What Is Skinny Agile?
Skinny 애자일이란?

Keith Hogan ***(PMI Florida Suncoast Chapter)*** - May 7, 2018

Topics: **Agile, Knowledge Shelf**

Agile 은 객체지향 언어와 XP(Extreme Programming)과 같은 반복적 개발의 변이를 사용하는 17 명의 전문 개발자에 의해 정의된 코드 개발 방법론입니다. 그들은 수준 높은 소프트웨어 제품을 생산할 뿐만 아니라, 좋은 근무환경을 만드는 4 개의 주요 가치와 12 가지의 원칙을 만드는데 협력하였습니다. 그 결과가 "*Agile Manifesto for Software Development* (Beck et al., 2001)"입니다.

Agile 을 채택 할 때, 경영자는 일부 기능을 불필요하거나 비효율적이라고 반대할 수도 있지만 Agile 에서 권장되는 일부 기능의 채택을 승인할 것입니다. 그런 다음 채택된 기능으로 구성된 축소 또는 "skinny" 버전의 Agile 을 구현합니다. 그런 다음 기존 IT 직원으로 구성된 팀을 구성하고 그들에게 대개 Agile 입문자 교육을 제공합니다. 그런 선형 순차적 프로젝트에 익숙한 직원이 Agile 을 배울 때는 그들의 선형 순차적 경험의 시각으로 그것을 볼 수 있습니다. 따라서 전통적인 환경에 선택적으로 도입된 Agile 의 어느 특정 기능은 완전히 실패한 것으로 보입니다. 사실, Agile 의 각 기능은 견고하고 많은 이익을 동반하는 비용 절감 방법을 만들기 위해 다른 기능과 함께 작동합니다.

새로운 팀은 각각의 스프린트에서 전문가들로 구성되어 있기 때문에, 분석, 디자인, 개발 그리고 테스트를 하는 전통적인 방식의 개발 활동을 진행합니다. 이것은 종종 미니 워터폴이라고도 합니다. 그것은 나쁜 방법은 아닙니다. 단지 새로운 Agile 팀의 시작 단계입니다.

"skinny" Agile 버전은 아래의 Agile 특성과 함께 스프린트가 도입되는 부분입니다.

- 사용자 스토리
- 백로그 정제
- 스프린트 계획
- 데일리 스탠드업
- 스프린트 칸반 보드
- 소멸차트와 팀 벨라서티
- 스프린트 회고

짝 프로그래밍, 테스트 주도 개발(TDD), 지속적인 통합(CI), 동일장소 근무 팀 등의 특성은 제외되었습니다.

Skinny Agile 의 장점

Skinny Agile 은 일일 상황 체크, 신속한 방해물 제거, 정보 방열기(칸반 보드)를 활용한 의사소통 개선 그리고 팀 상호간 지식교류 등의 이점이 있습니다. 가장 중요한 것은 "팀"사고 방식이 개개의 승패 사고방식을 대체한다는 것입니다. 회고 단계에서는 프로세스에 적용 할 수 있는 개선 사항을 식별합니다. 데일리 스탠드업은 팀원들을 당일 계획 한 업무에 집중하게 하고 진행을 저해하는 이슈를 제기하며 정보 방열기 (스프린트 간판 보드)를 반영한 일일 상황 보고서를 경영층에 제공합니다. 나는 이것이 변경 요청 목록을 처리하는 유지보수 팀에서 성공적으로 사용되어지는 것을 본 적이 있습니다.

그것은 나쁜 시작이 아니며, 생산성 향상을 가져올 것입니다.

그것은 다급한 개발자가 명확한 요구사항들(schedule over quality) 대신 추측을 활용하여 일을 진행할 때, 발생된 결점을 제거하지는 않습니다. 이것은 누군가의 질문에 즉시 대답할 수 있도록 부르면 들리는 거리에 각 구성원들이 있는 동일장소 근무 팀을 활용함으로써 개선됩니다. 그래서, 어떤 팀원이 토의되는 이슈를 들을 때, 그들은 해결책을 알수도 있을 것입니다.

다른 이점으로는 예산과 납품일의 정확성입니다. 더 빠른 납기는 선형 순차방식(Waterfall)의 다음단계로 일이 진행될 때, 핸드 오프를 제거함으로써 실현될 수 있습니다. 그러나, 결함은 일이 선형 순차적으로 진행될 때, 그 선형 체인을 따라 정보를 전달해야 하므로 발생합니다. 그것은 또한 몇몇 소수의 사람들이 여전히 실제 코딩의 전반적인 부분을 수행하기 때문에, 개발자 pool 의 크기는 증가하지 않습니다. 이러한 소수의 사람들이 훌륭히 수행할 수도 있지만, 새로운 기능의 제공을 지연시키는 병목현상을 유발합니다. 또한 단위 테스트 프로세스의 속도를 향상시키지 못하며, 시스템 통합 테스트를 개선하거나 결함을 줄이지도 못합니다.

"우리는 매우 빠르게 변화하는 환경에 처해 있습니다."라는 말을 들을 때, 그것은 비즈니스에 필요한 변경 사항을 구현할 때 IT 가 병목 현상임을 알려줍니다. 이는 비효율적인 프로세스에 기인한 것 같습니다.

Skinny Agile 의 비효율성

Skinny agile shops 은 프로세스의 낭비요소를 제거하려고 노력하는 lean shops 과 혼동해서는 안됩니다. Skinny agile shop 은 일반적으로 비즈니스 분석가 (BA), 데이터베이스 애널리스트 (DBA), 개발자, 품질 보증 (QA) 및 사용자의 전통적인 역할을 유지합니다. 따라서 그들은 한 전문가에서 다른 전문가로 작업을 전달합니다. 팀 구성원들은 동일장소에 배치되지 않으며 인수인계를 위한 준비 회의를 할 때 지연이 발생되며, 더한 지연은 프로세스에서 다음 스킬이 이용가능 할 때까지 기다리는 대기시간에서 발생합니다. 결함은 중요한 정보가 그 체인을 통해 전달될 때 발생됩니다. 사용자 요구사항 변경과 같은 정보가 손실되거나 오해 된 경우 나중에 품질 보증 또는 사용자 승인 테스트 (UAT)에서 수정해야하는 결과를 초래합니다. 프로세스 개선은 비효율적인 프로세스로 마감일을 맞추려 노력하는 바쁜 일정과, 매우 빠른 속도로 진행되는 일정으로 인해 생략됩니다.

팀원 수를 늘리면 팀 구성원 간의 의사소통 경로가 기하급수적으로 증가한다는 것을 기억하십시오. 따라서 분산 된 팀에서의 인원증가는 수익의 감소를 가져옵니다.

증상과 해결책
증상: 대부분의 스토리 포인트는 소수의 개발자에 의해 제공됩니다.
해결책: 각각의 개발자가 다른 팀원과 짝을 이루도록 개발자들을 쌍으로 엮어 주십시오. 이러한 행위는 개발자간 기술 수준을 향상시키고 코드 구성 및 명명 규칙을 표준화 합니다. 기술은 팀 구성원간에 전수됩니다. 속담에서 말하는 것처럼, 좋은 것은 더 나아지고, 더 나아진 것은 최고가 됩니다. 페어링은 모든 새 코드 개발에서 팀 구성원들에 의해 모든 코드를 검토하기 위해 권장되는 것입니다. 스프린트에서 페어링은 심지어 일부 시간이라도 이익을 가져다 줄 것입니다. 그리고 짝을 이루어 작업하는 것은 의사소통의 경로를 절반으로 줄어들게 합니다.

증상: 품질보증에서 하자가 많은 경우
해결책: Skinny Agile 을 도입할 때, 만약 여전히 별도의 품질보증 검사자를 운영한다면, 새로운 기능에 대한 작업을 수행할 때, 품질검사자와 개발자를 페어링 하십시오. 새로운 기능에 대한 작업이 끝나면 다시 페어링 하십시오. 사용자가 산출물을 사용해보면서 개발 중에 여러 변경사항들이 있음을 기억하십시오. 이러한 변경사항들은 새롭거나 또는 수정된 "요구사항"이 됩니다. 품질은 요구 사항에 대한 적합성으로 정의되기 때문에 품질 보증 담당자가 이 사실을 알아야 테스트 케이스를 준비 할 수 있습니다.

증상: 시스템 통합 테스트에 많은 결함이 있는 경우
해결책: 새로운 코드를 완성하여 통합 테스트를 통과시키거나 또는 실패하여 코드를 삭제할 때 새로운 코드를 다음 환경에 잘 융합시키는 지속적인 통합을 구현하십시오. 통합 테스트는 이전에 프로그래밍 된 단위 테스트에 새로운 코드로 된 신규 단위를 추가하여 실현하는 것으로 구성됩니다.

증상: 복잡한 기능으로 개발이 너무 오래 걸리는 경우
해결책: 코드가 개발되면 단위 테스트를 위한 코드도 개발해야 합니다. 그러면, 추가 코드가 더해질 때 기존 단위 테스트를 신속하게 다시 실행할 수 있습니다. 코딩하기 전에 요구 사항을 알아야 하므로 테스트 방법도 알고 있어야 합니다. 먼저 테스트들을 작성하십시오. QA, Dev 및 BA 가 필요한 테스트 케이스가 무엇인지를 알고 있는지 확인하십시오. 만약에 화면 형식과 같은 주관적인 문제인 경우 사용자가 확인하고 승인 할 수 있도록 신속하게 프로토 타입을 작성한 다음 (새로운 요구 사항이 됨) 사양에 맞게 코드를 작성하십시오. 루프에 QA 를 유지하십시오. 페어링으로 작업한 몇차례의 스프린트를 통해 QA 는 단위 테스트 케이스를 코딩하는 데 도움을 줄 수 있습니다. 코딩이 완료되면 개발자는 별도의 검사자가 검사할 필요없이, 필요할 때 마다 언제든지 테스트할 수 있습니다. 심지어 개발자와 검사자는 짝을 이루어 진행할 수도 있습니다. 스크럼 팀은 자기 조직화가 이루어지기 때문에 필요에 따라 페어링을 사용할 수 있습니다.

증상: 회의가 너무 많은 자료를 포함하고 있어, 모든 참석자에게 대부분 관련이 없는 경우
해결책: 대규모 응용 프로그램의 경우 에픽 및 기능을 여러 스크럼 팀이 수행 할 수 있는 합리적 그룹으로 나누고, 각 스크럼의 구성원을 최대 15 명으로 유지하십시오. 스크럼 중에 한 스크럼은 팀

Activity 를 조정하기 위해 활용될 수 있으며, 각 팀의 몇몇 대표로 구성할 수 있습니다. 스탠드 업은 15 분으로 제한된다는 것을 기억하십시오..

증상: UAT 에 결함이 너무 많은 경우
해결책: 요구 사항이 잘못 이해되었습니다. 기능 검토와 같이 프로세스에 사용자를 더 빨리 참여시켜야 합니다. 또한 개발 중에 발생된 변경사항을 전달해야 합니다. 3 가지 이상의 UAT 가 요구되어서는 안됩니다. 그 후에 사소한 결함이 백 로그에 들어갈 수 있습니다

증상: 책임완수를 위해 상당한 초과근무가 발생하는 경우
해결책: 팀의 사용 가능한 시간(용량)을 스프린트의 작업과 매치시키고 장애물을 매일 제거하도록 주의를 기울이십시오. KPI (Key Performance Indicator)의 일환으로 실제 일한 시간을 미래 예측시간의 정확도를 높이는 포인트로 매칭하십시오.

증상: 미스커뮤니케이션으로 인한 결함
Agile 이라는 용어는 많은 방법론에 적용됩니다. 일반적인 대화에서 동일한 단어를 말하고 있지만, 뜻은 완전히 다를 수 있습니다.
해결책: 새 기능을 시작할 때 BA, 개발자 및 QA 간의 동일한 이해에 도달 할 때까지 요구사항을 자세히 검토하십시오.

청구가능 시간
최근에 프로젝트에서 새로운 전환이 나타났습니다. 고객은 사용자 스토리 개발 이외의 다른 작업에 대한 청구를 허용하지 않는다는 조건을 명시했습니다. 어떠한 기반시설이나 결함 수정도 포함되지 않는다는 것입니다. 그들은 이러한 것들이 비용이 된다고 생각했고, 그것은 IT 가 예산을 반영하거나 부담해야 된다고 생각했습니다.

저는 개인적으로 IT 가 더 높은 품질의 코드를 제공하도록 유도하기 때문에 이것을 좋아합니다. 또한 이것은 증진과 실제 결함을 차별화해야 함을 의미합니다. 증진된 부분은 청구 가능합니다. 또한 그것은 IT 가 TDD (Test-Driven Development)를 규정할 때 개발 프로세스의 한 부분으로써 단위 테스트 케이스를 코딩하도록 권장합니다. 회귀 테스트는 TDD 자동화 테스트 케이스를 사용하여 수행할 수 있습니다.

더 적을수록 더 효과적
관여하는 사람이 적으면 적을수록 커뮤니케이션 오버헤드가 줄어듭니다. 프로세스에서 인수인계가 줄어들수록 재작업이나 결함이 줄어듭니다.

만약에 당신의 15 명 팀이 초기 요구 사항 (사용자 스토리)에서 실행까지 인수인계 없이 전체 프로세스를 수행할 수 있다고 상상해보십시오. 결과물은 유동적으로 움직일 것입니다. 이것을 각 스테이션이 제품을 다음 스테이션으로 넘겨주는 전문가 스테이션으로 구성된 조립 라인과 비교하여 보십시오. 이제 각 팀 구성원이 프로세스에 필요한 어떠한 단계도 수행할 수 있다고 상상해보십시오. 이러한 단계에서, 다음 가용 팀은 정해진 우선 순위 목록에서 다음 임무를 수행할 수 있습니다.

전문가를 기다릴 필요가 없습니다. 이는 항공사와 은행이 고객이 한줄로 서서 대기할때, 고객을 처리하는 방식과 마치 유사합니다. 이는 사용하기에 가장 효율적인 큐잉 방법임이 입증되었습니다.

프로젝트 관리 프로세스
소프트웨어 개발 라이프 사이클 (SDLC) 워터폴은 초기 단계에서 종료 단계까지가는 작업 분류 체계 (WBS)를 사용합니다. 그것은 가능한 1 년 또는 그 이상의 장래의 기능개발을 예측하는 것이기 때문에 실패합니다. 그것은 1 년 넘게 발생할 많은 변화를 설명하지 못합니다. 새로운 기능이나 수정된 기능에 유연하게 적용되도록 만들어져야 합니다. 우리는 모든 것을 한꺼번에 전달하는 빅뱅 방식에서 단계별 접근방식으로 그리고 반복적 접근 방식으로 지금은 연속적인 기반의 변화에 적응하는 Agile 접근방식으로 진보해왔습니다.

두 방법 모두 서로 강화하는 구성 요소가 있는 완전한 프로세스입니다. Agile 은 모든 에픽 및 기능의 계획에서 구현에 이르는 장기 WBS 를 대체합니다. 시작 및 종료 단계는 프로젝트 관리 지식 체계 (PMBOK® 지침) - 제 6 판 (PMI, 2017) 안내서의 프로세스 그룹에 포함되어 있습니다. 모든 프로젝트는 확실한 끝이 있습니다. Agile 은 끝이 없는 프로세스이며, 프로젝트마다 임무와 기능의 백로그를 반복하여 계속합니다. 기능은 하나 또는 여러 프로젝트에서 제공될 수 있습니다. Agile 팀은 한 프로젝트에서 다음 프로젝트로 함께 남아 있어야하며 시간이 지남에 따라 더 좋아질 것입니다.

따라서 프로젝트는 일반적으로 매출 전망에 의한 합당한 비즈니스 케이스에 의해 뒷받침되는 비즈니스 제품을 구성하는 상호 의존적이고 관련된 기능 세트입니다. 일부 프로젝트는 비용을 최소로 유지하면서 법적 또는 규제 요구 사항을 충족시키기 위해 정당화될 수 있습니다.

나무 분쇄기
Agile 팀을 뒤뜰에서 임대 나무 분쇄기를 운영하는 팀으로 간주하여 보십시오. 시간당 임대료와 연료비가 들게 됩니다. 그것을 사용하려면, 당신은 분쇄기를 통과하기에 충분히 작은 크기로 나무를 자르고, 조각 내어야 합니다. 그리고, 당신은 제한 비용까지 그것을 계속 사용하고 싶어합니다. 이전의 조각들이 분쇄되는 동안 새로운 조각을 준비하여 분쇄기의 사용을 극대화 합니다. 해당 나무는 가지가 조각으로 잘라져 분쇄기 근처에 준비되어져 있어야 합니다. (Agile 에서 이것은 백로그 정리). 분쇄 처리하는 동안 분쇄기의 처리속도 보다 앞서 있기 위해서는 더 많은 목재를 자를 수 있습니다. 이런 식으로 여러 개의 나무들을 분쇄할 수 있습니다.

세명의 행복한 이해관계자
만약 당신이 Skinny Agile 환경에 있다면, 다음 단계로 넘어가면 작업 환경이 크게 좋아질 것입니다. 시간 외 근무, 비상 사태 해결, 새로운 코딩 및 결함 수정이 줄어 듭니다. 요구되는 납품일에 대한 예측은 더 신뢰성을 가질 것입니다. 1. 행복한 고객, 2 행복한 직원 그리고 3. 행복한 경영자가 전제될 때, 고객은 신제품 출시를 미리 계획 할 수 있으며 새로운 소프트웨어 기능들은 사용 가능할 것입니다.

변화하려는 동기

변화에 대한 동기는 일정과 예산 계획이 반복적으로 지켜지지 않음으로써 이것이 제품 출시 지연과 사업 매출 손실을 유발할 수 있으므로 제기되었습니다.

다음은 Skinny Agile 프로세스를 향상시키는 데 사용할 수 있는 프로세스 변경 목록입니다:

- 짝 프로그래밍
- 동일장소 근무
- 제한된 팀 크기
- 풀타임 generalists vs. 매트릭스 운영 specialists
- 테스트 중심 개발
- 지속적인 통합
- 축소된 인수인계

기업 IT 조직에서 프로젝트 관리 vs. Agile

하나가 다른 하나를 대체한다는 생각은 맞지 않습니다. 프로젝트는 비즈니스 케이스를 기반으로 시작되며 예상 수익으로부터 도출된 납기일과 예산에 대한 기대 비용이 있습니다. 비즈니스 케이스는 실현되고 제품의 기대수명이 연장되기 시작하는 프로젝트로부터 예상되는 가치를 명시합니다. 완전한 제품이 생산되면, 프로젝트는 끝나고 종료됩니다. 프로젝트는 종종 기존 비즈니스 프로세스 및 설비를 변경하는 작업을 포함하며 하드웨어와 소프트웨어를 통합하는 것 일수도 있습니다.

프로젝트는 IT 소프트웨어 개발과 하나 이상의 IT 팀에서 수행 할 수있는 다른 IT 서비스를 필요로 할 수 있습니다. 스크럼 팀이 이미 존재할 수 있으며 그들의 백로그에 새로운 프로젝트 기능을 추가함으로써, 그 프로젝트 작업을 착수할 수 있습니다. 백로그에는 심지어 한 개 이상의 프로젝트에서의 작업도 포함될 수 있습니다. 이것은 초기단계에 팀 구성과 모집에서 오버헤드를 피할 수 있습니다.

사실 Agile 팀이라는 아이디어는 일반 IT 직원으로 구성된 팀이 지속적 개발 활동을 수행하고 짧은 개발 기간 (일반적으로 약 2 주) 내에 업무를 완료하도록 일정을 세우고 백로그의 작업을 수행하는 것입니다. 비즈니스 케이스를 개발하거나 예산을 관리하거나 비 IT 프로젝트 작업을 수행하지 않습니다. 한 프로젝트에서 IT 개발 작업이 완료되면 다른 프로젝트의 작업이 백로그에 추가될 수 있습니다. 그 팀을 지속 가동하여 당신이 분쇄 공급장치에 충분히 작은 크기의 나무 조작을 넣어주도록 기다리고 있는 나무 분쇄사업자로 생각하여 보십시오.

Agile 팀은 대개 웹 사이트의 IT 작업 (전체 스택)을 완료하는 데 필요한 모든 기술을 가진 IT 직원 그룹으로 구성됩니다. 초기에 그들은 BA, DBA, 개발자, QA 테스터, 사용자, 완성 된 코드 테스트를 수행하는 QA 와 같은 전문가들일 것입니다. 그들은 초기 스프린트에서 정의, 개발 및 테스트의 각 단계별로 각각의 기능을 이행합니다. 시간이 지남에 따라 크로스 트레이닝을 통해, 각 구성원은 솔루션을 설계하고 솔루션 검증에 필요한 테스트를 결정하며 단위 테스트를 코드화하고 솔루션을 개발하고 테스트를 실행하며 사용자와 솔루션을 검토하고 승인된 솔루션을 생산 준비 코드 기반에 통합합니다 (이전 및 현재 프로그래밍 된 테스트 케이스의 회귀 테스트 통과).

이는 다음 사용가능한 서버(개발자 쌍)가 백로그의 다음 작업을 수행하고 통합 테스트를 통해 생산 준비 코드 기반으로 옮길 수 있는 작업 대기열을 처리하는 가장 효율적인 방법입니다. 이것은 누락과 실수가 자주 발생하는 프로세스의 모든 인수인계를 제거합니다.

개발자는 일종의 소유권 기능을 주입하는 특정한 코드에 영구적으로 할당되지 않습니다. 울타리는 없습니다. 모든 개발자는 필요에 따라 코드의 모든 영역에서 작업합니다. 이것은 크로스 트레이능 및 지식 전달에 도움이됩니다. 그것은 개발 된 코드에서 햇살을 비추는 것입니다. 백도어는 없으며 코드는 표준을 준수합니다. 팀 구성원들은 팀 중심 사고체계의 팀으로 융합되어 함께 책무를 만들고 완수해냅니다. 각 구성원은 다른 구성원을 지원할 수 있습니다. 팀은 함께 있습니다.

이러한 Agile 팀은 하나 이상의 기존 생산 시스템에 대한 지속적인 유지 보수 및 개선 사항을 제공하는 데 활용될 수 있습니다. 또한 그들은 프로젝트의 한부분으로써, 새로운 애플리케이션에 대한 백로그를 제공받을 수 있습니다.

이는 마케팅 및 고객 서비스에 중요한 고객 웹 포털과 같이 지속적인 코드 개발이 필요한 기존 회사에 이상적인 Agile 설정입니다.

계약 환경에서의 Agile

기존의 Agile 팀이 없는 경우, 계약자와 핵심 인력은 프로젝트에 의해 할당되고 개발 수행에 배치됩니다. 여기서 학습 곡선과 팀 구성의 오버헤드가 도입됩니다. 그것은 프로젝트 예산의 추가 비용입니다. 그것은 당신의 회사가 최소한의 유지보수 직원만 유지하고 고정적인 Agile 개발 팀을 보유할 여력이 없는 경우 필요할 수 있습니다. 해마다 IT 직원을 필요로 하는 많은 프로젝트를 수행하는 회사의 연간 비용을 고려하십시오. 매트릭스 관리 IT 기술은 자원을 기다리는 대기시간을 유발합니다. – 그리고 그것은 하나가 늦게 시작하거나 완료 할 때 연쇄반응을 합니다. 이러한 비용과 지속적으로 운영하는 Agile 팀을 보유하는 것에 대비해 이러한 비용을 견줘보십시오.

Agile 프로세스는 어디에서 끝나나요?

Agile 팀이 개발을 통해 생산을 진행할 경우, 통합 테스팅과 생산 실행을 위한 추가적인 인수인계가 방지됩니다. 이 경우 Agile 팀이 새 코드를 생산 운영 지원 팀으로 넘기면 종료됩니다. 생산 단계로의 이관은 알 필요가 있는 모든 사람들에게 다가올 변화를 알 수 있도록 해야합니다. 이 단계는 프로젝트 매니져나 생산 지원 역할에 대해서 잘 알고 있는 별도의 "DevOps" 팀에 의해 수행될 수 있습니다. 보증 기간은 다음 스프린트의 일부으로 계획됩니다. Agile 팀은 생산 과정에서 발생할 수 있는 결함을 조사하고 해결할 준비가되어 있어야 합니다. 이것은 결함이 다음 스프린트와 출시를 위한 코드로 전달되는 것을 방지하는 데 중요합니다. 생산에서의 결함은 파급 효과가 분석되고 우선 순위를 정할 때까지 알 수없는 높은 위험요소를 지니고 있습니다.

Lean Agile

"린 (lean)"생산 시스템은 프로세스를 통해 제품을 가져옵니다. 제품이 납품이 완료되면 더 많은 제품이 배송 단계로 넘어가서 다시 프로세스의 시작 단계로 돌아갑니다. 이것은 낭비가 적고 재고가 적다는 장점이 있습니다. 소프트웨어 개발에서 계획에서 개발 설계로 이동하는 것은 힘들고 일정

잡이가 될 수 있습니다. 원하는 신제품에 대해 모든 것을 개념화하고 프로토 타입이나 심지어 프로토 타입을 여러 번 반복하지 않고 문서화하기는 어렵습니다.

Agile 은 소프트웨어에세 이것을 가능하게 합니다. 개발 팀은 마케팅 제품 개발과 같이 기획 및 설계 팀으로부터 설계 결정을 가지고 옵니다. 필요할 때 이것을 한 번에 가져오는 것은 매우 절감된 개발 방법입니다. 그것은 모든 정보를 사용할 수 있는 가능한 최신의 시점에 결정을 하는 것입니다. 마케팅 팀은 웹 페이지를 완성하기 전에 몇 차례 반복하여 화면 디자인을 보고 평가하기를 원합니다. 다른 한쪽에서는 새로운 세금 코드에 의해 규정된 회계 변경은 의견의 대상이 아닌 최종 요구 사항입니다. 테스트에서, 그것은 정확하거나 틀린 것입니다. 이것은 주관적인 것이 아니며 반복 접근을 요구하지 않습니다.

임베디드 소프트웨어에서의 Agile

정해진 일정과 예산으로 납품할 수있는 Agile 개발의 신뢰성은 회사 기획을 가능하게합니다. 제품 도입 계획에서 불확실성이 있습니다. 이것만으로도 회사에 가치를 제공합니다. 새로운 모델을 출시하려는 자동차 제조업체의 계획에 대한 일정의 중요성을 고려하십시오. 통합된 소프트웨어의 개발과 함께, 많은 Agile 팀은 여러 단계의 통합화와 요구된 사용자 승인 테스트와 함께 작동하고, 대형 시스템에 들어가야만 하는 하위 시스템에 코드를 제공합니다. 이것은 여러 단계의 복잡한 프로세스입니다.

Agile 팀은 생산단계에까지 길어진 추이를 발견했을 때, 결함을 제거하기 위해 계획 수립할 필요가 있습니다. 생산단계에서 코드가 성공적으로 작동 할 수 있도록 일부 기능이 이러한 용도로 사용 되어지도록 계획해야 합니다. 자동차 리콜은 매우 많은 비용이 듭니다. 제품을 사용할 때 결함이 발생되지 않도록 주의해야 합니다. 예를 들어, 임상실험 정보 시스템에서 위양성 결과를 초래하는 결함은 높은 재정적 위험 및 법적 위험뿐만 아니라 생명을 위협하는 결과를 초래할 수 있습니다.

성숙한 Agile

축소 또는 "Skinny"버전으로 시작하게 되겠지만, 지속적인 개선을 통해서 그것을 더 포괄적인 Agile 방법론으로 성숙시킬 것입니다. Agile 회고는 어떻게 다른 Agile 특성들이 프로세스를 향상시킬 수 있는지 밝혀 줄 것입니다. Agile 의 모든 특성이 전반적인 방법론을 지원하기 때문에 그렇습니다. Agile 은 머물러 있지 않고, 지속적으로 개선합니다. 우리의 기술이 발전됨에 따라 Agile 또한 향상됩니다. 똑똑한 사람들로 구성된 팀과 지속적인 개선 프로세스는 비즈니스를 지원하는 데 필요한만큼 프로세스를 개선하고 맞추어 나갈 것입니다.

References

1. Beck, K., Beedle, M., van Bennekum, A., et al. (2001). *Manifesto for agile software development*.
2. Project Management Institute. (2017*). A guide to the project management body of knowledge (PMBOK® guide)* – Sixth edition. Newtown Square, PA: Author.