1장

서론

> 현재의 우리는 반복적으로 하는 행동의 결과이다.
> 그러므로 탁월함이란 행동이 아니라 습관이다.
>
> – 아리스토텔레스

오늘날 수많은 소프트웨어 개발자들이 답답해하고 있습니다. 힘들게 야근을 하지만 진행 중인 프로젝트는 끝날 기미가 안 보입니다. 노력이나 열망이 부족해서 그런 것은 아닙니다. 팀 구성원 모두가 프로젝트를 깔끔하게 마무리 짓고 싶어합니다. 하지만 프로젝트를 어떻게 끝내야 할지 아는 사람이 없습니다. 뭐가 효과적인 방법인지, 직장에 어떻게 적용해야 할지 공부하고 실험해볼 시간을 마련하기가 쉽지 않습니다. 사람들 대부분은 일하느라 바쁜 탓에 이런 연구에 착수하기 힘듭니다.

그래서 이 책이 나왔습니다. 이 책은 다양한 규모의 회사에서 여러 프로젝트를 통해 현장에서 검증된 기본적이고 실천적인 충고를 모아놨습니다. 실제로 도움이 된다는 걸 우리가 직접 확인한 방법들입니다. 우리는 몇 주 동안만 회사에 들어와 일하는 컨설턴트가 아닙니다. 우리는 하루하루 출퇴근하며 일했습니다. 우리는 그럴듯한 아이디어에 달려들었다가 곧이어 다음 계약 건으로 넘어가거나 하지 않았습니다. 일이 잘못될 때에도 여전히 그 자리에 남아서 무너지는 걸 지켜봤습니다. 반대로 일이 잘 풀리는 것도 지켜봤습니다.

이 책의 아이디어 중 일부는 유명한 소프트웨어 방법론에서 노골적으로 가져온 것이고, 마땅히 공적을 받을 만한 곳에 공을 돌렸습니다. 나머지 아이디어는 피와 땀 그리고 눈물을 흘려가며 제련해낸 것입니다. 수많은 도구와 기술, 그리고 최고의 업무처리 기법(Practice, 실천기법 또는 프랙티스)을 실험했고, 쓸만할 땐 받아들였습니다. 실패한 것은 던져버렸죠. 여러분이 이 책에서 볼 내용 중 일부만이 원래 모습 그대로입니다(좋은 일이죠). 우리는 '거인의 어깨 위에 서서' [1], 산업계 최고 전문가들의 아이디어를 선별해서 적절하게 바꿔놨습니다.

오늘날 소프트웨어 개발팀의 50~70%는 기본적이고 유명한 소프트웨어 업무처리 기법을 사용하지 않고 있습니다. 이들은 무엇을 해야 하는지 몰라서 그런 게 아니라, 단지 어떻게 시작해야 될지 몰라서 그럴 뿐입니다. 우리는 사람들에게 개별 아이디어의 가치를 어떻게 납득시키는지 보여드리고, 초기에 밟아 나가야 할 실질적인 단계를 내놓을 것입니다. 그러고 나서 엉뚱한 길로 들어서지 않으려면 알아두어야 할 경고 신호를 제공할 겁니다.

'현장에서 발로 뛰는' 개발자들이 이 책을 썼습니다. 이 책은 이론이 아닌, 작은 벤처기업에서부터 세계에서 가장 큰 개인 소유의 기업(SAS)에 이르기까지 우리가 겪은 경험에 바탕을 두고 있습니다. 방법론이 아닌 프로젝트에 실제로 도움이 되는 지침을 제공합니다.

우리는 대중적인 '실용주의 시리즈'를 본받아 실제적이고, 가볍고, 읽기 쉽게 구성하려 했습니다. 아무쪼록 이 책이 다른 '실용주의' 책들이 다져놓은 기반 위에 올라서기를 바랍니다.

1.1 습관화된 탁월함

자, 아리스토텔레스의 명언이 여기에 어떻게 적용될까요? "현재의 우리는 반복적으로 하는 행동의 결과이다. 그러므로 탁월함이란 행동이 아니라 습관이다." 멋진 제품 하나

1 만약 내가 멀리 내다볼 수 있었다고 한다면, 거인의 어깨 위에 서 있었기 때문이다. – 아이작 뉴튼

를 또는 여러 개를 만들었다고 해서 훌륭하다고 단정 지을 수는 없습니다. 우리가 매일 하는 일이, 다시 말해 습관이 탁월함을 결정합니다. 정말 뛰어난 제품은 그저 좋은 습관이 가져온 부가물에 지나지 않습니다.

아리스토텔레스의 명언을 우리 자신의 삶(직업적인 측면이나 개인적인 측면 모두)에 적용해보면, 우리의 삶은 습관에 따른 부가물이라는 걸 깨닫게 됩니다. 그러니 신중하게 습관을 선택하는 편이 좋습니다. 사람들 대부분은 어쩌다 보니 지금의 업무절차를 택하게 됐을 뿐입니다. 이유는 가지각색인데, 그렇게 일하도록 배웠다던가, 상사가 그런 식으로 일했다던가 하는 식입니다. 우리는 더 잘 할 수 있습니다.

의식적으로 좋은 습관을 찾아보고, 일상 생활에 적용해보세요.

이런 식으로 해보세요. 개발 방법론 하나를 선택해서 연구해보고, 그 안에서 자신에게 맞을 것 같은 습관을 하나 고릅니다(다른 것과는 별도로 사용할 수 있는 습관이어야 합니다). 일주일간 적용해봅니다. 만약 그 습관이 마음에 들고 유익한 것 같으면, 한달 정도 계속 사용해봅니다. 새로운 습관이 자연스러운 일상이 될 때까지 연습합니다. 그러고 나서 앞서의 과정을 다시 반복합니다. 벽돌을 차곡차곡 쌓아나가듯, 이런 과정을 반복합니다. 이렇게 한번에 하나씩 새로운 습관을 익혀서 탁월함의 기초를 닦는 겁니다. 여러분 환경에 적합하지 않는 방법이라면 겁먹지 말고 제거해 버리세요. 그저 유명하거나 대중적인 방법이라고 여러분에게 맞지도 않는 방법을 계속 유지해선 안 됩니다. 여러분에게 무엇이 적합하고 뭐가 맞지 않은지 파악해서 자신만의 방법을 갈고 닦으세요.

"하루하루를 보낸다는 것은 다시 말해 삶을 보낸다는 말이다." [2] 이 말이 옳다면 하루하루를 어떻게 살아갈 것인지 신중하게 생각해야 할 것입니다.

2 Annie Dillard (U.S. author, poet, and Pulitzer prize winner in 1975 for nonfiction) 애니 딜라드 (미국의 작가이자 시인. 1975년에 논픽션 부분에서 퓰리처 상을 수상했다.)

TIP 조언 1

습관을 선택하세요

그저 되는대로 습관을 만들면 안 됩니다. 신중하게 습관을 선택하세요.

1.2 실용주의적 관점

이 책은 뭐가 좋고 그른지를 학술적으로 분석하지 않습니다. 여러분이 취사선택할 수 있는 방법론과 업무처리기법을 모아놓지도 않았습니다.

그 대신 이 책은 실제 소프트웨어 프로젝트에서 우리한테 잘 맞았던 것을 제시합니다. 우리는 효과적인 방법이라는 게 분명해지고서야 새로운 도구나 업무처리기법을 도입해서 사용했습니다. 우리는 쓸만했던 도구나 업무처리기법을 '소프트웨어 개발 도구상자'에 넣어두고 줄곧 사용해왔습니다. 어쨌거나 우리는 자신이 하고 있는 일이 무엇인지 제대로 알게 됐습니다. 이러한 도구와 업무처리 기법이 여러분에게도 유용하기를 바랍니다.

우리는 어떤 개발 방법론을 단지 '올바른 방법'이란 이유만으로 채택하기엔 여유가 없는 벤처기업에서 일한 적이 있습니다. 주변여건 때문에 당장 업무에 적용할 수 있는 검증된 아이디어를 찾아내야 했습니다. 우리는 막대한 자원과 기술을 마음껏 쓸 수 있는 큰 회사에서 일해보기도 했습니다. 그러나 큰 회사조차도 멋진 도구라서 또는 몇몇 전문가가 추천했다고 해서 특정 도구를 사용하려고 들진 않는다는 걸 알게 됐습니다. 회사는 당면한 문제를 빠르고 값싸게 풀어줄 해결책을 원합니다. 그래서 우리는 어떤 습관을 채택하기도 하고 버리기도 하면서, 다른 곳에도 적용할 수 있으면서도 문제를 효과적으로 풀어낼 수 있는 도구 집합을 만들어냈습니다. 이 책은 여러분의 회사에서도 변화를 불러일으킬 좋은 습관을 모아놨습니다. 그 결과는 놀라울 것입니다.

이 책이 가져다 줄 결과를 묘사하기 위해 '두 소프트웨어 회사 이야기'(찰스 디킨즈의

'두 도시 이야기'에 폐를 끼치겠지만)를 해보겠습니다.

첫 번째 회사는 엉망이었습니다. 꽤 비싼 소스 코드 관리 시스템을 구입하고도 설치조차 하지 않았습니다. 그 결과, 잠재적인 고객에게 보여줄 데모의 소스 코드를 잃어버리고 말았습니다. 어떤 기능이 제품에 포함되어야 하는지 아는 사람이 없는데도, 개발팀 전체가 제품에 달려드는 상황이었습니다. 코드는 불안정했고 거의 5분마다 충돌이 일어났습니다(더 나쁜 순간도 잦았는데, 실제 데모를 하고 있을 때 충돌이 일어나기도 했습니다). 이런 뒤죽박죽인 상황은 사기 진작에 도움이 되지 못했습니다. 회의는 고함치기 대회로 곤두박질치기 일쑤였습니다. 몇몇 개발자들은 하루 종일 사무실에 틀어박혀 그 같은 상황을 모면했습니다. 간단히 말해 일하기 나쁜 곳이었습니다. 심각한 문제가 있다는 걸 모르는 이가 없었지만, 아무도 문제를 해결하지 못했습니다.

두 번째 회사는 훨씬 나았습니다. 비슷한 수의 개발자가 있었지만, 그들은 주요 제품 세 개를 동시에 개발했습니다. 세 프로젝트의 코드는 소스 코드 관리 시스템 안에 들어 있었습니다. 팀 전체가 모여서 일일 회의를 했는데, 짧고, 프로답고, 그리고 효과적이었습니다. 각 프로젝트마다 종합 계획이 수립되어 있었기에 개발자 모두가 어떤 기능을 개발해야 하는지 알고 있었습니다. 사람들은 채석장 일꾼들의 신조에 따랐습니다. 돌덩이를 자를 때에도 마음 속에는 성당을 그려야 한다[HT00[3]]. 말인즉, 모든 사람이 보다 큰 협력체계 안에서 자신의 전문지식과 기술을 적용할 수 있었습니다. 그들은 매우 숙련된 개발자였기 때문에 야단법석 떨지 않고도 제품을 제때 전달해냈고, 심지어 제품은 안정적이기까지 했습니다.

가장 놀라운 점은 이 두 회사가 6개월도 안 되는 시간을 사이에 둔 같은 회사라는 사실입니다. 단지 이 책의 원리들을 적용했을 뿐입니다(여러분은 벌써 그러리라 짐작했을 겁니다. 그렇지 않나요?). 일정 기간이 지난 후, CEO는 우리가 '탁월함의 분위기'를 가져와 주었고, "알아보기 힘들 정도로 변했다"라고 말했습니다. 이 회사는 우리가 최

3 부록 238쪽 참고 문헌을 참고하세요.

근에 일한 곳 중 하나입니다. 여러분에게 제시하는 것과 거의 동일한 형태의 원리를 적용했었습니다. 그곳에서 겪은 변혁이 이 책을 쓰게 된 이유 중 하나입니다.

우리는 4명밖에 안 되는 작은 벤처기업에서부터 세계에서 가장 큰 개인 소유의 소프트웨어 기업인 SAS에 이르기까지 이러한 아이디어를 발견하고 적용해왔습니다. 솔직히, 회사 규모에 상관없이 이런 원리들이 효과적이라는 사실을 알고 우리 자신도 놀랐습니다.

이 책의 아이디어들을 큰 프로젝트의 기초쯤으로 생각하세요. 만약 여러분이 인프라스트럭처를 적절하게 설정하기 위해 미리 시간을 투자할 의향만 있다면, 프로젝트 생명주기 나머지 기간 내내 그 효과를 거둬들일 수 있습니다. 물론 모든 업무처리기법을 제대로 잡은 상태에서 프로젝트를 시작하는 편이 더 쉬울 겁니다. 건물의 깨진 토대마냥, 어떤 것은 쉽게 땜질이 되는 반면에 어떤 것은 매우 구조적인 문제라 고쳐서 원상 복귀시키려면 꽤 많은 노력이 듭니다.

여러분이 현재 프로젝트를 진행 중이더라도 좋은 습관들을 적용해보기에 늦은 것은 아닙니다. 지금 프로젝트에 이 책의 아이디어 중 일부를 도입해서 즉각적인 효과를 볼 수도 있습니다. 어떻게 하면 되는지는 마지막 장에서 다룰 예정입니다.

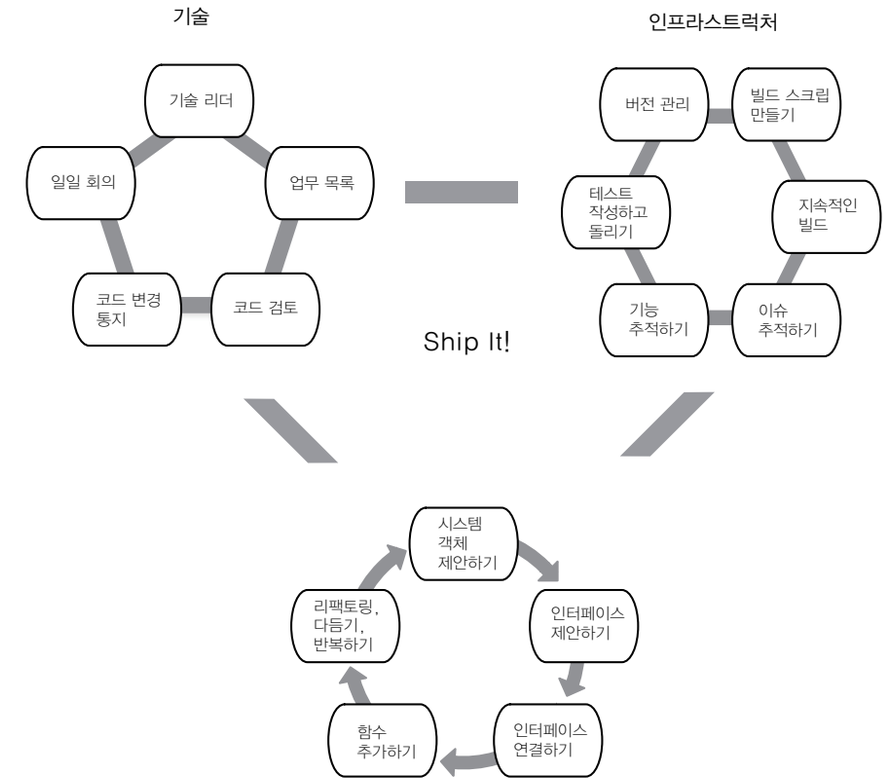
1.3 로드맵

아이디어들을 크게 세 영역으로 정리했습니다. 인프라스트럭처, 기법, 그리고 프로세스입니다(그림 1.1을 보세요). 이 영역들은 고객들이 원하는 제품을 지속적으로 제공하는 능력과 관련 있습니다.

인프라스트럭처

'도구와 인프라스트럭처' 장에서는 여러분 자신과 팀 전체의 삶을 편하게 해줄 소프트웨어 도구들을 다룹니다. 예를 들어, 좋은 소스 코드 관리 시스템은 프로젝트의 보석 같

그림 1.1 멋진 제품을 만드는 법



은 존재인 소스 코드를 안전하고 결실하게 지켜줍니다. 자동화된 빌드 시스템은 원하기만 하면 언제 어디서나 빌드를 반복할 수 있게 해줍니다. 또한 여러분이 무엇을 개발해왔는지 다른 사람들도 알 수 있게 하려면, 버그 보고서와 기능 요청서 그리고 다른 이슈들을 어떻게 추적하고 관리해야 하는지에 대해서도 논의할 겁니다. 마지막으로 좋은 테스트 장비를 어떻게 사용해야, 여러분이 생각했던 대로 코드가 작동하리라는 자신감을 갖게 되는지 보여드리겠습니다.

기법

'실용적인 프로젝트 기법' 장에서는 여러분과 팀이 매일 '힘들이지 않고, 보다 현명하게' 일하는 데 필요한 특정 기법들을 다룹니다. 팀에 기술 리더를 두어서 바깥 세상으로부터 여러분을 보호하고 알 필요가 있는 것들만 전달받는 방법을 알려드리겠습니다. 그룹이 제 방향으로 나아가길 원한다면, 여러분 자신의 일이나 팀의 업무를 업무 목록으로 정리하세요. 팀이 서로 대화를 나누지 않습니까? 누가 뭘 하고 있는지 모릅니까? 모두를 하나로 결속시키고 동시에 팀 동료의 마음을 얻을 기회가 필요하다면 일일 회의[4]를 시작해보세요. 짧은 코드 검토 과정을 통해 동료의 전문 지식을 활용하고 여러분의 지식도 공유할 수 있습니다. 그리고 검토가 끝나면, 코드 변경 내역을 공지해서 나머지 팀원도 알 수 있게 하세요.

프로세스

소프트웨어 개발에 관한 책치고 작가가 선호하는 개발 방법론을 다루지 않고 넘어가는 법이 없습니다. 이 책도 다르지 않습니다. 부족하지만 우리가 예광탄 개발이라고 이름 붙인 방법론을 추가했습니다. 예광탄 개발 방법을 사용하면, 거의 뼈대가 갖춰진 작동 가능한 시스템을 만든 다음에 구현 과정을 거치면서 빠진 부분만 채워넣으면 됩니다. 그러고 나서 실제로 구현하기 위해 빠진 부분을 채워 넣어서 만들기만 하면 됩니다. 예광탄 개발은 큰 프로젝트를 작게 쪼개서 각 팀이 하나씩 병행해서 작업하기에 좋습니다. 뿐만 아니라 자동화된 테스트에도 알맞습니다.

흔하게 벌어지는 문제와 문제를 해결하는 법

마지막엔, 흔하게 벌어지는 문제들을 제시하고, 앞서 언급한 도구와 기법, 그리고 프로세스를 사용해서 그런 문제를 어떻게 해결하는지 조언해 드리겠습니다. 수년간 이런 문제를 수도 없이 겪었습니다. 문제 중 일부는 해결했고, 어떤 것은 나중에야 해결법을 알게 됐습니다(지나고 나면 모든 게 확실해 보이는 법이죠). 여러분이 똑 같은 실수를 저지르지 않도록 우리의 경험이 도움이 되기를 바랍니다.

4 "실천가를 위한 실용주의 프로젝트 관리" 참조(위키북스, 2007)

무엇이 빠졌는가?

사람 문제와 요구사항 수집에 대해서 다루지 않습니다. 제대로 된 사람이라면 도구, 기법, 그리고 프로세스 따위를 프로젝트의 가장 중요한 요소라고 생각하지는 않습니다. 그러나 훌륭한 팀을 모으고 유지하는 것은 책 한 권이 따로 필요한 주제입니다(한 권이 아니라 시리즈가 필요할 수도 있습니다!). 대신 우리는 여러분의 팀이 이미 갖고 있는 기술을 활용하고 키우는 방법에 초점을 맞춥니다.

사람 문제와 마찬가지로, 제품의 요구사항에 대해 배우는 것도 또 하나의 큰 주제입니다. 메모 카드를 활용하는 방법부터 복잡한 시스템을 도입하는 것에 이르기까지 요구사항을 수집하는 방법은 다양합니다. 한 장에 담을 수 없는 커다란 주제를 다루려고 하기보단, 계속 변하는 요구사항을 다룰 수 있는 유연한 아이디어를 제공하기로 결정했습니다. 그 요구사항이 어떤 출처에서 나온 것이든 간에 말이죠. 이 책의 아이디어는 요구사항이 변하지 않는 프로젝트뿐만 아니라 요구사항이 끊임없이 변하는 프로젝트에도 쓸모가 있습니다. 그러니 여러분은 작은 종이 카드에서 추출한 요구사항 목록이든, 10,000쪽짜리 계약서에 근거를 둔 요구사항이든 상관없이 이 책의 아이디어를 사용할 수 있습니다.

여러분이 어떤 회사에 있든, 어떤 기술을 사용하건 간에 이 책의 아이디어를 활용할 수 있도록 일반적인 이야기를 하려고 노력했습니다. 그래서 인스톨러 기술이나 코드 최적화 도구에 대해서는 다루지 않습니다.

1.4 앞으로 나아가기

우리는 여러분이 이 책에서 제시하는 아이디어와 습관을, 그것들을 단조(鍛造)해낸 정신에 입각해서 사용해주길 바랍니다(다시 말해 실용적으로 활용해주기 기대합니다). 읽고 시도해보세요. 여러분 환경에 적합한 것은 남기고 나머지는 버리세요.

매 장을 읽고 나면 잠시 멈춰서 제시된 아이디어를 사용하고 있는지 판단해보세요. 사용하지 않고 있다면 '어떻게 시작해야 하나?'를 읽으세요. 벌써 사용하고 있다면 '내가 제대로 하고 있는 걸까요?'나 '경고 신호'를 읽으세요. 올바른 방향으로 나아가고 있는지 확인할 수 있습니다.

1.5 이 책을 어떻게 읽어야 하나?

프로젝트에서 여러분이 맡은 역할에 따라 이 책을 읽는 방법도 달라집니다. 당연한 말이지만, 여러분이 개발자이거나 테스터라면 팀 리더와는 다른 접근 방식을 택해야 할 겁니다. 하지만 어떤 역할이든 간에 이 책에서 가치 있는 것을 많이 얻을 수 있을 겁니다.

여러분이 개발자이거나 테스터라면

여러분이 현장 실무자, 즉 행동해야 하는 사람이라면 이 책을 처음부터 끝까지 읽으세요. 각 섹션은 여러분이 개인으로서나 팀 리더로서 매일 활용할 수 있는 실용적인 아이디어를 제시합니다. 간혹 팀 리더가 아닌 개발자가 팀에 중점을 둔 섹션을 넘어가는 경우가 있습니다. 정말 바람직하지 못한 생각입니다. 팀 환경이란 대부분 팀 구성원들이 무엇을 요청했는가, 무엇을 체험했는가에 따라 달라집니다. 어떤 도구나 기법, 그리고 프로세스가 회사에 긍정적인 영향을 미칠 수 있는지 알아뒀다가, 제안할 일이 있을 때마다 명확한 이유를 제시할 수 있어야 합니다. 개발자들이 특정 도구나 기법을 "이게 올바른 방식이예요."라며 옹호하는 걸 여러 번 봤습니다. 이런 식의 주장으로는 결코 관리자를 설득할 수 없을 뿐더러 사실상 반생산적이기까지 합니다. 아이디어를 제시하기 전에 팀에 어떤 이득이 되는지부터 확실히 해둬야 합니다.

어떤 제안이 마음에 드시나요? "Acme Code Systems 사의 소스 코드 관리 시스템이 필요합니다. 제품이 좋은데다가 모든 사람이 그걸 사용합니다. 이 제품이 최고입니다!" 다른 방식의 제안도 들어보시죠. "소스 코드 관리 시스템이 필요합니다. 이 시스템이 있으면 옛날 릴리즈에 접근할 수도 있고, 특정 코드 변경을 되돌리거나 개발자들이 코드 트리를 동시에 건드릴 수도 있습니다. 우리 회사가 개발에 투자한 것을 보호할 수 있는 가장 손쉬운 수단입니다. Acme Code Systems 사는 눈여겨봐야 할 훌륭한 제품을 만듭니다. 조와 저는 몇 개월 동안 그 제품을 사용해왔고, 생산성이 정말 크게 향상됐습니다. 이 제품이 어떻게 도움이 되었는지 목록을 작성해봤습니다. "

여러분이 프로젝트 팀 리더라면

팀의 주변여건과 작업 흐름을 점검하는 데 이 책을 사용하세요(이미 가끔씩 점검을 하고 있을 겁니다. 맞나요?). 팀이 일하는 방식을 재점검할 기회를 갖도록 하세요. 기초적인 요구사항을 다룰 수 있는 기본적인 도구 집합을 가지고 있습니까? 여러분 팀의 기술은 견실한 제품을 내놓고, 능력 있는 개발자를 배출할 만큼의 수준이 됩니까? 깔끔하고 잘 정의된 프로세스를 갖고 있나요?

팀이 일하는 방식을 검토할 때, 개별 항목이 타당한지 반드시 확인하도록 하세요. 예전에는 적합했지만 이제는 쓸모없어진 도구나 업무처리기법을 여전히 사용하고 있지는 않나요?

햄의 1/3을 잘라서 버리고 요리하는 여성에 대한 이야기를 들어보셨습니까? 왜 그렇게 요리하는지 묻자, 그녀는 어머니가 그런 식으로 요리했다고 대답했습니다. 어머니에게 이유를 묻자, 자신의 어머니가 요리하던 방식이라고 했습니다. 마침내 할머니와 대면했습니다. 할머니는 어렸을 때 햄을 통째로 담을 팬이 없어서 끝자리를 잘라내곤 했는데, 그것이 습관이 돼버렸다고 고백했습니다.

지난 해의 프로젝트나 할머니의 괴팍함이 아닌 지금 이 순간의 필요에 따라 습관을 형성하도록 주의하세요.

팀 전체가 필요한 도구, 기술, 그리고 프로세스에 접근할 수 있도록 하세요. 무엇이 왜 효과적인지 알아야만 팀을 효과적으로 이끌어나갈 수 있습니다. 각각의 섹션에는 막 도입할 시기에 도움이 될만한 팁이 제시되어 있습니다. 뿐만 아니라 걷잡을 수 없이 커지기 전에 문제를 파악할 수 있도록 경고 신호도 제시했습니다.

여러분이 관리자거나 깊게 관련된 고객이라면

상위 관리층이라면 적절한 정보를 요청하는 것만으로도 팀이 일하는 방식에 영향을 크게 미칠 수 있습니다. 이 책은 팀이 사용해야 할 핵심요소 일부를 보여주고, 어떤 질문을 던져야 하는지 알려줍니다. 예를 들어 지난 출시 때의 수정 건 목록을 요청하면, 그 정보를 원한다고 말하는 셈이 됩니다. 각 섹션을 읽으면서, 여러분이 원하는 방향으로 사람들을 이끌어 줄 산출물이 없는지 눈여겨 보세요. 팀 리더들에게 그 산출물을 제출하게끔 하면 됩니다. 그렇지만 이런 요청은 주의해서 해야 합니다. 관료주의적 잡일을 만들고 싶진 않을 겁니다. 사람들을 이끌 때는 신중하게 요청해야 합니다.

관리자가 돼서 일상 업무에서 제외되었으니, '어떻게 시작해야 될까요?' 장을 대충 훑어보고 넘어가려고 할지 모릅니다. 하지만 각 주제가 무엇을, 왜 다루는지 이해하고 싶을 겁니다.

개인이 모여 팀을 이룬다

이 책이 다루는 개념 중 대부분은 개인이 아닌 팀 구성원, 모든 팀, 그리고 관리자가 사용해오던 것입니다. 누군가가 업무처리 기법을 먼저 사용해보고 쓸만하다 싶어서 팀과 공유하는 일은 흔합니다. 우리는 이런 일을 반복해왔고, 다른 이가 그렇게 하는 걸 지켜봤습니다. 여러분도 똑같이 할 수 있습니다. 그렇게 해 본 누군가의 이야기를 들려드리겠습니다.

기민성 지원시스템의 빠른 성장

– CafePress.com의 도미닉 플란테, 저스틴 맥카티 공저

지난해 초 CafePress.com에서 막 일하기 시작했을 무렵, 관리층은 기민한 실천기법(Agile practices)을 도입하는 데 적극적이었습니다. 그러나 개발 환경은 자신감을 갖고 변화를 이뤄내는 데 필요한 기본적인 지원 시스템을 갖추고 있지 못 했습니다.

사람들이 자신만의 제품(티셔츠나 머그잔 같은)을 쉽게 디자인하고 구매할 수 있도록 CafePress의 핵심 기능을 확장하는 '직접 만들고 사기' 프로젝트를 시작했습니다. 프로젝트는 웹 프리젠테이션 계층에 더해 명확한 비즈니스 및 퍼시스턴스 계층을 도입하려는 첫 시도였습니다. 비즈니스 및 퍼시스턴스 계층은 대부분 테스트 주도 방식으로 개발됐습니다. 개발자는 NUnit 프레임워크를 사용해 테스트를 작성했습니다. 그와 동시에, 웹 계층에서 사용하는 클래스들을 컴파일하고 배치하는 반복적인 작업을 하려고 NAnt를 도입했습니다. 그런 다음 서브버전 코드 저장소로 들어오는 모든 체크인을 지속적으로 통합하려고(예를 들어, 컴파일하고 테스트를 돌려보기 위해) CruiseControl.NET을 묶어 넣었습니다. 마무리로, '치킨 리틀'이란 애칭을 붙인 작지만 눈에 띄는 (그리고 음향까지 나는) 워크스테이션에다가 CCTray[5]를 돌려서 빌드 상태를 알리게 했습니다.

서브버전, CruiseControl.NET, NAnt, 그리고 NUnit 간의 상호운영은 쾌적한 협력환경을 꾸리는 데 도움이 됐습니다. 이론이 분분한 벤더 분석이나 구매 결정은 없었습니다. 게다가 이런 지원 시스템들은 개발자가 원해서 도입됐습니다. 관리층의 지시 때문에 도입한 것이 아닙니다.

이렇게 자동화를 시작한 이래로 우리 팀은 커졌고, 여러 신참자들이 테스트 수트와 프로

5 빌드 실패를 알려주는 트레이 아이콘 애플리케이션입니다.

젝트 자동화 도구를 개선해나갔습니다. 최근 업데이트 중 일부는 자동화된 테스트 환경 배포 기능뿐만 아니라 100% 스크립트로 만들어진 개발환경구성 기능까지 포함합니다. 하지만 돌아보면, nant 테스트가 여전히 가장 많이 사용되는 타겟입니다.

이런 도구들이 출현하기 전에는 우리가 하루 동안 나누는 대화는 대부분 빌드 실패에 대해 사람들에게 질문하거나 API 변경을 통지하는 것이었습니다. CruiseControl.NET이 빌드를 처리해주기 때문에, 개발자들은 빌드를 원래 상태대로 유지하는 게 얼마나 중요한지 이해했고 그렇게 하려고 헌신했습니다. 누구도 잘못된 체크인 때문에 작업을 정지해야 하는 상황을 좋아하지 않았습니다. 지원 시스템을 제대로 갖춰놓으니, 자연스럽게 소프트웨어 설계와 구현에 대해 이야기 나누게 됐고, 주변 환경 때문에 발생한 좌절감으로부터 빠져나올 수 있었습니다.

우리가 처음에 들인 노력 전부가 테스트한 코드를 즉시 전달하는 데 도움이 됐고, 초창기에 한 투자 덕분에 '치킨 리틀'이 꿱꿱거릴 때마다 그 값어치를 할 수 있었습니다.

이 이야기처럼 개발자가 주도해서 변화를 일으키는 게 이상적이라고 생각합니다. 변화를 도입하려는 관리자를 실망시키려는 건 아닙니다. 하지만 가장 훌륭하고, 가장 적합한 변화 사례는 현장에서 나왔다는 걸 알게 됐습니다. 일을 하는 사람이 어떤 문제가 해결돼야 하는지 잘 알기 마련입니다.

그러니 여러분이 관리자이든, 개발자든, 테스터든, 또는 기술 리더이든 "이 책을 써먹으세요." 개인적인 일이나 회사업무에서 빠진 부분을 찾아내서, 어떻게 해야 여러분의 삶이 좀 더 편해질지 알아보세요.

Welcome any question! 무엇이든지 물어 보세요!

기민함이란 게 뭐죠?

기민함이란 소프트웨어 개발팀이 변화하는 환경에 재빨리 적응하는 능력을 일컫습니다. 이것은 변화하는 요구사항에 맞춰 재설계하는 걸 뜻하기도 하고, 새로운 버그에 신속히 대처하거나 새 기술을 빨리 도입하는 걸 뜻할 때도 있습니다. 일반적으로, 기민한 팀은 관료주의보다 결과에 신경씁니다. http://www.agilemanifesto.org/ 에서 기민한 소프트웨어에 대해 자세히 알아볼 수 있습니다.

웹 사이트에서 가져온 다음 인용문이 기민한 관점을 꽤 잘 요약하고 있습니다.

* * *

"우리는 직접 해보거나 다른 사람을 도와주면서 소프트웨어를 더 잘 개발하는 방법을 밝혀나가고 있다. 이 같은 작업을 통해 우리는 다음과 같은 가치에 도달했다.

프로세스나 도구보다는 사람과 상호작용에,

포괄적인 문서보다는 실제로 도움이 되는 소프트웨어에,

계약 협상보다는 고객과의 협력에,

계획을 따르는 것보다 변화에 맞추어나가는 것에,

왼쪽에 명시한 항목도 가치 있지만, 우리는 오른쪽 항목을 더 중요하게 여긴다."