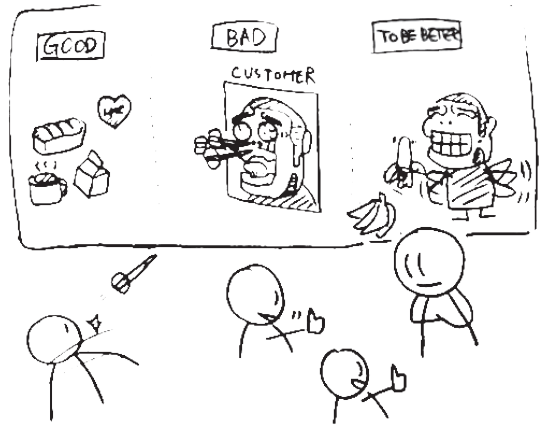
10 Continuous Integration



9 짝 프로그래밍 Pair Programming
10 지속적인 통합 CI
간결한 설계 Simple Design
리팩토링 Refactoring
테스트주도개발 TDD
11 속도 Velocity
작은릴리즈 Small Releae
작업현황판 Task Board
12 소멸차트 Burn-down chart
스프린트리뷰 Sprint Review
13 회고 Retrospective

12 Sprint Review

13 Retrospective



현장 고객
On-site Customer

ⓐ 요약

- 고객이 개발팀과 같은 장소에서 함께 개발에 참여하는 것입니다. 개발팀의 결과물에 대해 신속하게 피드백을 해줌으로써 개발팀이 엉뚱한 작업을 하지 않도록 도와줍니다.
- 스크럼의 제품 책임자와 역할이 유사해 보이지만, 현장고객은 고객들 사이에서 개발팀을 대표하는 사람이고 개발팀 내부에서는 여러 고객의 입장을 대표한다는 점에서 차이가 있습니다. 즉, 제품 책임자는 제품을 책임지는 개발 팀의 총괄 책임자이자 다양한 고객으로부터 개발팀으로의 단일 창구 역할을 수행합니다. 단, 고객이 모든 권한을 제품 책임자에게 일임했다면 제품 책임자를 고객으로 봐도 무방합니다.

ⓑ 역할자

- 최종 사용자: 구현된 제품을 실제로 사용하는 사람입니다.
- 고객: 제품을 구입하는 사람으로 최종 사용자와는 다를 수 있습니다.
- 내부 고객: 개발팀에 방향을 제시하거나 결과물을 전달받는 관련 부서가 내부 고객이 될 수 있습니다. 한 명 이상의 내부 고객이 존재할 수 있습니다.
- 개발팀: 제품을 개발하는 팀입니다.

ⓒ 적용상황

- 고객의 요구가 모호하여 업무를 추진하기가 어려울 때
- 고객이 원하는 기능의 수준을 파악할 수 없을 때
- 개발을 완료한 기능이 고객이 원하던 것이 아닐 때
- 고객과의 커뮤니케이션이 드물고 충분하지 않을 때
- 고객의 입장에서 개발팀의 진척상황이 궁금하고 불안할 때

ⓓ 준비물

- 고객이 상주할 수 있는 공간/책상

ⓔ 더 잘 하려면..?

- 고객은 개발 결과물인 제품에 대해 애정/책임을 가져야 합니다. (제품이 제대로 만들어지든 말든 관심이 없어서는 안됩니다)
- 고객은 제품을 개발하는데 필요한 분야의 지식과 의사결정권을 가지고 있어야 합니다.
- 고객은 전체 개발 일정/비용을 감안하여, 기능을 추가하거나 삭제할 수 있고 기능목록의 우선순위와 중요도를 변경할 수 있습니다.
- 개발팀은 고객과 대화하는 것을 긍정적으로 생각하여야 합니다.

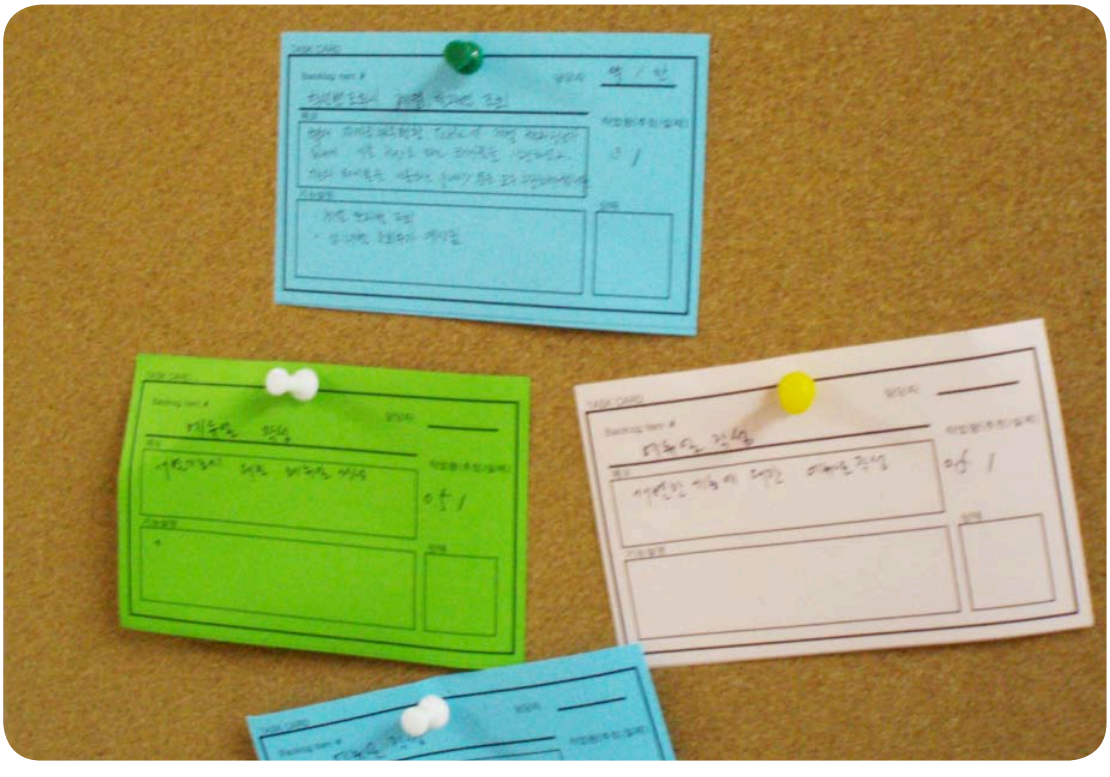
ⓕ 어떤 것이 해결되는가?

- 고객이 가장 중요하다고 생각하는 기능부터 구현할 수 있습니다.
- 애매모호한 상황을 개발팀이 자의적으로 해석/결정할 필요가 없습니다.
- 제품을 검증할 때 고객이 참여함으로써 완성도 높은 결과물을 만들 수 있습니다.
- 문서만으로 전달할 수 없는 행간의 세부적이고 감성적인 부분까지도 (고객과의 대화를 통해) 파악할 수 있습니다.

ⓖ 주의사항

- 참여한 고객이 사용하는 용어를 팀이 이해해야만 합니다. 필요하다면 용어집을 만듭니다.
- 고객과 이야기 할 때에는 고객의 언어로 이야기해야 합니다. 기술적인 용어의 사용은 지양하되 필요하다면 고객에게 충분히 설명/이해시켜야 합니다.
- 가끔 고객은 중간결과물을 보면서 요구사항을 계속 바꾸거나 확장하게 되는데, 일정과 비용을 감안하여 절충할 수 있어야 합니다.
- 고객은 개발팀의 의견을 존중해야 합니다.

사용자 스토리
User Story



ⓐ 요약

- 사용자 스토리는 개발해야 할 대상 제품이나 서비스의 기능을 정의하는 방식으로 사용자 입장에서의 비즈니스적인 가치를 정의하는데 초점을 두고 요구사항을 정리하는 실천법 입니다.

ⓑ 역할자

- 고객팀: 사용자 스토리의 초안을 제시합니다.
- 작성자: 가급적 업무를 잘 아는 사람(비즈니스 전문가, 분석자)이나 제품을 기획하는 사람(기획자)이 고객과 함께 있는 자리에서 질문을 하며 적습니다.

ⓒ 적용상황

- 고객이 쉽게 이해할 수 있도록 요구사항을 작성하고 싶을 때
- 사용자의 요구사항이 명확히 드러나 있지 않아 고객과 대화하며 상세화 하기 위해

ⓓ 더 잘 하려면..?

- 고객팀 혹은 고객팀 담당자가 사용자 스토리 작성에 참여할 의지가 있어야 하고 개발팀에서도 비즈니스 담당자/분석자/기획자가 존재해야 합니다.
- 만약 고객이 사용자 스토리 작성에 참여 의지가 없다면, 대체할 고객을 찾고 이에 따른 위험을 고객에게 설명해야 합니다.
- 시작할 때부터 가능하면 많은 고객이 프로젝트에 참여할 수 있는 환경을 만드는 것이 중요합니다.

ⓔ 어떤 것이 해결되는가?

- 팀과 고객간의 의사소통이 좋아지며, 이는 상호간에 '신뢰'를 쌓는데 가장 중요한 요소가 됩니다.
- 개발 대상에 대한 이해도가 높아집니다.
- 추정과 함께 사용하면 개발 계획 수립에 큰 도움이 됩니다.
- 명확하지 않은 것이 무엇인지 드러나게 해주며, 우선순위 추정에 도움을 줄 수도 있습니다.

ⓕ 주의사항

- 사용자 스토리만으로 개발을 진행하는 것은 규모가 큰 프로젝트에서는 적합하지 않을 수도 있습니다. 요구사항이 복잡해질수록 고객과 더 많이 대화해야 하며 고민해야 할 양도 그만큼 많아지기 때문입니다. 이 경우 사용자 스토리의 상세 내용으로 테스트 케이스에 준하는 완료조건, 테스트 데이터, 스토리 상세 설명, 화면 등을 추가할 수도 있습니다.
- 각각의 스토리는 하나의 기능만을 표현하도록 해야 합니다.
- 작성자뿐만 아니라 많은 사람이 혼동되지 않도록 가급적 단순한 문장으로 작성합니다.
- 테스트나 검증에 대한 내용은 따로 표시해 놓습니다. 이를테면 스토리 카드 뒷면에 적거나 항목을 따로 잡아서 표현합니다.
- 예 / [앞면] 사용자는 계좌이체를 통해 아이템을 구매할 수 있다.
  [뒷면] 실시간 계좌이체만 지원한다.
- 한 개의 사용자 스토리가 모호한 경우 두 개 이상의 상세한 내용으로 작성할 수 있습니다. 이를테면 '고객정보를 관리할 수 있다'라는 사용자 스토리 한 개 대신에 '고객프로필을 관리할 수 있다', '고객의 보유 카드 내역을 관리할 수 있다' 와 같이 보다 명확하게 나눌 수도 있습니다

ⓖ 순서

1 우선 해당 제품을 사용하게 될 고객의 분류를 판단해 정의합니다.
예) 상담원, 관리자, 지원부서기술자 등.

▼

2 요구사항을 정제하여 짧은 문장으로 기능을 표현합니다. 이때, "누가", "무엇을", "어떤 이유로" 원하는 가에 대해 정확히 기술해야 합니다.
예) As a Role : "고객분석담당자는"
So that : "현재 가입된 고객의 남여 성비와 사는 지역의 정보를 파악하기 위해"
I want : "전체 고객 명단의 목록을 성별과 지역으로 정렬해서 볼 수 있다"

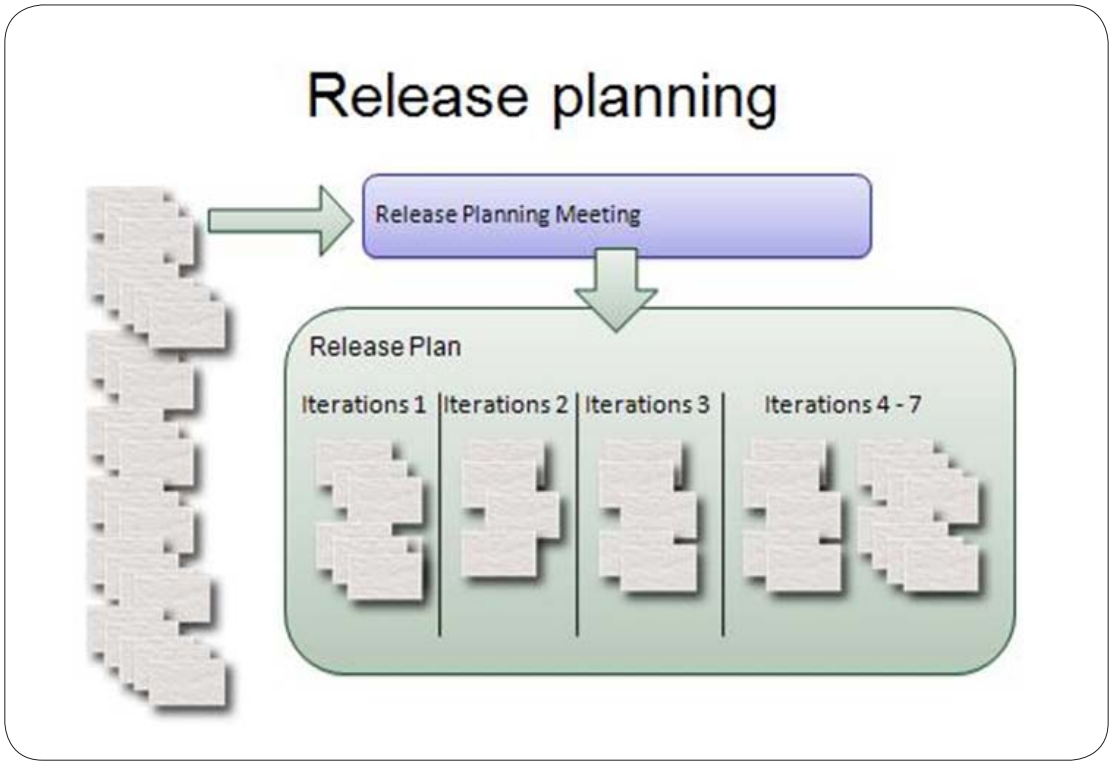
▼

3 일반적으로 '스토리 카드'라는 이름의 인덱스 카드나 종이를 사용해서 하나의 기능을 한 장의 카드에 적을 것을 권장합니다. 스토리 카드에 적어야 할 내용이 많은 경우, 애자일 관리 툴을 쓰는 것이 효과적일 수도 있습니다.

▼

4 이 사용자 스토리에 우선 순위를 정하게 되면 그에 따라 개발 분량과 일정을 정합니다.

릴리즈 계획
Release Planning



ⓐ 요약

- 릴리즈 계획은 고객과 함께 도출한 사용자 스토리를 기반으로 팀 전체가 업무량을 산정하고 팀 역량에 맞추어 스프린트 별로 사용자 스토리를 할당하며 프로젝트 전체적인 개발계획을 수립하는 실천법입니다.

ⓑ 역할자

- 진행자: 릴리즈 계획 진행 시 퍼실리테이터(회의나 교육 등의 진행을 원활하게 돕는 역할자) 역할을 수행합니다.
- 고객: 사용자 스토리에 대한 궁극적 책임을 지며, 사용자 스토리 내용에 대해 참여자에게 설명하고 여러 사용자 스토리의 우선순위를 결정합니다.
- 팀원: 모든 팀원이 참석하는 것이 가장 바람직 합니다. 팀 규모가 크고, 고객 숫자가 한정되어 있는 경우 적절한 규모의 분할된 팀 간의 릴리즈 계획이 어려울 수도 있습니다. 이런 경우 분석가 역할을 하는 인력이나 업무 도메인을 잘 이해하는 인력이 대표로 참여하여 개발 계획을 수립할 수도 있습니다.

ⓒ 적용상황

- 구현할 전체 업무 규모 산정이 어려울 때
- 조직이 해야 하는 할 일의 목록과 범위가 명확하지 않을 때
- 계획 수립 시 각자의 의견이 대립될 때
- 이해관계가 얽혀있어 각자 계획을 수립하기 어려운 경우 함께 이야기 하며 계획 수립 시 각자의 의견이 골고루 반영되도록 하고 싶을 때

ⓓ 준비물

- 인덱스 카드, 네임펜, 제품 백로그, 플래닝 포커

ⓔ 더 잘 하려면..?

- 상위수준의 비전과 비즈니스 목표를 사전에 공유합니다.
- 제품 책임자와 진행자는 도출된 사용자 스토리 목록을 준비하고 이를 설명할 준비를 합니다.
- 계획수립에 영향을 미칠 수 있는 개발준비 상황과 아키텍처에 대한 정보를 공유합니다.

ⓕ 어떤 것이 해결되는가?

- 해야하는 일의 규모가 파악되며 전체적인 개발 일정이 수립되고 각 릴리즈 별로 어떤 것을 구현해야 하는지 파악할 수 있습니다.
- 계획 수립 과정을 통해 이해 관계가 보다 명확하게 조정됩니다.
- 이 정보를 이용하여, 팀이 할 수 있는 일의 규모와 역량을 측정할 수 있고, 이 후 상황에 따라 다른 계획을 세우는 데 팀의 역량을 참고하여 반영할 수 있습니다.

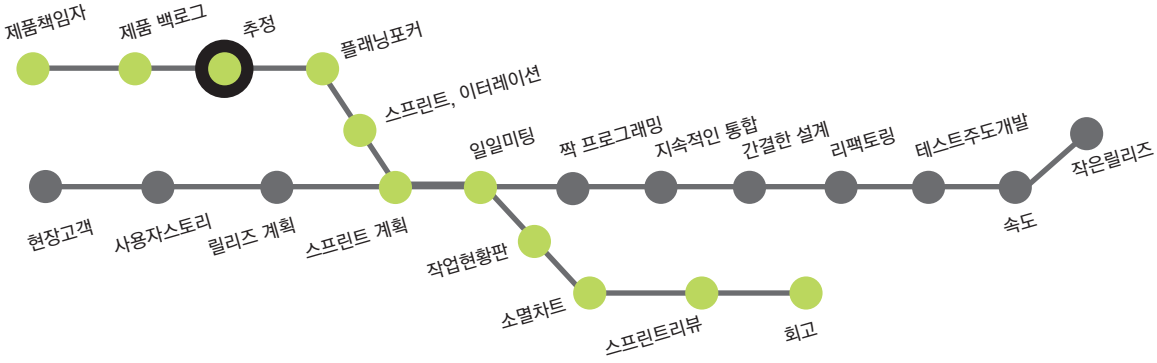
ⓖ 주의사항

- 적정 인원(5~6인)이 넘어가면 플래닝포커 게임을 하기에 시간이 많이 들고 비효율적이 될 수 있습니다. 이 때는 프로젝터를 활용하여 벽에 기 작성한 스토리 목록을 띄워놓고 함께 플래닝포커 게임을 할 수도 있습니다.
- 한번에 계획이 끝나는 것이 아닙니다. 스프린트 수행 후 실제 한 일을 측정하고 계획을 수정해 나가는 과정을 지속적으로 반복하여 수행할 수 있도록 합니다.
- 스토리 포인트로 표현된 추정치는 그 기능을 개발하는데 드는 노력의 양과 개발복잡도 그리고 거기에 내재된 위험성 등을 한데 모아 표현하는 값으로 산정(Calculation)이 아닌 추정(Estimation)을 통해 도출됩니다. 따라서 처음부터 너무 정확하게 추정을 하려는 것은 효과적이지 않습니다.

ⓗ 순서

1. 제품과 릴리즈 전체에 대한 비전과 일정 등을 공유합니다.
2. 사용자 스토리를 완료하는데 소요되는 공수를 스토리 포인트라는 단위로 나타냅니다. 이를 이용하여 규모를 추정(Estimation)합니다.
3. 스토리 포인트 추정은 플래닝포커 게임을 이용하여 수행합니다.
4. 현실에서 최초 정한 납기일까지 고객이 원하는 모든 기능을 완성하지 못할 수도 있습니다. 고객도 자신이 원하는 것이 무엇인지 잘 모를 수도 있고, 지금 현재의 계획이 현실성이 없을 수도 있습니다. 이 경우 가장 고객에게 가치가 높은 것부터 전달하는 것이 중요합니다. 이를 위해 우선 순위를 정합니다. 이 때 분석자는 비즈니스 측면에서 고객이 우선 순위를 만들도록 돕고, 개발자는 기술적 측면에서 고객을 돕습니다.
5. 팀에 맞는 적절한 스프린트 길이(기간) 및 완료조건을 정의합니다.
6. 우선순위, 개발 선후관계, 의존관계 등을 고려하여 사용자 스토리를 각 스프린트에 할당합니다.
7. 특정 팀의 속도(Velocity)를 측정한 자료가 있다면 이를 이용하여 스프린트에 할당된 사용자 스토리가 스프린트 내에 구현될 수 있는지 확인합니다.
8. 각 스프린트에 할당된 사용자 스토리의 구현을 기준으로 릴리즈 일정을 확정합니다.
9. 릴리즈 계획을 수시로 재점검하고, 변동 사항이 발생할 시 주기적으로 갱신하도록 합니다.

추정
Estimation



ⓐ 요약

- 사용자 스토리를 구현하는데 그 규모가 어느 정도인지 생각해 보는 과정입니다.
- 애자일 추정에서는 프로젝트 초기에 현실적인 판단이 어려운 'Man-hour', 'Man-day' 같은 시간 단위보다는 실제 수행한 것을 기반으로 공수를 추정해 낼 수 있는 스토리 포인트를 사용합니다.

ⓑ 역할자

- 스크럼 마스터: 추정 작업 시 퍼실리테이터 역할을 수행하기도 하며, 동시에 팀원들과 함께 업무량을 추정하기도 합니다.
- 제품 책임자: 업무를 추정할 사용자 스토리에 대해 팀원들에게 설명해 줄 수 있어야 하며, 추정에 도움이 되는 질문에 답변을 합니다.
- 팀원: 적극적으로 질의응답하며 사용자 스토리의 업무량을 추정합니다.

ⓒ 적용상황

- 릴리즈 계획 또는 스프린트 계획 시 전체적인 개발 계획을 수립하기 위한 추정이 필요할 때

ⓓ 준비물

- 인덱스 카드, 네임펜, 제품 백로그, 플래닝 포커

ⓔ 더 잘 하려면..?

- 제품 책임자와 스크럼 마스터는 도출된 사용자 스토리목록을 준비합니다.
- 제품 책임자는 사용자 스토리에 대해 업무 흐름을 중심으로 팀원들에게 설명할 수 있도록 준비합니다.

ⓕ 어떤 것이 해결되는가?

- 일의 규모가 보다 투명하게 파악되며 구체적인 계획을 수립할 수 있습니다.
- 추정을 한 결과로 나온 스토리 포인트를 반영한 소멸 차트를 활용하여 팀의 진행상태를 확인해 볼 수 있습니다.

ⓖ 주의사항

- 스토리 포인트로 표현된 추정치는 그 기능을 개발하는데 드는 노력의 양과 개발복잡도 그리고 그 안에 내재된 위험성 등을 한데 모아 표현하는 값으로 산정(Calculation)이 아닌 추정(Estimation)을 통해 도출됩니다.
- 추정의 경우 아무리 많은 시간을 들이더라도 직접 작업을 하기 이전에는 100% 정확해지지 않습니다. 추정은 추정일 뿐입니다. 완벽한 추정은 애초에 불가능하다는 것을 인정하고 포인트 추정에 너무 많은 시간을 들이지 않도록 합니다.
- 추정치는 팀에 속한 한 개인이 만들어 내기 보다는 여러 명이 참여하여 의견을 모으고, 서로 간의 이견을 조정하는 것이 중요합니다.

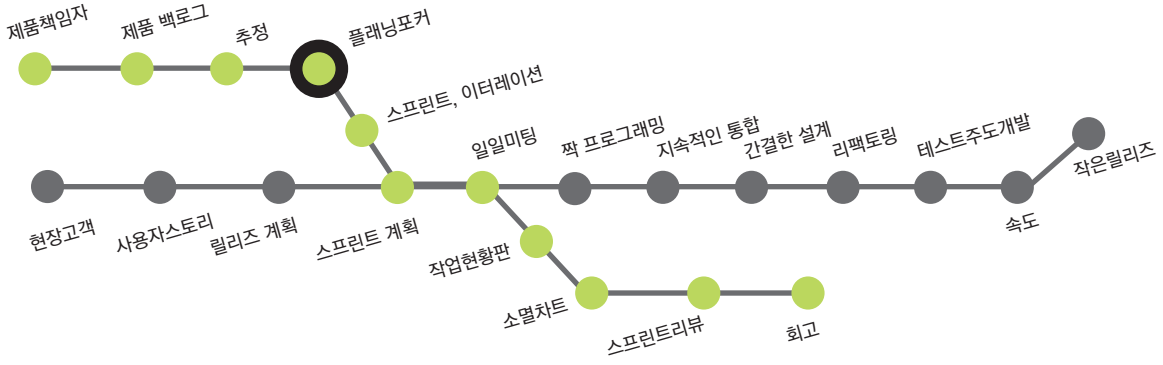
ⓗ 순서

1. 사용자스토리에 대한 설명 및 질문을 통해 팀원들이 사용자스토리에 대해 보다 깊이 이해 할 수 있도록 합니다.

▼

2. 플래닝포커 같은 실천법을 이용하여 사용자스토리를 추정합니다.

플래닝 포커
Planning Poker



ⓐ 요약

- 플래닝 포커는 사용자 스토리의 규모를 추정하는 방식입니다.
- 전통적인 방식과 달리 한 사람이 주도적으로 추정하는 것이 아니라 팀 전체가 같이 지혜를 모아 업무량을 추정하는 실천법을 말합니다.

ⓑ 역할자

- 스크럼 마스터: 플래닝 포커를 활용하여 업무량을 추정할 때, 퍼실리테이터 역할을 수행합니다.
- 제품 책임자: 업무를 추정할 사용자 스토리에 대해 팀원들에게 설명하고, 팀원들이 추정에 도움이 되는 질문을 할 때 답변하는 역할을 수행합니다.
- 팀원: 업무에 대한 공유, 질문을 통해 사용자 스토리의 업무량을 추정합니다.

ⓒ 적용상황

- 계획 수립 시 각자의 의견이 대립되거나 이해관계가 얽혀 있을 때
- 함께 계획을 수립하며 각자의 의견이 골고루 반영되도록 하고 싶을 때

ⓓ 준비물

- 인덱스 카드, 네임펜, 제품 백로그, 플래닝 포커

ⓔ 더 잘 하려면..?

- 제품 책임자와 스크럼 마스터는 도출된 사용자 스토리 목록인 제품 백로그를 준비합니다.
- 플래닝 포커 카드를 이용하면 더욱 좋으며, 플래닝 포커 카드가 없을 경우 직접 만들거나 손을 이용하여 진행할 수 있습니다.

ⓕ 어떤 것이 해결되는가?

- 다른 사람의 의견에 영향을 받지 않고 자신의 생각을 표현하고 공유할 수 있습니다.
- 업무에 대해 깊이 이해할 수 있습니다.
- 구현할 사용자 스토리의 범위와 제약사항 등에 대한 팀원 간의 이견을 드러내서 구현 시 참고할 수 있습니다.

ⓖ 주의사항

- 한 사람이 규모(크기)를 추정하도록 하지 않고 여러 사람이 참여하여 의견을 모으고, 서로간의 이견을 조정하면서 규모(크기)를 추정하도록 합니다.
- 누군가 먼저 "이 사용자 스토리는 5점 정도가 적당해" 라고 하면 다른 사람들이 이 말에 영향을 받을 수 있으므로, 모두가 동시에 의견을 내 놓도록 합니다.
- 스토리 포인트는 피보나치 수열로 1, 2, 3, 5, 8, 13, 20, 40 등으로 표현할 수 있습니다. 피보나치 수열을 이용하는 이유는 업무량을 대략적으로 추정하는 시점에서 큰 규모의 스토리의 경우 세세한 차이는 큰 의미가 없기 때문입니다.

ⓗ 순서

1. 사용자 스토리 중에서 적당한 사용자 스토리의 크기를 3으로 정하고, 이를 스토리 포인트라는 단위로 정합니다. 이 업무에 대한 내용을 설명하여 참여자 모두가 이해하도록 합니다. 참여자의 질문을 유도하여, 업무 크기에 대한 공감대를 형성합니다. 이 숫자를 인덱스 카드 한 귀퉁이에 기록합니다.

▼

2. 이제부터는 스토리 포인트가 3인 사용자 스토리를 기준으로 상대적으로 포인트를 정하면 됩니다. 3보다 두 배 정도 큰 일이면 5, 절반 정도의 일이라면 1이나 2를 선택합니다. 이때 다음과 같은 게임의 형식을 취합니다.

▼

3. 추정 하고자 하는 인덱스카드를 고르고, 제품 책임자가 추정할 사용자 스토리를 설명하면, 팀원들은 이에 대해 질문 하고, 제약사항, 위험 등에 대해 간략히 토론을 하여 업무에 대해 이해 합니다.

▼

4. 설명이 끝나면 하나~둘~셋 하면 각자 생각하는 숫자가 적혀있는 카드가 위로 향하도록 내밉니다.

▼

5. 이번 라운드에서 만장 일치가 아니고 차이가 발생하였다면 이제 의견 조정을 위한 이야기를 시작 합니다.

▼

6. 가장 낮게 나온 사람이 낮은 이유를 이야기 하고 가장 높게 나온 사람이 높은 이유를 이야기 합니다.

7 이야기 나눈 것을 바탕으로 다시 게임을 반복합니다.

8 만장일치가 될 때까지 진행하는 것을 원칙으로 하지만, 만약 3회가 되도록 만장 일치가 나오지 않는다면 중간 포인트를 선택하고 다음 게임으로 넘어갑니다.

9 40 이상의 일이라고 생각한다면, 사용자 스토리가 너무 커서 분할해야 하는 것은 아닌지 다시 한번 확인합니다.

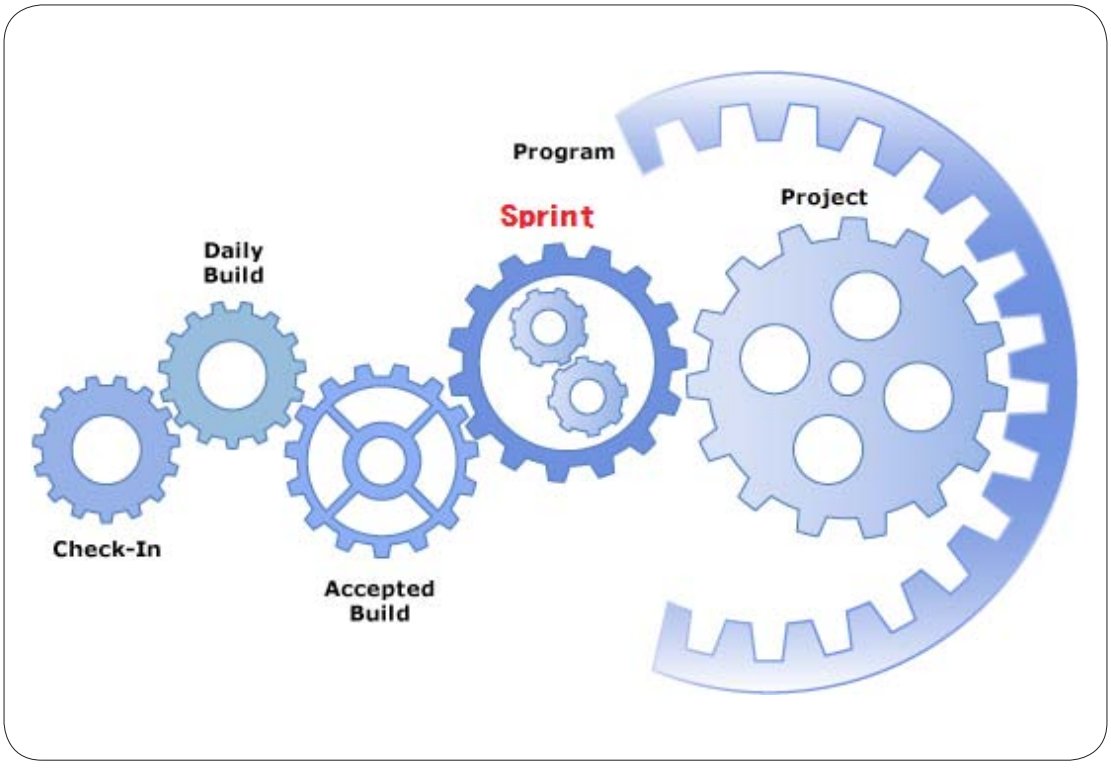
10 나눌 필요가 있다고 판단되는 일에 대해서는 세분화된 할 일을 인덱스 카드에 적고 이에 대해 플래닝 게임을 한 뒤, 이전의 인덱스 카드를 폐기합니다.

11 모든 할일 목록에 대해 플래닝 게임을 진행하면 추정을 마칩니다

스프린트

Sprint


제품책임자
제품 백로그
추정
플래닝포커
스프린트, 이터레이션
일일미팅
짝 프로그래밍
지속적인 통합
간결한 설계
리팩토링
테스트주도개발
작은릴리즈
현장고객
사용자스토리
릴리즈 계획
스프린트 계획
작업현황판
소멸차트
스프린트리뷰
회고
속도


Program
Project
Sprint
Daily
Build
Check-In
Accepted
Build

ⓐ 요약

- 개발팀은 스프린트라고 불리는 한정된 기간(Timebox)동안 계획한 일을 수행합니다. 프로젝트 상황에 따라 다르지만 주로 1주 ~ 4주 정도를 스프린트 기간으로 정합니다. 이를 다른 용어로 이터레이션(Iteration)이라고도 부릅니다.
- 팀의 역량에 따라 (속도(Velocity)에 근거하여) 스프린트 기간 동안 완료하기로 약속한 업무를 지속 가능한 리듬(Sustainable Pace)으로 작업을 수행합니다.
- 팀은 스프린트 기간 동안 스프린트 백로그(Sprint Backlog)에 스스로 계획한 목표를 달성하기 위하여, 분석/설계/개발/테스트를 통해 실행 가능한 소프트웨어를 개발합니다.

ⓑ 역할자

- 스크럼 마스터: 스프린트 동안에 팀 외부로부터의 업무나 간섭 등으로부터 팀을 보호합니다.
- 팀원: 스스로 계획한 일을 스프린트 종료일까지 완료하기 위해 노력합니다.
- 제품 책임자: 스프린트 목표를 정하고 명확한 요구사항을 전달하도록 노력합니다.

ⓒ 적용상황

- 고객이 자신이 투자한 비용으로 얼마나, 어떻게 구현되었는지 빨리 확인하고 싶어할 때
- 많은 변경이 예상되어, 먼저 동작하는 제품에 대해 고객의 확인을 받고 싶을 때

ⓓ 더 잘 하려면..?

- 스프린트 계획을 통하여, 팀원 동의 하에 스스로 수행할 수 있는 정도의 작업량을 정하고 스프린트를 수행합니다.
- 팀원들이 최대한 스프린트 기간 동안에 목표량에 집중하여 일할 수 있는 환경이 제공되어야 합니다.

ⓔ 어떤 것이 해결되는가?

- 분석/설계 단계에는 상대적으로 업무 강도가 낮고 개발/테스트 단계에는 상대적으로 강도가 높던 전통적인 방식에서 벗어나, 프로젝트 전체 기간 동안의 업무 강도가 평준화 됩니다.
- 실패하더라도, 빨리 적게 실패하여 프로젝트 전체에 부정적인 영향도를 줄일 수 있습니다.
- 다음 스프린트에서 더 나은 방법으로 스프린트를 수행할 수 있습니다.

ⓕ 주의사항

- 스프린트 길이를 정할 때는 제품 책임자와 팀원 사이에 합의를 통해 적절한 기간으로 정해야 합니다. 일반적으로 제품 책임자는 짧은 스프린트를 선호하고, 팀은 긴 스프린트를 선호합니다.
- 스프린트 길이를 균일하게 가져가는 것이 좋습니다. 왜냐하면, 스프린트 길이가 변경되면 모든 사람이 특정시기에 해야 할 일에 대해 혼란을 겪을 수 있기 때문입니다.

스프린트 계획
Sprint Planning

ⓐ 요약

- 스프린트 동안 수행할 스프린트 목표와 이를 이루기 위한 작업 상세 내역을 정합니다.
- 작업 상세 내역은 스프린트 동안 수행할 작업을 목록화한 스프린트 백로그로 정리합니다.
- 스프린트 내 완료기준을 정하고 팀원들과 공유합니다

ⓑ 역할자

- 스크럼 마스터: 스프린트 계획의 퍼실리테이터 역할을 수행합니다.
- 제품 책임자: 스프린트 목표를 정하고, 스프린트 백로그 내 스토리의 내용에 대해 팀원들에게 설명합니다.
- 팀원: 스스로 작업할 스토리와 작업량을 결정하고, 능동적으로 제품 책임자에게 구현 내용에 대해 질문하며, 요구사항을 상세화 합니다.

ⓒ 적용상황

- 스프린트 내 목표가 있고 팀이 이를 구체화 할 필요성이 있을 때
- 작업자를 명확히 결정할 필요가 있을 때
- 팀별 작업 내용에 대해 공유할 필요가 있을 때

ⓓ 더 잘 하려면..?

- 변경사항이 발생한 경우, 제품 백로그의 내용 및 우선순위를 업데이트 하고 스프린트 계획을 시작합니다.
- 제품 백로그에 있는 사용자 스토리 중 완료된 것과 미완료된 것은 명확하게 구분할 수 있어야 합니다.
- 제품 책임자는 스프린트 계획 대상 업무에 대해 팀원들에게 상세히 설명할 수 있을 만큼 충분히 인지하고 있어야 합니다.
- 이전 스프린트 수행 결과인 소멸 차트를 업데이트하고, 팀 속도를 확인해 둡니다
- 가능한 한 팀원 모두가 참여하여, 스프린트 동안의 수행해야 할 작업을 공유합니다.

ⓔ 어떤 것이 해결되는가?

- 팀원 모두가 팀이 결정한 완료기준을 재차 명확히 인지합니다.
- 팀원 모두가 스프린트 내 성취해야 할 명확한 목표를 설정하게 됩니다.
- 팀원들 스스로 자신이 작업할 내용을 결정하게 되므로 작업완료에 대해 책임감을 가지게 됩니다.
- 타 팀원의 작업 내용을 알게 되므로, 협업이 필요한 작업을 식별하게 됩니다.
- 팀원 모두가 자신이 해야 할 일을 보다 상세하게 인지하게 됩니다.
- 작업량이 많은 팀원과 적은 팀원을 인지하여, 서로 돕거나 배려할 수 있습니다.

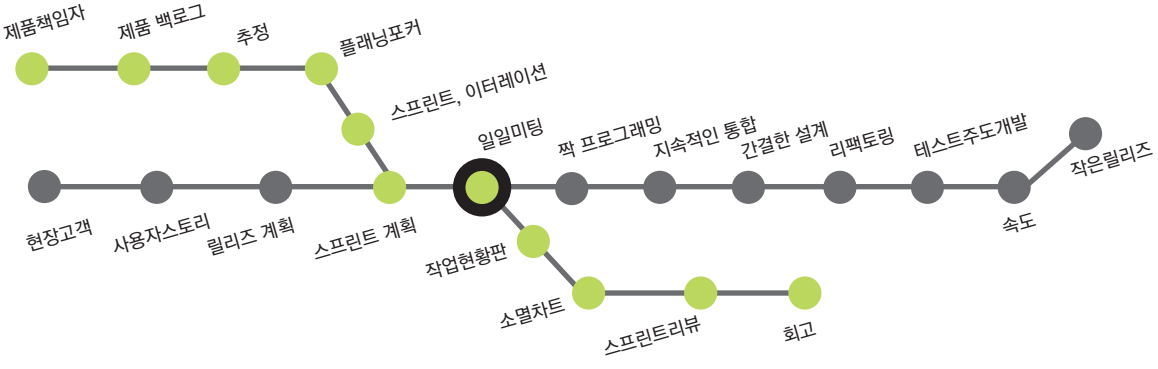
ⓕ 주의사항

- 완료기준에 대해 팀원들과 공유하지 않으면, 팀원 별로 결과물에 대한 품질 차이가 발생할 수도 있습니다.

ⓖ 순서

1. 스프린트 내 수행해야 할 목표를 제품 책임자가 팀원들과 공유합니다. 이 목표는 주로 스프린트 시작 시 제품 백로그 상의 가장 우선순위가 높은 내용들을 말합니다.
2. 스프린트 내 완료기준을 스크럼 마스터가 팀원들과 공유합니다. 혹시, 완료기준이 변경되어 수정해야 한다면, 제품 책임자를 포함한 팀원 모두의 합의하에 수정합니다.
3. 팀원들은 정해진 스프린트 기간 동안 수행할 사용자 스토리를 제품 백로그에서 선택합니다. 짝 프로그래밍을 한다면, 두 명이 함께 선택할 수도 있습니다.
4. 작업하기로 한 스토리 포인트가 개인별로 지나치게 차이가 많이 난다면, 합의 하에 조정합니다.
5. 팀 전체가 선택한 스토리 포인트의 합을 구하고, 그 합이 이전 스프린트들의 팀 속도보다 지나치게 높거나 낮을 경우 조정합니다.
6. 특정 사용자 스토리에 대해 스토리를 완료하기 위한 상세 태스크를 도출합니다.
7. 도출된 태스크에 담당자와 예상 작업시간을 적습니다.
8. 스크럼 마스터는 태스크에 적힌 개인별 시간의 합을 구하고, 스프린트 내 작업일(Work Day)에 비추어 적절한지 판단하고 조정합니다.
9. 제품 책임자와 스크럼 마스터는 팀원들이 세운 계획을 리뷰하고, 태스크에 식별되지 않은 작업이 있다면 추가합니다.
10. 스프린트 동안 작업할 계획에 대해 요약하여 팀원들과 공유합니다.
11. 팀원들 모두 정해진 스프린트 기간 동안 계획한 것을 완료하기로 서약(Commitment)합니다.

일일 스크럼
Daily Scrum



ⓐ 요약

- 일일 스크럼은 팀원 전체가 정해진 시각에 팀의 작업 현황판 앞에 모여 15분 간 서서 진행하는 회의입니다.
- 팀원들은 돌아가면서 각자가 어제 했던 일, 오늘 할 일, 문제가 되는 일에 대해 이야기 하면서 팀 전체 진행상황과 이슈를 서로 공유합니다.
- 스크럼의 대표적인 3가지 활동(스프린트 계획, 일일 스크럼, 스프린트 리뷰) 중 하나입니다.

ⓑ 역할자

- 스크럼 마스터: 팀원들이 일일 스크럼에 참여하도록 유도하고 시간이 길어지지 않도록 합니다. 가능하면 동등한 발언 기회를 제공합니다.
- 팀원: 개발자 외에도 분석가, 설계자, 테스터, 시나리오 담당자 등이 함께 할 수 있습니다. 스크럼에서는 기능별(Functional) 팀 보다 교차기능팀(Cross-functional team)이 되어 한 팀 내에서 협업하는 것이 좋습니다.

ⓒ 적용상황

- 애자일의 가장 기본적인 실천법을 시도해 보고 싶을 때
- 팀원 간 원활한 정보 교류가 필요할 때
- 팀 내 방치되는 업무가 존재하거나, 같은 업무를 함께 하거나, 개별 일정이 공유되지 않아 대기해야 하는 경우가 많을 때
- 같은 문제를 여러 팀원이 다른 시점에서 겪을 때
- 팀의 진척상황이나 이슈가 잘 드러나지 않을 때
- 팀원들 간 업무 작업량이 각자 달라 특정 인원에게 일이 몰릴 때
- 그날그날 무엇을 해야 하는지 몰라서 방황할 때

ⓓ 준비물

- 타이머 (15분)
- 벽면이나 화이트보드로 구성된 작업 현황판
- 포스트 잇, 마커 펜

ⓔ 더 잘 하려면..?

- 팀원 모두가 참여합니다.
- 기타 이해당사자들은 선택적으로 참석하되 팀이 이야기 하는 것에 귀를 기울이고 가능하면 미팅 이후에 자신의 이야기를 합니다.
- 제품 책임자와 자주 진행상황과 방향을 이야기하는 것이 좋습니다. 팀원들의 자율적인 커뮤니케이션을 위해 제품 책임자를 배제하게 되면, 자칫 제품 책임자가 불안함을 느낄 수 있습니다.

ⓕ 어떤 것이 해결되는가?

- 팀의 이슈를 일일 단위로 공유하고 논의하게 만들어 줍니다.
- 불필요한 회의를 줄일 수 있습니다.
- 이미 한 팀원에 의해 해결된 문제를 어떤 팀원이든 또다시 해결하려 하지 않게 해줍니다. 해결된 문제의 경우는 기존에 해결했던 사람이 도와줄 수 있게 됩니다.
- 프로젝트 후반부에 문제점이 갑자기 발생하는 것을 효과적으로 막아줍니다.
- 모든 사람들의 참여로 팀 전체 상황을 공유할 수 있습니다.

ⓖ 주의사항

- 사전에 정해진 시간과 장소에서 진행합니다.
- 15~20분 이내로 간단하고 빠르게 진행합니다.
- 하루 일과를 시작할 때 진행하면 좋습니다. (가능하면 오전에 진행)
- 한 사람도 빠짐없이 이야기 합니다.
- 기타 이해당사자들은 가능한 참여하지 않습니다. 대신 일일 스크럼을 마치고 장애요인에 대해서는 해결하도록 적극 나섭니다.
- 전문가나 말이 많은 사람이 말을 계속 주도하지 않도록 합니다.
- 불가피하게 일일 스크럼 미팅에 참석 못할 경우 반드시 사전에 모두가 알도록 합니다.
- 작업 현황판은 개인 별로 업데이트합니다.
- 소멸 차트를 쓴다면 같이 업데이트 합니다.
- 완료로 카드를 옮길 때에는 팀이 정한 '완료 정의 '를 만족하는지 확인합니다.

ⓗ 순서

1. 모든 사람이 "어제 무얼 했는지", "오늘 무엇을 할 것인지", "어떤 장애요인이 있는지"를 이야기 합니다.
2. 각자 작업 항목을 완료할 때까지 남은 시간을 이 시간을 통해 추정해 봅니다.
3. 문제점과 장애요인을 드러내고 필요하다면 목록을 갱신합니다.
4. 소멸 차트를 갱신합니다.

짝 프로그래밍
Pair Programming

ⓐ 요약

- 두 명의 작업자가 한 개의 PC (보통, 한 개의 키보드 및 마우스를 가진 작업공간)를 공유하며 작업하는 것을 말합니다.
- 키보드를 소유하고 작업을 진행하는 사람을 “드라이버(Driver)”라 하고, 직접 작업을 하지는 않지만 목표와 방향을 점검하며 드라이버의 작업을 검토하며 리드하는 사람을 “네비게이터(Navigator)” 라고 합니다.
- 대개 네비게이터가 작업의 방향을 제시하고 드라이버가 질문과 제안을 합니다.

ⓑ 역할자

다양한 수준 별로 짝을 지을 수 있습니다.

- 초보자와 초보자: 효율이 떨어지지만 서로 배우며 학습합니다.
- 초보자와 전문가: 숙련자의 경험을 전수받을 수 있습니다.
- 전문가와 전문가: 서로 신선한 자극을 받으며 개선점을 모색합니다.

ⓒ 적용상황

- 새로운 기술을 배우며 경험할 때
- 비슷한 실력의 개발자가 어려운 기능을 함께 개발할 때
- 팀원의 실력을 상향 평준화 하고 싶을 때
- 신입 개발자를 팀 내에 빠르게 적응시키려고 할 때
- 업무 난이도가 높고 실수하기 쉬운 문제를 해결해야 할 때
- 해결하기 어려운 버그를 처리해야 할 때

ⓓ 준비물

- 두 명이 함께 앉을 수 있는 작업공간과 함께 작업을 할 PC
- 타이머
- 함께 의논할 수 있도록 종이와 펜, 화이트보드

ⓔ 더 잘 하려면..?

- 관리자 및 팀원이 짝 프로그래밍에 대한 필요성을 인지할 수 있도록 합니다.
- 작업 규칙에 대해 팀원이 인지하도록 합니다.
- 서로 상대방을 신뢰하며 존중합니다.
- 대화를 논쟁으로 생각하지 않습니다. (승자/패자가 생겨서는 안 됩니다)
- 함께 수행할 작업에 대해, 대화 또는 데모 등의 방법으로 두 역할자(드라이버 &네비게이터)의 정보를 공유 합니다.

ⓕ 어떤 것이 해결되는가?

- 코딩을 하면서 자연스럽게 동료(Peer) 검토가 수행되어 에러가 발생할 확률이나 위험이 줄어듭니다.
- 팀원끼리 지식 공유가 가능해집니다. 게다가 짝을 지속적으로 변경하면, 팀 전체 지식 공유 또한 가능해집니다. 팀원 한 명이 자리를 비우더라도 누구나 그 작업을 대신할 수 있습니다.
- 신입 개발자는 숙련된 개발자로부터 다양한 경험을 짧은 시간에 배울 수 있습니다.
- 프로그래밍을 위해 사용하는 툴의 유용한 팁(Tip)을 공유할 수 있습니다.

ⓖ 주의사항

- 일반적으로 짝 프로그래밍은 혼자 프로그래밍을 하는 것보다 15% 정도의 시간이 더 소요됩니다(By Laurie Williams 의 경험적 연구). 때문에 코드의 품질을 얻는 것이 15% 시간을 더 소요하는 것보다 중요하다고 판단될 때 사용해야 합니다.
- 짝 프로그래밍 시 반드시 타이머를 두어 정해진 시간에 휴식을 취해야 합니다. 4 시간 이상의 짝 프로그래밍은 지양하도록 합니다.
- 너무 오랫동안 드라이버, 네비게이터 중 한 가지만의 역할을 수행하지 않도록 짝을 교체합니다.
- 홀로 작업하는 것을 선호하는 팀원의 경우, 짝 프로그래밍을 굳이 강요하지 않습니다. 하지만, 모두가 함께 하는 것을 권장합니다.

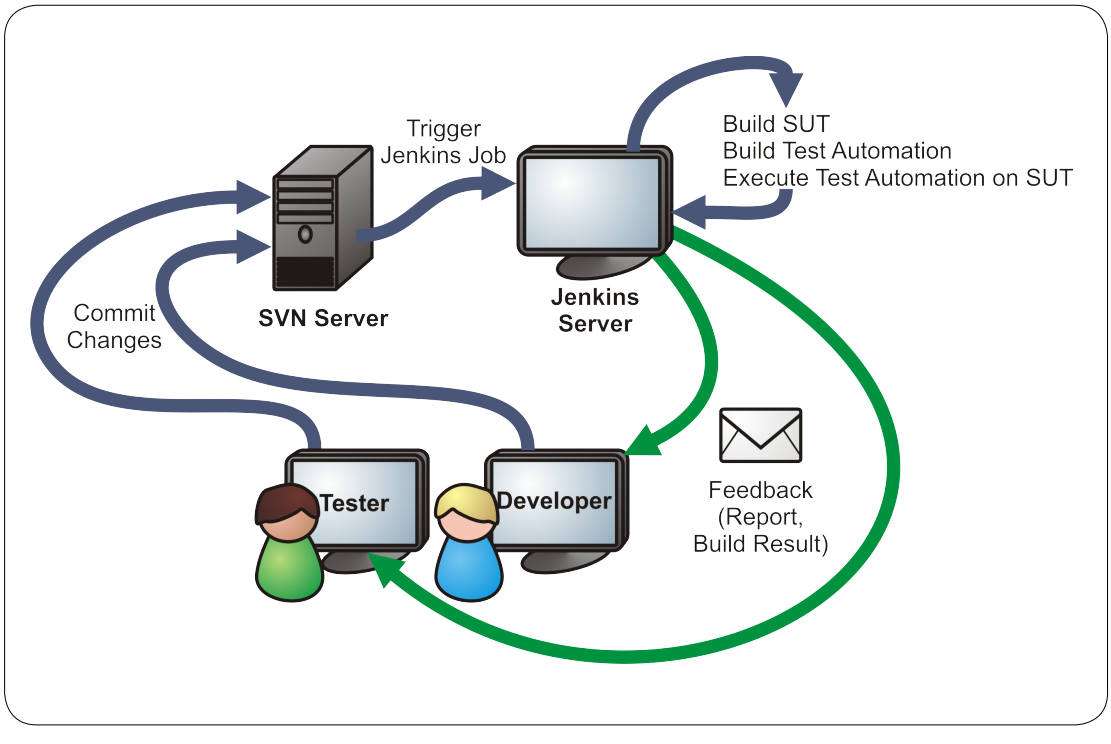
다음의 경우 짝 프로그래밍을 사용하지 않습니다.

- 팀원의 기술력이 뛰어나고 작업에 이미 익숙한데 쉬운 작업을 수행하는 경우 개별로 작업하는 것이 더 나을 수 있습니다.
- 팀원의 기술력이 매우 부족하고, 경험도 없는 경우임에도 어려운 작업이 맡겨졌다면 적절한 교육을 선행하여 수행하는 것이 더 나을 수 있습니다.

ⓗ 순서

1. 1 대의 PC에 드라이버와 네비게이터가 함께 앉습니다. 이 때 일반적으로 기존에 해당 코드를 작업하던 팀원이 드라이버를 맡습니다.
2. 드라이버는 키보드에 손을 올려 놓습니다.
3. 네비게이터는 금일 작업에 대한 목적 및 수행 태스크를 설명합니다.
4. 코드 수정 시 드라이버는 수정 내용을 설명하면서 코드를 짜고, 네비게이터는 큰 그림에 대한 비전과 아이디어를 제공합니다. 유용한 툴 사용 팁을 설명할 경우, 네비게이터가 잠시 키보드를 사용할 수 있습니다.
5. 사전에 정한 시간이 되거나, 필요에 따라 합의 하에 드라이버와 네비게이터가 역할을 바꾸어 수행합니다.
6. 정한 시간마다 역할자를 바꾸면서 3~4회 반복합니다.

CI
Continuous Integration



ⓐ 요약

- CI는 팀이 작업한 소스코드, 데이터베이스 스크립트, 코드검사(Inspection), 자동화 테스트 등을 가능하면 자주 통합하는 소프트웨어 개발 실천법 입니다.
- 자주 통합하고 검증함으로써 최신 코드가 항상 건강한 상태인지 확인할 수 있으며, 통합 주기를 짧게 가져감으로써 오류 발생 시 원인 파악을 신속하게 할 수 있습니다.
- 최소 하루에 한번 이상, 여러 번의 통합을 진행합니다.
- 자동화된 빌드를 통해 가능한 빨리 통합 에러가 없는지 검증하는 작업입니다.

ⓑ 역할자

- CI 담당자: 프로젝트 초기에 프로젝트에 맞는 CI 환경을 구성하고 관리합니다.
- 팀원: 자신이 작업한 것에 의해 통합 빌드가 실패하면 최대한 빨리 고치고, 내가 원인을 제공하지 않았더라도 실패를 발견한 경우에는 다른 팀원에게 알려줍니다.

ⓒ 적용상황

- 팀의 빌드가 깨져 복구하는 일이 빈번할 때
- 반복적으로 팀원들이 작성한 코드를 통합 해야 할 때
- 테스트, 검사, 배포를 자동화하고 싶을 때
- 언제나 배포 가능한 기능의 상태를 만들고 싶을 때

ⓓ 준비물

- 버전관리 저장소 예) SVN, CVS, GIT
- CI 통합서버 예) Hudson, Cruise Control
- 빌드 도구 예) Ant, Maven
- 통합빌드 장비

ⓔ 더 잘 하려면..?

- 빌드 할 대상이 거의 없을때부터 CI 환경을 구축하는 것이 좋습니다.
- 통합빌드는 가능한 한 모든 과정을 자동화하여 중간에 휴먼에러(Human error)가 유입되지 못하도록 하는 것이 필요합니다.

ⓕ 어떤 것이 해결되는가?

- 통합 위험을 줄여줍니다.
- 재작업을 줄여줍니다.
- 배포할 수 있는 소프트웨어 상태를 유지시켜 줍니다.
- 통합빌드로 인해 지속적인 상태 모니터링이 가능하기 때문에 프로젝트 가시성을 높입니다.
- 제품 완성도에 대해 팀원들이 보다 큰 자신감을 갖게 됩니다.

ⓖ 주의사항

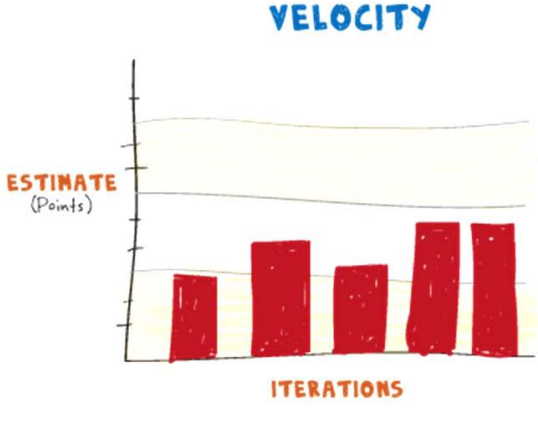
- 가능한 한 자주 빌드 서버에 커밋(Commit)하는 것이 좋습니다. 커밋 하는 주기가 하루 이상이 되면, 통합작업 자체에 많은 시간이 소모 될 수 있습니다. 이 때문에 다른 개발자들이 최신 변경사항을 자신의 로컬 환경에서 이용할 수 없게 되거나 추가적인 코드 머지(Merge) 작업에 많은 시간이 소모할 수도 있습니다.
- 불완전한 코드를 커밋해서는 안 됩니다. 특히, 통합을 관리하는 사람이 모든 개발자가 완전한 코드만 커밋할 것이라고 가정하는 것은 매우 위험한 생각입니다. 때문에, 누군가는 적절한 모니터링을 해야 합니다. 동시에 개발자들에게 장인정신(Craftmanship)같은 적절한 동기부여를 해주면 더욱 좋습니다.
- 빌드가 실패하면 즉시 고쳐야 합니다. 내가 고칠 수 없다면, 고칠 수 있는 팀원에게 가능한 한 빨리 알려야 합니다. CI 통합환경이 모두 녹색으로 바뀔 수 있도록 해야 합니다.
- 자동화된 개발자 테스트도 CI 환경에 포함됩니다. 이를 위해 JUnit과 같은 xUnit 프레임워크로 테스트를 자동화합니다.
- 테스트와 검사는 반드시 모두 통과해야 합니다.
- 개발자들 모두에게 로컬에 별도의 빌드 환경을 갖게하면 통합빌드 서버에서 실패할 확률이 더 적어집니다.

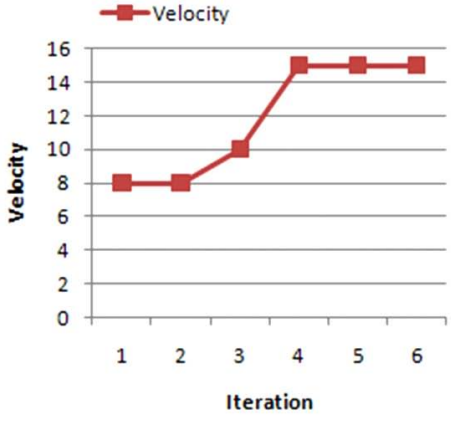
ⓗ 순서

1. 소스코드 컴파일: 소스코드 컴파일은 CI의 가장 기본적이고 일반적인 요소입니다.
2. 데이터베이스 통합: DDL, DML 스크립트를 이용하여 데이터 베이스를 생성하고 경우에 따라서는 데이터를 넣을 수도 있습니다.
3. 테스트: JUnit과 같은 자동화 단위 테스트 도구를 이용하여 테스트를 자동화합니다.
4. 검사(Inspection): 정적분석, 동적분석과 같은 자동화된 코드검사는 코드를 불완전하게 만들 수 있는 문제를 조기에 찾을 수 있도록 돕습니다.
5. 배포: CI를 통해 지속적인 배포를 수행하면, 항시 고객에게 사용 가능한 소프트웨어를 제공할 수 있습니다.

속도
Velocity


제품책임자
제품 백로그
추정
플래닝포커
스프린트, 이터레이션
일일미팅
짝 프로그래밍
지속적인 통합
간결한 설계
리팩토링
테스트주도개발
작은릴리즈
현장고객
사용자스토리
릴리즈 계획
스프린트 계획
작업현황판
소멸차트
스프린트리뷰
회고
속도


VELOCITY
ESTIMATE
(Points)
ITERATIONS


Velocity
16
14
12
10
8
6
4
2
0
Velocity
1
2
3
4
5
6
Iteration

ⓐ 요약

- 한 팀이 단위 스프린트 기간 내에 완료시킨 스토리 포인트 총 합을 속도(Velocity)라고 합니다.
- 이 때 완료의 정의는 고객과 사전에 정한 조건에 부합된 것이어야 하며, 모든 팀원이 함께 인지해야 합니다.

ⓑ 역할자

- 팀 리더: 속도를 측정합니다. 이를 위한 정보 획득을 위해 팀원들에게 상시 현재 상태에 대한 업데이트를 독려 합니다.
- 팀원: 팀원은 팀의 속도를 알고 있어야 하며 나의 팀 기여도를 함께 생각해 보아야 합니다.

ⓒ 적용상황

- 특정 팀이 자신들이 할 수 있는 일의 양에 대해 잘 모르는 경우 속도를 이용하여 추정할 수 있습니다.
- 고객과 이야기 할 때도 만약 팀의 속도를 이미 알고 있다면 좀 더 객관적인 사실을 기반으로 업무와 일정을 조율할 수 있습니다.

ⓓ 더 잘 하려면..?

- 사용자 스토리의 완료 기준을 반드시 사전에 고객과 함께 정의 해야 합니다.
- 그리고 이 완료 조건에 대해 팀원 모두가 동일한 생각을 가져야 합니다.

ⓔ 어떤 것이 해결되는가?

- 여러 스프린트를 거치며, 팀의 속도가 지속적으로 비슷한 수치가 나올 경우 예측 가능성이 높아져 향후 매우 정확한 계획이 가능해 질 수 있습니다.
- 팀의 속도를 이용하여 남아있는 사용자 스토리(범위), 리소스(인력), 출시일(납기) 등에 대한 적합성 여부를 예측 및 판단할 수 있어 위험관리에 좋습니다.

ⓕ 주의사항

- 새로운 팀과 새로운 사람들이 모인 경우, 최초 스프린트에 대한 속도 목표를 너무 낙관적으로 정하지 않도록 합니다.
- 매 스토리 마다 팀의 실제 일하는 날짜 및 인력의 투입/철수 등을 고려해야 합니다.
- 팀 간의 생산성 비교를 위해서 속도를 사용하지 말아야 합니다. 추정하는 대상과 작업 주체가 다르기 때문에 동일한 기준의 비교 지표로 사용할 수 없습니다.

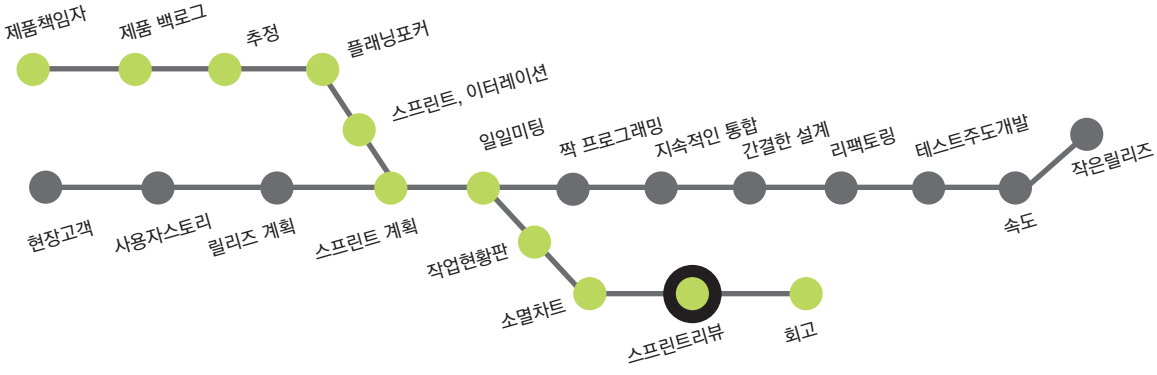
ⓖ 순서

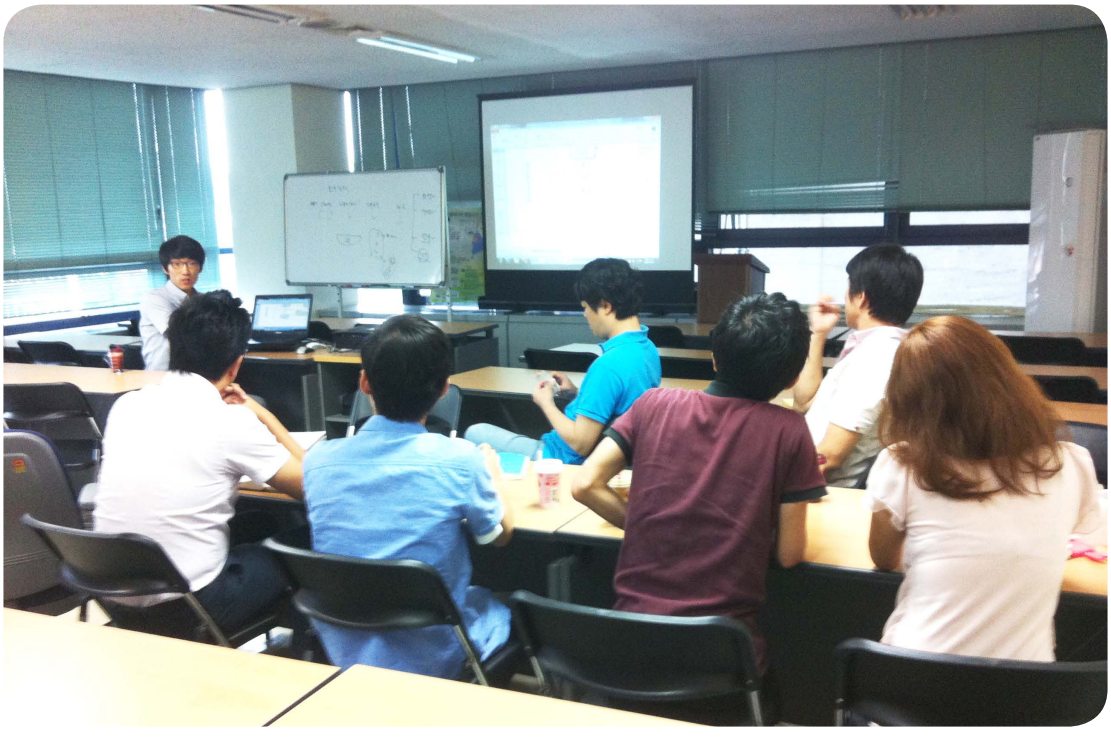
1. 스트린트#0: 팀의 속도를 모르기 때문에 전체 스토리 포인트의 합을 반복주기에 균등분할 하거나, 양이 적더라도 고객과 팀이 합의한 사용자 스토리로 스프린트#0의 작업 양을 정합니다.

▼

2. 스프린트#N: 해당 스프린트에서 완료한 속도를 측정, 스프린트#N+1에 활용 합니다.

스프린트 리뷰
Sprint Review





ⓐ 요약

- 스프린트 끝에 고객의 요구사항이 동작하는 기능으로 구현이 되면 스프린트 리뷰를 통해 구현된 기능을 고객에게 보여주고 신뢰와 피드백을 이끌어내는 활동 입니다.

ⓑ 역할자

- 전체 참석자: 질문, 관찰, 토론하고 제안할 수 있으며 이러한 활동을 서로 장려해야 합니다.
- 스크럼 마스터: 스프린트 리뷰 회의의 진행과 퍼실리테이션(리뷰회의 준비, 스프린트 개요 설명, 안건정리 등)을 책임집니다.
- 팀원: 고객을 대상으로 해당 기능이 포함된 업무 흐름을 간략하게 설명하고 기능 중심으로 데모 합니다.
- 고객, 제품 책임자: 제품 데모를 보고 피드백을 주어야 합니다.

ⓒ 적용상황

- 제품 기능에 대한 데모를 진행하고, 고객의 요구사항이 결과물에 잘 반영이 되었는지 피드백을 받고 싶을 때

ⓓ 준비물

- 스프린트 결과물
- 해당 스프린트 사용자 스토리 목록 (데모 조건, 완료 조건 참조)
- (필요 시) 참여자의 피드백 시트
- 약간의 다과

ⓔ 더 잘 하려면..?

- 데모는 고객이 원하는 가치를 전달하는 가장 중요한 활동입니다. 사전에 내부 테스트를 수행하여 품질을 지키는 것이 중요합니다.
- 개발한 소프트웨어가 전반적으로 작동이 안될 경우 이는 스프린트 내에서 테스트 활동을 제대로 하지 않았다는 의미입니다. 이런 상황이 발생하면 반성의 기회로 삼으면 좋습니다.
- 담당자가 결함이 두려워 데모에 소극적이라면, 고객의 요구사항이 제대로 구현되었는지 여부를 검증 받는 것이 중요하므로 작은 결함은 크게 문제가 되지 않는다고 설득합니다.
- 데모는 고객 및 이해당사자 중심으로 진행하지만 실 사용자 그룹을 별도로 선정하여 데모 후 테스트에 참여시키면 훨씬 더 효과적 입니다.

ⓕ 어떤 것이 해결되는가?

- 데모는 고객 및 이해당사자로부터 자신들이 투자한 비용이 어떻게 기능으로 구현되었는지 확인하는 자리입니다.
- 우리 팀이 어떤 결과물을 냈는지 이해관계자들이 알게 됩니다.
- 프로젝트 팀원들은 스스로 만들어낸 작업 결과에 대해 성취감을 맛볼 수 있습니다.
- 팀이 다음 스프린트에 무엇을 해야 하는지 결정하는데 도움이 되는 정보를 얻을 수 있습니다.

ⓖ 주의사항

- 여러 팀이 있는 경우 프로젝트의 스프린트 시작일과 종료일은 가능한 동기화하는 것이 좋습니다. 또한 스프린트 리뷰 일정도 일관된 날짜에 진행할 수 있도록 사전에 공유하고 실시하는 것이 좋습니다. (고객 시간 배려 및 집중)
- 스프린트 데모에 팀원은 가능한 모두 참여하여 고객의 피드백을 함께 공유하도록 합니다.
- 완성되지 않은 기능은 스프린트 리뷰 시 시연되지 않도록 데모 시나리오를 계획 해야 합니다.
- 수행 시간은 4시간 이내(유연하게 적용)로 하되, 전체적인 흐름을 확인하여 비즈니스 가치를 점검하는데 집중합니다. 해당 스프린트 계획 상 비즈니스 흐름을 확인하기 어려운 경우 사용자 스토리 별로 기능 중심의 리뷰를 진행할 수 있습니다.

ⓗ 순서

1. 개발팀은 고객 및 시스템 유관 조직에 데모일자, 장소, 기능(사용자 스토리)에 대하여 미리 공지합니다.

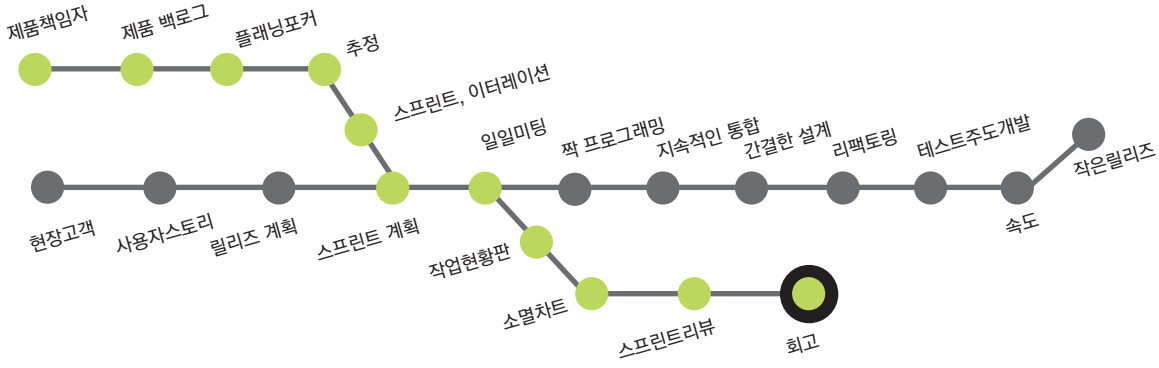
▼

2. 개발팀은 고객 비즈니스 관점에서 쉽게 기능 검증이 될 수 있도록 업무 흐름을 기반으로 데모합니다.

▼

3. 스프린트 리뷰 결과서 를 작성하여 고객과 공유합니다.

회고
Retrospective



ⓐ 요약

- 일정기간 동안 프로젝트를 수행한 뒤 팀이 프로세스를 개선하고 추후 동일한 문제점을 반복하지 않기 위해 실시합니다.
- 온고이지신(**溫故而知新**)의 개념으로 과거를 돌아보며 새로운 것을 깨닫는 활동입니다.
- 팀이 스스로 학습하고 개선해 나갈 수 있도록 하는 장치로, 애자일 실천법 중 가장 먼저 도입할 것을 추천합니다.
- 팀의 단점을 찾기보다, 강점을 찾아 극대화 하는 것을 목표로 합니다.

ⓑ 역할자

- 퍼실리테이터: 회고의 진행이 매끄럽게 수행되도록 참가자들의 관심과 참여를 유도합니다.
- 참가자: 함께 프로젝트를 수행한 팀원들이 참여합니다.

ⓒ 적용상황

- 팀이 같은 실수를 반복한다고 느껴질 때
- 조직이 비슷한 상황을 반복해서 겪게 될 때
- 팀의 불만이나 제안이 리더에게 전달되지 않을 때

ⓓ 준비물

- 벽면 혹은 화이트보드
- 포스트 잇 (색깔 별)
- 네임 펜
- 약간의 다과

ⓔ 더 잘 하려면..?

- 같은 프로젝트 팀 단위로 회고를 진행하는 것이 좋습니다.
- 팀은 스스로 더 나은 방향을 찾기위해 노력하는 자세를 가져야 합니다
- 애자일 실천법을 적용할 때 일단 작게 도입하고 회고를 통해 팀 스스로 실천법이 추구하는 방향을 자연스럽게 밟아갈 수 있도록 유도합니다.

ⓕ 어떤 것이 해결되는가?

- 바쁘다는 핑계로 커뮤니케이션이 없던 사람들 간에 소통이 시작됩니다.
- 팀이 반복적으로 실수하는 것들이 줄어들거나 없어집니다.
- 팀의 역량 수준이 점차 증가됩니다.

ⓖ 주의사항

- 회고 자체가 목적이 되어서는 안됩니다. 팀 스스로 필요하다고 느끼는 문제의식의 자각이 필요합니다.
- 팀이 바쁘고 분주할수록 스스로를 점검하고 되돌아보는 시간을 갖는 것이 필요합니다. 리더가 팀원들이 회고하는 시간을 잡담하거나 노는 시간으로 간주하여 회고 자체를 할 수 없게 막아서는 안됩니다.
- 회고를 하는 간격이 너무 길어서는 안됩니다. 사람이 과거의 사건과 실수를 기억하는데 한계가 있기 때문입니다. 따라서 수개월 단위로 회고를 할 경우 중간중간 사건들을 기록해 두어야 합니다.
- 리더의 지시에 의해 수행하는 경직된 분위기의 회고는 바람직하지 않습니다. 하향식 보다, 상향식으로 진행될 수 있도록 분위기를 유도/조성합니다.
- 리더는 팀 스스로가 개선하려는 의지가 있다고 믿어야 합니다.
- 회고에 참여한 모든 사람들이 의견을 개진할 수 있도록 해야 합니다.
- 팀이 결정한 활동이 지속될 수 있어야 합니다.
- 서로를 비난하는 방향으로 진행되면 진행자는 회고를 중단할 수 있습니다.

ⓗ 순서

1. 사전 준비하기: 회고에 참여할 사람들을 소집하고 회고할 수 있는 분위기를 만듭니다. 경직되지 않은 열린 마음을 가질 수 있도록 합니다.

▼

2. 자료 모으기: 기간 동안에 벌어졌던 다양한 자료들을 수집합니다. 시간이나 사건에 따라 모을 수 있습니다. 잘 했던 점, 아쉬웠던 점들을 나열합니다. 잘 했던 점을 더 잘하게, 혹은 미진했던 점을 극복할 수 있는 방법이 무엇인지를 모습니다.

▼

3. 통찰 이끌어내기: 이렇게 모여진 아이디어들을 바탕으로 투표를 하여 실제 다음 기간 동안 실행에 옮길 아이템을 선정합니다.

▼

4. 무엇을 할지 결정하기: 구체적이고, 측정할 수 있고, 달성 가능하며, 적절하고, 시기적절한(S.M.A.R.T) 목표를 세우도록 합니다.

▼

5. 회고 끝내기: 회고에 참석한 사람들에게 감사를 표현합니다. 회고 자체를 더 잘하기 위한 방법을 나눕니다.

03

CASE STUDY

사례1. 팀이 소통하기 시작하다

사례2. 지속적인 통합의 핵심은 지속적인 관심

사례3. 애자일팀에 개발자 말고 또 누가 있나요?

사례4. 애자일이 스며들다

사례5. 애자일 드러내놓고 해볼까?

사례 1 팀이 소통하기 시작하다

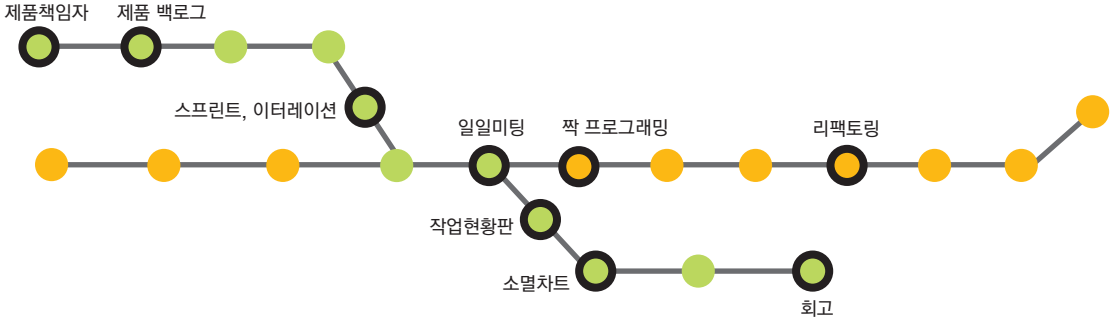


그림 3-1. 사례 1의 프랙티스 맵

PL 이 되다

핸드폰 애플리케이션을 개발하던 김 선임은 2008년 조직개편에서 PL 이 되었다. 김 선임을 추천한 상무님은 회사의 개발문화를 바꾸기 위해 사내 전문가를 통해 단위 테스트(Unit Test)와 리팩토링(Refactoring)을 교육/훈련시켰는데, "지금과 같은 땜빵 개발 방식으로는 더 이상 안된다" 라는 생각을 가지고 열심히 참여하여 좋은 성과를 내었던 김 선임을 눈여겨 봐두었던 터였다. 쟁쟁한 책임연구원들을 제치고 그것도 젊은 여성이 리더를 맡게되자 시샘하는 사람들이 많았고 김 선임은 무척이나 부담이 되었다.

맡게 된 애플리케이션 팀은 총 7명. 주임 연구원 4명과 출산휴가에서 복귀한 선임 연구원 1명, 신입사원 2명이었다. PL로서 어떤 역량을 갖추어야 하는지 제대로 배워본 적은 없지만, 자신의 지시를 기다리고 있는 7명을 어찌됐건 이끌어야만 한다.

김 선임은 열정적이고 의지에 찬 (약간은 떨리는) 목소리로 말했다

"여러분, 솔직히 저는 신참 PL 입니다. 많이 부족하지만 제가 꿈꾸는 팀의 모습이 있습니다. 매일매일 성장하는 느낌을 받을 수 있는 팀을 만들고 싶어요. 나중에 어떤 애플리케이션을 개발하더라도 다른 조직으로 이동을 하더라도 우리 팀에서의 경험이 매우 값지게 느껴지게 만들고 싶습니다. 함께 잘 해 봅시다."

하지만 팀원들의 반응은 냉랭했다. 잦은 조직변경에 여러 명의 리더를 겪었던 기존 멤버들은 그저 인사치레로 하는 말이려니.. 혹은 뜬 구름 잡는다는 식으로 받아들이는 것 같았다.

뜨뜨미지근한 반응을 뒤로한 채 '그래도 열심히 해보자' 라는 마음으로 업무를 파악하고 진행한 지 얼마 후, 팀원들의 성향과 역량의 차이를 느낄 수 있었다.

출산휴가에서 복귀한 선임 연구원은 과거에는 한 가닥 하던 개발자였지만 복귀 후 플랫폼이 바뀌어 일을 진행하는 속도가 더뎠다. 주임 연구원 4명도 개발능력이 뛰어난 2명과 다른 일을 하다가 합류하여 코드를 분석/파악하느라 속도가 더딘 2명으로 나뉘었다. 신입사원은 거의 말 수가 없는 여자 연구원과 컴퓨터를 전공하지 않은 남자 연구원으로 업무에 거의 도움이 되지 않았다. 결국 개발 능력이 뛰어난 2명의 주임 연구원에게 팀의 사활을 맡겨야 하는 상황이었다. 개발도 해주면서 나머지 개발자들도 이끌어주면 좋겠다는 생각을 하게 되었다.

반항적인 핵심 개발자

핵심 개발자인 이핵심 주임과 박잘난 주임은 같은 시간에 남들보다 많은 이슈를 해결하고 기능을 구현하고 있었다. 특히 이핵심 주임의 실력은 팀 외부에서도 인정해줄 정도로 뛰어났고 덕분에 팀원들도 이핵심 주임에게 물어보는 일이 많았다.

하지만 이핵심 주임도 결혼을 앞두고 자꾸만 자신에게 일이 몰리는 것이 부담이 되었고 점차 팀원들의 요청에 짜증 섞인 반응을 보이게 되었다. 그도 그럴 것이 동료를 도와주다 보니 근무시간에 자기 일을 많이 못하고 야근을 하는 날이 늘어가는데다, 설상가상으로 도움을 요청했던 동료들은 이 핑계 저 핑계로 퇴근을 해버리니 얄미운 마음이 들었던 것이다.

피로가 누적되서일까 이핵심 주임이 작성한 코드에서 큰 사고가 몇 차례 터지게 되었다. 막상 원인을 알고 보니 사소한 코딩 실수였고, 요구사항이 바뀌면서 수정했던 코드에서 예전에 잘 동작하던 기능이 오동작하는 것이었다. 김 선임이 코드를 함께 살펴보니 단위 테스트 코드도 전혀 없었고, 코드는 매우 지저분했다. 순환 복잡도 수치가 57 이고 함수 길이는 450 라인이었다.

김 선임: “이 주임, 요새 많이 피곤하지? 고생이 많아”

이 주임: “아.. 네.. 사실 요새 야근을 많이 해서 그런지 몸이 예전 같지 않네요”

김 선임: “가뜩이나 지쳤을텐데, 요 며칠 사고 터진 것 때문에 맘 고생도 많았겠어”

이 주임: “....”

김 선임: “이 주임을 나무라는 게 아냐. 우리 모두가 그러니까”

김 선임: “그런데 코드가 꽤 복잡하고 테스트 코드가 없던데.. 알고 있었어?”

이 주임: “네? 제가 짠 코드가 복잡하다구요?”

김 선임: “응.. 함수 길이도 그렇고, 복잡도 값이...”

이 주임: (말을 자르며) “함수 길이는 적당하구요. 간결하진 않지만 복잡하진 않은데요”

이 주임은 자신의 코드를 비난하는 줄 알았는지 매우 방어적이고 예민하게 반응했다.

김 선임: "이 주임.. 이 주임을 탓하는게 아니야. 코드를 고칠 때마다 회귀 테스트를 해서 과거에 정상동작하던 기능들이 여전히 문제없는지 확인해야 하는데..."

이 주임: "저도 나름 확인을 한 거라구요! 너무 촉박하고 정신이 없었을 뿐이예요"

김 선임: "그래.. 알아! 나도 그랬으니까. 회귀 테스트라는 걸 해야하는데 바빠서 놓치게 된다는거.. 그런데 테스트 해야하는 걸 코드로 작성해 놓고 그걸 기능을 수정할 때마다 실행시켜 본다면 회귀 테스트를 금방할 수 있어"

이 주임: "가뜩이나 촉박한 일정에 기능 구현도 벅찬데.. 테스트 코드까지 작성하라구요? 게다가 제 코드는 UI 처리가 많아서 테스트 코드를 어떻게 작성하라는 거죠? 차라리 직접 손으로 테스트하는게 더 빠를 것 같은데요?"

김 선임은 자신이 상황을 더 악화시킨 건 아닌가 걱정이 됐다. 다음 날 부터 이핵심 주임이 자신을 대하는 태도가 냉랭해졌고 팀원들과 회의를 할 때에도 다소 부정적인 반응을 보이게 되었기 때문이다.

이 주임은 필경 자신의 코드에 대한 관찰과 조언이 자신을 비난하는 것으로 받아들였던 것 같다. '넌 그런 것도 모르냐' 는 식으로.. 이 주임이 그렇게 반응하자 기술적인 부분에서 김 선임과 이 주임의 입장이 다르면 팀원들도 이 주임의 말에 휩쓸리곤 했고 김 선임은 마음이 착잡해졌다.

사내 전문가의 도움을 구하다

김 선임은 이대로라면 팀 분위기가 더욱 악화될 것이 우려되었고 단위 테스트와 리팩토링을 교육받았던 사내 전문가를 만나기로 마음먹었다. 큰 기대를 하지 않고 그냥 간단한 조언이라도 건질게 있을까 하는 마음으로 만났다.

상황을 전하는 동안 한가닥 책임은 알 수 없는 미소를 띠며 고개를 끄덕였다. 김 선임의 말이 끝나자 그는 흥미롭다는 눈빛으로 '재미있네요. 팀에 단위 테스트, 리팩토링을 적용해보고 싶은 김 선임의 요청을 어떻게 거절하겠습니까. 그리고 제가 이핵심 주임을 만나볼게요. 단, 팀에 제 자리를 하나 만들어 주세요. 몇 주 간 함께 일하고 싶네요' 라고 말했다.

어떤 도움을 줄 수 있을지는 모르겠지만 한가닥 책임의 실력은 교육을 받으며 매우 뛰어나다고 느꼈으니 여러모로 도움을 받을 수 있을 것 같았다. 신입사원 충원을 위해 비워둔 자리가 두 자리 남아 있었기 때문에 한가닥 책임의 요청은 쉽게 들어줄 수 있었다.

학습 분위기가 형성되다

한가닥 책임은 우선 신입사원들과 짝 프로그래밍을 하거나 코드 리뷰를 시작했다. 그러면서 그들이 잘 몰랐던 개념이나 기법에 대해 간단히 정리하여, 거의 매일 팀 내 작은 화이트보드 앞에서 조촐하게 세미나를 했다. 세미나는 누구나 참여할 수 있었고 화이트보드는 팀 자리의 중앙에 위치해 있었기 때문에 지나다니다 슬쩍슬쩍 어떤 주제로 세미나를 하는지 알 수 있어 좋았다.

초기에는 개발 역량이 뛰어난 두 명의 주임 연구원을 빼고 시작했는데, 중간중간 한가닥 책임이 전체가 알아야 할 주제나 실수가 있는 경우 모두를 불러서 세미나를 실시했다. 신입사원은 매일매일 바쁜 선배사원에게 전수받지 못하는 코딩 스타일과 기법을 배울 수 있었고, 중요한 개발 팁은 기존 사원에게도 도움이 되었다. 설명을 하다가 의도적으로 한가닥 책임은 단위 테스트와 리팩토링 실천법에 대해 언급하였고, 결국 팀원들이 먼저 소개해달라고 요청을 하여 강의를 하기에 이르렀다.

그러던 어느 날, 팀에 중대한 결함이 발견되어 그 문제를 해결하기 위해 이핵심 주임과 한가닥 책임이 짝 프로그래밍을 하게 되었다. 이핵심 주임의 설명을 듣고 코드를 살펴본 한가닥 책임은 문제 상황을 (강의에서 다루었던) 단위 테스트 코드로 작성해볼 것을 제안하였다. 이핵심 주임은 마뜩한 표정이었지만 한가닥 책임의 리드 아래 테스트 코드를 어떻게 작성하는지, 우리가 살펴봐야 할 문제가 무엇인지 설명을 들었다. 그리고 한가닥 책임이 직접 키보드를 잡고 코드를 작성하기 시작했다.

기본적인 테스트 코드를 몇 개 작성한 후, 한 책임은 코드가 어떤 의도로 작성되었는지 잘 표현할 수 있도록 Extract Method 기법을 사용하여 분리/정리하였다. 그리고 각각의 함수에 대해 다시 테스트 코드를 작성하였다. 하나하나 코드를 수정하고 분리할 때마다 테스트 코드를 작성하고, 테스트를 실행하여 문제가 없다는 것을 확인하며 진행했다. 문제는 금방 발견되어 수정을 마쳤지만 한가닥 책임은 멈추지 않았다.

지금 우리가 알고 있는 스펙은 일단 충분히 테스트 코드로 작성해 두죠

한가닥 책임의 말이었다. 이핵심 주임의 입에서 '디버깅 해보고 코드 몇 줄 고치면 되는데.. 뭐하러 저렇게 무리를 한담?' 이라는 말이 튀어나갈 뻔 했다. 그렇게 30~40분을 더 투자한 후 한 책임은 키보드에서 손을 내려놓았다.

보니까 이 코드는 수정이 많이 됐던 코드더군요. 다음 번에도 또 수정될 가능성이 있을 것 같아 조금 무리해서 테스트 코드를 달았어요. 좀 비효율적으로 보이죠?

한가닥 책임의 말에 이핵심 주임은 마음을 들킨 것 같아 얼굴이 발그레 해졌다.

그로부터 며칠 뒤, 예언이라도 한 것처럼 한가닥 책임이 붙잡고 있던 코드에서 문제가 발견됐다. 사실 코드에 문제가 있어서는 아니고 요구사항이 바뀌어서 기존 코드의 동작이 결함으로 올라온 것이었다. 한가닥 책임은 이번에도 이핵심 주임과 짝 프로그래밍을 하자고 했다.

마음을 연 핵심 개발자

한가닥 책임은 먼저 변경된 요구사항을 테스트 코드로 작성하고, 삭제된 요구사항 테스트 코드는 주석으로 처리했다. 한 책임은 그 테스트 코드를 통과시켜 보자며 이핵심 주임에게 코드 수정을 요구했다. 이핵심 주임은 만족스럽게 코드 수정을 마쳤다며 키보드에 손을 뗐다. 한 책임은 기존 테스트 케이스들을 전부 실행시켜 보았다. 두 개의 테스트 코드가 실패하는 것이 아닌가! 이핵심 주임은 당황하여 디버깅을 해보았다. 코드를 수정할 때 if 문을 하나 빠뜨렸는데 그것이 원인이었다.

부끄러워 하지 마세요. 스펙이 테스트 코드로 존재하니까 수정할 때 안전망이 되어주죠? 만약 이런 안전망이 없었다면 나중에 품질부서나 고객의 결함으로 발견이 되어야만 찾아서 수정할 수 있을 거예요. 기존 코드도 그렇게 수정돼 왔었는지 많이 지저분한 상태였어요.

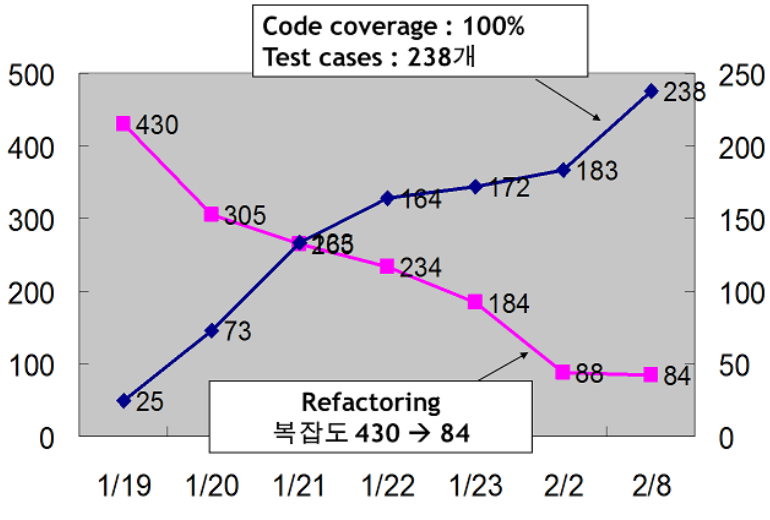


그림 3-2. 안전망으로서의 테스트 코드를 갖추다

한가닥 책임은 웃으며 설명했다. 이 날의 경험은 이핵심 주임의 태도를 180도 바꾸었다. 불필요해 보이던 테스트 코드를 작성하는데 본인이 더욱 열심이었고, 한 책임에게 궁금한 것들을 자주 물어보고 배운 것을 팀 내에서 세미나하게 되었다. 팀의 기술적인 리더인 이핵심 주임의 세미나는 한 책임의 세미나 만큼이나 팀원들을 집중하게 만들었고, 팀에 "잘 모르는 것은 누구라도 공부해서 알려주자" 라는 분위기를 형성하였다. 이핵심 주임은 단위 테스트와 리팩토링의 팀 내 전도사가 되었고, 자신이 만든 샘플 코드들을 세미나에서 소개하였다.

스크럼을 도입하다

이핵심 주임의 태도가 바뀌면서 팀의 분위기가 매끄러워졌고 김 선임은 매우 만족했다. 이 무렵, 프로젝트 관리 경험이 부족한 김 선임이 업무를 분장하고 진척을 확인하는데 있어서의 애로사항을 이야기하니 한 책임은 애자일 진영의 프로세스인 스크럼을 권유하였다

한가닥 책임이 스크럼을 소개하고 스크럼 마스터의 역할을 맡기로 했다. 팀원들은 여전히 새로운 것에 적응해야 한다는 불만이 있었지만, 그래도 팀에 많은 도움을 주는 한 책임의 권유라 한 번 믿어보기로 했다.

그림 3-3. 팀의 스크럼 현황판

팀을 강제로 순환휴가 보내다

일일 스크럼 미팅을 진행하는데 팀원들의 반응이 시큰둥했다. 다들 각자 맡은 업무가 있고 담당 업무 별로 기능을 구현하거나 문제를 그날그날 해결해야 퇴근할 수 있으니.. 다른 사람들의 이슈나 코드를 알려고 하지 않는 분위기였다. 출시 일자가 다가오면 어느 기능에서 문제가 터질지 모르니 주말이든 공휴일이든 출근해야 했다. 5분 대기조가 되어 회사를 지켜야 하는 상황에서 그런 심리는 당연한 것일 수 있었다.

고민 끝에 김 선임은 팀원들을 강제로 휴가 보내기로 결심했다. 첫째는 휴가 소진을 권하라는 지시사항을 따르기 위함인데다, 둘째는 지친 개발자들을 잠시라도 쉬게 하고 싶었다.

처음 휴가를 가라고 했을 때 팀원들은 불안해 했다. 당장 출시해야 하는 급박한 이벤트가 없는 시즌인데도 말이다. 특히 이핵심 주임부터 시작하려니 책임감이 강한 이핵심 주임은 자기 업무를 백업할 사람이 없으니 휴가를 반납할 기세였고, 팀원들은 이핵심 주임이 없는 동안 일이 터지면 어떻게 해결할 수 있을지 안절부절했다. 사실 제일 불안했던 사람은 김 선임이었다. 이핵심 주임이 자리를 비운 사이에 대박 사고라도 터지면 자신이 모든 책임을 져야만 하기 때문이었다. 김 선임은 한가닥 책임도 계시니까 안심하라고 팀원들을 설득하고 본인의 결정대로 강행하였다.

팀이 소통하기 시작하다

이핵심 주임은 담당하던 기능을 팀원들에게 설명하고 휴가를 떠났다. 김 선임은 웃으며 회사에서 오는 전화는 일체 받지 말라고 했다. 팀원들은 불안해 했지만 큰 문제 없이 일주일을 잘 넘겼다.

그렇게 몇 번의 차례를 거친 후에 팀원들 사이에 두 가지가 마음에 새겨진 것 같았다. 첫째는 '나도 평일이나 주말에 휴가를 갈 수 있구나' 하는 것이고, 둘째는 '다른 사람의 기능/코드도 관심을 가져야 하는구나' 하는 것이었다.

그 뒤로 일일 스크럼 미팅에서 사람들의 눈빛이 바뀌었다. 관심 없던 사람들이 다음에 휴가를 누가 언제 떠날지 모르기 때문에 다른 팀원들의 업무를 귀담아 듣기 시작했다. 그리고 자신이 겪었던 삽질(?)에 대한 경험도 나누면서 팀원들이 같은 실수를 반복하지 않도록 했다. 팀이 하나가 되어 문제를 해결하려는 모습이 보이자 김 선임은 매우 만족해 했다.

남은 숙제

기술적으로는 단위 테스트와 리팩토링을 적용하여 개발을 진행하고, 지속적인 통합 시스템을 구축하여 매일매일 그 날 작성된 코드를 빌드하고, 작성된 테스트 코드를 실행하여 그 결과를 메일로 알려주도록 하였다. 팀은 스스로가 성장하고 있다는 느낌을 갖게 됐다.

이 무렵 애플리케이션 개발에 싫증이 났다며 조금 도전적인 프레임웍 코드를 작성해보고 싶다는 팀원이 다른 조직으로 이동하겠다는 의사를 밝혔다. 김 선임은 별 차이가 없고 여기서 더 많은 것을 배우고 성장하는 것이 본인의 경력에 도움이 될 것이라며 설득했지만, 그 친구는 결국 마음을 굳히고 팀을 떠났다.

그런 아픔이 있었지만 김 선임은 주변에 자신의 사례를 소개하고 애자일과 스크럼을 권했다. 그 새 김 선임은 "다른 팀은 스크럼과 애자일 기법을 사용하지 않으면서 어떻게 관리하는지 모르겠다" 고 말할 정도로 애자일 옹호론자가 되었다. 하지만 여전히 "우리 상황에는 맞지 않는다" 고 반발하는 NAH(Not Applicable Here) 신드롬에 빠진 PL 들은 PL 경험이 적은 나이 어린 김 선임의 권유를 잘 받아들이려 하지 않았다.

사례 2

지속적인 통합 Continuous Integration의 핵심은 지속적인 관심Continuous Interest

그림 3-4. 사례 2의 프랙티스 맵

지속적인 통합(CI) 실천법 도입 계기

대형 인터넷 서비스 업체인 M사는 시스템이 커지면서 오랫동안 유지보수를 해야 하는 레거시 코드가 많아지게 되었다. 점점 유지보수가 힘들어지는 시스템이 생기면서 대규모 개편작업을 필요로 하는 경우가 빈번해졌다. 개편작업에는 많은 비용과 시간이 드는 것이 당연했다. 이런 일이 반복적으로 생기지 않도록 만들자는 목소리가 커졌다. 그래서 M사 애자일 개발의 시작은 "SW품질을 높이자" 였다. 그 중에서도 코드 품질을 높이는 데에 집중하고자 했다. 그러기 위해서는 코드 품질에 대한 가시화가 필요했다. 이런 일을 진행하기 위해 우선적으로는 인프라를 구축하는데에 노력을 기울이기 시작했다.

CI 도입 방법

이러한 인프라 구축을 위해 BDS(Build Deploy System), BTS(Bug Tracking System, JIRA), CI서버 등 일종의 Quality Dashboard(QD) 같은 사내 시스템을 구축하게 되었다. BDS는 SVN 코드 저장소를 중심으로 한 형상관리 시스템이고, BTS는 아틀라시안 JIRA를 중심으로 한 태스크 관리 시스템이다. CI서버에 대해서는 이후 자세하게 설명하기로 하고 QD에 대해 먼저 설명하자면 QD는 CI서버를 통해 수집한 각종 SW 품질 관련 지표들을 일목요연하게 보여주는 시스템이다.

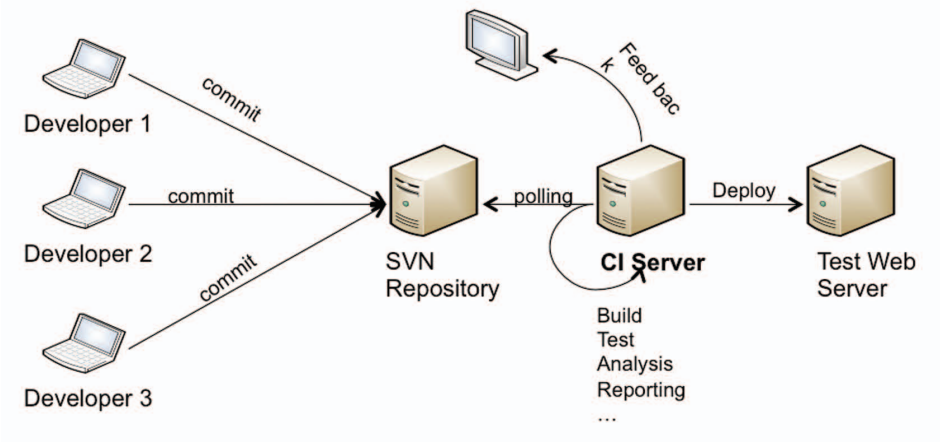
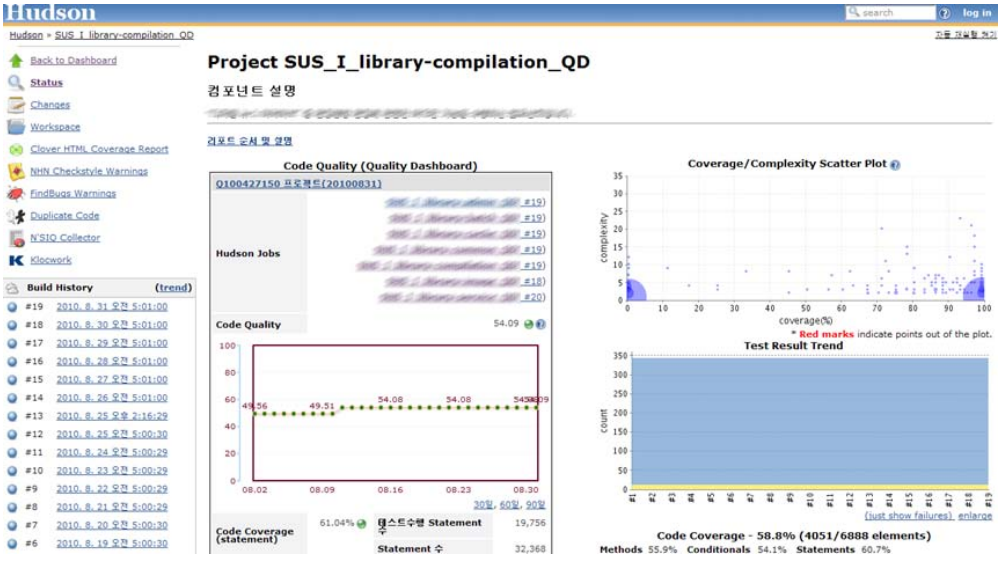


그림 3-5. 기본적인 CI 구축 구성도

CI를 도입할 때 가장 먼저 고민했던 점은 중앙 집중형으로 갈 것인지, 팀에서 각자 CI 서버를 구축할 것인지에 대한 결정이었다. 결론적으로, 각 팀에서 CI 서버를 개별적으로 구축하도록 가이드하기로 하였다. 이유는 각 팀에서 만드는 서비스가 다양하므로 중앙에서 한 가지 방식을 정해 가이드하기에는 무리가 있었기 때문이다. 하지만 각 팀에서 CI 구축에 대해 지식과 경험을 가진 사람이 많지는 않았다. 그리하여 해당 작업을 도와주는 내부 전문가들을 육성할 필요성이 대두되었고, 팀을 꾸려 인력을 키워나갔다.



| 그림 3-6. 팀 별 CI 서버 화면

CI서버로는 오픈소스 CI 서버인 젠킨스(Jenkins)를 전사 표준으로 결정했다. 내부 전문가 팀은 CI서버에 대한 교육을 진행했으며, 불편사항에 대응하는 일을 하게 되었다. 전담팀에서는 각 팀에서 필요로 하는 기능이 있으면 플러그인을 찾아서 표준 플러그인으로 가이드하거나, 없을 경우에는 직접 만들어서 배포하기도 하였다.

빌드 스크립트가 뭔지도 모르는 팀도 있었고 CI라는 개념조차 생소한 팀도 있었다. 서비스 개발환경도 팀 별로 매우 달랐기 때문에 표준 CI서버조차 어떤 식으로 적용해야 하는지 어려워 하는 팀도 있었다. 이를 보완하기 위해 내부 전문가 팀은 인력을 지원하는 것 뿐 아니라 사내 CI 교육도 정기적으로 개설/진행하였다.

또한 각 기술 환경에 맞는 지침서를 작성해서 배포하였다. 그리고 매 분기마다 각 팀의 CI서버를 점검해서 그 결과를 바탕으로 개선해 나가는 일을 3년에 걸쳐 진행하게 되었다. CI 지표들의 종합점수에 해당하는 CI 서버 거버넌스라는 것을 만들어서 각 팀에서는 해당 팀의 CI 서버 운영 수준이 어느정도 인지 쉽게 알 수 있도록 도왔다.

		● 24	● 30	● 20	● 3	프로젝트 수		
프로젝트명	조회기준일	Code Quality	코딩 표준 준수율 (%)	Code Coverage (Branch) (%)	Code Coverage (Statement) (%)	잔존 정 석 결함 (개/KL		
[illegible]	[illegible]	19,75	41.68	15.69	측정안함	5.		
[illegible]	[illegible]	49,25	39.29	62.01	측정안함	[illegible]		
[illegible]	[illegible]	52,07	100	75.56	측정안함	[illegible]		
[illegible]	[illegible]	49,25	39.29	62.01	측정안함	[illegible]		
[illegible]	[illegible]	59,16	96.74	63.22	측정안함	0.		
[illegible]	[illegible]	58,83	97.62	측정안함	71.03	0.		

ㅣ 그림 3-7. 사내 CI 종합관리 화면

CI 도입 시 어려웠던 점

CI 서버를 구축 하는 것 자체에는 별 부담감이나 거부감이 없었다. 이미 나름대로의 CI서버를 운영하던 팀들이 있기도 했다.

QP (Quality Practice)

그런데 단순히 CI서버만을 운영하는 것만으로는 부족한 점이 있었다. SW 품질을 가시화 하기 위해서 어떤 지표를 쓸 것인지 고민하게 되었다. 그래서 만들어진 것이 QP(Quality Practice)라고 하는 품질지표이다. QP는 코드 커버리지(테스트 코드가 제품 코드의 어느 정도를 실행하며 검증하였는지를 측정하는 지표)를 측정하고, 코딩 표준 준수 여부를 확인하며, 복잡도가 높은 코드와 중복코드를 찾아내서 개선하도록 유도하고, 테스트 실행 결과 등을 챙기도록 가이드 하는 일련의 활동을 지칭하는 사내 고유 용어로 이후부터 QP를 각 팀을 비롯하여 전사적으로 챙기게 되었다.

각 팀은 자신이 운영하는 CI서버에서 해당 팀의 QP정보를 관리할 수 있게 되었으며, 해당 정보는 또한 중앙으로 모아서 QP Dashboard (QD) 를 통해 많은 관리자들이 살펴볼 수 있게 했다. 이런 정보를 바탕으로 지속적인 관심과 개선을 유도할 수 있었다.

이런 활동이 지속되면서 점차 규모가 커졌고, 2011년에는 200대 이상의 젠킨스 서버가 운영되었고 CI로 관리되는 프로젝트의 수는 2,000개 이상으로까지 증가되었다. 이정도 규모로 CI서버를 운영하는 사례는 국제적으로도 드문 경우에 속한다.

서서히 드러나는 문제점들

하지만 좋은 점만 있는 건 아니었다. CI서버 자체는 개발자와 개발팀을 위해 만들어진 것이었으나, 일부 개발자 입장에서는 자신이 작성한 코드나 제품에 대한 내용이 마치 발가벗겨져서 공개되는 듯한 느낌이라고 이야기하는 경우도 있었고, 이로 인해 거부감을 갖는 경우도 생겼다.

QP 또한 개발자를 지원하고 코드 품질 향상을 도와주기 위해 만들어졌으나, 관리자가 사용하는 지표의 하나처럼 느껴지거나 활용되는 경우가 생김에 따라 마찬가지의 부작용이 발생했다. 예를 들자면 테스트 커버리지의 경우 커버리지의 숫자를 높이는 것이 목적이 아닌 테스트를 신경써서 잘 만들자는 것임에도, QP 지표의 목표가 숫자를 높이는 것에만 집중되는 현상이 발생하곤 했다.

결론

하지만 결국에는 모든 팀과 개발자들이 CI서버 사용을 당연한 활동으로 받아들이게 되었다.

성공적으로 CI를 도입하려면 1차적으로는 상위 조직장의 강한 의지와 지원이 필요하고 이에 더해서 지속적인 기술 지원과 점검이 함께 따라야 한다고 본다. 즉, 지속적인 통합(Continuous Integration)의 핵심은 지속적인 관심(Continuous Interest)인 것이다.

사례 3

애자일팀에 개발자 말고 또 누가 있나요?

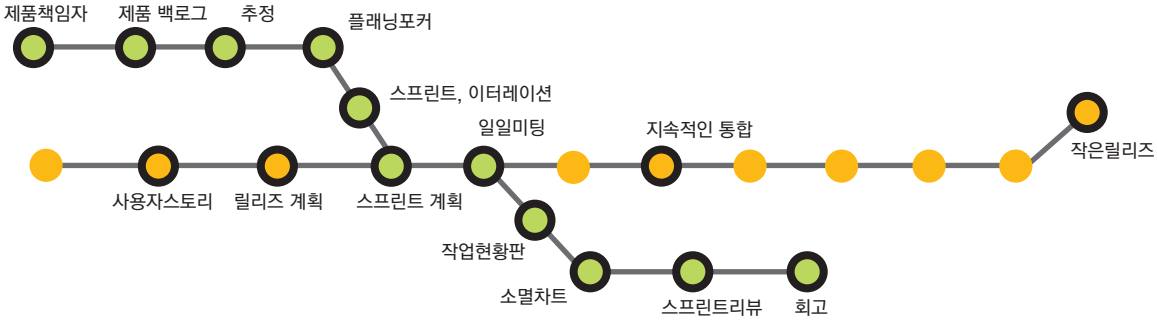


그림 3-8. 사례 3의 프랙티스 맵

스크럼을 기준으로 보면 애자일 팀에는 제품 책임자, 스크럼 마스터, 그리고 팀원이 있다. 일반적으로 팀원은 분석/설계/개발자, SA, 테스트 엔지니어 등으로 구성되는데, 본 사례에서는 제품 책임자와 테스트 엔지니어의 활약에 힘 입어 성공한 애자일 프로젝트를 소개하겠다.

프로젝트의 상황은?

애자일 방법론의 도입이 결정된 조직은 이미 출시되어 있는 제품의 추가 기능을 개발하면서 유지보수하는 팀이다. 개발팀과 운영팀이 분리되어 있음에도 불구하고 긴급 패치 진행 등 운영팀 지원 업무가 많아 늘 운영에 대한 부담을 가지고 있는 구조였다. 프로젝트가 진행된 이후 팀의 업무 비중을 분석해 본 결과, 개발이 70%, 운영이 30% 정도였는데 이러한 구조는 애자일 도입에 있어 잠재적인 이슈로 파악되었다.

프로젝트 수행 시 8 주를 반복적인 릴리즈 기간으로 정하였다. 이 기간 동안 3주씩 두 번의 스프린트를 수행했고, 2 주의 내/외부 QA(품질보증)기간을 거쳐 출시하는 리듬으로 제품 개발을 진행하였다.

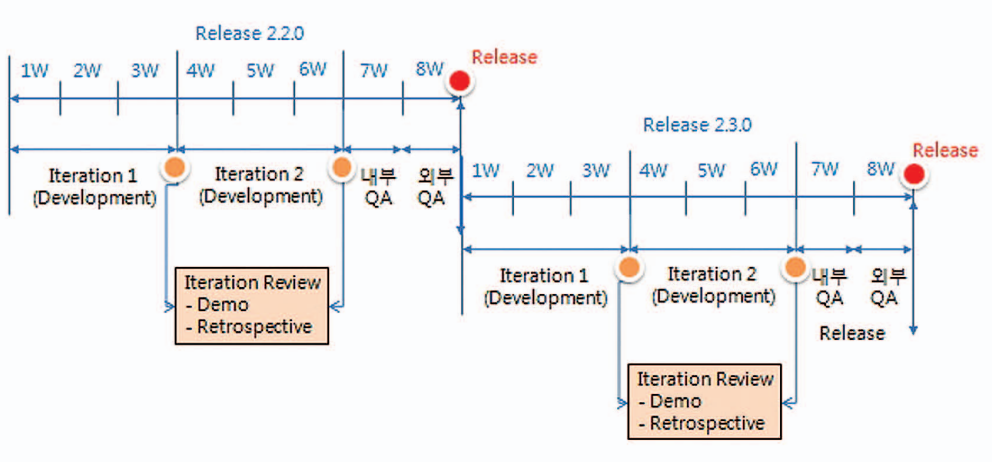


그림 3-9. 8주로 구성된 릴리즈 주기

스폰서십과 팀원 간의 미묘한 갈등

이번 프로젝트는 솔루션 개발을 위한 것으로, 애자일 도입을 위해 초기부터 팀 간 협의를 거쳐 선정되었다. 따라서 팀장 선에서는 공감과 이해를 통해 스폰서십을 받을 수 있었던 반면, 팀원들은 그에 대한 불신과 수동적인 자세를 가지고 있었다. 기존의 개발 방식에 익숙해져 있다보니 애자일이 낯설고 불편했지만 위에서 결정된 사안이므로 따라야 하는 입장이었다.

PM은 애자일 적용에 대해 그다지 적극적이지 않아 일단 관망하겠다는 입장이었고, 스크럼 마스터 역할을 하는 리더 역시 Command & Control 스타일(팀의 자율과 협동을 중시하는 스크럼과는 어울리지 않는)로 팀원들을 적극적으로 변화시키고자 하는 노력이 쉽지는 않았다.

그나마 다행인 것은 팀원 중 일부가 새로운 개발 방식을 도입하는데 흥미를 느끼고 있었다는 것이다. (어차피 팀에 정형화된 개발 방식이 없고, 임기응변적으로 개발하고 있었다. ;-)

제품 책임자의 참여를 이끌다

가장 먼저 시도한 것이 제품 책임자를 정해 팀에 적극적으로 참여시키는 일이었는데 시작부터 순탄치가 않았다. 애자일에 대한 정규 교육과 세미나 등을 통해 제품 책임자의 역할과 피드백이 프로젝트 성공에 필수임을 강조했지만, (서로의 상황에 대한 이해가 부족해서인지) 충분히 공감을 이끌어내지 못했고 피상적인 정보 전달에 그친 면이 있었다.

프로젝트 요구사항은 불특정 다수 고객을 대상으로 (주로 고객의 목소리(VoC : Voice of Customer)와 Help Desk를 통해) 수집되었다. 이러한 요구사항에 대해 비즈니스 중요도에 따라 우선순위를 부여하고 개발 계획을 구체화하기 위해 제품 책임자의 전념이 시급했다.

천만다행으로 릴리즈 계획을 진행하면서 제품 책임자가 스스로 본인 역할의 중요성을 깨닫고 적극적인 참여를 선언하였다. (이는 나름 성공적으로 마무리된 본 프로젝트에서는 중요한 전환점이었다.)

그 이후 제품 책임자는 일일 스크럼에 참석하겠다는 쉽지 않은 결정을 했다. 참석하지 못하는 경우에는 의사소통 도구로 사용했던 시스템을 통해 사용자 스토리에 대한 인수조건(Acceptance)에 해당하는 내용을 기술하여 개발자들과 지속적으로 대화했다.

그림 3-10. 관건은 사용자 스토리의 인수조건

개발자와 테스트 엔지니어는 (이처럼 적극적인 제품 책임자의 참여 덕분에) 사용자 스토리의 인수조건을 통해 구현된 모습을 명확히 파악한 다음 개발과 테스트를 수행하게 되었다. 이 부분은 나중에 스프린트 데모나 회고를 통해 가장 좋았던 점 중 하나로 언급되었다.

첫번째 회고를 진행할 때, 제품 책임자는 1년 이상 함께 개발 및 버전업을 하면서 개발팀의 개발자 이름과 얼굴을 잘 알게된 것은 이번이 처음이라고 하였다. 기존에도 리더급 개발자 등 일부 팀원들과는 이슈 관련 회의를 하곤 했지만, 비즈니스와 개발이라는 장벽을 가지고 자신의 입장을 내세우고 다소 공격적인 자세로 업무를 공유해 왔다고 고백했다.

애자일 도입 이후의 변화는 긍정적이었다. 사용자 스토리에 기초하여 개발자 및 테스트 엔지니어와 인수조건을 놓고 이야기하고, 개발에서 일어나는 여러 상황에 대해 의견을 나누는 과정에서 개발자와 개발팀의 입장에 대해 좀 더 이해할 수 있게 되었다.

팀을 완성하다

소규모 개발팀의 경우 개발자가 테스트까지 병행하는 경우도 많지만, 이번 프로젝트에서는 애자일 테스팅도 하나의 실천법으로 정립하고자 테스트 엔지니어를 별도로 팀에 배정하였다.

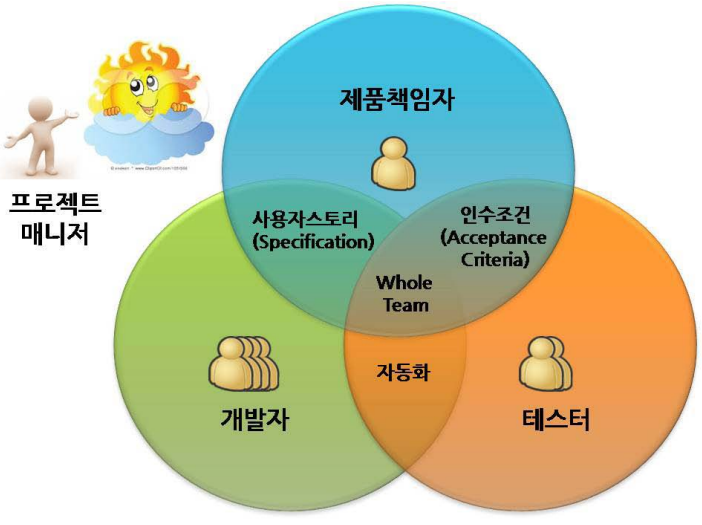


그림 3-11. 스크럼 팀과 애자일 테스팅

처음 PM에게 전담 테스트 엔지니어를 요청했을 때는 비용 상의 이유로 추가 인력투입이 어렵다는 답변이 돌아왔다. 그동안 진행해온 테스팅 방식과 차이가 있다는 것도 한 가지 이유였다. (기존의 방식은 개발 기간 중에 개발자가 테스트까지 수행하고, 이후 1 주일 간 자체 QA 팀에서 별도로 검증한 다음, 최종적으로 외부 QA팀에서 1 주일간의 테스트 수행을 통해 기준을 통과하면 릴리즈하는 것이었다.)

어려운 설득 과정을 통해 (한시적이기는 했지만) 사내 테스트 팀으로부터 테스트 엔지니어 한명을 팀에 합류시키기로 하였다. 테스트 팀의 문화가 개발팀과 달라 초기에 적응이 어려울까 우려했지만 결과적으로 테스트 엔지니어는 빠르게 개발팀에 융화되어 갔다.

제품 책임자가 그러했듯이 사용자 스토리라는 매개체를 통해 이야기를 할 시간과 기회가 많았던 것이 큰 도움이 되었다. 새로이 팀에 합류한 테스트 엔지니어는 테스트 시나리오에 기반한 화면 중심의 테스트 뿐 아니라 개발과 테스트 코드 작성이 가능한 인력이었다. (아마도 개발 능력이 있는 엔지니어였기 때문에 팀과 쉽게 융화되고 팀에 기여하는 속도가 빨랐던 것 같다.)

짝 프로그래밍 말고 짝 개발은 어떤가요

테스트 엔지니어는 스프린트 계획 회의, 일일 스크럼, 회고 등 개발자가 참여하는 활동에 늘 함께 참여했다. 일일 스크럼에서 테스트 내용을 팀과 공유하고, 이후 필요에 따라 개별 미팅을 통해 구체적인 사항을 함께 풀어 나갔다.

짧은 스프린트 주기 중에도 테스트 가능한 코드가 나오기까지는 일정한 시간이 필요하다. 이 기간 동안 테스트 엔지니어는 사용자 스토리에 대한 테스트 방법, 시기 등을 능동적으로 계획하여 팀과 공유하고, 제품 책임자와 함께 테스트 시나리오를 작성하였다.

모든 사용자 스토리에 적용하지는 못했지만 개발자가 코드를 매일 빌드하면 테스트 엔지니어가 개발된 코드를 공유하고, 개발자가 코드를 작성하는 동안 테스트 엔지니어는 테스트 코드를 작성하였다.

테스트 엔지니어와의 협업은 점차 테스트 주도 개발(TDD: Test-Driven Development)과 짝 프로그래밍(Pair Programming)의 혼합 유형으로 진화되었다. 즉, 개발자가 테스트 엔지니어의 테스트 코드를 받아 기본 기능을 구현하고, 개발자가 기능을 구현하는 동안 테스트 엔지니어는 다시 예외 상황 등을 고려한 테스트 코드를 추가하여 개발자가 기능을 완성하도록 유도하였다.

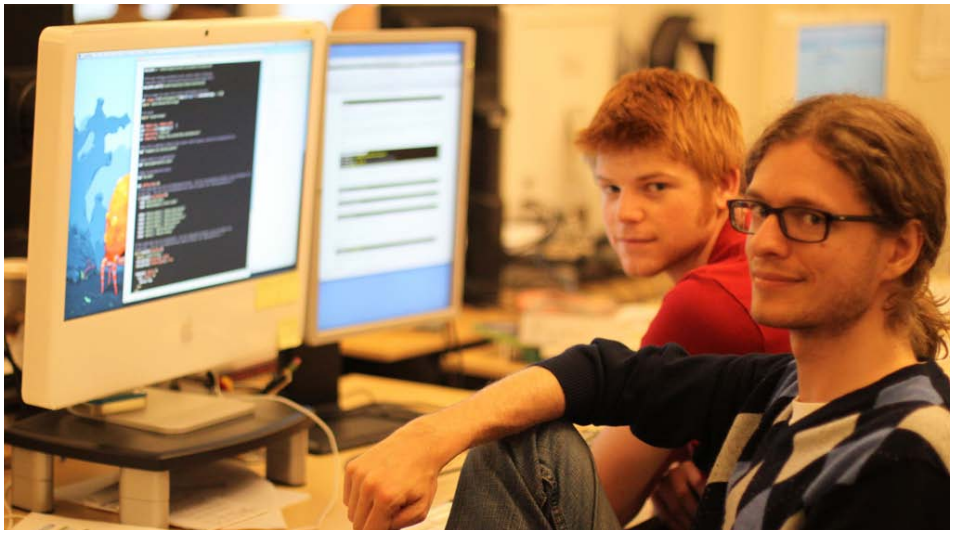


그림 3-12. 애자일 프로젝트를 성공으로 이끈 짝 개발

이렇듯 테스트 엔지니어가 테스트 코드 중 기본 흐름인 Positive Case를 작성한 것은, 개발자로 하여금 테스트 코드 작성을 습관화하기 위한 기본 틀을 마련해주고 추후 이를 토대로 보완하기 위함이었다. 궁극적으로는 개발자가 Positive Case를 기본 테스트 코드로 작성하고, 테스트 엔지니어가 제 3자 입장에서 Negative Case를 보강하는 방식이 더 적절하다고 판단된다.

테스트 엔지니어의 이러한 활동은 테스트가 스프린트 종반에 집중되는 소규모 폭포수(Mini Waterfall) 방식의 개발을 방지하는데 큰 도움이 되었다. 테스트 엔지니어의 조기 투입 및 활용을 위해, 비즈니스적으로 우선순위가 높은 기능부터 빨리 작동/확인 가능하도록 사용자 스토리의 작업을 분할했다. 또한 비즈니스적으로 중요한 기능을 우선적으로 작동시키기 위해 화면-비즈니스 로직-데이터 계층을 관통하는 방식으로 개발 계획을 수정해야 했다. (이와는 대조적으로, 애자일 도입 이전의 기존 방식은 개발 초기에 데이터베이스와 유틸리티, 라이브러리를 구축하느라 테스트 엔지니어가 실제 사용할 수 있는 기능이 거의 없었다. ;-)

사용자 스토리에 기초한 기능 테스트 시나리오는 테스트 엔지니어의 주도로 작성되었는데, 스프린트가 진행 됨에 따라 비즈니스 수준의 의미있는 통합 테스트 시나리오로 활용하게 되었다.

일반적으로 테스트 조직과 개발조직 간의 관계는 제품 책임자와 개발자의 관계보다 적대적인 면이 있다. 테스트는 결함을 많이 발견해야 하고 개발은 결함을 최소화해야 하는 상반된 역할을 가지고 있으니, 결함으로 인정하고 안 하고의 문제로 서로를 비난하기 십상인 탓이다. (일례로, 테스터 입장에서는 테스트를 하려고 하면 기본적으로 화면도 제대로 안 올라오는 상황이 비일비재하고, 개발 입장에서는 잡아내는 결함이 업무의 중요 로직이 아니라 사소하거나 발생 확률이 희박한 예외상황이라 무가치하다고 받아들이기도 한다.)

하지만 테스트 엔지니어의 투입과 병행된 애자일 도입 이후 반전이 일어났다. 프로젝트 회고 시 개발자들은 테스트가 (독립된 단계로 분리되어 있는 것이 아니라) 동일 스프린트 내에서 테스트 엔지니어와의 협업으로 수행 된 것이 매우 좋았다고 했다. 테스트 엔지니어가 매 스프린트 마다 결함을 발견하고 개발팀이 수정해 나가는 과정에서 팀으로 협업하고 서로 이해하는 분위기가 이루어졌기 때문이다. 덤으로 제품의 품질도 향상되었다. 이후 개발자와 테스트 엔지니어가 협업하는 이러한 방식을 “짝 개발(Pair Development)”이라 부르게 되었다.

대부분의 개발자는 테스트가 (부담스러운 것이 아니라) 본인의 코드를 검증해주어 개발에 도움이 된다며 테스트 엔지니어에게 고마움을 표했다. 이 부분은 앞서 언급한 제품 책임자의 경우와 일맥 상통하는 점이다. 함께 이야기할 수 있는 시간과 기회를 많이 가짐으로써 서로에게 도움이 되는 관계로 발전한 것이다.

짝 개발은 결과적으로 이번 프로젝트를 성공으로 이끌었는데, 릴리즈 이전에 수행하는 외부 품질인증 부서의 테스트에서 결함이 단 한 개도 발생하지 않는 이변을 낳았다. 처음에 비용 문제로 팀 내에 테스트 엔지니어 인력을 투입하기 어렵다고 했던 PM 은 이후부터 프로젝트 자체 비용으로 테스트 엔지니어를 고용하기 시작했다.

나름 많은 준비를 하고 스크럼과 XP 의 여러 좋은 실천법을 적용해 보았던 프로젝트였다. 이후 애자일이라는 이름으로 여러 프로젝트를 수행하면서 “애자일은 목적이 아니라 프로젝트가 잘 되고자 하는 목적을 달성하기 위한 수단”이라는 깨달음을 얻었다. 애자일을 적용하면서 애자일을 위한 프로젝트가 되고 있는건 아닌지 가끔 돌이켜 보기를 바란다.

정교하고 논리적이고 공학적인 실천법보다 소통의 경험을 통해 팀이 완성되고, 서로를 이해하고 이에 따라 변화된 모습을 가져오는 것이 애자일의 참된 가치가 아닐까?

풀지 못했던 숙제

제품 책임자 혹은 고객의 참여가 어려운 경우에는?
스크럼 마스터 역할을 맡은 리더가 Command & Control 인 경우에는?
팀에 개발과 유지보수성 업무가 혼재하는 경우는?
위에서 다룬 이상적인 테스트가 지속될 수 있을까?

사례 4

애자일이 스며들다

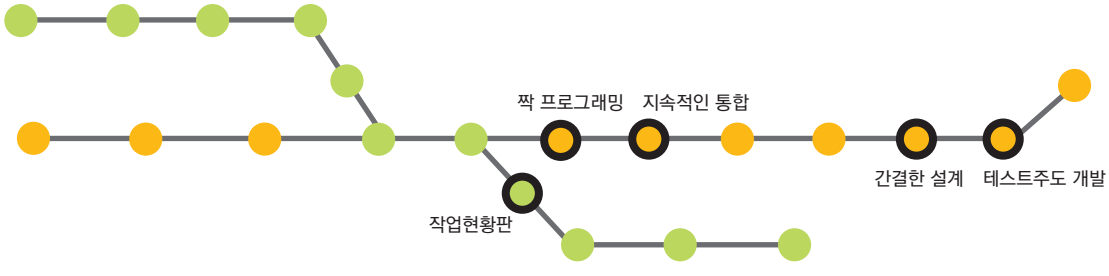


그림 3-13. 사례 4의 프랙티스 맵

고생스런 유지보수 경험을 하다

2005년 겨울, 신개발은 대형 SI 사업에 개발자로 참여했다. 그는 이 프로젝트에서 다른 149명의 프로젝트 참여 인력과 마찬가지로 13개월 동안 고되게 야근을 해야 했다. 사실 이번 프로젝트는 개인적으로 현재까지 회사 생활 중 가장 힘든 경험이였다.

우여곡절이 많았지만 다행스럽게도 납기를 지키고 시스템은 오픈 했다. 신개발은 내심, 가능한 빨리 이 프로젝트를 떠나고 싶었지만, 운영 PM으로 내정된 김피엠의 끈질긴 설득으로 유지보수에 남기로 했다.

그는 곧 후회하게 된다. 왜냐하면 유지보수 팀에서 예상보다 굉장히 많은 일을 떠맡게 되었기 때문이다. 물론, 개발 프로젝트와 달리 유지보수 프로젝트의 인력은 일반적으로 한 사람이 여러 가지 역할을 맡는다. 하지만, 당시를 회고하면 3년 차 엔지니어에게 맡겨진 일 치고는 많아도 너무 많았다.

엔지니어로서는 시스템 관리 업무 개발, 로그인/보안 등의 공통기능을 개발하는 일, 서버를 운영하는 일, 분배(Deploy)를 담당하는 일 등을 해야 했다. 게다가 관리자로서 PL을 맡아 고객과 신규 및 변경 요구사항에 대해 협상하는 최접점 역할을 수행했다. 그가 맡은 일은, 불과 6개월 전만해도 8명이 하던 일 이었다. 이를 모두 인수인계 받았을 때 신개발은 현기증이 나고 머리가 폭발할 지경이었다.

유지보수 프로젝트가 시작된 후, 세 달 가까이 야근과 주말 출근이 반복되었다. 이 때 그의 작업 강도는 개발 당시보다도 훨씬 센 것이었다. 당시 근무시간외 수당이 시간당 5,000원었는데, 신개발은 월급을 제외하고 이 수당으로만 한 달에 100만원 가까이 모을 수 있었다.

그는 이 시간을 견디기 힘들어 했다. 매일 오후 4시쯤에 화장실 거울을 통해, 스트레스 때문에 얼굴 전체가 붉어진 스스로를 발견할 수 있었다. 그는 항상 이 때쯤 세수를 하며 짜증과 열을 식히곤 했다. 일할 시간이 부족하여 점심을 거르는 경우도 종종 있었다.

애자일을 알리다

보통 사람은 스스로가 편안할 때 보다 힘든 시기에 현재 상태를 바꾸고자 하는 의지가 강해진다. 신개발도 당시에 그랬다. 이러한 상황이 지속되자 “이렇게 살다간 안되겠다 뭔가를 해야지” 라는 원초적(?) 의지가 점점 커졌다. 이 때 신개발 주변에 매우 좋은 자극을 준 선배가 있었는데 그 선배는 ‘애자일’이란 것의 신봉자였다. 그 선배를 통해 신개발은 제약이론(Theory of Constraint)의 창시자 골드랫 박사의 “The Goal” 이란 책을 읽을 수 있었다. 또한 그를 통해Cruise Control (후에 이 도구는 젠킨스(Jenkins) 라는 이름으로 바뀌게 된다) 이라는 지속적인 통합 툴을 배울 수 있었다.

신개발은 이 선배를 통해 현재 상황에 대해, 객관적으로 관찰하고 문제점이 없는지 고민하는 방법을 배울 수 있었고, 스스로 하는 일들을 훌륭하게 관리하는 방법을 깨우칠 수 있었다. 그리고 그 때부터 애자일에 대한 공부를 하기 시작했다.

유지보수 시작 후 3개월 정도가 지나자 시스템은 점차 안정화되어 갔다. 자연스레 신개발의 근무시간외 수당도 점점 줄어갔다. 고생할 당시는 매우 답답함을 느꼈으나 그 때의 경험은 (사람들과의 관계를 생각하면) 꽤나 값진 것이었다. 이 기간을 통해 그는 매우 중요한 자산을 얻었다. 그것은 ‘관리자와 고객의 무한 신뢰’ 였다. 그가 헌신한 만큼의 충분한 신뢰가 쌓인 것이었다. 이때부터 신개발이 하는 많은 것들이 수월해졌다. 관리자와 고객들은 그를 믿기에 그와 대화하는 것을 좋아했다. 하물며 엉뚱한 것을 제안하더라도 (반드시 무슨 이유가 있을 것이라 생각하고) 신개발의 이야기를 긍정적으로 들어주었다.

그 후, 신개발은 주변 사람들과 많은 대화를 시도했다. 그리고 특히 김피엠에게 그가 공부한 애자일이란 것이 무엇인지에 대해 설명하기 시작했다. 그 접근 방법은 매우 소극적이고 점진적인 접근법이었다. 그는 의견의 충돌보다는 다음과 같은 유연한 타협으로 대화를 시작했다.

신개발: “PM 님, 이 책 한번 읽어보실래요? 미국에는 이런 애자일이란게 있데요. 근데 참 재미있어요. 짝 프로그래밍이라고, 둘이서 한 키보드를 두고 같이 개발을 한데요. 하하하. PM 님이 들으면 기가 막힐 이야기죠.”
김피엠: “그래? 하하 그거 재밌네. 근데 그렇게 프로젝트를 하면 돈이 남을까?”
신개발: “에이.. 물건너 얘긴데요 뭘. 근데, 미국은 여러 가지 상황이 좋으니까 남긴 많이 남는데요. 프로젝트 상황 괜찮아지면, 우리도 한번 해볼까요?”
김피엠: “하하, 신입 사원 들어오면 한번 해봐라. 그렇게 하면 얘가 일을 얼마나 잘하는 앤지 몇 시간 안에 알 수 있을 것 같은데?”
신개발: “와! 좋은 생각이세요. 그럼, 신입이 들어오면 제게 맡겨주세요. 한번 교육을 시켜 볼께요.”

신개발은 2개월 후 들어온 신입사원 3명에게, 실험군과 대조군을 만들어 짝 프로그래밍을 시켰다. 그리고 그 실험결과를 리포트로 만들어 김피엠에게 주었다. 이 리포트를 통해 김피엠은 개발에 전혀 관심이 없던 신입사원이 어떻게 변화하는지를 관찰하며 매우 즐거워했다. 김피엠이 짝 프로그래밍을 보는 시선은 자연스레 달라졌다. 짝 프로그래밍은 그에게 더 이상 공수를 갉아먹는 말도 안 되는 방식이 아니라, 교육을 시키며 관리를 하는 매력적인 도구로 비쳐졌다.

무모한 실험

신입사원을 대상으로 한 실험이 성공적으로 끝나자, 신개발은 더 많은 욕심이 생겼다. 무엇이든 더 할 수 있을 거라는 자신감으로 가슴이 가득 찼다. 하지만 남들에게 무엇인가 함께 하자는 제안을 할 용기는 아직 없었다. 유지보수팀의 구성원 대부분이 그보다 많은 사회 경험을 가지고 있었으므로, 이러한 선배들에게 새로운 것에 대한 제안을 하는 것은 매우 어려운 일이었다. 그래서 먼저 혼자서 할 수 있는 것을 찾아보기 시작했다.

‘리팩토링’ 이란 것이 눈에 들어왔다.

"리팩토링은 [기능은 그대로이지만 코드의 디자인을 개선하는 것]이다. 이것은 내 코드에 나 혼자서도 할 수 있는 것이다. 누구에게도 이야기할 필요 없이…"

이때부터 그는 리팩토링을 수행할 사냥감을 찾기 시작했고, 가장 먼저 눈에 들어온 소스는 로그인 부분이었다. 당시 로그인 소스는 말 그대로 스파게티 코드였다. 최초의 코드는 단지 한 개의 클래스로 이루어져 있었다. 라인 수도 500줄 정도라 가독성도 그리 나쁘지 않았다. 하지만, 여러 업체가 로그인에 보안 및 암호화 솔루션을 추가로 심으면서 코드는 점점 변질되어갔다. 게다가 고객들의 빗발친 요구사항으로 인해 해당 솔루션은 규칙 없이 여러 조각의 긴 클래스들로 커스터마이징되었다. 결과적으로, 코드 덩어리는 빗물에 쓸려가는 찢어진 신문지 같이 지저분해졌다.

신개발은 이 코드를 리팩토링하기로 했다. 클래스를 분할하고 10 줄 이상되는 메소드는 추출하여 적절한 책임에 맞는 클래스에 매핑시켰다. 1,000 줄 정도 되는 코드는 클래스당 100 줄 정도씩으로 단순화 되었다. 그는 매우 기뻤다. 그는 그 코드가 그의 노력의 산물이고 결국 시스템에 훨씬 유익한 일이라 확신했다. 다만, 그는 아무에게도 그 코드를 리팩토링 했다는 이야기를 하지 않았다. 사실 할 필요가 없었다. "누가 이 엄청난 작업을 이해하겠는가?" 라고 오만을 떨며 즐거워했다. 그리고 1 달간 리팩토링한 코드를 배포했을 때, 그는 엔지니어로서의 짜릿함을 느꼈다.

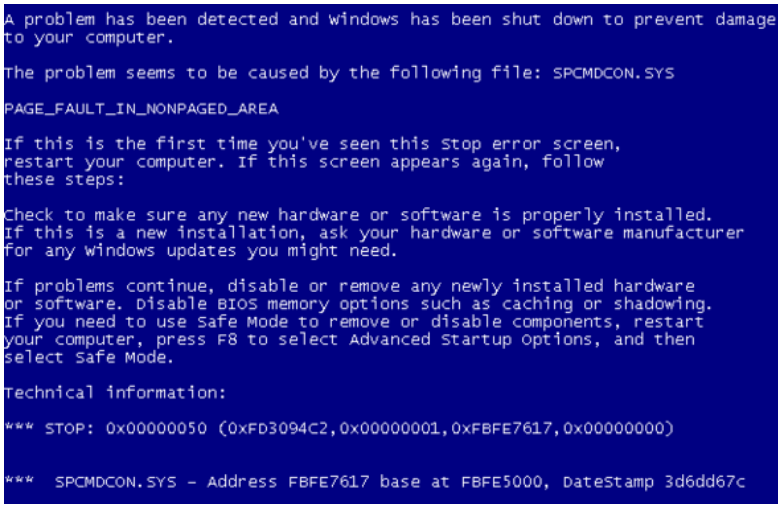


그림 3-14. 대형사고를 치다

하지만, 그 짜릿함은 오래가지 않았다. 새벽 1시에 분배된 내용은 다음날 아침 엄청난 후 폭풍으로 밀려왔다. 35만명의 사용자가 4시간 동안 로그인을 하지 못했다. 신문에 날 사고를 친 것이었다. 고객 팀장이 사무실로 부랴부랴 와서 처음에는 화를 내다가 몇 시간 후 부터는 살살 신개발을 달래기 시작했다. 고객 팀장의 얼굴은 새파랗게 질려 있었고 그가 식은땀을 흘리며 다음과 같이 이야기 했다.

> 저기.. 복구는 가능한거지?
> 응?
> 되는거지?

신개발은 이해가 되지 않았다. 개발환경에서 분명히 완벽하게 돌던 소스였다. 무엇인가 이상했다. 그리고 억울했다. 그는 4시간에 거쳐 이를 겨우 복구해냈다. 김피엠은 모든 상황이 종료된 후 다음과 같이 이야기 했다.

> 네가 가진 신뢰가 이번 한번에 완전히 무너졌다

김피엠은 이 일의 후속 조치로, 다시는 이 같은 일이 발생하지 않을 것이고, 만약 발생한다면 책임지겠다는 각서까지 썼다.

신개발은 끝까지 리팩토링을 하여 소스코드에 문제가 생긴 것이라는 걸 누구에게도 말하지 않았다. 그냥 예전에 고쳤어야 할 기존 코드의 동작이 결함으로 올라와 사고가 난 것이라고 이야기 했다. 그는 도저히 그 모든게 그가 한 일이라고 말할 수 없었다.

이러한 문제가 생긴 이유는 무엇일까? 사실 이유는 단순했다. 그는 당시 거의 자살행위나 다름없는 방식으로 소스를 분배한 것이다. 신개발은 테스트 코드 없이 리팩토링을 했고, 결과적으로 매소드 하나하나가 잘 동작하는지 전혀 검증하지 않았다.

이 실험을 통해 그는 많은 교훈을 얻었다. 이후 신개발은 비슷한 방식으로 리팩토링을 혼자 시도해보려는 엔지니어들을 보면 “절대 생각조차 하지말라” 고 이야기하기 시작했다.

유지보수에 사용해 본 칸반(Kanban)

수 개월이 지나 이 전의 사고가 잊혀질 무렵 이 유지보수 팀은 착한 팀이 되어 있었다. 스스로 맡은 바 책임을 다하는 것은 물론, 다른 팀원이 힘들 때는 기꺼이 도와주기도 했다. 하지만 여전히 고질적인 유지보수 조직의 문제를 어느정도 가지고 있었다.

유지보수 조직은 보통 업무 별로 인력이 할당되기 때문에, 시기에 따라 바쁜 사람과 바쁘지 않은 사람이 갈리게 된다. 이를 효과적으로 구분하고 바쁜 사람의 부담을 서로 분담하는 체계가 필요했다. 이를 위해 신개발은 "큐(Queue)제어 방식인 칸반을 도입하자" 라는 제안을 했다. 팀원 모두 이 문제에 대해 공감하고 있었기 때문에 쉽게 동의했다.

먼저, 눈으로 보이는 즐거움을 주기 위해 예쁘게 코르크 보드를 한쪽 벽면에 크게 붙였다. 색 테이프를 이용하여 구간을 표시했다. 그리고 김피엠에게는 우리가 일을 더 잘해보려고 이런 보드를 쓰려고 하는 것이니 특별히 신경쓰지 말아달라고 부탁했다. 김피엠이 누구를 억압하거나 강요하는 스타일은 아니었지만 최대한 팀이 자발적으로 선택하여 적용한다는 느낌을 주고 싶었다. 일단 보드를 붙이는 것만으로도 김피엠과 고객은 좋아했다. 전시효과가 있었다. 이 보드는 정적인 유지보수 조직에 무엇인가 돌아가고 있다는 모습을 효과적으로 보여줄 수 있었다.

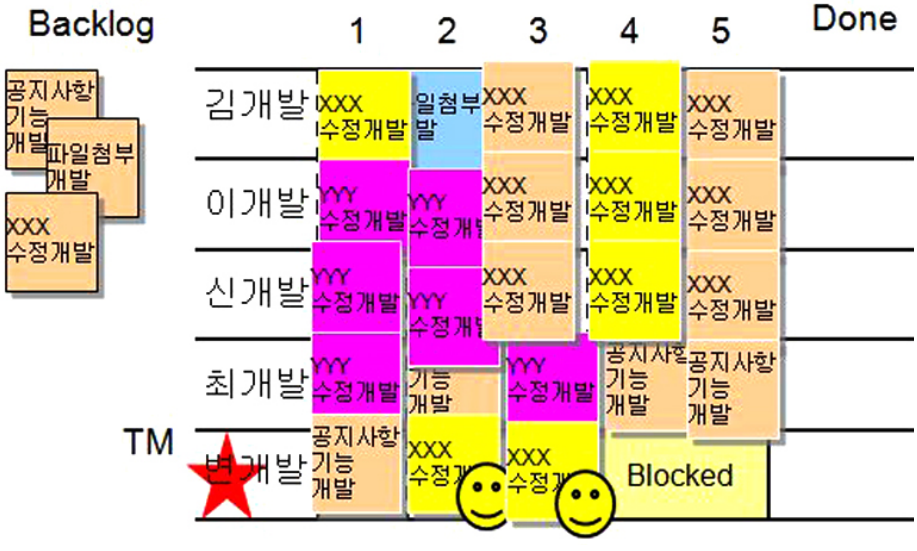
그림 3-15. 칸반보드로 업무 현황을 가시화하다

코르크 보드에 유지보수 팀원의 이름을 각각 넣고, 한 명 당 5개의 슬롯을 주었다. 그리고, 고객의 요구사항이 시스템으로 들어오면 포스트 잇에 해당 요구사항의 내용과 번호를 간단히 적고 백로그 부분에 올려 놓았다.

그리고, 개인당 가진 슬롯의 개수까지만 포스트잇으로 적힌 일을 가져가고, 이것이 넘치는 경우, 다른 사람이 자신의 업무가 아니더라도 도울 수 있도록 했다. 모든 사람의 슬롯이 다 찬 경우에는 김피엠에게 현재 프로젝트가 감당할 수 없는 정도의 일이 들어오고 있다고 이야기 했다. 김피엠은 이 상황에 대해 매우 현실적인 의사결정들을 했다. “8 시까지 근무하자. 대신에 저녁밥은 프로젝트가 펑크가 나더라도 내가 책임진다”. 또는 고객에게 가서 “우리 정말 너무 힘들다. (칸반 보드를 보여주며) 저거 봐라 애들 죽는다. 웬만한 요구사항은 우리에게 넘기지 말고 너희가 좀 알아서 처리해줘라” 라는 식으로 프로젝트의 일을 줄이기 위해 노력했다.

그림 3-16. 칸반보드: 모든 개발자의 슬롯이 가득찬 경우

한 달 정도 이 보드를 사용한 후, 일상적으로 아침마다 커피를 함께 마시는 자리에서 팀 내 누군가가 테스트를 너무 개인의 역량에 의존하는 것 같다는 이야기를 꺼냈다. 그들은 즉시 어떻게 하면, 효과적으로 테스트를 할 수 있을까를 이야기했고, 그들 스스로 '테스트 매니저' 라는 역할자를 만들어 냈다. 당시 화요일과 목요일에 정기 분배 작업이 있었는데, 이 전에 분배할 모든 기능에 대해 담당 테스트 매니저가 한 번씩 제 3 자의 눈으로 확인하기로 했다. 매주 돌아가며 '테스트 매니저'의 역할을 담당했는데, 이때는 두 개의 슬롯을 줄여 주는 방법으로 테스트로 인한 공수를 분산시켰다.



I 그림 3-17. 칸반보드: 테스트 매니저(TM)에게는 적은 슬롯이 주어짐

그 뒷 이야기

신개발은 이후에도 다양한 여러 애자일 기법을 프로젝트에 적절히 녹이는 노력을 했다. 굳이 모든 실천법을 한번에 적용할 필요는 없었다. 조직이 필요로 하는 적절한 시점에 한 가지씩 꺼내어 해보자고 제안했다. 사실 애자일이란 말을 할 필요 조차 없었다. 그저, 팀을 개선하기 위해 여러가지 시도를 하고 지속적인 개선을 한 것이었다.

그림 3-18. 유지보수팀의 회고

대신, 팀 전체가 현재의 문제를 공감하는 상황을 만들어 내고 개선하도록 대화하는 자리를 마련했다. 이런 식으로 3년이 지났다. 이 유지보수 조직은 좋은 습관이 계속 쌓여 지속적으로 발전하는 팀이 되었고, 김피엠은 애자일에 대해 점점 더 큰 관심을 갖게 되었다. 그리고 “익스트림 프로그래밍”, “스크럼”, “불확정성과 화해하는 프로젝트의 추정” 등을 읽으며 김피엠은 사무실에서 여러 차례 혼잣말을 했다.

야, 이거 완전 우리 얘기잖아!!!

그 뒤로 김피엠은 스스로를 애자일 PM이라고 소개하기 시작했으며, 회사 내 애자일 전도사가 되었다.

사례 5

애자일 드러내놓고 해볼까?

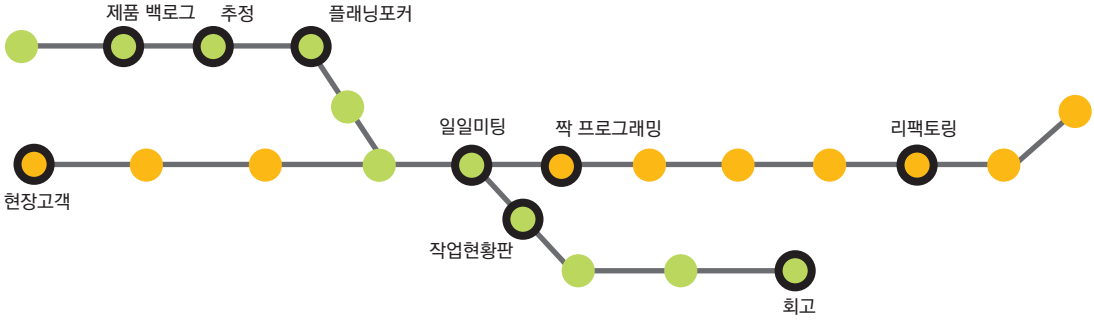


그림 3-19. 사례 5의 프랙티스 맵

"스마트폰" 개발, 발 등에 불이 떨어지다

2009년 말부터 2010년 초 IT시장의 화두는 "스마트폰" 이었다. Android 플랫폼 기반의 개발 경험이 없던 국내 기업은 앞다투어 앱 개발에 필요한 Java 개발자 확보 전쟁에 나섰다. 본 사례에서는 SI(시스템통합) 업체로서 고객사 스마트폰 앱 개발에 투입되어 애자일 기반의 Android 개발 프로젝트를 성공적으로 수행한 경험을 소개하겠다.

연구 과제에 "애자일" 을 녹이다

당시 사내 애자일 확산 및 코칭 업무를 수행 중이던 강 코치는 Android SW 개발체계 구축 과제에 투입되었다. 과제 기간은 2010년 5월부터 12월 까지였고 개발 프로세스를 Android 개발에 적용하면서 체계를 완성하는 전략을 가져갔다.

과제 목표 세 가지 가운데 강 코치는 “Android 환경에 적용 가능한 품질체계” 구축 과제 담당이었고, 그 중에서도 개발 프로세스 체계를 수립하여 적용하는데 주력하였다. 그는 고객사의 기존 제품 개발 프로세스에 애자일 프로세스를 하이브리드하게 접목할 수 있는 방향을 고민 하기 시작했다.

고객사의 스마트폰 개발 조직은 스마트폰 모델 아래 앱 별로 팀이 구성되어 있었다. 팀 구성은 일반적으로 고객사 연구원이 PM 을 맡고 팀원으로 고객사 인원을 포함하여 SI 업체, 외주 협력사 등이 한 팀으로 적게는 2~3명 에서 20명 전후가 대부분이었다.

프로젝트는 2010년 5월부터 시작될 예정이었지만 한 달 이른 4월부터 팀원들이 투입되었다. 현황 파악을 위한 인터뷰를 통해 개선 포인트를 찾고 동시에 애자일을 활용할 수 있는 부분을 염두에 두며 과제를 수행하였다. 더불어 고객사, SI 업체, 외주 협력사를 대상으로 애자일 교육을 수행하였다.

애자일을 제안했지만 외면 받다

5월에 강 코치는 애자일에 기반한 SW개발 프로세스를 만들어 고객사 PM 에게 설명하고 내심 자원하는 팀이 나오길 바랬다. 그런데 새로운 시도(“애자일”)에 대한 부담 때문인지 단 한 팀도 먼저 나서려 하지 않았다. 팀장들의 한결같은 피드백은 “지금은 바빠서 힘들구요. 여유가 생기면 그 때 다시 이야기 하시죠…”

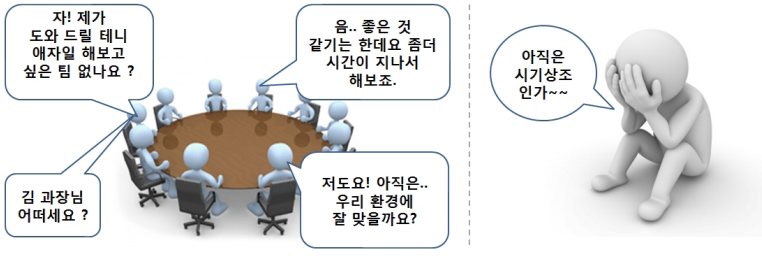


그림 3-20. 외면받았던 부끄러운 과거

숨어있는 스크럼 팀을 발견하다

"아.. 아직은 시기상조인가.." 하며 좌절하고 있던 때였다. 고객사 사무실을 둘러보니 매일 아침 3명이 조그만 판을 들고 회의실로 매일 약 20 분 정도 사라졌다 나타나기를 반복하는 것을 발견하였다. 궁금하여 무엇을 하는 것인지 슬며시 물어보았다. 그러자 담당 PL 은 매일 스크럼 미팅이라는 것을 하는데, 옆에 있는 고객과 다른 사람들에게 방해될 것 같아 회의실로 작업 현황판을 들고가서 하는 것이라고 설명해 주었다. 이 말을 듣고 뛸 듯이 기뻐 제안을 하게 되었다.

> 사실 저는 애자일 코치입니다. 스크럼을 하신다니 놀랍네요! 이렇게 숨어서 할 것이 아니라 드러내놓고 해보시지요. 제가 적극 도와드리겠습니다. 그리고 PM 에게 애자일을 하는 것에 대해 제안하고 설득해보겠습니다.

고객사 PM의 열린 생각

다음날 강 코치는 고객사 PM인 장 책임과 이야기를 나누었다. 새로운 방식을 적용하는 것에 대해 처음에는 경계하는듯 했지만, 충분한 설명과 대화를 통해 동의를 얻었다. 잘 되면 다른 팀으로 확산하고 싶다는 긍정적인 의지도 내비쳤다. 출발이 좋았다.

그림 3-21. PM 의 지지를 얻은 애자일 코치

장 책임이 담당하는 앱은 "Android SNS (소셜 네트워크 서비스: Twitter 나 Facebook 앱)" 이다. 이 팀은 3개 파트로 나누어져 있고, 전체 팀원은 20여명이었다. 이 가운데 스크럼 미팅을 조용히 하고 있던 3 명도 포함되어 있었다.

작업 현황판 앞에서 스크럼 미팅을 하다

그림 3-22. 작업 현황판과 일일 스크럼 미팅

일일 스크럼 미팅은 통상 길어지는 것을 방지하기 위해 서서 하라고 권고하고 있다. 하지만 이 팀의 경우는 앉아서 하면서도 회의가 늘어지지 않았으며 특정인이 발언을 독점하지 않고 비교적 훌륭하게 진행되었다.

고객과 함께 한 회고 미팅

강 코치는 SNS 앱 팀의 애자일 적용을 지원하면서 특히 일일 스크럼과 회고 만큼은 지속적으로 실천할 수 있도록 도와주고 싶었다.

1 Retrospectives Info

Stage	회고일자	장소	진행자	참석자
	2010.05.06(목) 17:00~18:00			

2 Retrospectives 내용

What was Good?

1. 현재 Issue에 대하여 한눈에 진행사항을 확인할 수 있었고, 함께 대응하고 해결사례를 전파 및 공유 했던점. - 4
2. 각 담당자의 업무Load 확인이 가능하여 업무 배분시 참고될 수 있었고 Scrum Master의 의해 업무 조정이 이루어짐.
3. Dash-board 및 Post-it 활용한(Memo) 업무관리(시간관리가 잘이루어지고, Backlog update 잘이루어짐 - 7
4. Pair Programming 및 Cross check 수행 이
(Done 기준 : Cross Check 완료후) - 2
5. Intercasting 소스 활용
→ 역으로 Intercasting 소스 Inspection

Any Bright Ideas?

1. 집중근무 시간을 정했으면(오전 10~11, 오후
2. Issue 해결 정리 필요(Mantis 활용)
3. 휴일 채택근무 대처
4. Post-it에 추가정보 관리(예정 Working-day/실제
5. Sanity Test 자동화 - 4
6. Stress Test 자동화

3 실천방안 - S.M.A.R.T

실천항목	실천내용	비고
Scrum Meeting 시간 Ground-rule 정의	• 9:00 에 못하면 9:30 에 시작하자.(그외 Scrum Master가 판단) • 부재중인 인원이 있어도 수행하자.(부재 인력엔 후 공유) • Scrum Meeting 시간에 되면 Dash-board 앞으로 자연 스럽게 모이자	오전에 못할 상황이면 오후에 할 수 도 있음
Dash-board 와 Backlog 관리 및 동기화 쉽게...	• Backlog 항목 설명 및 필수/옵션 정리하여 제공 • Backlog 작성기이드 Sheet 추가하여 제공	
Issue 해결 정리	• Mantis 를 활용하여 공유 • Issue 관리대장 문서 아니라 실제 Working-day 함께 표시하자	개발리더가 판단하여 활용
Post-it 추가 정보 관리	• Working-day 뿐만 아니라 Backlog 에도 Update 하여 유지함 • Backlog 에도 Update 하여 유지함	표시방법 : 3/2 (3월예상, 2일만에 완료) 4
Sanity Test 자동화	• Sanity Test를 Robotium 을 이용한 Black-box 테스트 자동화 수행 가능	테스트 전문화 Task 에서 진행중
Stress Test 자동화	• Application 에 부하를 주는 Stress Test 가능 여부에대한 기술 검토 • 결과에 대한 Feedback 예정	테스트 전문화 담당자에게 전달

기타 • 정리 : 개인의 의견이 비슷한경우 그룹핑 하여 정리함
• 투표 : 공감하는 내용에 개인별 총5개의 스티커로 점투 (표 숫자)

그림 3-23. 회고와 회고록

그래서 PM 을 고객으로 참여시켜 회고를 진행하였고 회고록(실천사항 포함)을 작성하여 참여자 모두와 공유하였다. 이는 매월 지속되었다. PM은 이 3 명의 팀원이 수행하는 방식을 마음에 들어했다. 진행 상황도 잘 보였고 회고를 통해 개선하고자 하는 의지와 노력을 높이 샀다.

PM이 애자일 확산에 동의하다

강 코치는 나머지 파트까지 애자일을 확산해 보자는 제의와 함께 장 책임에게도 지원사격을 부탁했다. 강 코치는 파트 리더 3 명과 논의를 통해 애자일 적용 방안을 만들어 공유하고 연말까지 스크럼 미팅과 회고를 지속하기로 했다.

'스크럼의 스크럼' 을 도입하다

스크럼 팀으로 협업하기에 가장 적합한 인원은 5~9 명이다. 따라서 20명이 넘는 전체 팀을 업무 관련성이 높은 각 파트 별로 나누어 진행하기로 했다. 전체 팀이 함께 모여 일일 스크럼 미팅을 할 장소도 마땅치 않았고, 한 사람씩 이야기를 하다보면 미팅이 길어져 지루해지는 문제도 발생할 수 있어 내린 결정이었다.

결국 파트 별로 현황판을 개발/설치하여 시간 차이를 두고 각각 일일 스크럼 미팅을 수행하였다. 사실 파트 별로 일일 스크럼 미팅 시간을 나눈 이유는 순전히 각 파트가 익숙해질 때까지 강 코치가 스크럼 미팅에 참석하여 피드백을 주기 위해서였다. 모든 파트가 일일 스크럼을 마치고 나면 각 파트의 스크럼 마스터가 참여하는 '스크럼의 스크럼(Scrum of Scrums)'을 수행하였다.

그림 3-24. 파트 별 현황판과 일일 스크럼 미팅

팀을 남겨두고 복귀하다

이처럼 3명으로 시작된 스크럼 팀에 SNS 앱팀 전원이 조인하게 되었고, 일일 스크럼과 회고 뿐이었지만 애자일 실천법 적용을 유지할 수 있었다. 팀이 스스로 애자일을 지속하는 것을 보며 기쁨을 느낄 무렵 강 코치는 본사에 다른 업무가 생겨 복귀하게 되었다.

그 후 강 코치는 매월 회고 지원을 위해 이 고객사를 방문하였고 중간중간 온라인으로 소식을 전해 들으며 진행상황을 지속적으로 공유하였다.

자기조직화(self-organizing)팀이 되다

이렇게 5월부터 12월까지 지원을 하면서 팀에 어떤 실천과 변화가 있었는지 돌이켜 보았다.

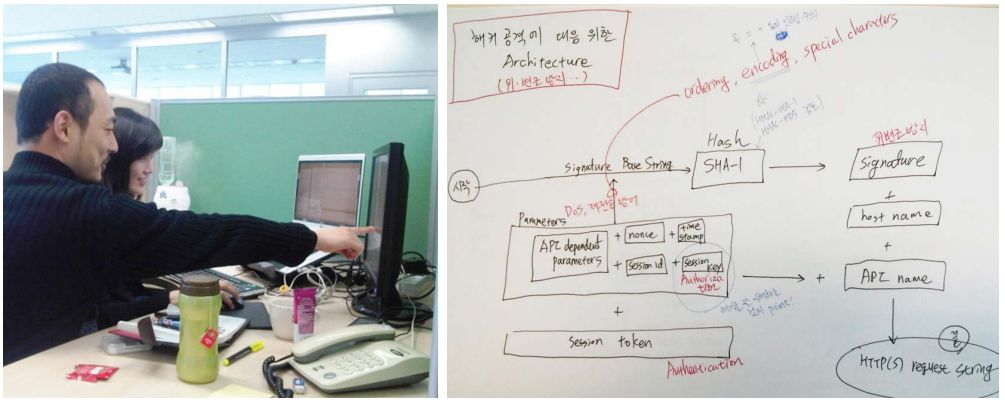
그림 3-25. 회고를 통해 조금씩 발전한 팀의 모습

처음은 “일일 스크럼” 과 “회고” 로 시작하였던 팀이었다. 일일 스크럼 미팅을 통해 팀은 서로가 어떤 상태인지 어떤 문제에 직면해 있는지를 알게 되었다. 간혹 일일 스크럼 미팅이 PM 을 위한 업무보고 스타일이 되거나, PM 이 참여하여 업무를 지시하는 형태로 진행될 때도 있었다. 그러면 회고 때 그런 것들이 드러나게 되고 강 코치는 팀이 초심을 잃지 않도록 독려하였다.

회고 활동을 지속하면서 팀은 2 주마다 자신을 돌아보며 실수했던 점과 잘 한 점을 공유하며 실수를 줄이기 위해 노력하는 모습을 보였다. 문제 해결을 위해 강 코치는 새로운 애자일 실천법을 시도해보도록 권하거나 다양한 제안을 하곤 했다.

그렇게 적용한 애자일 실천법으로 “짝 프로그래밍”, “리팩토링”, “애자일 추정”, “사용자 스토리”, “짝 디버깅” 등이 있다. 이들은 모두 회고 때 제안되어 팀의 동의 하에 팀 실천법으로 채택되었다.

더불어 “1 장짜리 설계”, “Bug Find Day” 등의 새로운 시도도 있었다. 누군가 위에서 시켜서 하는 게 아니라 팀이 공감하고 합의를 이끌어 내는 모습, 이것은 애자일에서 말하는 자기조직화 팀의 모습이었다.



| 그림 3-26. 짝 프로그래밍과 1장짜리 설계

10번의 회고에서 매번 간단한 설문조사를 실시한 결과, ‘애자일을 다른 사람들에게 추천할 의사가 있는가’ 라는 질문에 80% 가 추천할 의사가 있다고 답변했다. 짝 프로그래밍에 대해서도 ‘업무가 즐거웠는가’ 라는 질문에 75% 이상이 즐거웠다고 답하였으며, ‘많이 배운 것 같은가’ 라는 질문에도 82% 이상이 많이 배웠다고 응답 하였다.

더불어, 회고에서 나온 실천사항이 채택된 경우는 49건, 이 가운데 정말로 실천한 경우는 30건으로 실천율이 61% 정도 되었다.

팀 아이디어: Bug Find Day 이벤트

회고 중에 나온 아이디어인 “매주 1회 Bug Find Day” 가 채택되어 6월부터 매주 실시하였는데 그 효과는 놀라웠다. (매주 금요일 오후 2:00 ~ 4:00)

항목	기존 테스트	Bug Find Day
테스트 시간	0.5시간 x 5일 x 12명 = 30시간	2시간 x 1일 x 12명 = 24시간
결함 건수	30개	70개
효율성	시간당 1개	시간당 2.9개
효과성	-	테스트 커버리지 94% Major Defect 발견율 증가 팀 시너지 발휘 UI시나리오 학습 효과

2~3 명 단위의 6 개 팀으로 나누어 실시간으로 결함을 찾는 경진대회 처럼 운영하였다. ‘누가 더 결함을 많이 찾는가’ 를 겨루는 방식이었는데 상품권을 걸고 진행해서인지 모두가 열심이었다.

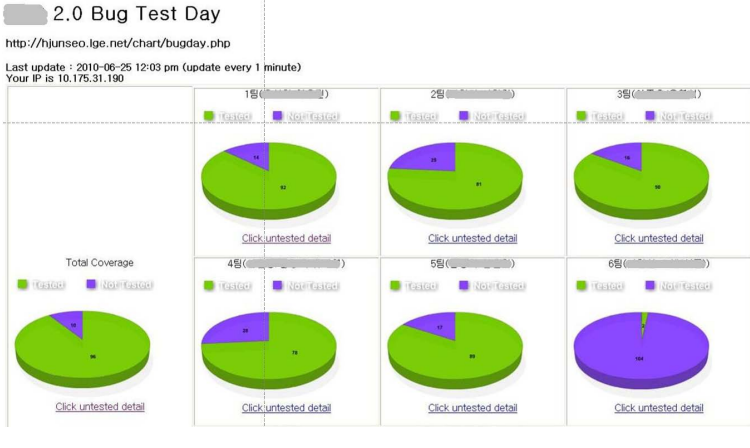


그림 3-27. 실시간으로 찾은 결함에 대한 관리 화면

고객사 PM인 장 책임의 피드백

개발 당시 팀원 간에 기능과 역할이 너무 세분화되어 있었고 프로젝트 초기에 개발자간 개발 능력에도 많은 차이가 있었습니다. 그것을 메울 수 있는 방안이 필요했는데 애자일의 스크럼 미팅, 회고, 짝 프로그래밍과 같은 협업 방식이 유용했던 것 같습니다.

사실 휴대폰의 개발 특성상 일정이 매우 촉박하여 새로운 프로세스 도입에 우려가 있었습니다. 그런데 저희 내부 개발자들의 적극적인 참여 의지가 있어 우려 반 기대 반으로 시작하게 된 것이죠.

각 파트별 일일 스크럼 미팅을 통해 이슈 사항에 대해 공유하고 대응 방안을 서로 모색하는 시간이 되었습니다. 작업 현황판을 활용하여 개인의 이슈 사항에 대한 진척도를 공유 함으로써 투명하게 이슈가 관리되고, 실제 처리 기간을 줄이는데도 도움이 되었습니다.

회고 미팅을 통해 스스로 주간 세미나와 Bug Find Day, 주말 대응 프로세스와 같은 좋은 프로세스 개선 과제를 도출 하였습니다. 또한 여러 참신한 아이디어 덕분에 팀 분위기 개선에도 많은 도움이 되었습니다. 짝 프로그래밍 도입으로 고참자의 지도 하에 신규 인력의 개발 실력이 빠른 시일에 향상되어, 업무 전환을 할 때에도 신속하게 백업이 이루어질 수 있었습니다.

다음 프로젝트에도 이번에 도출된 개선 사항이 지속적으로 반영될 수 있도록 관리가 필요할 것 같아요. 매월 진행되는 회고에서 만족도가 낮은 팀원과 실행률이 낮은 원인에 대하여 문제점을 도출하고, 꾸준한 개선 방안을 모색한다면 팀이 점진적으로 더 나아질거라는 확신이 듭니다.

출시 일정에 무리가 가지 않는 선에서 다른 조직에도 점차 확산되어 하나의 문화로 자리 잡기를 기대해 봅니다.

성공, 그 뒷 이야기

대부분 “모바일, 스마트폰, 임베디드, 안드로이드 플랫폼, 높은 수준의 품질, 쉽지 않은 테스트 등…” 이런 환경에 애자일을 적용하는 것은 무리이고 힘들다고 예상한다. 하지만 이런 환경에서도 누군가는 애자일의 가치와 장점을 보고 몸소 실천하고 있었으며 그것이 작은 씨앗이 되었다. PM인 장 책임 역시 어렵다고 단정짓지 않고 한번 해보자는 열린 마음이 있었기에 애자일 도입을 조직의 성과로 발전시킬 수 있었다.

짝 프로그래밍은 자연스런 업무 백업 체계가 되어 야근이 많았던 상황을 개선하였고, 주말 대응도 돌아가며 할 수 있게 되었다. 이처럼 주변의 다른 팀보다 야근 및 주말 대응에 유연하게 대처하는 팀이 된 것도 중요한 성과로 평가되었다.

04

APPENDIX

애자일 적용시 빠지기 쉬운 함정

애자일 회고

FAQ

용어사전

애자일 적용 시 빠지기 쉬운 함정

1 관리도구로서의 애자일 (무늬만 애자일)

애자일 도입이 (의사소통과 협업을 증가시키기 위함이 아니라) 일정 중심의 반복개발을 통해 관리자가 가시적인 진척도를 정기적으로 확인하기 위한 관리 도구 확보로 변질될 수 있습니다. 이를 경험한 팀원들은 (애자일 프로젝트를 경험했으나 안 좋은 기억으로 인해) 애자일은 더 힘들고 다시는 적용하고 싶지 않은 개발 방식으로 인식하게 되기 쉽습니다.

2 중요한 건 '애자일' 한 마음

애자일을 적용하는 팀에서 중요한 건 애자일한 마음과 태도라는 것이 크게 강조되는 상황이 발생하곤 합니다. 하지만 정작 애자일 한 마음이 어떤 것인지는 각자가 편한 대로 생각한 채 개발이 진행됩니다. 이를테면 미리 계획을 세우지 않고 임기응변 식으로 대응하는 것을 애자일하다고 이야기하거나, 형식이나 절차를 버리고 일하는 것을 애자일 하다고 이야기하기도 합니다. 심지어는 실천법이나 개발 절차 자체의 가치조차 폄하해서 지키지 않는 것이 더 애자일 하다는 것으로 오해하기도 합니다.

3 NAH(Not Applicable Here) 신드롬: 그 기법은 우리 환경에 맞지 않아요

애자일 기법 가운데 편하거나 받아들이기 좋은 것만 도입하려고 하는 현상이 있습니다. 하지만 현 프로젝트에서 쉽게 적용하기 어려운 것을 도입해야 실제 프로젝트에서 개선을 이룰 수 있는 상황도 많습니다. 이러한 때는 팀이 합심해서 노력해야 실현 가능한 경우가 대부분입니다. 팀에 도입이 용이해서 적용했지만 결과적으로 애자일 도입 후 좋아진 점을 찾아보기도 전에 프로젝트가 끝나버릴 수도 있습니다.

4 목표를 잊은 일일 스크럼 미팅

스크럼 미팅이 애자일 실천법 가운데 가장 쉽게 적용할 수 있는 것 중 하나지만 사실 제대로 적용 하기란 생각만큼 쉽지 않습니다. 스크럼 미팅을 잘하기 위해 잊지 말아야 할 몇 가지가 있습니다.

“프로젝트 목표/스프린트 목표는 항상 공유되고, 진행 상태는 매일 갱신 되어야 하며, 위험과 이슈는 투명하게 식별되고, 문제는 팀이 함께 고민하고 해결하기 위해 꾸준히 노력해야 한다”는 것입니다. 목표를 잊으면 재미도 없고 애자일 적용 자체가 형식적으로 전락하게 됩니다.

5 잘못 흘러가는 스크럼 미팅의 징후들

스크럼 미팅을 하면서 다음과 같은 징후가 보인다면 미팅이 제대로 진행되고 있지 못함을 인지 해야 합니다.

어느 시점부터 누군가에게 보고하는 느낌을 받거나, 누군가가 우리의 한일을 감독하는 것 같은 느낌을 받고, 스크럼 미팅에서 실수나 완료하지 못한 작업에 대하여 숨기거나 변명을 하려는 분위기가 많아지고, 특정 사람만 말을 많이 하고 일방적인 전달 형태로 진행되고, 누군가 출근시간 체크 목적으로 활용한다는 느낌을 받는다면, 팀원이 함께 진지하게 고민해야 합니다.

6 누가 스크럼 마스터인가

스크럼 마스터는 매우 중요한 역할임에도 불구하고 부적절한 사람이 스크럼 마스터를 맡는 경우가 종종 생깁니다.

첫째, 자원한 사람에게 스크럼 마스터 역할을 맡기는 경우입니다. 누가 스크럼마스터를 하겠냐는 질문에 다들 주저하는 상황을 생각해보겠습니다. 그때 자원한 사람에게 스크럼 마스터를 맡기고 다른 팀원들 “아 나는 아니니 다행이야” 라고 느끼는 상황을 말입니다. 자발적인 의지도 중요하지만 그 보다는 팀을 하나로 통합하고 문제 상황을 빠르게 인지해서 해결책을 찾아 도입할 수 있도록 만드는 스크럼 마스터로서의 역량과 자질 이 더 중요할 수 있습니다.

둘째, 팀원들이 돌아가며 스크럼 마스터 역할을 맡는 경우입니다. 언뜻 보면 스크럼 마스터라는 역할을 팀원 모두가 경험할 수 있어서 좋다고 여길 수 도 있습니다. 하지만 아무나 스크럼 마스터를 맡아도 상관없다는 인식이 생길 수 있으며 프로젝트가 진행되는 내내 스크럼 마스터가 누구인지 모르는 혼란이 생길 수 있습니다.

마지막으로 팀장이 스크럼 마스터가 되는 경우입니다. 스크럼 마스터는 팀을 보호하는 사람입니다. 그래서 어떤 경우에는 관리자와 반대 입장에서 팀을 대변해야 하는데 팀장이 스크럼 마스터라면 역할 분리가 불가능해 집니다.

7 분석/설계 없이 코드 개발 바로 하기

애자일을 적용하면서 가장 많이 받는 오해가 분석/설계 없이 바로 개발을 한다는 것입니다. 짧은 스프린트 기간 동안 동작하는 소프트웨어를 만들고 데모를 통해 확인 받는 것에 더 주목하기 때문일까요? 그보다 분석/설계 단계를 개발에 필요한 논리적인 흐름을 정리하고, 모델링한다고 여기기 보다는 문서를 만드는 시간이라고 생각하기 때문입니다.

분석/설계 결과를 정리하는 형태와 상관없이 (스케치를 하던, 메모로 작성을 간략히 하던, UML 모델을 만들던) 개발을 위해 필요한 논리를 구조화하는 작업은 반드시 필요합니다. 다만, 규모가 크거나, 운영/유지보수를 위해 정형화된 문서가 필요하거나, 감리가 필수적인 프로젝트에서는 형식을 갖춘 문서 작업이 필요하기도 합니다.

8 스토리 포인트에 의한 생산성 비교

두 개 이상의 애자일 팀이 있을때, 많은 관리자들은 사용자 스토리가 반영된 소멸 차트를 가지고 생산성을 측정하고 싶어합니다. 게다가 한 명의 고객이 두 팀에 대해 요구사항 전달 및 관리에 책임이 있는 경우 이러한 현상이 두드러집니다. 하지만 스토리 포인트 자체가 팀마다 다른 기준으로 정의되기 때문에 서로 다른 두 팀을 평가하는 자료로 활용하는 것은 잘못된 접근 방식입니다.

가능하다면 (팀의 진척을 숫자로 나타내는 것보다) 지속적으로 고객과 대화하면서 스프린트 리뷰마다 얼마나 고객을 만족시킬 수 있는지를 기준으로 팀의 성공여부를 판단하는 것이 좋습니다.

고정가격 계약 기반의 프로젝트로 진행되는 상황에서는, 전체 분량 대비 어느 정도 개발이 완료되었는지 고객이 자료나 지표를 요구할 수도 있습니다. 이런 상황에서는 어떤 방식으로든 이를 측정해야 합니다. 이런 경우 고객과 완료 기준에 대해 서로 명확히 이해한 후 계획을 수립할때 사용자 스토리에 대한 크기를 어느 정도 맞추고, 사용자 스토리로 진척을 표시하는 방법도 가능해 집니다.

9 불평이 가득한 회고

때로는 (팀 내부에서 원인과 개선점을 찾기보다) 팀 외부의 조직이나 제품 책임자(혹은 PM)가 제대로 일을 하지 않기 때문이라는 불평이 쏟아져 나올 때가 있습니다. 이런 경우 대개 초기에는 부정적인 한 두 명의 입에서 의견이 나오는 식으로 시작해서, 회고가 반복되면서 불만의 에너지는 팀 전체로 전염이 되고 결국, "그래.. 우리만 잘 하면 뭐해~" 라는 식의 무기력한 분위기가 팀 전반에 자리잡게 됩니다. 특히 팀의 자유로운 의견 개진을 위해 제품 책임자(혹은 PM)를 회고 때 배제할 경우 이러한 분위기로 빠져들기 쉽습니다.

10 야근 불가

애자일 실천법 중에 “지속 가능한 속도로 일하기(Sustainable Face)”가 있습니다. 과도한 업무는 오히려 팀의 생산성을 떨어뜨리기 때문에 잔업은 지양해야 한다는 실천법입니다. 애자일을 도입하면 야근을 하지 말아야 하는 걸까요? 애자일 방식으로 프로젝트를 진행하고 있으니 야근을 하면 안 된다고 이야기하시겠습니까?

애자일에서 이야기하는 많은 가치와 원칙이 있습니다. 어떤 상황이냐에 따라 이런 가치와 원칙이 더 중시될 수도 있습니다. 예를 들어 팀에서 운영하는 지속적인 통합(CI) 서버가 빌드에 실패했다면 어떻게 해야 할까요? 빌드가 깨진 채로 퇴근하는게 맞을까요?

빌드를 깨먹은 사람은 빌드가 성공할 때까지 퇴근할 수 없다는 원칙을 가진 애자일 개발팀이 많습니다. 야근을 강요하는게 아니라 빌드가 깨져서는 안 된다는 팀의 원칙을 중시하는 것입니다.

애자일 개발을 프로젝트에 도입하는 자체가 야근을 없애주는 게 아닙니다. 다양한 애자일 실천법을 도입하고 개발 속도를 높이기 위해 노력함으로써 지속 가능한 속도로 일하는 애자일 팀을 만들 수 있습니다.

11 팀 자율을 빙자하여 나태해진 팀

팀 리더가 업무를 할당하는 것이 아니라 팀 전체가 책임을 공유하고 팀 스스로가 해야 할 일을 결정할 수 있다는 것이 애자일 팀의 특징입니다. 그러나 이런 특징이 자칫 팀의 조직운영 체계와 리더십을 부인하는 분위기로 전락해서는 안됩니다.

오히려 팀이 나태해지고 권한만을 강조하며 책임을 회피하는 모습을 보이게 되면, 초기 긍정적이고 개방적으로 나섰던 팀 리더에게 상처를 주게 되고 결국 팀은 이전보다 더욱 세밀한 관리와 통제를 받게 됩니다.

12 모든 스프린트에 통합테스트하기

스프린트 시작 전에 계획을 통해 동작하는 소스코드를 만들어 내는 것을 스프린트의 목표로 설정합니다. 그리고 이를 검증하기 위해 스프린트 마다 사용자 스토리 단위의 기능테스트(Functional Test)와 통합 테스트 시나리오 기반의 통합테스트를 수행합니다.

비즈니스적으로 의미가 있는 통합테스트 시나리오를 검증하기 위해 한 스프린트 내에서 모든 사용자 스토리를 구현하기란 현실적으로 어렵기 때문에, 이 시점의 테스트는 통합 테스트라기보다 기능 테스트에 가까운 형태로 수행합니다.

13 운영조직에서 스크럼 적용하기

업무 특성상 개발과 유지보수 업무를 동시에 진행하거나 운영만을 하는 조직이 애자일을 도입하는 경우 쉽게 접할 수 있는 것이 스크럼입니다. 하지만, 각 기능 별 소요기간이 천차만별이어서 사용자 스토리에 기반한 백로그 관리가 어렵고, 계획 자체가 불가능한 경우가 발생하기도 합니다. 이러한 경우 칸반과 같은 운영조직에 맞는 방법을 적용하는 것이 바람직합니다. “애자일 = 스크럼”이 아니며 조직 특성에 맞는 프로세스를 도입하는 것이 중요합니다.

애자일 회고

애자일 회고란?

애자일 회고는 이터레이션 동안 팀이 일해온 방식과 결과물에 대해 팀이 함께 이야기 하면서 개선할 점을 찾고, 다음 이터레이션에 보다 나은 팀이 되기 위해 개선 사항을 적용하는 실천법입니다.

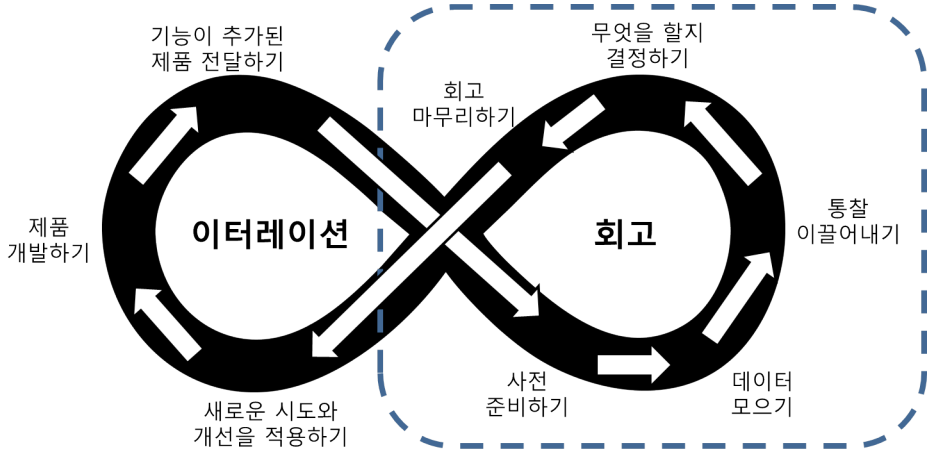


그림 4-1. 이터레이션과 회고, 회고의 절차

유용한 단계 별 회고 기법들

단계	기법 예제
1 체크인/회고 준비하기	① 이터레이션에 대한 감상 공유 ② ESVP ③ 체크인
2 데이터 모으기	① 좋았던 점 / 나빴던 점 모으기 ② 학습 매트릭스 ③ 화남, 기쁨, 슬픔 ④ 팀 만족도 그래프 그리기
3 통찰 이끌어내기	① 점 투표 하기
4 무엇을 할 지 결정하기	① SMART 한 회의 ② EXACT 한 목표기준 세우기
5 회고 마무리하기	① 회고활동에 대한 회고 ② 서로에게 감사하기 ③ Follower 구하기

1 체크인/회고 준비하기

사람들이 손쉽게 회고에 집중하기 위해, 분위기를 조성하는 활동으로, 지난 이터레이션 결과물(완료항목과 테스트 결과)에 대한 공유를 수행합니다.

이터레이션에 대한 감상 공유

Time Box: 10분

수행 방법

1. 팀원 한 명씩 번갈아 가며, 이번 이터레이션에서 느꼈던 것을 짧게 이야기 하도록 합니다.
2. 내용은 자유롭게 말할 수 있도록 하되, 시간은 개인별로 1분 30초 이내로 제한합니다.

주의할 점

- 내용 자체가 너무 부정적으로 흐르지 않도록 합니다.
- 팀 내 인원에 대한 불만을 이야기 하지 않도록 합니다.

기대효과

- 이터레이션을 수행하면서 느꼈던 감정과 데이터들을 정리할 수 있는 시간을 제공합니다.
- 나 이외에 다른 팀원들이 지난 이터레이션 동안에 생각나는 내용을 공유합니다.

ESVP

Time Box: 10분

수행 방법

1. 팀원 한 명 당 한 개의 포스트잇을 나누어 줍니다.
2. 참가자들은 익명으로 회고에 대한 자신의 태도를 ESVP 중 알파벳 하나로 적게 합니다.

> E: Explorer (탐험가)
> 새로운 아이디어나 직관을 쉽게 발견하는 사람. 가능한 모든 것을 배우고 싶어하는 사람
> S: Shopper (쇼핑하는 사람)
> 가능한 한 모든 정보를 둘러보고 그 중 유용한 아이디어를 취하고 행복해하는 사람
> V: Vacationer (휴양객)
> 현재 회고에 흥미는 없지만, 매일 반복되어 지겨운 일상에서 잠시나마 떠나 있는 행복을 느끼는 사람
> P: Prisoner (죄수)
> 억지로 앉아 있다고 느끼고, 차라리 다른 일을 하고 싶어하는 사람

3. 결과를 칠판에 붙여, ESVP 가 각각 몇 명인지 팀원 전체가 알게 합니다.
4. “이 결과에 대해 어떻게 생각하나요?” 라는 질문으로 보다 심도 있는 대화로 진행하는 기초로 활용합니다.

기대효과

- 이터레이션을 수행하면서 느꼈던 감정과 데이터들을 정리할 수 있는 시간을 제공합니다.
- 나 이외에 다른 팀원들이 지난 이터레이션에 생각난 내용을 공유합니다.

체크인

Time Box: 5분

수행 방법

1. 회의시작 전에 구성원들에게 특정 주제에 대해 짧게 돌아가며 이야기 합니다.

주제 예시

- 지금 자신이 느끼고 있는 감정을 한 마디로 표현해 주세요
- 지금 자신에게 가장 필요한 것을 한 마디로 표현해 주세요.
- 최근 있었던 에피소드와 그로 인해 생긴 변화는 무엇이 있나요.
- 요즘 자신이 가장 좋아하는 음식은 무엇인가요
- 자신을 동물에 비유한다면 어떤 동물일까요.

주의할 점

- 심리적인 장벽을 낮춰줄 뿐이지, 궁극적으로 모임이나 회의의 문제점을 해결해 주지 않습니다.

기대효과

- 적극적으로 참여하지 않았던 사람에게 무조건 참여하는 기회를 줌으로써, 참여에 대한 두려움과 심리적 장벽을 없애 보다 회의에 적극적으로 참여하도록 합니다.
- 모임이나 회의에서 보다 다양한 의견들이 나옵니다.

2 데이터 모으기

지난 이터레이션 동안 일어난 일에 대해 팀원이 느낀 공통 분모를 찾는 활동

좋았던 점 / 나빴던 점 모으기

Time Box: 20분

수행 방법

1. 참여 인원에게 두 가지 색깔의 포스트잇을 각각 3 장씩 나눠줍니다.
2. 참여 인원에게 이터레이션 동안 일하는데 도움이 되었던 요소와 방해가 되었던 요소 3 가지를 각각 다른 색의 포스트잇에 적게 합니다.
3. 작성이 완료된 사람의 포스트잇부터, 왼쪽에는 도움이 된 요소, 오른쪽에는 방해가 되었던 요소로 벽에 붙이고 비슷한 내용끼리 그룹핑합니다.
4. 진행자는 그룹핑된 내용을 간략하게 정리하여 전체 인원과 공유합니다.

수행 팁

- 인원이 많을 경우(10 명 이상) 시간을 단축시키기 위해, 포스트잇을 3 장 대신 2 장만 사용합니다.
- 포스트잇이 많을 경우 벽에 붙이는 시간을 줄이기 위해, 팀원 중 한 명에게 도움을 요청합니다.

기대효과

- 아주 가벼운 내용부터 무거운 내용까지 다양한 의견이 공유될 수 있는 장을 만듭니다.
- 비슷한 내용이 많다면 꼭 개선해야 할 내용일 수 있습니다.

학습 매트릭스

Time Box: 30분

수행 방법

1. 화이트 보드에 X/Y 축을 그리고, 왼쪽 위부터 잘된 점(+), 나빴던 점(-), 고칠 내용(C), 불확정(?) 을 적습니다.
2. 진행자는 해당 내용에 대해 팀원들에게 설명하고 20 분 동안 자유롭게 화이트 보드에 와서 이터레이션에 대한 아이디어를 적도록 합니다.

수행 팁

- 고칠 내용이 다음 이터레이션에 적용가능한지 팀에 스스로에게 물어보게 합니다.

기대효과

- 팀 스스로 이슈 및 현안에 대해 자유롭게 공유할 수 있게 만들어 Self-Organizing을 연습할 수 있게 해줍니다.

화남, 기쁨, 슬픔

Time Box: 20분

수행 방법

1. 참여자에게 각각 다른 색깔(빨간색, 노란색, 파란색)의 포스트잇 3장을 을 나눠줍니다.
2. 이터레이션 동안 기억날 만한 이벤트들을 상기시켜 줍니다.
3. 이터레이션 동안 가장 화가 났던 이벤트를 빨간색 포스트잇에 적고, 가장 기뻤던 이벤트를 노란색 포스트잇에 적고, 가장 슬펐던 이벤트를 파란색 포스트잇에 적게 합니다.
4. 진행자는 포스트잇을 수집하여 3 그룹으로 나눠 왼쪽, 가운데, 오른쪽에 각각 화남, 기쁨, 슬픔을 적습니다.
5. 진행자는 해당 내용을 리포팅하면서 각각 이벤트들에 대해 상기시킵니다.

수행 팁

- 이 회고 방식은 부정적인 의견을 보다 많이 도출시키기 때문에, 긍정적인 내용을 나중에 이야기 해야 좋은 분위기로 나머지 회고를 진행할 수 있습니다. 따라서 슬픔→화남→기쁨 순서대로 리포팅하는 것이 좋습니다.

기대효과

- 부정적인 감정을 긍정적인 감정과 적절히 균형을 유지하여, 문제를 팀이 직시하고 해결할 수 있는 장을 만듭니다.

팀 만족도 그래프 그리기

Time Box: 20분

수행 방법

1. 참여 인원들에게 포스트잇 한 장씩을 나누어 줍니다.
2. A4 지에 다음이 적혀있는 종이를 나누어 주고, 포스트잇에 개인이 생각하는 팀의 수준을 숫자로 적게 합니다.
3. 각 레벨에 대한 정의를 모두에게 공유합니다.

> Lv 5: 우리 팀은 세계 최고의 팀이다.
> Lv 4: 나는 내가 이 팀의 일원이어서 기쁘고 우리 팀이 협력하는 방식에 만족한다.
> Lv 3: 제법 만족한다. 우리는 협업을 잘하는 편이다.
> Lv 2: 만족할 때도 있지만 그렇지 않을 때도 있다.
> Lv 1: 불행하다. 팀 워크 수준이 불만족스럽다.

4. 벽에 별 수치를 알기 쉽게 숫자 별 일렬로 포스트잇을 나열하여 붙입니다.
5. 팀원들에게 5분 정도의 시간을 주고, 우리 팀 전체의 만족도가 현재 상태인 이유를 고민하게 합니다.
6. 팀원 한 명씩 차례로 우리 팀의 상태에 대한 원인을 추측한 내용을 발표하게 합니다.

수행 팁

- 개선할 내용이 다음 이터레이션에 적용 가능한지 팀 스스로 정해야 합니다.

기대효과

- 팀 스스로 이슈 및 현안에 대해 자유롭게 공유할 수 있게 만들어 Self-Organizing을 연습할 수 있게 해줍니다.

그림 4-2. 팀 워크 만족도 그래프

3 통찰 이끌어내기

모아진 데이터를 이용하여 어떠한 주제에 대해 토의할 지 논의하여, 보다 개선된 팀이 되기 위해 무엇을 해야 할 지에 관한 실천계획(Action Plan) 을 만드는 활동

점 투표하기

Time Box: 10분

수행 방법

1. 점 스티커를 팀원들에게 나누어 줍니다.
2. 팀원들에게 마음 속으로 모아진 데이터에 대해 고민하도록 하고, 팀에 가장 도움이 되었던 것과 가장 방해가 되었던 것을 우선순위화 하도록 합니다.
3. 팀원들에게 우선순위가 높은 순으로 두 가지에 대해 모아진 데이터가 있는 곳에 직접 가서, 투표하도록 합니다.
4. 표가 많이 있는 도움이 된 내용과, 방해가 된 내용 2 가지 주제에 대해 리포팅 합니다.
5. 15분 간 팀원들에게 도움이 된 내용은 어떻게 더 도움이 될 지, 방해가 된 내용은 어떻게 개선할 지에 대해 토의를 합니다.
6. 결론을 각 그룹의 대표가 이야기하게 합니다.

수행 팁

- 토의에 적절한 인원은 4 명 정도입니다. 6 인 이상이면 그룹으로 나누어 토의합니다.

4 무엇을 할 지 결정하기

SMART 한 회의

Time Box: 20분

수행 방법

1. SMART 한 목표에 대해 설명합니다.

   S: Specific (상세하고)
   M: Measurable (측정 가능하고)
   A: Attainable (달성 가능하고)
   R: Relative (팀의 개선과 관련이 있으며)
   T: Timely (이터레이션 안에 실행 가능해야 합니다)

2. 팀에게 15분의 시간을 주고, 주어진 데이터에 대해, 다음 이터레이션 동안 무엇을 SMART 하게 수행하면, 팀에 도움이 될 지 토의하도록 합니다.
3. 그룹 별 회의 결과를 그룹에서 한 명씩 타 팀과 공유할 수 있도록 리포팅합니다.

수행 팁

- 인원이 4 인 이상이면 그룹으로 나누어 토의합니다.

EXACT한 목표기준 세우기

Time Box: 20분

수행 방법

1. EXACT 한 목표에 대해 설명합니다.

> EX: Exciting (긍정적이고 에너지가 넘치는 영감을 주고)
> A: Assessable (성취한 것을 확인 가능하며)
> C: Challenging (도전적인 목표이고)
> T: Timely (기한이 있어야 합니다)

2. 팀에게 15분의 시간을 주고, 주어진 데이터에 대해, 다음 이터레이션 동안 무엇을 SMART 하게 수행하면, 팀에 도움이 될 지 토의하도록 합니다.
3. 그룹 별 회의 결과를 그룹에서 한 명씩 타 팀과 공유할 수 있도록 리포팅합니다.

수행 팁

- 인원이 4 인 이상이면 그룹으로 나누어 토의합니다.

5 회고 마무리하기

수행한 회고를 마무리하고, 다음 이터레이션의 시작을 알리는 활동

회고활동에 대한 회고 (+/델타(Delta))

Time Box: 10분

수행 방법

1. 플립 차트에 T 자 형태의 표를 그리고 제한 시간을 알려줍니다.
2. 강점과 변화를 마음껏 말하도록 유도합니다.
3. 사람들의 피드백에 감사하며 더 발전시킬 것과 변화시킬 것을 정합니다.

수행 팁

- T 자 형태의 표를 기준으로 피드백의 수를 제한하는 방법도 있습니다.
- 변화시킬 것이 결정되었다면 담당자를 정해 추적하게 하는 것이 좋습니다.

기대효과

- 회고를 끝내기 전에 다음 회고에서 이번 회고의 무엇을 유지하며 어떤 변화를 시도할지 결정할 수 있습니다.

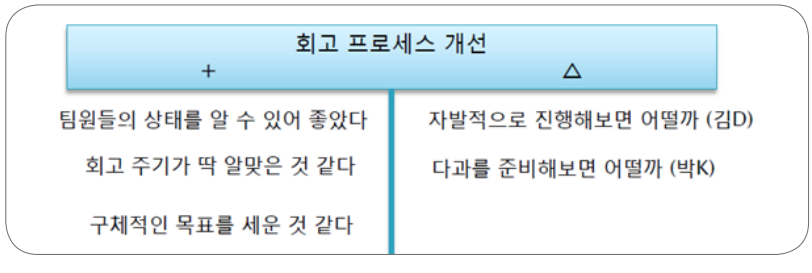


그림 4-3. 회고 활동에 대한 회고: +/Delta

Follower 구하기

Time Box: 10분

수행 방법

1. 회고를 통해 결정한 사안의 중요성과 가치에 대해서 공유합니다.
2. 회고를 통해 결정한 사안에 대해 추적 관리할 담당자를 구합니다.
3. 자발적으로 문제를 추적 관리할 담당자가 결정되면 진행자는 일의 중요성을 다시 강조하며 모두 함께 담당자를 격려합니다.

수행 팁

- 강요 되지 않도록 주의합니다.

기대효과

- 팀원 사이에 자발적인 스폰서십을 구하여 자기조직적인 팀 분위기를 형성합니다.

FAQ

Q1 왜 고객이 사용자 스토리를 작성해야 하나요?

각 스토리는 기술적 전문 용어가 아닌 비즈니스 언어로 작성해야 합니다. 그래야 고객이 스토리를 어느 이터레이션이나 릴리즈에 포함시킬지 우선순위를 정할 수 있습니다. 따라서 기획의 주체자가 스토리를 작성하는 것이 제품의 동작을 가장 잘 설명할 수 있습니다.

Q2 사용자 스토리에 구체적인 세부사항도 적어야 하나요? 이를테면 '이름, 이메일 주소, 아이디, 제목, 본문' 등과 같은 검색조건에 대한 내용들의 경우 모두 표시해야 하나요?

스토리 문장은 짧고 간결하게 적는 것이 좋습니다. 따라서, 필요하다면 마찬가지로 따로 항목을 따로 만들어 적어 둡니다.
세부사항을 모두 스토리로 작성하기보다, 관련하여 개발팀과 고객이 서로 합의한 내용을 스토리로 구현했는지 확인하는 (테스트)형태로 문서화 하는게 좋습니다.

Q3 저희는 기존에 도출된 WBS 가 이미 있는데 사용자 스토리를 또 작성해야 하나요?

내용과 우선순위가 고객 관점인가 개발 관점인가를 살펴보시길 바랍니다. 고객 관점으로 작성되어 있다면 기존 WBS를 스토리로 간주할 수 있습니다. 하지만 대개 WBS는 구현 관점으로 작성되어 있어서 고객의 관점이 반영되기는 힘듭니다!

Q4 스크럼 미팅에서 할 말이 많아서 15분이 짧습니다. 어떻게 해야 하나요?

팀이 모두 공통적으로 느끼고 있다면 일일 스크럼 미팅 시간을 20분~25분 정도로 늘려서 진행합니다. 단, 너무 오래 서있어서 팀이 피로하게 느끼지 않을 정도의 시간이어야 합니다.

특정 주제를 가지고 토론하느라 길어진다면, 해당 내용을 일일 스크럼 미팅을 끝나고 관련자들끼리 따로 모아 별도 진행하도록 유도합니다.

팀원이 너무 많아서 오래 걸리기도 합니다. 더 작은 팀으로 나누어서 진행합니다.

Q5 팀 전체가 짝 프로그래밍을 해야 하나요?

꼭 그럴 필요는 없습니다. 팀원 중 참여하고 싶은 사람들부터 시작하십시오. 또한 프로젝트 전체 기간이 아니라 일정 기간에만 하도록 유도하는 것도 좋은 방법입니다.

Q6 왜 개인의 업무속도가 아닌 팀의 업무 속도 인가요?

일은 개인이 아닌 팀이 하는 것이기 때문에 중요한 건 팀의 속도이지 개인의 속도가 아닙니다.

개인의 속도를 추적하고 공유하면 프로젝트 성공을 가로막게 됩니다. 개인을 평가하게 되면 다른 사람을 돕는 일을 하지 않으려 할 것 입니다. 결국 서로 협력하지 않게 되어 팀으로서의 시너지가 일어나지 않습니다.
또한 팀의 속도로 팀 간 비교와 평가를 하려고 해서도 안됩니다. 스토리 포인트를 정확하게 이해 한다면 비교할 수 있는 항목이 아니라는 것을 알 겁니다. 개개인 혹은 팀을 존중하고 배려하는 것은 애자일의 핵심가치이기도 합니다.

Q7 스토리 포인트(Story Point: SP)와 FP(Function Point)를 활용하여 생산성을 측정하는 것과 어떤 차이가 있나요?

FP와 SP는 목적이 다르기 때문에 둘 사이의 연관성은 거의 없습니다.

흔히 FP는 SW규모 및 비용을 산정하는데 흔히 사용됩니다. 그리고 FP를 잘 이해하는 사람이라면 누가 산정하더라도 동일 프로젝트에 대해서는 거의 비슷한 수치로 산정 됩니다.
이에 반해 SP는 애자일 프로젝트 업무 복잡도, 특성 그리고 참여자의 경험을 반영한 추정입니다. 측정하는 팀마다 모두 다르며 편차도 클 수 있으며 상대적입니다.

Q8 한 스프린트의 길이를 2주 이내로 짧게 잡으면 어떤가요?

장점: 스프린트 기간이 짧으면, 회사는 더 자주 방향을 올바르게 수정할 수 있게 됩니다.
(짧은 스프린트 ⋯› 짧은 피드백주기 ⋯› 더 잦은 고객의 피드백 = 잘못된 방향에 따른 시간낭비 감소)

단점: 스프린트 기간이 짧아지면 짧아질수록, 고객의 변경 요청 분량이 늘어날 수 있습니다. 또한 개발팀은 짧은 스프린트 기간 내에 동작하는 SW를 개발해야 하므로 심리적인 압박이 무척 커집니다.

Q9 한 스프린트의 길이를 2주 이상으로 길게 잡으면 어떤가요?

장점: 스프린트 길이가 길면, 팀이 추진력을 얻기 위한 시간을 갖게 되고, 발생하는 문제를 해결하면서 스프린트를 달성할 수 있는 충분한 여력을 가질 수 있습니다.

단점: 스프린트 기간이 길면, 고객 확인과 피드백을 받는 주기가 길어지는 만큼 프로젝트 리스크가 증가하게 됩니다.

Q10 추정을 틀릴 수 있다는 가정을 한다고 하는데, 이에 대한 보정을 할 수 있는 방법이 있나요?

추정치의 불확정성과 오류는 속도(Velocity)로 보정할 수 있습니다.

몇 번의 스프린트를 통해 사용자 스토리를 구현해 나가다 보면 팀의 속도를 비교적 객관적으로 측정할 수 있고 경험을 통해 팀의 속도를 가질 수 있습니다. 스토리 포인트 기반 접근법의 강점은 속도 개념을 적용하기 때문에 계획상의 오류가 스스로 교정(Self Correcting)된다는 점입니다.

가령, 어떤 팀이 프로젝트의 규모를 200점으로 추정했다고 가정했을 때, 처음에 이 팀은 스프린트 당 25점의 스토리를 구현할 수 있을 것으로 예상하였지만 프로젝트가 시작되자 실제 속도는 20에 불과했습니다. 하지만 재 추정 작업 없이도 프로젝트의 기간이 스프린트 8번 대신 10번으로 늘어날 것이라는 사실은 쉽게 알아낼 수 있습니다.

Q11 추정한 결과가 맞지 않는다면 추정을 다시 해야 하나요?

추정을 다시 하는 작업은 스토리 포인트가 크게 바뀐 경우에 한정해서 진행하길 권장합니다. 따라서 단순히 프로젝트 진도가 생각만큼 빠르지 않다는 이유로 재 추정을 하는 건 바람직하지 않습니다.

구현해야 할 각 기능의 추정치는 다른 기능의 추정치와 견주어 상대적으로 정해집니다. 그러므로 추정치가 다소간 부정확하더라도 큰 문제는 발생하지 않으며, 추정치들 간에 일관성을 유지하는데 집중하도록 합니다.

Q12 이번 스프린트에는 눈에 보이는 데모를 진행할 내용이 없어요!

어떤 스프린트는 작업 결과물을 직접 눈으로 보기 어렵거나 준비하기가 애매한 경우가 있습니다. 그런 경우 데모 진행의 생략을 고려하게 됩니다. 하지만 고객이 볼 수 있는 화면이 없다고 해서 스프린트의 데모를 생략해서는 안됩니다.

이런 상황이라면 스프린트 데모 시간을 Peer Review, Code Inspection 등의 시간으로 활용하면 좋습니다. 그리고 팀이 합의하여 정리한 완료정의 기준을 만족하는 항목들이 있다면 그 기준에 근거하여 팀원들이 검토/확인하는 시간으로 활용할 수 있습니다.

Q13 스프린트 데모 후 고객으로부터 피드백을 받은 내용을 해당 스프린트에서 해결해야 하나요?

스프린트 리뷰는 스프린트 종료 2~3일 전에 실시 하도록 합니다. 피드백 가운데 1~2일 사이에 해결할 수 있는 것은 반영하고 나머지는 다음 스프린트로 넘겨 다시 스프린트 (또는 릴리즈) 계획 회의를 할 때 결정하게 됩니다. 만약 스프린트를 마치고 데모를 한다면 고객의 피드백 사항을 다음 스프린트에 반영합니다.

Q14 고객 참여가 미흡하거나 고객이 원거리에 있어 수행이 원활 하지 않을 때는 어떻게 해야 할까요?

팀 내부 검토 활동 기회로 활용하거나, 팀 내에서 고객 역할 대행을 통해 스프린트 리뷰를 수행합니다. 이 때 중요한 것은 제품 책임자는 고객의 눈높이로 피드백을 해야 한다는 것입니다.

Q15 고객이 반드시 팀과 상주해서 하루 종일 있어야 하나요?

팀과 같은 공간에 있으면 수시로 의문점이 있을 때 추측이 아닌 고객의 의사결정을 통해 진행할 수 있어서 좋습니다. 하지만 고객도 SW개발 지원 이외에 자신의 업무를 병행해야 하는 상황이라면 팀과 인접한 별도의 공간에 있는 것도 가능합니다. 단 접근성이 좋아야 합니다.

Q16 고객이 24시간 팀의 질문에 응대해야 하나요?

고객 역시 자신의 일을 수행하는데 집중할 수 있어야 하므로, 그럴 경우 팀에 상주하거나 팀의 질문에 답을 해주는 시간을 정기적으로 정해서 운영할 수 있습니다.

Q17 고객이 여러 명이고 각 고객들이 파트타임으로 참여합니다. 괜찮을까요?

고객들 간에는 서로의 관점/입장이 다를 수 있습니다. 그들을 서로 다른 시간에 만나 의견을 듣게 되면 개발팀에 혼란을 초래할 수 있습니다. 고객들끼리도 함께 만나서 의견을 조율할 수 있도록 해야 합니다.

Q18 고객이 없는 경우는 어떻게 하나요?

팀의 산출물을 전달받거나 사용하는 조직을 고객으로 간주하고 참여시킵니다.

Q19 애자일은 산출물 없이 개발을 하나요?

그렇지 않습니다. 애자일은 동작하는 SW중심의 개발과, 낭비 제거를 지향하므로 다른 방법론에 비하여 산출물이 적으며 내용 또한 단순하게 작성하는 것을 권장 합니다. 또한 유지보수에 꼭 필요한 산출물과 내용의 적정성은 고객과 합의를 합니다.

Q20 애자일을 적용하면 프로젝트 성공률이 높아지나요?

당연한 이야기입니다만, 애자일 적용이 곧 성공을 보장하진 않습니다. 이건 어떤 방법론이라 할지라도 마찬가지입니다. 다만 애자일 방법론을 적용하실 경우, 프로젝트의 불확실성을 좀 더 투명하고 가시적으로 관리함으로써 프로젝트 성공의 예측 가능성을 높여줄 수 있습니다.

다시 정리하겠습니다. 성공율이 높아지나요? 네. 높아집니다. 하지만 세상에 쉬운 일은 없습니다. 다른 부분에서 더 노력해야 할 수도 있습니다. 애자일을 잘 적용하기 위해 지속적으로 관심을 갖고 꾸준히 개선하는 노력이 필요합니다.

Q21 애자일은 기존보다 더 많은 인원이 투입 되는 게 아닌가요?

그렇지 않습니다. 기존의 인원투입 계획과 애자일로 진행할 때 인원 투입은 분명 차이가 있으며, 애자일은 프로젝트 전반부부터 후반부까지 투입인력에 대하여 "자원 평준화" 하여 계획을 세울 것을 권장합니다. 주요 리딩(업무, 기술) 인력은 프로젝트 초반에 투입하는 것이 좋습니다.

Q22 방법론과 관계없이 모든 요구사항을 관철시키려고 하는 고객을 어떻게 설득하면 좋을까요?

이터레이션 마다 결과물을 보여주니, 고객의 피드백이 강력할 수 있습니다

- 고객의 요구사항이 개발의 방향을 벗어 나지는 않았는지 우선 확인합니다.
- 고객의 요구사항에 공감하며 반영의 의지를 보입니다.
- 하지만, 이때 새로운 요구사항이 얼마의 일정이 소요하게 될 것인지를 확인해서 알려주어야 합니다.
- 팀 미팅을 통해 일정을 추정해 냅니다.
- 고객에게 소요기간을 이야기하며, 우선순위를 정하도록 합니다. 이로 인해 일정이 변경되는 부분에 대해서 알려준 다음, 선택하게 합니다.

Q23 팀 멤버 구성 비율을 어떻게 가져가야 할까요?

기존에 팀원들이 담당하고 있는 역할들을 기반으로 교차기능팀을 구성할 수 있는지를 고민하셔야 합니다.

- 분석과 설계가 가능한 개발자, 개발할 수 있는 DBA, 모델링이 가능한 AA 가 함께 하면 이상적입니다만, 그건 말 그대로 이상적입니다.
- 팀원들만으로 개발 전 영역을 커버하기에 부족하다면, 외부에서 점검/지원해 줄 수 있는 비상주 인력을 소싱합니다.
- 특정 역량이 부족해 팀 개발이 삐걱일 때는 팀 마스터가 Rain Maker 역할을 할 수 있어야 합니다.
- 특정 역량이 뛰어난 팀원이 있을 경우에는 구심에 놓고 적극적으로 활용합니다. (ex. 뛰어난 AA : 공학도구나 연계, 특정 개발분야에서 최대 퍼포먼스를! 훌륭한DBA: Data모델링이나 DB설계 등에서 최대 퍼포먼스를! 등등등)

Q24 기존 방법론에 익숙한 프로젝트 리더의 사고 전환을 어떻게 해야 하나요?

현황보드와 ToDo목록 관리만으로는 불안해 할 수 있습니다.

- 이번 프로젝트를 성공적으로 만들기 위해서, 애자일 방법론이 여타 방법론보다 어떠한 장점을 줄 수 있는지 스스로 비교 판단하도록 유도합니다. 만일 장점이 단점보다 크지 못하다는 결과가 나오면 방법론 선택이 잘못된 셈입니다. (이래서는 안되겠죠)
- 애자일을 불안해 한다면 정확히 불안해 하는 요인이 무엇인지를 파악해서 개방된 곳에서 해당 내용을 볼 수 있도록 만듭니다. 그리고, 팀원들은 해당 사항에 대해 어떻게 생각하는지 의견을 모읍니다.
- 특별히 작성했으면 좋겠다고 생각하는 문서가 있으면, 팀원과 협의를 합니다. 단, 이때 팀원들에게 부하를 주는지, 개발을 방해하게 되는지 살펴보고 판단합니다.

Q25 “(팀이 힘들어 하니까) OOO 실천법은 제외할까요?”- 현실과 타협하고 싶어집니다.

- 타협은 할 수 있지만, 이득을 얻지 못하게 되는 부분이 어떤 점인지는 고객/팀원 모두 인지해야 합니다.
- 그리고, 최대한 적용할 수 있는 부분까지 적용하기 위해 노력합니다. (ex. 특별히 민감한 모듈에 대해서는 자동화 테스트를 구축)
- (필요에 따라) 실천법을 적용하려다 보면 결국 팀의 역량을 향상시키게 됩니다.

Q26 애자일에 대한 개발자들의 오해와 마음가짐에 대해서는 어떻게 생각하시나요?

'애자일 하면 산출물 안 만드는 거 아냐?' 라고 생각하는 사람들이 적지 않습니다.
만들어야 한다고 하면 급 실망하며 '결국 비슷하잖아!' 라고 돌아서기 쉽습니다.

- 팀이 하고 있는 활동 중에 애자일의 가치에 부합하는 것이 있다면, 그것이 이미 애자일이라는 사실을 인식시키십시오. 현실을 부정한다고 느끼면 저항하거나 방어기재를 펼칠 수 있습니다.
- 애자일의 가치가 산출물 미작성에 있는 것이 아님을 인식시킵니다..
- 경우에 따라서는 사용하는 개발 방법론이 애자일임을 숨길 수도 있습니다. (단, 이런 경우 일부 개발자에게 줄 수 있는 '애자일 프로젝트로 진행한다'는 자긍심을 심어주지 못하게 될 수 있습니다.)
- 필요한 산출물은 팀원이 합의를 유도하고 자발적으로 수행하도록 합니다.
- 불필요한 산출물은 최대한 억제시키고 있음을 이해시킵니다.

Q27 실천법에 대한 숙련도가 떨어집니다. 그래도 지속해야 하나요?

책 읽고, 교육 한번 받고 진행하면 헤매기 딱 좋습니다. 기존 방법론처럼 (갑갑하지만) 명확하고 구체적인 틀이 없기 때문입니다.

첫 번째 적용 프로젝트일 경우 좌충우돌하며 학습해야 하는 상황을 피할 수 없습니다. 단, 빠른 시일 내에 적응하도록 팀 내에서 학습이나 경험 공유 등의 활동을 진행합니다.
경우에 따라서는 필요한 고객의 지원을 얻도록 적극 노력 합니다.

Q28 혹시, 사내에서 반드시 준수해야 하는 프로세스나 시스템이 있다면 어떻게 해야 할까요? 예외로 할까요?

- 팀의 리더(혹은 PL, PM)가 해결해 주어야 하는 문제 중 하나입니다.
- 경량화된 형태로 팀을 운영하는데 표준 프로세스나 시스템이 주는 부담을 최소화해야 합니다.
- 관련 팀과 협의하여 사용 예외로 처리하고, 대체할 산출물을 찾습니다.
- FP 계산을 요구할 경우, 마찬가지로 대체 추정 가능한 산출물을 준비합니다. (ex. Stroy Point의 workday/MM 변환 등)

Q29 '전체비용절감'을 생각하는 애자일에서 봤을 때, 개발에서 배포까지의 비용은 오히려 예전보다 높을 수 있습니다. 실제 비용 절감을 체감하려면 운영 및 유지보수까지 진행되어야 하는데 개발만 하고 빠지는 SI프로젝트의 경우 그걸 감당하려는 PM이나 팀을 만들 수 있을까요?

일반적으로 애자일 실천법(테스트 주도 개발, 지속적인 통합, 짝 프로그래밍 등)을 적극 도입할 경우 개발 기간이 더 소요되는 것처럼 느끼게 될 가능성이 큽니다.

그래서 하나의 팀이 기획부터 개발, 유지보수 및 기능 개선까지 지속적으로 진행하는 경우에 비해, 일정이 고정되어 있고 '개발팀'과 '유지보수팀'이 분리되어 있는 SI산업의 경우가 애자일 적용이 더 어렵다고 알려져 있습니다.

사실 이 부분은 선택의 문제인데요. 가장 좋은 방법은 선택지를 제공하고 고객이 선택하도록 하는 방식입니다. 즉, 기존 방식을 버리고 애자일을 잘 적용하면 결함 수정 비용을 포함한 전체 개발비용을 감소시킬 수 있습니다. 가급적이면 구체적으로 어느 정도 감소시킬 수 있는지 자료를 제공하고 도입/미도입에 따른 효과/비용 논의를 통해 고객이 스스로 결정하게 합니다.

그렇게 해서 기간이나 개발 범위에 대해 고객과 따로 협의를 이끌어 내어야 합니다. 이상적으로는 처음부터 이러한 내용에 기초하여 차별화된 방식으로 수주를 하는 것입니다. 애자일 적용 자체가 목적은 아니므로 상황에 맞게 결정하는 것이 중요합니다.

Q30 지속적인 통합을 실천하려고 합니다. 자동화 빌드 시스템을 꼭 갖추어야만 하나요? 전문가가 없습니다.

꼭 빌드 시스템을 자동화할 필요는 없습니다. 기존에 빌드 담당자가 있다면, 그 담당자가 매일 정해진 시간에 빌드를 돌려 검증하도록 하세요. 만약 담당자의 업무가 가중되어 힘들다면 팀원들이 그 역할을 번갈아 해주면 됩니다.

중요한 것은 최신 소스를 자주 빌드하여 코드 베이스가 건강(빌드에 오류가 없고, 시스템의 기능이 정상 동작하는 상태)한지 확인하고, 건강한 코드를 받아 개발을 지속할 수 있도록 하는 것입니다. 사람이 하더라도 그 가치를 얻을 수 있다면 출발로는 괜찮습니다. 점점 다른 일이 많아진다면 그 때 자동화하는 것을 고민해보셔도 좋습니다.

용어사전

교차기능팀
Cross-functional team

서로 다른 기능(분야)의 사람들로 이뤄진 팀을 지칭한다. 예를 들어, 개발자들만 모인 팀이 아니라 분석가, 설계자, 개발자, 테스터가 한 팀을 이룰 때 교차기능팀이라고 한다.

기능점수
Function Point (FP)

"소프트웨어 규모 측정 기법이다. 애플리케이션에서 표현되고 사용된 정보의 양을 정량화하는 방식으로 크게 트랜젝션 기능과 데이터 기능으로 나누어 소프트웨어의 규모를 산정한다."

네비게이터
Navigator

짝 프로그래밍을 할때 직접 작업을 하지는 않지만 목표와 방향을 점검하며 드라이버의 작업을 검토하며 리드하는 역할자이다.

대기행렬 이론
Queueing theory

대기행렬(queue, waiting line)을 수학적으로 다루는 이론이다. 이 이론은 대기행렬에 도착하는 것과 대기하는 것 그리고 서비스되는 일련의 프로세스들에 대한 수학적, 확률적 분석을 가능하게 한다. 흔히 시스템의 평균 대기시간, 대기행렬의 추정, 서비스의 예측 등을 현재 상태에 기초한 시스템의 확률을 기반으로 하여 성능을 측정하는 유용한 도구로 쓰인다.

드라이버
Driver

짝 프로그래밍을 할때 키보드를 소유하고 실제 작업을 진행하는 역할자이다.

리팩토링
Refactoring

소프트웨어를 보다 쉽게 이해할 수 있고 적은 비용으로 수정 가능하도록 겉으로 보이는 동작의 변화 없이 소프트웨어의 내부 구조를 개선하는 실천법이다.

릴리즈 목표
Release Goal

릴리즈 기간동안의 목표를 뜻하며 릴리즈 회고등을 통해 목표에 도달했는지 점검을 한다. 일반적으로 릴리즈 계획을 하면서 목표도 함께 정한다.

릴리즈 소멸차트
Release Burn-down Chart

릴리즈 기간 내 완료되는 일의 추이를 보면서 동시에 남아있는 일의 양을 확인할 수 있는 차트이다.

릴리즈 계획
Release Planning

고객에 의해 도출된 사용자 스토리를 기반으로 팀이 함께 전체 업무량을 산정하고, 팀의 역량에 맞추어 스프린트 별로 사용자 스토리를 할당하여 제품백로그(Product Backlog)를 작성한다. 이와 같이 이터레이션에 기초하여 전체적인 개발 계획을 수립하는 실천법이다.

반복점증적
Iterative and Incremental

일정 기간에 적합한 작업 범위를 선정해서 개발하는 방식이다. 즉, 분할해서 전체를 정복하는 식으로 개발을 진행한다. '이터레이션(iteration)', '스프린트(sprint)' 등이 이러한 방식을 따르는 대표적인 실천법이다.

백로그 아이템
Backlog Item

하나의 사용자 스토리를 시작할때 해야할 일을 액티비티 단위로 나눈 것이다.

사용자 스토리
User Story

실제 사용자에게 정말 가치있는 기능을 간단명료하게 기술한 것으로 스크럼의 제품 백로그(Product Backlog)와도 유사하다. 결국 고객의 요구사항을 효과적으로 명세하기 위한 실천법이다.

소멸차트
Burn-down chart

번다운 차트라고도 한다. 업무 시작시에 계획했던 일들의 총합을 초기값으로 정하고, 매일매일 초기값에서 완료된 작업을 차감하여 추이를 나타내는 차트. 시간의 흐름에 따라 값이 점점 작아진다고 하여 소멸차트라고 부른다.

속도
Velocity

팀이 주어진 기간(스프린트) 내에 완료 할 수 있는 업무의 양으로 단위는 팀에서 추정한 스토리 포인트가 된다.

스크럼 마스터
Scrum Master

스크럼 팀의 주요 역할자로 팀이 완전히 생산적이고 기능적이 되도록 보장한다. 즉, 외부의 간섭과 방해로부터 팀을 보호하고 프로젝트 내 이슈 및 문제 해결을 적극적으로 지원는 등 애자일 프로세스가 준수되도록 보장한다.

스토리 포인트
Story Point

사용자 스토리의 크기를 나타내는 단위이다. 즉, 사용자 스토리를 구현하는데 소요되는 비용으로 릴리즈 계획 및 승인 테스트 계획의 토대가 된다.

스프린트
Sprint

이터레이션과 같은 의미로 스크럼에 기초한 반복점증적 개발을 위한 기본 단위이다. 스프린트 계획에 따라 분석/설계/코딩/테스트 등이 모두 하나의 스트린트 안에 이루어질 수 있다.

스프린트 계획
Sprint Planning

제품 책임자, 스크럼 마스터 포함 팀 전체가 모여 다음 번 스프린트 동안 개발해야 할 기능 목록을 정의하고 관련 세부 계획을 세우는 회의이다.

스프린트 리뷰
Sprint Review

반복주기가 끝날때 마다 팀이 완료한 사용자 스토리를 고객 및 이해 당사자에게 데모하고 다양한 피드백을 받는 실천법이다.

스프린트 소멸 차트
Sprint Burn-down chart

스프린트 내 남아 있는 업무량 추이를 보여주는 차트이다. 팀이 프로젝트 진행현황을 하루 단위로 공유하고 상황에 따라 민첩하게 대응하기 위한 자료로 활용된다.

완료정의
Definition of Done

작업이 종료되었다는 것을 정의한 기준. 개인 별 완료기준이 다르면 팀 수준의 결과물의 완성도를 저하시킨다. 팀이 합의하는 작업의 완료 기준을 의미하며, 작업을 계획할 때 완성되었을 때의 모습을 구체적으로 서술하도록 한다.

워크플로
Work flow

개발 프로젝트에서 단계별 작업흐름을 지칭한다.

일일 스크럼
Daily Scrum

매일 팀원 모두가 정해진 시간에 팀의 작업 현황판 앞에 모여 15분 간 진행하는 회의로 참여자는 "어제 했던 일, 오늘 할 일, 애로사항"을 공유한다. 스크럼(Scrum)의 대표적인 3 가지 활동 중 하나이다.

자기조직화
Self-organizing

하나의 팀이 한 두 사람에 의해 움직이거나 개발이 진행되는 것이 아니라, 팀 구성원 각자가 능동적으로 업무를 찾고 문제를 해결해 나가는 식으로 움직이는 자율적인 팀의 모습을 지칭한다.

작동하는 소프트웨어
Working Software

(스프린트 리뷰에서) 스프린트 기간 중에 작업한 내용을 실제 데모할 수 있는 결과물을 의미한다. 애자일 선언문에서 언급하고 있는 애자일의 핵심 가치 중 하나이다.

작업 현황판
Task board

팀 내 프로젝트 진척상황을 한 눈에 알아볼 수 있도록 만든 차트이다. 팀원 누구나 쉽게 접할 수 있는 곳에 크게 붙여 놓는다. 대개 팀의 할일, 이벤트, 위험요인, 휴가계획 등의 정보가 게시된다. 큰 시각적 차트(Big Visible Chart)라고도 부른다.

작은 릴리즈
Small Release

실행 가능한 모듈을 실제 환경에 가능한 빨리 적용하는 것을 목표로 한다. 고객이 소프트웨어가 어떻게 작동하는지 최대한 짧은 기간에 볼 수 있도록 짧은 주기로 업데이트된 모듈을 릴리즈 하는 것을 말한다.

제품 백로그
Product Backlog

우선순위가 매겨진 요구사항 목록으로 가장 높은 가치를 맨 위에 둔다. 제품 책임자가 우선순위를 결정하고 팀에 의해 추정된 사용자 스토리로 표현된다.

제품 책임자
Product Owner

스크럼 팀의 주요 역할자로 제품의 특성과 기능을 정의하고 출시 일자와 내용을 결정하며 제품의 수익성(ROI)에 대한 책임을 진다. 시장 가치에 따라 구현할 특성과 기능에 우선순위를 매기고 제품 백로그에서 특성 및 기능, 우선순위를 변경할 수 있다. 작업의 결과물을 승인 또는 거절하며 비즈니스 부서와 커뮤니케이션한다.

지속적인 통합
Continuous Integration

팀이 작업한 내용을 지속적으로 통합하는 실천법이다. 최소 하루에 한번 이상, 매일 여러 번의 통합이 이루어지는 것을 권장한다. 자주 통합하고 검증함으로써 최신 코드가 항상 건강한 상태인지 확인할 수 있으며, 통합 주기를 짧게 가져감으로써 오류 발생 시 원인 파악을 신속하게 할 수 있다.

짝 프로그래밍
Pair Programming

한 컴퓨터에서 두 명의 프로그래머가 협력하며 완성하는 프로그래밍 방식이다.

추정
Esitmation

사용자 스토리를 구현하는데 있어 그 규모가 어느 정도인지 생각해 보는 과정이다. 보통 'man-hour', 'man-day' 같은 시간 단위보다는 스토리 포인트와 같은 상대적인 크기를 통해 표현한다.

테스트 주도개발
TDD

개발 기법의 하나로 개발을 진행하기에 앞서 테스트 코드를 먼저 만들어 놓고 해당 테스트 코드를 만족하는 업무코드를 작성하는 식으로 진행한다. 결과적으로 테스트가 개발을 이끌어가는 방식이 되기때문에 Test-Driven Development(TDD)라고 불린다

속도
Velocity

애자일 팀이 일정 기간 동안 완료한 사용자 스토리에 대한 스토리 포인트의 총합이다. 남아있는 사용자 스토리, 리소스, 출시일 등의 적합성 여부를 예측 및 판단하는데 활용 가능하다.

퍼실리테이터
Facilitator

회의의 목표를 벗어나지 않고 매끄럽게 진행되도록 참가자들의 관심과 참여를 유도하는 역할자이다.

플래닝 포커
Planning Poker

사용자 스토리의 규모를 추정하는 방식이다. 전통적인 방식과 달리 한 사람이 주도적으로 추정하는 것이 아니라 팀 전체가 함께 지혜를 모아 업무량을 추정하고자 카드를 이용하여 계획을 하는 게임이다.

현장 고객
On-site Customer

고객이 개발팀과 같은 장소에서 함께 개발에 참여하는 실천법이다. 개발팀의 결과물에 대해 고객이 신속하게 피드백을 해줌으로써 업무 효율을 높일 수 있다. 실제 최종 사용자나 구매자를 접촉하기 어렵다면 조직 내 고객 역할을 담당할 내부 고객을 지정할 수 있다.

회고
Retrospective

프로젝트의 끝에 행해지는 의식으로, 그 간의 일들을 돌아보고 경험했던 것을 통해 학습하고 더 나아지기 위한 변화를 계획하는 일련의 활동을 말한다.

참고문헌

1. 애자일 선언문 원본링크
   Manifesto for Agile Software Development, http://agilemanifesto.org/

2. 애자일 소프트웨어 개발에 대한 한국어 Wikipedia
   애자일 소프트웨어 개발, http://ko.wikipedia.org/wiki/애자일 소프트웨어 개발

3. 익스트림 프로그래밍 (인사이트 / 켄트 벡)

4. 린 소프트웨어 개발의 적용 (위키북스 / 메리 포펜딕, 톰 포펜딕)

5. 스크럼: 팀의 생산성을 극대화시키는 애자일 방법론 (인사이트 / 켄 슈와버, 마이크 비틀)

6. 애자일 회고 (인사이트 / 에스더 더비, 다이애나 라센)

7. 스크럼(인사이트 / 켄슈와버)

8. 사용자 스토리 (인사이트 / 마이크 콘)

9. 스크럼과 XP (인사이트 / 헨릭 크니버그)

10. 불확실성과 화해하는 프로젝트 추정과 계획 (인사이트 / 마이크 콘).

11. 지속적인 통합 (위키북스 /폴 M 듀발, 스티븐 M 마티야스, 앤드류 글러보)

12. 애자일 도입 성공요인 분석
    애자일 실천법 세미나 (김창준), http://agile.egloos.com/5299932

13. The Costs and Benefits of Pair Programming (Alistair Cockburn, Laurie Williams),
    http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF

14. Paired Programming in the Software Factory Questions and Answers
    (Menlo Institute / Thomas Meloche, James Goebel and Richard Sheridan)

15. Pair Programming Illuminated (Addison-Wesley Professional / Laurie Williams)

16. An anatomy of a retrospective, http://www.retrospectives.com/pages/Anatomy.html

애자일 SW개발 101

독자에게 한마디

하얀 눈 위에 발자국은 반갑고 친구와 같은 느낌이었던 것 같습니다. 애자일 적용에 첫발을 내딛으려는 팀에게 도움이 되었으면 좋겠네요.

경기원, LG CNS 전문기술교육팀

애자일을 통해 내가 하고 있는 무엇인가를 더 낫게 만들려는 욕심 있는 분들께 추천합니다.

신황규, 삼성SDS 기술혁신팀

현실을 부정하지 말고 일단 작게 시작하세요.
관찰하고 적응하세요. 특정 방법에 얽매이지 말고 애자일 선언문을 기억하세요. :-D

심우곤, LG전자 Agile개발팀
http://wgshim.com

어제보다 나은 오늘을 만드는데 도움이 되었으면 좋겠습니다.

이승룡, 삼성SDS 기술혁신팀

애자일을 적용하려는 팀에 도움이 되었으면 좋겠어요!

채수원, NHN 기술전략팀
http://blog.doortts.com

여러분들이 애자일과 더 친해졌으면 합니다.

황상철, NHN 기술전략팀
http://pragmaticstory.com

애자일SW개발 101

공동집필 | 경기원, LG CNS 전문기술교육팀
신황규, 삼성SDS 기술혁신팀
심우곤, LG전자 Agile개발팀
이승룡, 삼성SDS 기술혁신팀
채수원, NHN 기술전략팀
황상철, NHN 기술전략팀

기획/총괄 | 이세영, 정보통신산업진흥원 부설 SW공학센터
편집/디자인 | 이경진

주관/후원 | nipa 정보통신산업진흥원

발행일 | 2013년 1월 7일

애자일 SW개발 101

정보통신산업진흥원 부설 nipa 소프트웨어공학센터 서울시 마포구 상암동 1605번지 누리꿈스퀘어 연구개발타워 13층 Tel. 02-2132-1338 Fax. 02-2132-1358 http://software.kr/