애자일 방법을 사용해서 일정 문제를 관리하기

Brian Button, VP Engineering
Asynchrony, Inc., www.asolutions.com

번역: tb

사용자 end users 를 실망시키는 소프트웨어 프로젝트가 일정과 예산 안에 구현되는 경우가 거의 없다. 최초의 계획에만 맞추어 작업 되거나 사용자의 요구에 맞춰 변경되어 작업되는 경우 위에 언급한 조건 중 하나만 만족할 가능성이 높다. 두 가지 모두를 만족시키는 것은 어느 정도 통찰을 필요로 한다. “곰과 왈츠 춤을 추기: 소프트웨어 프로젝트에서의 리스크 관리”의 저자인 디마르코와 리스터는 일정 문제 Schedule Flaws 를 자신들이 뽑은 소프트웨어 프로젝트 관리의 5 대 리스크 중 하나로 꼽았다.

이 아티클에서 우리는 일정 문제에 관한 몇 가지 증상과 그 원인 그리고 일정 대비 당신 팀의 진척을 추적하는데 사용할 수 있는 현존하는 메트릭과 다이어그램에 대해 논의하고, 이러한 리스크들을 처리하는 애자일 방법을 설명할 것이다. 일정 문제의 리스크는 한 프로젝트가 시작할 때 어떤 일정을 세우더라도 프로젝트 종료시점에는 거의 일정이 맞지 않을 것이라는 점 그리고 정확한 완료일, 내용, 비용 추산을 해서는 안 된다는 점과 관련이 많다. 소프트웨어의 불확실성 uncertainties 과 추상성 intangibles 때문에 프로젝트 시작 시점에 얼마나 많은 시간과 공수가 투입되는 일정이 세워지든 상관이 없었다. 왜냐하면 그 일정은 개발이 진행되면서 필연적으로 변경될 것이기 때문이다.

원인

일정 문제에 관해서는 서로 다른 종류의 두 가지 원인이 있다. 한 종류는 프로젝트를 둘러싼 환경의 비 예측성과 직접적으로 관련이 있다. 즉 인력, 하드웨어와 네트워크 이슈, 휴가 일정, 날씨 등이 작업 완료에 직접적으로 영향을 끼친다. 두 번째 종류는 충분한 크기의 소프트웨어가 구현되고, 테스트되고, 배포 준비가 되는 때를 정확하게 예측하기 어려운 것과 관련이 있다.

환경적인 이슈는 특히 다루기 어려운데, 그들을 예측할 수 없기 때문이다. 사람들은 아프기도 하고, 눈보라 폭풍이 닥치기도 하고, 케이블 선이 갑자기 끊어지기도 한다. 대개 이것들은 당신의 프로젝트를 오래 잡아 끌지는 않지만, 일반적으로 당신의 영향력 내에 있지 않다. 하지만 그것들의 효과는 고려되고 예측되어야 한다. 또 이런 종류와 관련하여 당신의 영향력 내에 있는 것은 사람들을 시스템 개발로부터 멀어지게 만드는 회의의 횟수와 시간이다. 만일 팀의 생산성을 저해하고, 사기를 꺾는 한 가지 상황이 있다면

그것은 몇몇 문화에서 나타나는 끊임없는 회의 맹신자 multiple meeting mania 일 것이다.

두 번째 종류와 관련하여 시간 문제는 인생사 바로 그 자체이다. (Regarding the second category, the time issue is just a fact of life.) 소프트웨어는 놀랄 만큼 복잡하고 자연의 법칙 그 어느 것에도 구애 받지 않는다. 그리고 올바르게 동작하기 위해서는 많은 수의 독립적인 조각들이 일관성 있는 전체로서 서로 꼭 들어 맞아야 한다. 이러한 사실과 더불어 어떤 소프트웨어 플랜도 고객과의 첫만남의 상태로 살아남지 못한다. (역자주: 소프트웨어 플랜은 결국 변경된다는 뜻) 그리고 당신의 플랜도 실제 벌어지는 일들에 대응하기 위해 변경되어야 하는 상황에 놓이게 된다. 이것이 아래에서 우리가 좀더 자세히 살펴볼 리스크 이다.

증상

때때로 일정 문제로 어려움을 겪고 있는 팀은 아래의 증상 중에 하나 또는 그 이상의 증상을 나타낸다.

1. 고객과 이해관계자들로부터 변경 요청이 빈번하다

이론상 이해관계자들이 프로젝트에서 원하는 바는 프로젝트가 발생하기 전에 정리되는 것이 논리적인 것처럼 보인다. 이 명제에서의 결점은 고객들이 시스템을 새로 구축하거나 기존에 없던 혁명적인 것일 경우 그들 자신이 원하는 바를 거의 알지 못한다는데 있다. 그들이 동작 중인 시스템의 일부분을 보자마자 아이디어가 떠오르기 시작할 것이며, 결국 변경 요청을 하게 된다. 이것들 중 몇몇은 이전에 이미 발견되었던 새로운 요구사항일 것이며, 몇몇은 이미 구현된 것을 세부적으로 바꾸는 것이 될 것이다. (Some of these may be new requirements that they've just discovered, and some may be refinements on work that has already been done.) 또 다른 경우 프로젝트 시작 시점에는 알려지지 않았던 새로운 일을 의미하기도 한다.

2. 믿을 수 없는 추정

만들어지는 모든 소프트웨어 조각들은 본질적으로 새로운 것이다. 이러한 사실 때문에 개별 조각을 만드는 시간은 정확하게 측정하기 어렵다. 심지어 잘 알고 있는 도메인에서조차 팀이 선택한 특정 솔루션을 반복하는 일이 거의 없다. 왜냐하면 프로젝트가 현재 위치한 상황이 똑같지 않기 때문이다. 또한 단위 작업이 예측한

시점보다 일찍 끝나기보다는 상당히 늦은 시점에 완성될 가능성이 높다. 부정확한 빌드 예측은 큰 프로젝트의 일정을 지연시킬 수 있다.

3. “비공식적인" 작업량의 부담

팀은 대개 두 가지 유형의 업무를 하는데, “공식적인" 업무 또는 일정에 포함되는 일 그리고 “비공식적인” 업무로 모든 사람들이 그 일에 대해서 알지만, 아무도 얘기하지 않아서 아무도 계획에 고려하지 않는 업무가 있다. 이것들에는 소프트웨어 출시를 위해서는 반드시 해야 하는 활동 같은 액션 아이템이나, 부하와 확장성 같은 특별한 종류의 테스팅 또는 짧은 마감일의 관점에서는 수정 계획도 없고 출시되지 않을 쳐내야 하는 것들이 포함된다. 모든 팀은 이런 것들을 가지고 있으며, 대개 이런 것들은 프로젝트의 마지막 날까지 일정 문제로 드러나지 않는다.

4. 불확실한 품질

불확실한 품질은 “비공식" 업무의 보다 구체적인 형태이다. 일상적인 사용에서 충분한 품질을 확보하지 못한 소프트웨어 프로젝트가 대단히 많다. 그들은 프로젝트 라이프사이클의 후반부가 될 때까지 전체 시스템 빌드가 이루어지지 않으며, 개발 기간 동안 제한적인 테스팅만 이루어지고, 소프트웨어가 “완성"될 때까지 성능이나 보안 테스팅에서 비켜나 있거나 프로세스의 후반부가 될 때까지 테스팅을 지연시키는 요소들이 존재한다.

이런 영향으로 프로젝트 라이프사이클의 가장 좋지 않은 시점에서 이미 완료되었어야 하는 알 수 없는 작업들이 존재하는 잠재적인 프로젝트 리스크가 발생한다. 결국, 출시 바로 전 시점이 일정이 된다. (The effect of this is that there is a potential project risk of an unknown of work that needs to be done at the very worst time in a project’s lifecycle - at the very end, right before delivery is scheduled.)

5. 매트릭스 조직으로 구성된 팀원

모든 회사에는 여러 프로젝트들의 성공에 필수적인 특별한 지식을 가진 사람이 존재한다. 이러한 스태프 인력은 여러 팀에 조언을 해줄 수 있는 아키텍트일 수도 있고, 성능

테스팅, 사용편의성, 접근성, 보안성에 특화된 전문가 이거나 일반적인 테스터 일 수도 있다. 또한 이들은 다양한 범위에서 팀이 필요로 하는 여러 역할을 맡은 경우도 있다. 대부분의 경우 회사는 보유한 개발자보다 더 많은 업무와 팀을 가지고 있다. 이러한 희귀한 인력을 최대한 활용하려는 의도에서 이러한 인력들은 동시에 여러 팀을 지원하라는 요청을 받는다. 결국 이들은 한 개 팀의 작업에 병목이 아니라, 그들이 기여해야 하는 모든 팀의 병목이 된다.

메트릭스

잘 수집된 시간 이력이 있는 메트릭스 metrics 를 수집하는 것은 언제 일정에 문제가 발생했는지 무엇이 일정에 영향을 끼쳤는지 이해하는 핵심이다. 일정 문제를 묘사하는 가장 기본적인 메트릭은 간단한 번다운 burndown 차트이다. 번다운 차트는 시간 대비 완료된 업무의 그래프로, 가끔은 실제와 예상치의 작업/타임라인을 표시하기도 한다. 한 프로젝트의 실제와 계획된 진척상황이 일치하면 제대로 되고 있는 것이다. 출시일자 대비 당신의 진척상황을 묘사하는 확실한 메트릭은 프로젝트를 유지하는 가장 중요한 척도이다. 왜냐하면, 이것은 문제가 있는지 없는지 알려주는 선행 지표이기 때문이다. 다음의 예를 보자.

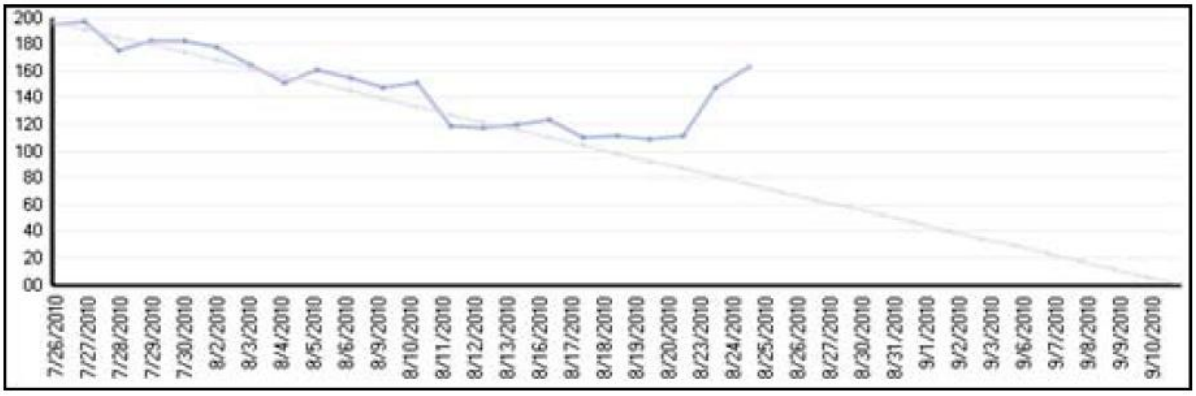

그림 1 - 번다운 차트의 예

이 다이어그램에서 우리는 수 주일 동안 기본적으로는 번다운 차트의 이상적인 곡선을 따라가는 프로젝트를 볼 수 있다. 이 릴리즈에서 남아있는 순수 작업량은 프로젝트가 예상 일자에 완료될 수 있도록 점차로 감소하고 있다. 사실상 이것은 일정대로 진행한다는 기록물이다. 하지만 갑자기 프로젝트가 잘못되고 있다.

하강하는 번다운 라인의 위쪽으로 가는 선에서처럼 다수의 작업이 릴리즈에 추가되었다. 그리고 프로젝트의 완료일자에 곧바로 문제가 생겼다. 프로젝트를 성공시키려면 프로젝트 범위가 축소되거나 시간이 추가되어야 한다.

위의 차트는 프로젝트에서 남아있는 순수 작업량을 확인하고, 완료 일을 예측하는데 유용하다. 하지만 이것은 추가된 작업량 대비 완료된 작업을 절대량으로 나타내지 못한다. 이것을 표현하는데 적합한 다른 몇 가지 종류의 그래프가 있다. 그 중 하나는 완료된 작업량 대비 남아 있는 작업량을 나타내는 쌓여진 바 차트 stacked bar chart 이다.

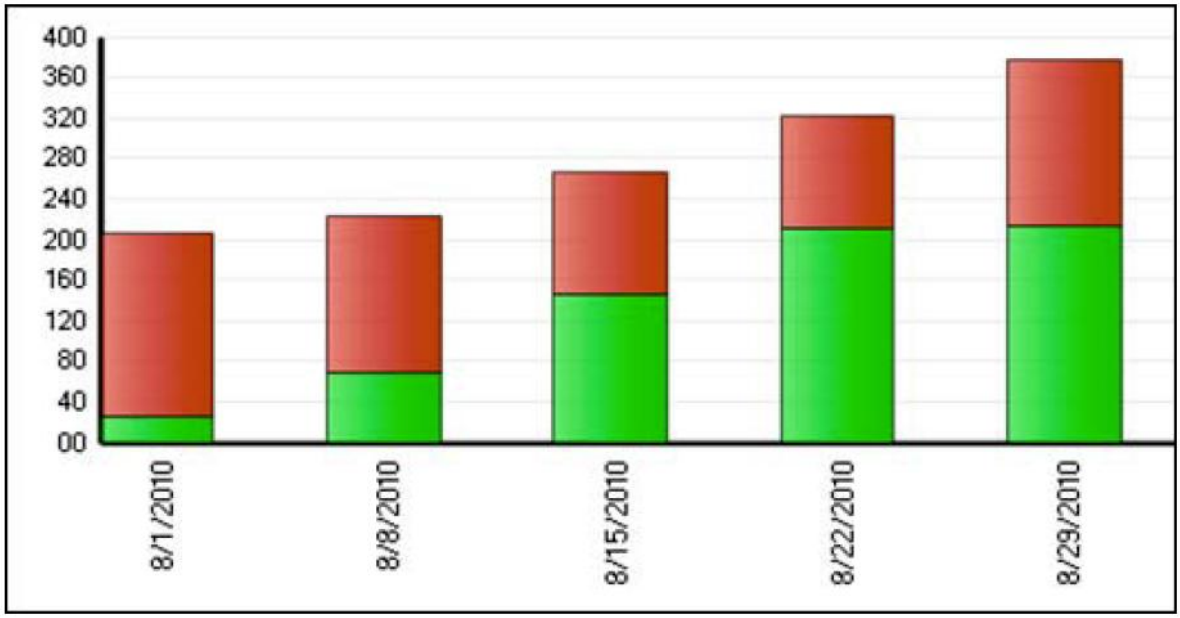

그림 2 - 번업 차트 burn up chart 의 예

위의 차트에서 모든 바의 전체 높이는 프로젝트에 존재하는 전체 작업량을 나타낸다. 반면에 아래 녹색은 완료된 작업 그리고 위쪽 빨간 부분은 남아 있는 작업량이다. 다시 말해 프로젝트의 전체 범위는 각 바의 높이가 다른 것과 비교해서 변함이 없는 한 고정이라는 뜻이다. 만일 전체 높이가 증가한다면 프로젝트에 (구현해야 할) 추가적인 범위가 더 생겼다는 말이다. 이 시점에서 작업이 완료되자마자 재빨리 추가된다면 완료 선은 우측으로 가더라도 그대로 유지 되는 것을 볼 수 있다.

이 두 개의 그래프는 동일 프로젝트의 동일 백로그를 나타내고 있다. 하지만 각 그래프에서 얻을 수 있는 정보는 다르다.

원인을 찾기 위한 메트릭스

프로젝트가 일정을 따르지 않는다고 판단되면, 왜 그런지 알기 위해 추가적인 조사가 있어야 한다. 아래는 일정 문제의 근본적인 원인을 아는데 사용할 수 있는 몇 가지 메트릭들이다.

1. 처리량의 변화

팀이 처리하는 작업량이 단위 시간 동안 매우 다른 차이를 보인다면, 가능한 이유 중 하나는 팀의 가용 대역폭이 시간이 지남에 따라 급격하게 변경된 것으로 생각할 수 있다. 특정한 팀 구성원이 여러 팀에 매트릭스로 묶여있다면, 그가 일하는 수 주 일 동안 집중력이 떨어져서 팀의 작업량이 떨어진 것일 수 있다. 이런 경우라면 일일당 또는 스프린트당 가용한 전체 시간을 나타내는 간단한 그래프로 이 이슈를 파악하기에 충분하다. 아래에 예가 있다.

그림 3 - 스프린트당 처리량

확실히 이 팀과 관련된 인력의 업무 시간에 차이가 있다. 왜 이렇게 투입시간이 많이 차이가 나는지 알기 위해서는 추가적인 조사가 필요하다. 이 이런 일이 발생했는지 와 상관없이 이 팀의 작업세기 velocity 는 이터레이션과 이터레이션 사이에 별 차이가 없을 것이다.

2. 추정 정확도의 부실

추정의 정확도를 판단하는 것은 메트릭을 분석하는데 있어 가장 어려운 부분 중 하나다. 수정되어야 하는 특별한 것을 추정하는 것이 중요할까 아니면 많은 수의 피쳐 대비 수정해야 하는 것을 전체적으로 추정하는 것이 중요할까? 내가 관리했던 최근의

프로젝트에서 실제 달성한 것과 비교하여 추정을 했었다 (이것을 처음 시도한 경우였다). 여기서 우리가 배운 것은 우리가 개별 피쳐나 스토리 추정은 아주 좋지 못했지만 전체적으로 봤을 때는 아주 정확한 추정을 했었다는 것이다. 다시 말해 개별 추정은 편차가 심하게 과하거나 덜 했지만, 실수들이 서로 간에 상쇄되어 전체적인 추정은 매우 정확했다!

추정 정확도를 알아볼 때 가장 중요한 부분은 추정의 주류 main body 로부터 벗어나는 이야기들을 확인하는 것으로, 어떤 이유로 그들이 그렇게 많이 벗어나 있는지 이해하는 것이다. 이 프로젝트에서 내가 했던 것은 동일한 추정치를 가진 모든 이야기들을 (여기서는 1 과 8 “포인트" 사이) 수집해서, 실제 걸린 작업 시간에 해당하는 스토리들을 배치했다. 다음의 나의 그래프에서는 1 포인트짜리 피쳐를 나타낸다.

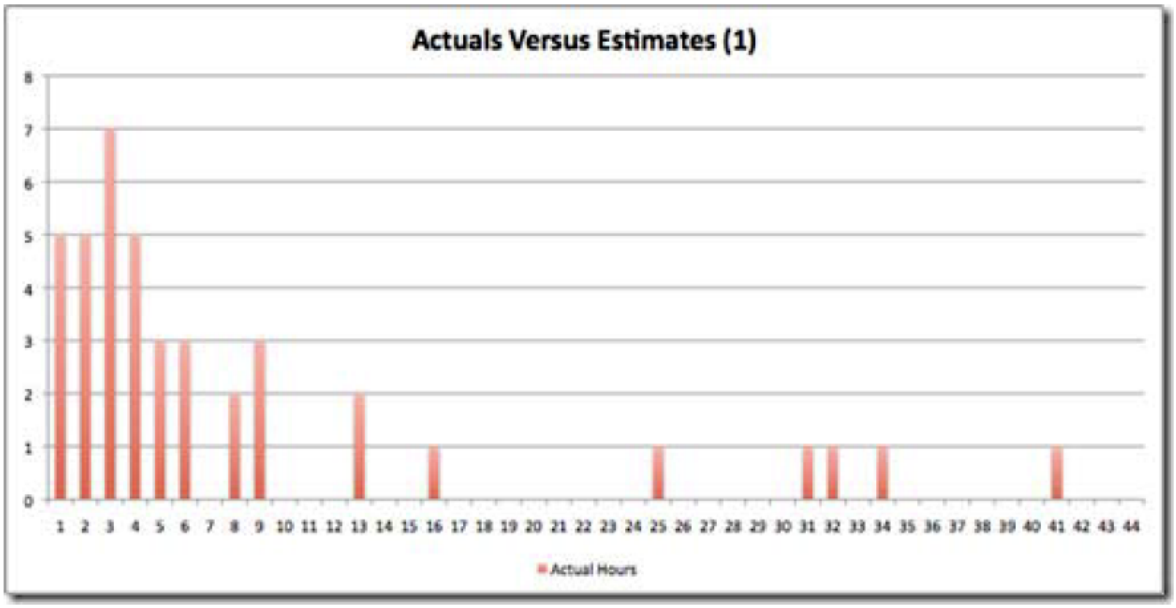

그림 4 - 1 포인트 스토리에 대한 실제 대비 추정치

이 그래프에서 Y 축은 X 축을 따라서 볼 수 있는 해당 시간 안에 완료된 피쳐의 수를 나타낸다. 우리 프로젝트에서는 4 시간 작업량을 1 포인트짜리로 예상했다. 따라서 대부분의 경우 1 포인트의 피쳐들은 아주 정확하게 추정되었다 (그들 대부분은 1 시간에서 6 시간이 걸렸다).

4 시간 미만으로 추정된 것들도 몇 개 있었는데, 대부분 우리 추정 상으로는 분수시간으로 처리할 수 없었기 때문에 우리가 만들 수 있는 가장 작은 단위인 1 포인트를 할당했다. 하지만, 논의의 주제가 될 수 있는 주류에서 벗어난 추정들도 있었다. 대부분의 경우 기존에 존재하던, 레가시 legacy 코드에 존재하던 결함을 발견했거나 요구사항이 모호한 것 같이 시간이 걸릴 만한 타당한 이유가 있는 것이었다.

더 복잡하고 큰 스토리의 추정에 대해 비슷한 그래프가 만들어졌다. 누구나 예상하듯이 스토리에 대한 추정이 커질 수록 추정의 불확실성 또한 증가했다. 이것에서 팀이 배운 중요한 교훈은 추정의 정확도는 큰 스토리 보다는 작은 것이 낫다는 것이었다. 예를 들어 추정한 스토리의 크기가 두 배가 되면 그림 5 에서처럼 추정의 분산도는 급격하게 변했다.

아래의 그래프에서 보듯이 2 포인트로 추정한 스토리에 걸린 실제 시간 중 가장 높은 것은 6 또는 7 시간 사이의 어딘가에 해당되며, 이것은 1 포인트에 4 시간 작업으로 계산한 결과와 매우 유사하다. 하지만 이런 식으로 스토리 크기를 단순히 늘리면 더 많은 불확실성이 추정에 포함되어 주류에서 많이 벗어난 추정들이 생겨난다.

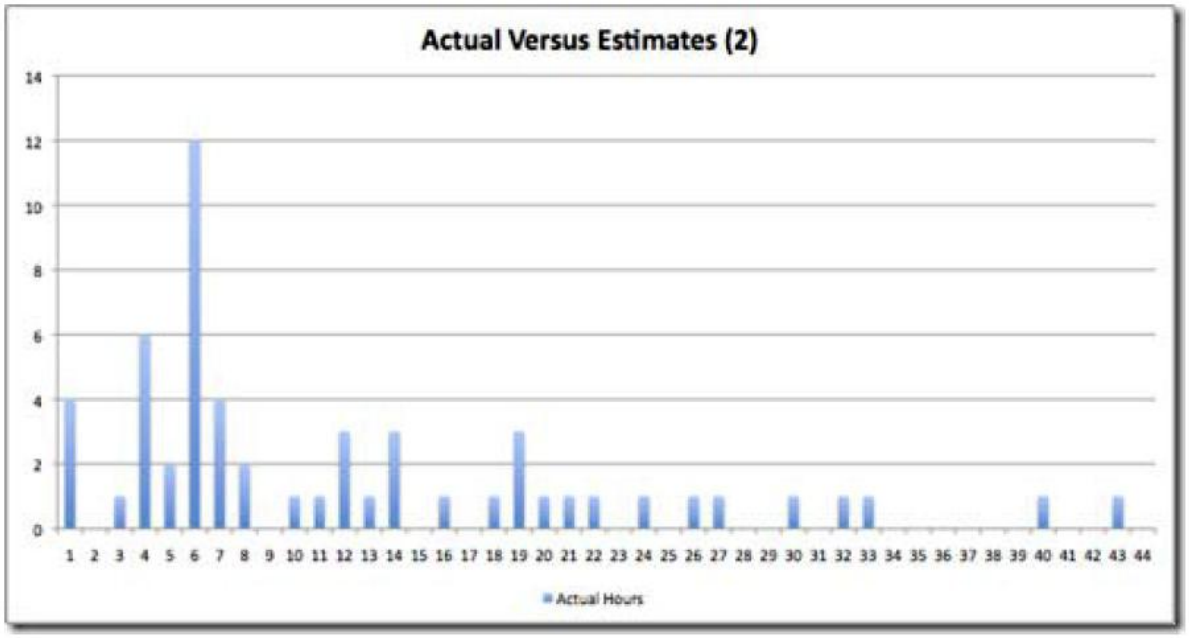

그림 5 - 2 포인트 스토리에 대한 실제 대비 추정치

3. 불확실한 품질과 “비공식" 업무

이전에도 언급했지만 이 두 가지 증상은 서서히 퍼진다. 내가 아는 한 대규모의 프로세스 변경을 도입하지 않고는 이것들을 직접 측정할 방법은 없다. (조금씩 애자일이 도입되고 있으며, 아래와 같은 이유 때문에) 내가 참여했었던 팀에서는 이 두 가지 이유를 팀 내 모든 사람들이 알고 있었지만 아무도 인정하지 않았다. 이 두 가지 문제의 효과를 이해하는 가장 좋은 방법은 사람들이 경험하고 있는 밑바닥의 정서를 알기 위해 매니저를 팀과 충분히 밀착시켜서 일하게 하는 것이다. 이러한 밑바닥 정서를 느끼게 되면 팀원들은 품질과 완전성, 적합성에 대한 대화를 시작해야 한다. 팀이 이러한 대화를 주저하는 시간이 길

어질 수록 프로젝트 막바지에서 불쾌하고 놀라운 일들이 더 많아질 것이다.

애자일 계획 & 로드맵

아마도 당연하겠지만 이 전체 아티클의 주제는 애자일이 되고 애자일처럼 생각하고 애자일의 방식으로 행동하는 것이다. 그렇게 되면, 위에서 얘기한 고통을 느끼지 않고 당신의 프로젝트는 항상 적시에, 주어진 예산 안에서, 고상한 품질을 가진 채로 출시될 것이다. 물론 이론뿐일 수 있다. 하지만 실무 적용에 있어서 당신은 이런 목표에 좀더 다가갈 수 있는 지식을 가질 수 있다.

애자일 팀은 다른 방식으로 계획한다. 물론 그들에게도 플랜과 일정이 있다. 하지만 그 플랜은 시간이 지나면서 변경될 것을 가정한다. 계획을 세우는 일은 일상적인 활동으로 프로젝트 전반을 통해 서로 다른 레벨과 서로 다른 리듬rhythms으로 수행된다. 계획은 프로젝트를 수행하는 과정 내내 리스크를 관리하는 방식으로 이루어진다. 이러한 계획을 세우는 서로 다른 단계는 다양한 방식으로 위에서 언급한 이슈들을 처리하는데 기여한다.

최상위의 레벨에서 애자일팀은 합의된 일정에 고객에게 출시할 수 있는 (개발) 능력을 계획한다. 이러한 능력은 가능한 한 완성된 피쳐 명세를 만들지만 약간씩의 여지를 남겨둘 수 있도록 느슨하게 정의된다. 이 약간의 여지라는 말은 외견상 말이 안 되는 것처럼 보인다. 하지만 이것이 왜 이런 스타일의 계획 세우기가 그렇게 성공적인지에 대한 실제적인 핵심 포인트이다. 우리는 이것에 관해 짧게 이야기 할 것이다.

이런 계획의 결과로 각각의 (개발) 능력이 제공할 것에 대한 상세함과 장래 어느 시점에 제공할 것인지에 대한 (개발) 능력의 로드맵이 완성된다. 이것은 장기간의 계획, 마케팅, 판매에 적합해야 한다. 그들은 대략적인 로드맵을 가지고 거의 확실한 출시를 보장한다.

아주 높은 레벨에서 이런 장기간의 계획을 유지하면, 이 시점에서 사람들은 적은 비용과 적은 리스크를 가지고 계획의 변경이 자유롭게 된다. 이 레벨의 계획은 연간 몇 차례에 걸쳐 발생한다.

계획 & 실행

로드맵/포트폴리오 레벨의 계획세우기에서 한 단계 낮은 레벨이 릴리즈 계획이다. 이것은 대개 4-12주 동안에 출시할 피쳐들을 언제 어떻게 결정할 것인지를 계획한다. 로드맵에서 (개발) 능력이 선택되고 더 작은 범위로 세분화하여 최소한의 팔릴 수 있는 피쳐Minimal Marketable Features (MMFs)라고 부르는 좀더 이해할 수 있는 단위로 만든다. 처음에 선정된 이러한 피쳐들은 비즈니스 이해관계자에게 최대한의 가치를 주고 리스크를 감소시키고, 조직 내부에 비전을 주기 위한 의도를 가지는 경향이 있다. 낮은 가치를 주는 피쳐들은 프로젝트 일정의 후반부로 밀리거나 완전히 배제될 수도 있는데, 특히 그것들의 가치가 개발 비용을 정당화할 만큼이 되지 않으면 결코 구현되지 않는다.

MMFs는 조직이 사용자나 고객에게 흥미나 흥분을 줄 수 있는 최소한의 기능 단위를 나타낸다. 이들은 다양한 (개발) 능력에 영향을 미치며 시스템의 여러 곳을 이용할 수도 있다. 하지만 그것은 항상 즉시로 나타날 수 있는 것이어야 하고, 누군가에게 판매될 수 있는 가치를 지닌 것이어야 한다. (They always represent something of immediate, marketable value to someone.) 이 레벨에서 이들은 로드맵 상의 연대기라기 보다는 좀 더 잘 정의된 것이 된다. 하지만, 추가적인 정의는 구체적인 상세함이 필요하게 될 때까지는 의도적으로 연기된다. 이전에 했던 것처럼 상세한 결정은 의도적으로 이후로 연기된다. 결정이 미뤄지는 이유는 이른 결정은 잘못될 리스크를 증가시키기 때문이다. (The reason decisions are deferred is that deciding early increases the risk of being wrong.) 결정을 미루는 것은 결정을 내리기 전에 가능한 한 많은 학습을 할 시간을 주어 올바른 선택을 할 가능성을 높여준다. 이 상황 뒤에 있는 이론은 Last Responsible Moment의 Lean 원리로 표현된다.

MMFs는 그것을 구현하는 실무자에 의해 추정된다. 그리고 이들은 릴리즈의 중요성에 따라 우선순위를 정한다. 이 레벨의 계획세우기는 릴리즈당 한 번 발생하기 때문에 1년에 4-12번 사이가 된다.

가장 빈번한 계획세우기의 형태는 이터레이션 계획세우기로 매주 한 번이나 두 번 발생하며, 진가가 비로소 나타나는 곳이다. 소수의 MMFs가 팀에 전달되고, 다시 “유저 스토리"로 분해하여 MMFs 피쳐의 일부분을 제공하는 낮은 기능으로 바뀌어 제공된다. 하지만 이런 유저 스토리의 핵심적인 특성은 이들이 시스템의 이해당사자나 사용자에게 얼마간의 흥분을 제공해야 한다는 점이다. 몇 개의 스토리들이 합쳐져서 하나의 MMF를 구성하는 경우가 많다.

그 팀은 이터레이션 계획세우기를 하면서 이터레이션 안에서 MMFs를 구성하는 유저 스토리를 어떻게 구현할 것인지에 대한 계획을 세우고, 어떻게 하면 각각의 MMF들이 낮은 비즈니스 레벨에서 어울릴지를 논의한다. 각 스토리는 완료를 알 수 있는 인수 통과조건을 포함해 가능한 한 구체적으로 정의한다. 이 인수 통과조건은 스토리가 완료되었고 스토리의 끝을 알 수 있는 측정 가능한 표준으로 사용된다. 이렇게 해서 측정할 수 없고 알려지지 않은 작업들이 프로젝트의 후반부에 구현되도록 미뤄진다. 결국 최종 단계에 와서 모든 유저 스토리들이 추정되었다. 이 시점에서 이렇게 세부적으로 쪼개진 작업들이 하루 또는 하루에 못 미치는 작업량으로 바뀐다. 위에서 언급했듯이 작은 스토리들은 더 정밀하게 추정이 가능하다.

이터레이션 계획세우기 기간 동안 사용된 (개발) 능력 계획세우기의 일부분으로써 팀 능력에 대한 시간 누적 값들도 추적해서 1-2주 기간 안에 약속된 작업량을 줄이는데 사용된다. 이러한 계획세우기planning, 구현하기committing, 실행하기executing, 출시delivering의 일상적인 리듬이 프로젝트에 심장박동을 제공해서 진행상황에 대해 측정하고 추적할 수 있도록 한다. (This regular rhythm of planning, committing, and delivering gives the project a heartbeat that allows its progress to be measured and tracked.)

퍼즐의 마지막 조각은 실행이다. 이것은 소프트웨어를 만드는 기간 동안 발생한 일정 문제의 원인을 다루는 것이 해당된다. 팀 내의 모든 개인은 첫 사용자 스토리부터 마지막 라인의 코드까지 품질 있는 제품을 만들 책임이 있다. 모든 사람이 실행하고, 모든 사람이 테스트하며 모든 사람이 품질책임을 가진다. 품질은 한 팀에게 있어서 모호한 것이 되어서는 결코 안 된다. 팀 구성원이 만드는 각각의 단계는 품질 있게 만든다는 시각으로 작업된다. 보안성, 부하, 확장성, 성능을 포함한 모든 것들에는 자동화된 테스트가 존재한다. 대부분의 테스트는 하루에 십여 회씩 수행되고 모든 테스트는 밤이 되기 전까지는 최소 한 번 실행된다. 이 시스템은 생성되고, 배포되고, 테스트 되기를 지속적으로 반복한다.

확실히 이러한 품질 수준에 도달하기 위해서는 공수가 많이 든다. 하지만, 이러한 공수의 이익은 언제나 출시할 수 있는 코드가 바로 준비된다는 것이다. 어떤 피쳐가 완성되면 정말로 끝난 것이다. 코딩이 된 후 피쳐나 시스템 레벨에서 테스트되고 필요한 문서가 작성되면 나갈 준비가 된다. 이렇게 하면 프로젝트 전반을 통해 개발 진척이 가치의 완성이라

는 측면에서 추적되며, 작동하는 기능을 초기에 증감식으로incremental 출시하게 된다.

이 아티클에서 상세화된 애자일 활동과 메트릭에 집중한다면 팀은 일정 문제로 유발될 수 있는 리스크를 확인하고 관리할 수 있게 된다. 이러한 메트릭들은 리스크를 드러나게 할 뿐만 아니라 리스크를 관리하는 툴을 제공한다. 두 가지를 조합한 중간쯤을 사용해서 팀은 자신들의 이해관계자들에게 빠르고 효과적이며 고품질의 가치를 전달하게 된다. 가치를 전달하는 것이 우리가 여기에 있는 이유 아닌가? (And delivering value is what we're here for, isn't it?)

.Fin.