**기계적인 애자일의 다섯 가지 징후**

Daryl Kulak
Pillar Technology, http://www.pillartechnology.com

@ Method & Tools Spring 2010 Page 13

Ver. 0.2

나는 직업적인 애자일 코치이지만, 실제로는 이야기작가이다. 이 아티클에서 다섯 가지의 이야기를 할 생각인데, 대개는 어떻게 애자일이 기계적으로 변하는지 알 수 있도록 도움이 되는 실제의 이야기에 바탕을 두고 있다. 나는 이 이야기에서 구체적인 어떤 사람들을 지칭하지 않도록 약간은 우스꽝스러운 이름들로 변경했다.

**애자일 전문가 신드롬**

옛날에 네버랜드 보험이라는 회사가 있었다. 거기에는 개발팀도 있었고, 애자일 컨설턴트도 있었다. 이들 모두는 애플리케이션을 빌드하기로 마음 먹었다. 그 팀의 인원은 약 10 여명 정도였고, 그 애자일 컨설턴트는 명석함과 헌신적인 애자일리스트로 정말 정말 명성이 높았다. 그 개발팀은 그와 함께 있는 것을 행운으로 생각했다. 그 애자일 컨설턴트의 이름은 Sticky LaGrange 였다.

Sticky 는 팀이 애자일 활동을 할 수 있도록 도왔고, 그들 모두를 가르쳤다. 팀은 신중하게 Sticky 의 말을 따랐고, 무엇을 해야 할지 잘 모르는 경우에는 언제라도 그에게 답을 구했다. Sticky 는 아주 도움이 되었고 인내심이 있었다. 그뿐 아니라, Sticky 는 사람들을 코칭하는데 많은 시간을 할애하고도, 팀 내의 다른 사람들보다 자신이 더 많은 코드를 작성했다는 사실에 자부심이 있었다. Sticky 는 정말 놀라운 사람이었다.

최종적으로 그 애플리케이션이 기한과 예산에 맞게 완성되었다. 네버랜드에 있는 모든 사람들은 깊은 감명을 받았다. 팀은 아주 자랑스러워 했다.

그리고 나서 네버랜드의 사악한 왕은 예산을 줄여버려, 그 팀은 Sticky 와 헤어지게 되었다. 이제는 Sticky 없이도 업무를 성공적으로 수행할 수 있었다. 그가 아주 잘 가르쳤기 때문이다. 그래서 그 팀은 다른 애플리케이션의 개발 업무를 맡아 개발을 시작했다. 그들은 Sticky 가 말했던 모든 것을 지키려고 애썼다. 의심이 드는 일이 있다면, 그 팀에 있는 누군가는 반드시 질문을 했다. “Sticky 라면 어떻게 했을까?” 그 그룹은 Sticky 가 했었던 것을 기억해내고는 충실해 그대로 따라 했다. 하지만 일이 잘못되어

가기 시작했다. 팀 내 서로 다른 사람들이 Sticky 의 충고를 서로 다르게 기억하고 있었다. 그리고 때때로 설명할 수는 없지만 Sticky 의 충고가 잘못된 것처럼 보였다! 이 새로운 상황에서는 Sticky 의 충고는 쓸모가 없었다. 어떻게 Sticky 의 충고가 잘못될 수 있을까?

팀원 몇몇은 자신들이 Sticky 의 충고를 무시해야 한다고 말했다. 하지만 다른 사람들은 그건 끔직한 생각이라고 여겼다. "그것은 진정한 애자일이 아니야!" 그들은 울부짖었다. 왜 처음에는 잘 들어맞았던 Sticky 의 충고가 어긋났을까?

불행히도 네버랜드의 그 팀은 내가 **애자일 전문가 신드롬** Agile Expert Syndrome 이라 부르는 상황에 빠진 것이다. 이것은 사람들이 전문가나 컨설턴트, 책이나 아티클의 충고를 따라야 한다고 믿는 상황이다.

한 팀이 이렇게 추종 모드 compliance mode 에 진입할 때마다 그들은 곤란을 겪는다. 그들은 애자일을 기계적으로 수행한다. 그리고 기계적인 애자일을 수행하는 팀은 항상 어려움을 겪는다. 왜냐하면 전문가로부터 받은 조언과 다른 상황을 마주치게 되면 항상 어려움에 부딪히게 될 것이기 때문이다. 그리고 그 상황이 발생할 때마다 자신들의 사고 intelligence 를 동원해서 생각할 수 있는 최선의 행동을 하기 보다는 전문가가 했을 것으로 추정되는 것을 판에 박힌 듯 수행할 것이다.

**의사결정과 실제 업무의 분리**

옛날에 Cheap Green Shirts.com 이라는 옷 제조회사가 있었다. 그 회사에는 일을 잘하고 높은 생산성을 보이는 몇몇 애자일 팀이 있었다. 그들의 성과는 높았고, 그 팀은 만족했으며 비즈니스의 성과는 좋았다.

"내 생각엔 이 팀의 성과를 더 높일 수 있어요" 부사장 중 한 명이 말했다. "내가 보기엔 A 팀이 다른 팀보다 훨씬 성과가 좋기 때문에, A 팀의 활동 중에서 베스트 프랙티스를 뽑아서 다른 팀들에게 적용하죠. 그런 식으로 해서, 우리의 모든 팀이 A 팀의 성과를 달성할 겁니다. 정말 굉장할 거예요!"

그래서 부사장은 여러 팀들에게 조언을 해줄 수 있는 소프트웨어 엔지니어링 프로세스 그룹 Software Engineering Process Group (SEPG)이라 이름 붙인 애자일 전문가 그룹을 만들었다. SEPG (seepage 라고 발음되는) 전문가들은 각 팀에서 하는 활동들을 주의 깊게 조사, 수집해서 최선의 활동을 추려내었고 다시 여러 팀들에게 적용하도록 했다.

놀랍게도 그 팀들은 새로운 활동들 중 대부분을 할 수가 없었다. SEPG 전문가들은 화가 났다. “이 팀들은 배우려고 하지 않아요.” SEPG 전문가들이 탄식했다. “SEPG 사람들은 우리의 업무를 이해하고 있지 않아요.” 팀들이 말했다.

그래서 부사장은 강제로 SEPG 전문가들이 제시했던 것들을 그룹이 사용하도록 했다. 그 그룹들은 새로운 활동들을 의무적으로 이행했다. 이 시점에서 모든 팀의 성과는 곤두박질 쳤다. 어떻게 이런 일이 벌어졌지? SEPG 전문가들이 설명했다. “그 팀들은 (우리가 제시한) 활동을 잘못 적용했어요!” 팀들은 대답했다. “(이들이 제시한) 활동들은 우리와 맞지 않아요.”

이 부사장은 우리가 “의사결정과 실제 업무의 분리"라고 부르는 기계적인 애자일의 문제를 느낀 것이다. 의사 결정이 작업을 하는 사람과 멀어지면 멀어질수록, 결과는 잘못될 것이다. SEPG 그룹은 (실제 작업을 하는) 팀들과 상당히 괴리가 있었고, 그들 일상업무의 상세한 부분까지는 알지 못했다. 그래서 그들이 전문가라 할지라도 불행히도 그들의 추천은 실패했다. 처음에는 그 팀들이 어떤 활동을 할지 어떤 것이 자신들에게 잘 맞을지 의사결정을 했다. 부사장이 (실제 업무를 수행하는 팀이) 의사결정을 하지 못하게 하고 별개의 그룹에서 결정을 하도록 해서, 성과와 팀 만족도는 감소했으며, 모든 사람들이 문제를 안게 되었다. 부사장은 팀들이 사람이 아니라 기계라고 생각했던 것이다. (The vice-president was thinking that the teams were machines, not people.)

**시스템이 아닌 사람을 비난하기**

한 회사에 서로 협력해서 일하는 두 팀이 있었다. 한 팀은 다른 팀이 코드로 구현할 요구사항들을 제공했다. 첫 번째 팀을 Uppities 라고 부르겠다. 왜냐하면 그들은 프로세스에서 상류 upstream 에 있기 때문이다. 두 번째 팀을 Downers 라고 부르겠다. 왜냐하면 그들은 프로세스에서 하류 downstream 에 있기 때문이다.

맨 먼저 Uppities 와 Downers 들은 서로 다른 버전의 애자일로 작업했다. Uppities 는 6 주의 긴 이터레이션 단위로 일하는 반면에 Downers 는 2 주간의 이터레이션을 사용했다. 하지만 모두들 약간씩 애자일의 색채를 띤 채 일했다.

여전히 Uppities 는 Downers 들과 잘 협업하기가 어려웠다. Uppities 들은 항상 시급한 경향이 있었고, 심지어 이터레이션 중간인데도 Downers 에게 높은 우선순위의 업무들을 쏟아 부었다. (The Uppities tended to have big emergencies and would dump high-priority work into the Downers area, even in mid-iteration.) Uppities 의 업무가 회사 내에서 가장 중요하기 때문에, Downers 들은 따를 수 밖에 없었다. (Since the Uppities work was the most important work in company, the Downers had to comply.)

어느 날 나는 downstream 팀의 누군가와 대화 중이었다. 그를 Donny 라고 부르자.

나는 Donny 에게 Uppities 와 Downers 간의 충돌의 원인을 뭐라고 생각하느냐고 물었다. “음, 물론 당연히 Uppities 죠!” 그가 말했다. “그들은 다른 팀에 대한 존경심이 없어요! 그들은 그들만 최고인줄 안다고요!”

나는 Uppities 의 시각이 무엇인지 떠올릴 수 있냐고 물었다.

“오, 문제 없어요. 나는 거기서 일한 적이 있어요.” Donny 의 말에 나는 놀랐다. “내가 Uppity 였을 때, Downers 들이 문제였던 건 명백했어요. 하지만, 이제 나는 무엇이 진짜 문제인지 알죠. Uppities 가 문제 덩어리들이죠. Downers 가 아니라고요!”

나는 내가 “시스템이 아닌 사람을 비난하기”라고 부르는 기계적인 애자일의 또 다른 측면에 Donny 가 빠진 것인지 궁금해졌다. 아무리 그가 현재 속한 조직이 아닐지라도 Donny 는 시스템이 문제가 아니라 다른 팀이 문제라고 생각하고 있었다. 내가 말하는 시스템이라 함은 소프트웨어 애플리케이션이 아닌 두 팀이 상호작용하는 방법이나 문화를 의미하는 것이다. 아마도 이러한 갈등을 조장하는 문제가 전체 시스템 안에 있을 것이다. 그리고 그러한 구조적인 문제가 “다른 사람의 잘못”처럼 비쳐졌을 것이다.

우리가 누구의 잘못인지 판단하는 비난게임 blame-game 에 (힌트: 그것은 항상 다른 사람의 문제다) 빠지기 보다는 항상 “자 이제 함께 우리 시스템의 문제를 고쳐봅시다”라는 자세를 가진다면, 우리는 발전할 수 있을 것이다. (If we can always take an attitude of “Let’s fix the system together” rather than getting into a blame-game of determining who’s at fault (Hint: it’s always the other guy), we can make much more progress.)

**보스, 그냥 내게 뭘 해야 되는 지만 말해 주세요**

나의 회사인 Pillar Technology 에는 개발자를 뽑는 아주 엄격한 인터뷰 프로세스가 있다. 우리 프로세스 안에는 응시 지원자의 테스트 주도 개발자로서의 능력을 평가하기 위해 Pillar 의 전문가 중 한 명과 협업을 해야 하는 조건이 있었다.

팀의 사무실을 지나가며 한 번 목격할 기회가 있었는데, 응시 지원자가 방안으로 들어가 우리 Pillar 의 전문가 옆에 앉았다. 우리는 그 지원자를 Emo 라 부르겠다. 이것은 테스트 주도 개발에 대한 것이기 때문에, Pillar 의 전문가는 Emo 에게 기존 코드를 보고, 구현되어 있는 유닛 테스트를 확인해서, 테스트가 어떻게 통과하는지 말해보라고 요청했다.

그 테스트는 아주 간단했다. 입력 필드를 제공하고, 변수가 “12”와 같은지를 리턴하는 단언문 assert 이었다. Emo 는 코드를 한 번 보더니, 커서를 그곳에 가져다 놓고 전체를 지워버렸다. 그리고는 그곳에 리턴 변수를 “12”로 만드는 코드를 작성해 넣었다. 그리고는 Pillar 의 전문가를 보고는 미소를 지었다.

이것 모두는 내가 본 가장 짧은 페어링 인터뷰이었지만, 가장 긴 인터뷰 회고 retrospective 이기도 했다.

Emo 는 내가 “보스, 그냥 내게 뭘 해야 되는 지만 말해주세요.” 라고 부르는 기계적인 애자일의 한 측면에 빠진 것이다. 만일 우리 팀원들이 자신들의 보스가 지시 하기만을 바라는 사고를 가졌다면 우리 모두는 어려움을 겪었을 것이다. 우리는 모든 팀원이 모든 문제에 대해 자신의 모든 판단력을 총동원하여 대응하기를 원한다. (We need every team member to be applying his or her full intelligence to every problem.) 그들이 단순히 지시를 따르는 모드로 들어갈수록, Emo 가 인터뷰에서 했던 그런 해결책을 만나게 될 것이다.

팀원들은 이런 사고에 아주 쉽게 빠져든다. 이러한 팀원들을 강제로 만드는 매니저가 있는데, 그건 정확히 매니저가 의도한 것과 다른 어떤 결과를 두려워하기 때문이다. 하지만 문서에 “당신이 해야 하는 것”이라고 적혀있기만 해도 이런 유형의 팀원들을 만들 수 있다.

우리는 우리의 팀원이 불쌍한 Emo 와 같은 사고체계에 빠지는 것을 무슨 수를 쓰더라도 막아야 한다.

**사람이나 팀들간의 경쟁**

통신 회사의 부사장에게는 모두들 대단한 성과를 내는 여섯 개의 애자일 팀이 있었는데, 부사장에게 갑자기 아이디어가 떠 올랐다. 그는 이달의 우수 팀의 타이틀을 걸고 각 팀을 서로 경쟁시켜서 “더 성과를 낼 수 있다”는 것을 깨달았다.

우리는 이 부사장을 Emeril 이라고 부르자.

Emeril 는 성과에 있어서 다른 팀들을 압도하는 팀에 대해 무료로 점심을 제공한다고 발표했다.

경쟁이 시작되었다. 그 팀들은 즉시 시스템의 규칙을 확인하기 시작했다. 결국 그들은 스토리점수 storypoints를 적게 만들면 추가 작업 없이 더 많은 성과를 낼 수 있음을 발견했다. 그 팀들은 부사장이 무엇이 어떻게 되어 가고 있는 건지 깨닫기 전까지는 몇 달 동안 "스토리점수 인플레이션"의 기간을 겪었다. 결국 부사장은 스토리점수는 모두 동일한 크기여야 한다고 못박았다.

그러고 나니, 팀들은 품질과 결함 수가 측정 대상이 아니라는 사실을 알게 되었다. 따라서, 각 이터레이션마다 사용하는 테스팅 시간을 줄여버렸다. 확실히 점수는 올라갔다. 하지만 결함도 올라갔다. 그래서 결국 부사장은 결함도 측정 대상이라고 선언했다. 더 많은 결함이 발견될 수록 팀 점수는 내려갔다.

여기까지는 좋았다. 하지만 부사장은 테스터 또한 팀의 일원이었다는 점을 깨닫지 못했다. 그들 또한 무료 점심의 대상이었다. 따라서, 그는 테스터에게 "많은 결함을 찾아내지 못하면, 너희들의 무료 점심은 없을 것이다." 라고 세 번째로 선언했다.

그리고는 이후로도 계속 진행되었다. 당신도 이러한 경쟁이 일을 잘 하는데 도움이 되지 않는다는 걸 알 것이다. 팀이라는 장벽이 없어지면 자연스럽게 협력이 자리잡을 것이다. 왜 우리의 비용을 써가며 다른 팀의 성과에 도움을 주어야 할까?

부사장은 "사람이나 팀들간의 경쟁"이라 부르는 기계적인 애자일의 징후로 고생한 것이었다. 사람이나 팀을 서로간에 경쟁에 놓이도록 하면, 그들이 그 일을 하기 위해 당신의 조직을 망가뜨려야 하더라도 당신의 목표는 달성할 수 있을 것이다. (Put people or teams in competition with each other and you will ensure that they will meet your targets, even if they have to destroy your organization to do it.)

**다섯 개의 징후 - 다섯 개의 이야기**

이것들은 당신의 애자일 팀을 키워나가고 계속 유지시키는 동안에 발생할 수 있는 문제이다. 그리고 이것들은 사람을 사람이 아닌 기계로 취급하는 것과 관련이 있다.

기계적인 애자일을 해결하기 위해 새로운 활동이나, 린 Lean, 칸반 kanban 또는 식스 시그마 Six Sigma 그 어느 것도 도움이 되지 않을 것이다. 스크럼의 스크럼 또한 도움이 되지 않을 것이다. 미팅을 더 자주한다고 도움이 되지 않을 것이다. 오로지 사람을 기계로 생각하는 태도를 바꿔야지만 도움이 될 것이다.

**도움이 되는 아이디어들**

아래에 당신 팀의 역량을 늘리고 계속 유지하는데 도움이 되는 제안들이 있다. 이것들이 위에 나온 다섯 개의 징후에 대한 유일한 해결책은 아니며, 다섯 개의 해결책 모두가 모든 다섯 개의 징후에 대한 해결책이 됨을 알 것이다.

- 당신이 길 옆에서 베스트 프랙티스를 보았다면, 무시하라.

“베스트 프랙티스" 아이디어는 매우 기계적인 될 가능성이 높다. 그것은 우리가 이미 논의한 문제 중 애자일 전문가 신드롬으로 연결될 가능성이 있다.

나는 당신이 어디에선가 프랙티스를 보고 당신 상황에 적용하지 않을 것이라고 단언할 수는 없다. 다만, 확실하게 말할 수 있는 건 맹목적으로 따라 해서는 안 된다는 점이다.

맹목적으로 “베스트 프랙티스”를 따라 하는 건 항상 우리를 어려움에 빠뜨린다. 우리는 프랙티스가 어딘가에서 사용되었다는 것을 이해하지만 질문을 할 수 있어야 한다. “그것이 여기서는 어떻게 활용될 수 있을까? 그것을 적용하려면 그것을 어떻게 바꿔야 할까?”

- 의사결정과 현업 간의 차이를 좁혀라

SEPG 전문가의 사례에서 보았듯이 의사결정이 업무를 수행하는 사람과 멀어질 때 상처를 입힌다. 차이가 커질수록 상처도 깊어진다. 가능한 한 의사결정과 현업간의 거리를 좁혀야 한다. 만일 업무를 수행하는 사람이 그 업무에 대한 모든 결정을 내릴 수 있다면 최상이다. 하지만 어떤 이유로든 그 둘 사이가 분리되어야 한다면, 가능한 한 차이를 좁히도록 해야 한다 (이상적으로는 같은 팀 내가 될 것이다).

- 팀간의 경계를 무너뜨려라

매니저의 가장 큰 책임은 팀간의 경계를 무너뜨리는 것이다. 팀들을 서로 서로 경쟁하게 하면 경계가 세워진다. 매니저가 다른 팀과의 의사소통을 위해 “모든 것은 나를 통해야 한다”라고 말해도 경계는 세워진다.

최고의 애자일 매니저는 경계를 무너뜨리는 작업을 할 것이다. 팀들이 다른 팀과, 다른 그룹과, 다른 부서와 제한 없이 의사 소통할 수 있는 방법을 찾아라.

- 비구조화의 가치

우리는 거의 항상 문제를 발견하면 그것을 고치기 위해 어떤 구조를 발명하려고 한다. 두 애자일 팀이 의사 소통하는데 필요한 것이 더 있나? 스크럼 미팅의 스크럼을 소집하라.

하지만 구조화되지 않을 때의 가치 또한 존재한다. 종종 더 많은 구조, 더 많은 미팅, 더 많은 역할, 더 많은 문서, 더 많은 준비 없이 문제를 해결하는 방식을 찾는 것이 가치가 있다.

상황을 개선시킬 수 있는 우리 자신의 태도에 대해 변경할 것들은 없나? 만일 그런 것이 있다면, 우리 자신의 태도를 변화시키기 위해 무엇을 해야 하나? 우리 팀 내부에 업무가 완료되는 방식과 관련한 문화를 바꿀 수 있을까?

- 당신의 프로세스들을 과도하게 엔지니어링 하지 말라

우리는 종종 우리가 어떻게 소프트웨어를 만드는지 설명하는데 소프트웨어 엔지니어링을 사용한다. 소프트웨어는 한 번 생성되고 나면 확실히 기계적이다. 따라서, 무언가 기계적인 것을 만드는 것에 대한 우리의 생각이 소프트웨어 생성을 설명할 때 들어맞는 측면이 있다.

하지만, 이것을 가지고 엔지니어 프로세스를 생각할 때는 난관에 봉착하게 된다. 우리의 프로세스는 위에서 얘기한 나의 이야기들에서처럼 기계적이지 않다. 그들은 손쉽게 적용이 가능해야 하는 살아있고, 숨쉬는 프로세스여야 한다. 이것처럼 살아있는 프로세스를 가지는 유일한 방법은 사람들이 생각하기에 옳다고 생각하는 일에 사람들을 풀어놓는 것이다. 우리는 사람이 사람답게 행동할 수 있게 해야 하며, 사람을 사람을 위해 만드는 기계 모델 속으로 집어넣지 않아야 한다. (The only way to have processes that are alive like this are to have openings in them for people to do what they think is the right thing to do. We need to allow people to act like people, and not try to force them into a machine that we've created for them.)

.Fin.