지속적인 통합

위험을 줄이기 소프트웨어 품질을 높이고

폴 M. 듀발
스티브 M. 마티아스
앤드류 글로버

최재훈

위키북스

목 차

역 자 서 문

아침 일찍 출근하고 컴퓨터를 켠다. 윈도우 로고가 뜨는 사이, 가방 정리를 하고 컵을 씻고 홍차를 준비한다. 그러다 윈도우 로그인 창이 뜨면 암호를 입력한다. 윈도우 바탕화면이 뜨고 웹 브라우저를 연다. 차 한 모금을 입 안에 담고, 느긋하게 RSS 구독기를 통해 어제 밤 사이에 올라온 글을 읽는다.

뭔가 이상하지 않습니까? 다시 해볼까요?

암호를 입력한다. 윈도우 바탕화면이 뜨면, IDE를 실행시키고 소스 코드를 열심히 살펴본다.

그래도 여전히 뭔가 이상한데요? 어떤 사람은 출근해서 웹툰부터 볼지도 모릅니다. 너무 오래 만화 감상에 열중하지 않는다면, 만화를 보며 긴장을 풀어도 좋겠지요. 하지만 여기서는 딱딱하게 '회사에선 일만 해라' 라고 말하려는 게 아닙니다. 게임 개발사에선 '게임을 즐기는 것조차 일'이라고 하더군요. 그런 면에선 저는 직무 유기를 한다는 소리를 들어야 할지도 모르겠습니다. 그렇다면 처음 제시한 시나리오가 왜 이상하다는 걸까요? 한 번만 더 해보겠습니다.

암호를 입력한다. 시작 프로그램들이 뜬다. 윈도우 작업 표시줄에 CCTray 트레이 아이콘이 뜨더니, 이내 반가운 녹색 불빛이 들어온다. '아, 오늘 아침도 프로젝트는 안녕한가 보구나'. 안심하고 마음 편히 '마음의 소리'(웹툰)을 보러 간다.

CCTray는 CruiseControl .NET이라는 빌드 자동화 서버와 통신하고, 빌드가 깨지지 않았는지, 테스트가 실패하지 않았는지 알려줍니다. 녹색 불빛은 '걱정하지 말라'는 표시입니다. 아침에 출근해서 손수 소스 코드를 빌드해보지 않아도, 프로젝트가 잘 돌아가는지 알게 해주니 참 고마운 도구입니다.

생각해보세요. 여러분은 아침에 출근하면서, 전날에 밤을 샜다면 낮에 출근하면서 이런 생각해본 적 없습니까? '오늘은 제발 프로그램을 띄워보기라도 하자'. 컴파일은 어떻게든 되는데, 막상 돌려보면 어찌된 영문인지 이상한 행동만 보이고, 뭐가

문제인지 알고 보면 한 달 전에 겪었던 문제이고, 끝이 없습니다. 회사 갈 생각만 해도 머리가 지끈거리니 참 큰일입니다.

이 책은 두통거리를 제대로 다루는 방법에 대해 말합니다. CCTray만 봐도 안심하고 일할 수 있는 그런 환경을 만드는 법을 알려줍니다.

사실 역자 서문이란 게 그리 흥미로운 대목은 아닙니다. 독자들이 알고 싶어하는 내용은 전부 저자들이 설명해놨고, 역자가 할 일이라곤 그저 이 책이 얼마나 훌륭한지, 왜 이 책을 읽어야 하는지 간단히 설명하는 정도일 뿐이죠. 그러니 이쯤에서 마침표를 찍겠습니다. 앞서 제시한 시나리오만으로도 명석한 여러분은 이 책이 왜 쓸모 있는지, 이 책에서 무엇을 배우게 될지 알았을 겁니다.

그럼, 즐거운 시간이 되시길 바랍니다.

2008년 3월 어느 날,

역자 최재훈.

저 자 · 역 자 소 개

폴 M. 듀발 | Paul M. Duvall

폴 M. 듀발은 컨설팅 회사인 Stelligent 사의 CTO이자, 소프트웨어 생산을 최적화함으로써 개발 팀이 안정적이고 신속하게 더 나은 소프트웨어를 만들도록 돕는 사고 리더입니다. 그는 개발자와 테스터에서부터 아키텍트와 프로젝트 관리자에 이르기까지, 소프트웨어 개발 프로젝트의 사실상 모든 역할을 다 맡아봤습니다. 폴은 금융, 주택, 정부, 보건, 그리고 큰 규모의 독립적 소프트웨어 벤더를 비롯해 다양한 산업 분야에서 고객들에게 컨설팅을 해왔습니다. 그는 여러 앞서가는 소프트웨어 컨퍼런스에서 대표 연설자를 맡고 있습니다. 그는 IBM developerWorks에 사람을 위한 자동화(Automation for the People) 시리즈를 썼고, 『NFJS 2007 Anthology』(실용주의 프로그래머, 2007)의 공저자이며, 『UML 2 Toolkit』(Wiley, 2003)의 공헌 저자이기도 합니다. 폴은 특허 신청을 해놓은 의료 연구 데이터 관리 시스템의 공동 발명가입니다. 그는 www.testearly.com와 www.integratebutton.com에서 열심히 블로깅을 합니다.

스티븐 M. 마티야스 3세 | Stephen M. Matyas III

스티브 M. 마티야스 3세는 5AM Solutions 사의 서비스 부분인 AutomateIt의 부사장인데, 이 회사는 조직이 자동화를 통해 소프트웨어 개발을 개선하도록 돕습니다. 스티브는 응용 소프트웨어 공학에서 다양한 배경을 갖고 있으며, 민간 기업과 정부 고객 모두를 경험해보았습니다. 스티브는 아주 다양한 역할을 수행해왔는데, 비즈니스 분석가와 프로젝트 관리자부터 개발자, 설계자 및 아키텍트까지 경험해봤습니다. 그는 『UML 2 Toolkit』(Wiley, 2003)의 공헌 저자입니다. 스티브는 애자일과 Rational Unified Process(RUP)를 비롯해 다양한 반복적이고 점진적인 방법론을 실천하는 사람입니다. 그는 방법론과 소프트웨어 품질, 그리고 프로세스 개선에 있어 특화를 해야 하는 자바/J2EE 사용자 주문 소프트웨어 개발 및 서비스 산업에서 풍부한 실무 경험을 가지고 있습니다. 스티브는 Virginia Polytechnic Institute and State University(버지니아 공대)에서 컴퓨터 과학 분야의 학사 학위를 획득했습니다

앤드류 글로버 | Andrew Glover

앤드류 글로버는 컨설팅 회사인 Stelligent 사의 사장이자, 소프트웨어 생산을 최적화함으로써 개발 팀이 안정적이고 신속하게 더 나은 소프트웨어를 만들 수 있도록 돕는 사고 리더입니다. 앤디는 북 아메리카에서 벌어지는 다양한 컨퍼런스에서 자주 연사로 활동하며 No Fluff Just Stuff 소프트웨어 심포지움 그룹에서도 연설을 맡습니다. 그리고 앤디는 『Groovy in Action』(Manning, 2007), 『Java Testing Patterns』 (Wiley, 2004), 『NFJS 2006 Anthology』(Pragmatic Programmers, 2006)의 공동 저자입니다. 그는 IBM의 developerWorks와 오라일리의 ONJava, ONLamp, Dev2Dev 등 다양한 온라인 출판물의 저자이기도 합니다. 또 www.thediscoblog.com와 www.testearly.com에서 소프트웨어 품질에 대해 열심히 블로깅을 합니다.

최재훈 http://kaistizen.net, kaistizen@gmail.com

게임 개발을 업으로 삼는 프로그래머다. 게임 개발에 있어선 이제 막 반년이 지난 초보이지만, 흥미롭고 생소한 기술을 많이 접하고, 익숙하진 않지만 독특한 분위기를 누리느라 항상 즐겁다.

여전히 스타크래프트는 무한맵에서 하고, 올해에 나올 게임 기대작으론 스포어(Spore)를 손에 꼽는다. 게임 개발자답지 않게 평소엔 게임을 즐겨 하지 않지만, 기대작이 나오면 다른 일은 제쳐놓고 삼매경에 빠진다. 유머 사이트에 떠도는, '슈퍼로봇대전 OGS'의 엔딩을 보고 싶다고 적은 휴가 계획서를 보고 그 용기가 멋지다고 생각하는 사람이다.

락 밴드는 U2가 최고라 생각하고, 15년째 '배철수의 음악 캠프'를 들은 애청자이다.

『Ship it! 성공적인 소프트웨어 개발 프로젝트를 위한 실용 가이드』에 이어 이번이 두 번째 역서이다. 번역을 마치고 몇 번이나 검토를 해도, 여전히 고칠 부분이 많아 부끄럽다. 그저 전작 때보단 좀더 제대로 번역했다는 소리를 듣고 싶다.

마 틴 파 울 러 [1] 의 머 릿 말

1) 마틴 파울러는 마틴 파울러 시리즈의 편집자이며, ThoughtWorks 사의 선임 과학자입니다.

소프트웨어 업계에서 근무한 지 얼마 되지 않았을 무렵엔 개별적으론 잘 동작하던 모듈도 한데 모으면, 도대체가 이해하기조차 어려운 형태로 모든 게 엉망이 되어 버리곤 했습니다. 하지만 지난 몇 년간, 통합은 프로젝트에서 가장 고생스러운 일에서 별 것 아닌 일로 축소되었습니다.

이렇게 된 가장 근본적인 원인은 좀더 자주 통합하는 실천 방법입니다. 한때는 매일 빌드하는 것을 야심찬 목표로 생각한 적도 있었습니다. 지금 이야기하는 대부분의 프로젝트들은 하루에도 여러 차례 통합을 합니다. 참 이상하게도, 고생스러울수록 좀더 자주하는 것이 상책인 듯합니다.

지속적인 통합에 있어 흥미로운 점 중 하나는 지속적인 통합이 주는 효과에 사람들이 종종 놀란다는 사실입니다. 우리는 사람들이 지속적인 통합의 효과를 별로 중요하지 않은 이점으로 치부해버리는 모습을 발견합니다. 하지만 지속적인 통합은 프로젝트에 완전히 다른 분위기를 불러일으킬 수 있습니다. 문제를 더 빨리 탐지해내기 때문에 가시성이 훨씬 좋아집니다. 잘못을 저지르고 그것을 발견해내는 간격이 줄기 때문에, 뭐가 변경됐는지 쉽게 살펴볼 수 있어서 원인을 찾아내는 데 도움이 되므로 결함을 찾아내기 더 쉽습니다. 이것이 테스트 프로그램과 결합하면, 버그를 획기적으로 줄일 수 있습니다. 그 결과, 개발자는 디버깅하는 데 시간을 덜 쓰고 기능을 넣는 데 시간을 더 쓰게 되며, 견고한 기초를 닦고 있다는 자신감을 갖게 됩니다.

물론, 통합을 더 자주하라고 말로만 해선 충분하지 않습니다. 이 간단한 캐치 프레이즈 뒤에는 지속적인 통합을 현실로 만들어 주는 원칙과 실천 방법이 한 다발 있습니다. 원칙과 실천 방법에 관한 조언들은 대부분 여러 책과 인터넷에 흩어져 있었기 때문에 스스로 찾아서 탐구해야 했습니다.

그래서 폴이 이러한 정보들과 최고의 실천 방법들을 응집력 있는 한 권의 책으로 묶어서 이러한 책을 기다렸던 사람들에게 지침서를 만들어준 걸 보고 기뻤습니다. 간단한 실천 방법이 으레 그렇듯, 세세한 부분으로 들어가면 어려워집니다. 지난 몇 해에 걸쳐 우리는 그 같은 세부 사항에 대해 많이 연구했고 어찌 대처하면 되는지를 공부했습니다. 이 책은 지속적인 통합이 소프트웨어 개발에 탄탄한 기초를 제공하듯, 지속적인 통합을 위한 탄탄한 기초를 마련해주는 교훈들을 모아놓았습니다.

폴 줄 리 어 스 의 머 릿 말

누군가 하루 속히 이런 책을 써주길 바라왔습니다. 이런 책을 쓰게 될 사람이 제가 되길 남몰래 바라왔지요. 하지만 폴, 스티브, 그리고 앤드가 마침내 모든 것을 한데 묶어 응집력 있고 사려 깊은 전문 서적으로 만들어내서 기쁩니다.

저는 영원처럼 느껴질 정도로 오랜 세월 지속적인 통합에 푹 빠져 지내왔습니다. 2001년 3월에 CruiseControl 오픈소스 프로젝트를 공동 설립하고 관리자로 일하기 시작했습니다. 낮에는 ThoughtWorks 사에서 컨설팅 업무를 맡아 고객이 지속적인 통합의 원칙과 도구를 활용하여, 테스트 솔루션을 구조화하고 빌드하며 배포하는 일을 도왔습니다.

CruiseControl 메일링 리스트에서의 활동은 2003년에야 시작됐습니다. 수천 가지의 서로 다른 지속적인 통합 시나리오에 대한 글을 읽을 기회가 있었습니다. 소프트웨어 개발자들이 겪는 문제는 다양하고 복잡했습니다. 개발자들이 이러한 작업에 달려드는 이유가 점점 명확해지고 분명해졌습니다. (빠른 피드백, 빠른 배포, 그리고 반복 가능한 자동화된 테스트 같은) 지속적인 통합의 장점은 그 복잡함을 훨씬 웃돌았습니다. 하지만 이런 종류의 환경을 만들 땐 목표를 잘못 겨냥하기 쉽습니다. 그리고 제가 처음 CruiseControl을 출시할 때는 사람들이 자신의 소프트웨어 개발 프로세스를 개선하기 위해, 지속적인 통합을 이토록 흥미로운 방식으로 쓸 줄은 몰랐습니다.

2000년, 저는 대규모 J2EE 애플리케이션 개발 프로젝트에 참가했는데 명세가 제공하는 모든 기능을 사용했습니다. 애플리케이션은 그 자체로 훌륭했지만, 빌드하기가 힘들었습니다. 이때 빌드라 함은 컴파일하고, 테스트하고, 자바 아카이브를 생성하며 기능 테스트를 수행하는 과정을 말합니다. Ant는 아직 초기 단계였고 자바 애플리케이션을 위한 사실상의 표준이 되기 전이었습니다. 우리는 섬세하게 조정한 일련의 셸 스크립트를 이용해 모든 걸 컴파일하고 단위 테스트를 돌렸습니다. 또 다른 일련의 셸 스크립트로는 모든 걸 묶어서 배포 가능한 아카이브로 묶어냈습니다. 결국, JAR 파일을 배포하고 기능 테스트 수트를 실행시키려면 하나하나 직접 해나

갈 수밖에 없었습니다. 말할 것도 없이, 이러한 과정은 인내를 요하고 지루하기 때문에 실수가 따르기 마련이었습니다.

그래서 '버튼 하나'만 누르면 재현 가능한 '빌드'를 만들어주는 뭔가를 만들려는 저의 모험이 시작됐습니다(그 당시에 마틴 파울러의 이야깃거리 중 하나였습니다). Ant가 교차 플랫폼 빌드 스크립트를 만드는 문제는 해결해줬습니다. 남은 부분은 배포, 기능 테스트, 결과 보고하기 같은 지루한 단계를 다뤄 줄 무엇이었습니다. 당시에 저는 기존 솔루션을 조사했지만 쓸 만한 게 없었습니다. 그 프로젝트에 적용하고픈 방식대로 모든 게 작동하는 걸 찾을 수 없었습니다. 그 애플리케이션은 성공적으로 개발을 마치고 실제 환경으로 넘겨졌지만, 저는 그 일들을 좀더 잘 할 수 있었다는 걸 알았습니다.

그 프로젝트가 끝나고 다음 프로젝트가 시작되기 전, 바로 그 사이에 저는 해답을 찾았습니다. 마틴 파울러와 매트 포멜Matt Foemmel이 지속적인 통합에 관한 세미나 기사를 때마침 발표했던 겁니다. 운 좋게도, 저는 파울러와 포멜이 제시한 시스템을 재사용할 수 있는 솔루션으로 만들어내는 일을 하는 ThoughtWorks 사의 직원 몇 명과 함께 일하게 됐습니다. 과장하지 않더라도 저는 무척 흥분했습니다! 지난 프로젝트 때부터 해온 기도에 대한 대답이란 걸 알았던 거지요. 몇 주 만에 우리는 모든 준비를 마치고 우리가 만든 새로운 시스템을 여러 프로젝트에 사용하기 시작했습니다. CruiseControl의 베타 테스트 사이트를 방문해서 베타 버전을 대규모 프로젝트에 설치해 보기도 했습니다. 그 직후 우리는 이 시스템을 오픈소스로 공개했습니다. 저의 경우엔, 이를 후회한 적이 한 번도 없었습니다.

ThoughtWorks 사의 컨설턴트로서, 저는 몇몇 매우 복잡한 엔터프라이즈 배포 아키텍처 작업에 참여했습니다. 고객들은 광고가 약속한 이점을 대충 이해한 나머지 문제를 빨리 고칠 방법을 찾을 때가 많습니다. 어느 기술이라도, 기술이 기업을 쉽게 변화시킬 수 있다는 잘못된 믿음이 조금씩은 있습니다. 만약 컨설팅을 해온 수년간의 경험으로 배운 게 있다면, 보기만큼 쉬운 건 없다는 겁니다.

저는 고객들에게 지속적인 통합의 원칙들을 실용적으로 적용하는 법에 대해 즐겨 말합니다. 지속적인 통합의 이점을 진정으로 활용하려면 개발 '리듬'을 바꾸는 게 중요하다고 강조하길 좋아합니다. 만약 개발자들이 한 달에 한 번만 체크인을 하고, 자동화된 테스트에 주의를 기울이지 않거나, 깨진 빌드를 고쳐야 한다는 아무런 사회적 책무가 없다면, 지속적인 통합의 혜택을 전부 거두기 위해 반드시 해결해야 할 큰 문제가 있는 겁니다.

그렇다고 이러한 실천 방법들이 정착되기 전까진 IT 관리자들이 지속적인 통합에 대해 잊어야 한다는 뜻일까요? 아닙니다. 사실, CI(지속적인 통합) 실천 방법들을 사용하는 것이야말로 변화를 향한 가장 빠른 동기부여 방법 중 하나가 될 수 있습니다. 저는 CruiseControl 같은 지속적인 통합 도구를 설치하면 소프트웨어 팀이 고무되어, 적극적인 정도를 넘어 미래를 내다보고 반응하게끔 된다는 걸 발견했습니다. 변화는 하룻밤 사이에 일어나지는 않습니다. 여러분은 자신의 기대를 적절하게 설정해야 합니다. 끈기를 갖고 밑에 깔린 원칙들을 잘 이해한다면, 가장 복잡한 환경일지라도 이해하기가 더 쉬워지고, 테스트하기가 더 쉬워지며, 빨리 생산하기도 더 쉬워질 수 있습니다.

저자들은 이 책으로 프로젝트 개발 현장을 다듬어 왔습니다. 저는 이 책이 포괄적이면서 폭넓다는 걸 알게 됐습니다. 지속적인 통합의 가장 중요한 측면을 깊이 있게 다루었기 때문에 독자들이 제대로 알고 결정을 내릴 수 있게 해줄 겁니다. 폭넓은 범위의 주제는 오늘날 지속적인 통합의 광경을 지배하는 접근 방법들을 광범위하게 다루며, 독자들이 결정해야 할 교환조건(tradeoff)을 저울질하는 데 도움이 될 겁니다. 마지막으로 지속적인 통합 커뮤니티에서 여러 사람이 달성하려 그토록 노력했던 작업이 더 나은 혁신을 위한 기초로써 형태를 갖추게 되어 매우 기쁩니다. 이러한 이유로 저는 이 책을 지속적인 통합의 마술을 이용하여 대규모 애플리케이션의 복잡함을 이해하는 데 있어 꼭 필요한 책으로 추천합니다.

서 문

경력 초창기에 저는 한 잡지에서 엔터 키처럼 생기고 '통합하기Integrate'라 적힌 키보드 키가 한 쪽 가득 그려진 광고를 보았습니다(그림 P-1을 보세요). 그 그림 아래에는 "이렇게 쉽게 일이 풀려준다면,"이란 문구가 있었습니다. 누구를 위한 광고인지, 무엇을 위한 광고인지 확실하진 않지만, 마음에 와 닿았습니다. 소프트웨어 개발에 있어서 그런 일은 실현 가능성이 없다고 생각했습니다. 왜냐하면 제가 참여한 프로젝트의 경우, 대부분의 프로젝트 일정표 끝 무렵에서 무수히 많은 소프트웨어 컴포넌트를 꾸역꾸역 기워 맞추느라 며칠씩 '통합 지옥'을 헤매곤 했습니다. 하지만 전 그 개념이 좋았고, 그래서 광고를 잘라서 벽에 붙여놓았습니다. 제게 있어 그 광고는 능력 있는 소프트웨어 개발자가 되고자 하면 달성해야 할 중요한 목표 중 하나(반복가능하고 오류가 발생하기 쉬운 프로세스를 자동화하는)를 나타냈습니다. 게다가 그 광고는 소프트웨어 통합을 '별 것 아닌 일'(nonevent, 마틴 파울러가 이렇게 불렀듯이)로 만들고자 하는 저의 믿음을 구체화시켰습니다 – 물론 그렇게 되었죠. 지속적인 통합(CI)은 통합을 별 것 아닌 일로 만드는 데 도움을 줄 수 있습니다.

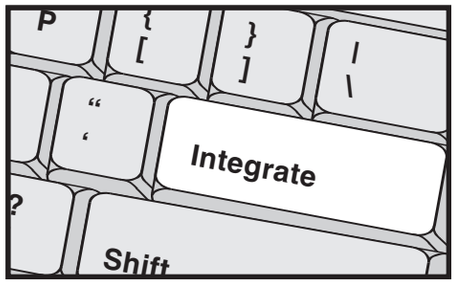


그림 P-1 통합하기!

무엇을 위한 책인가?

소프트웨어 프로젝트에서의 보다 전형적인 개발 프로세스를 한번 생각해봅시다. 코드를 컴파일하고, 데이터베이스를 통해 코드를 정의하고 조작합니다. 테스트를 하고, 코드를 검토하고 최종적으론 소프트웨어를 배포합니다. 이와 더불어, 소프트웨어 상태와 관련해서 여러 팀이 서로 의사소통 해야 할 필요성도 분명히 있을 겁니다. 버튼 하나만 눌러서 이런 모든 프로세스를 처리할 수 있다면 어떨지 상상해보세요.

이 책은 소프트웨어 개발 프로세스 상당수를 자동화하는 가상의 '통합' 버튼을 어떻게 만드는지 보여줍니다. 이뿐만 아니라, 우리는 이 '통합' 버튼을 지속적으로 눌러서, 배포 가능한 애플리케이션을 만드는 데 방해되는 뒤늦은 결함 발견과 저품질의 코드와 같은 위험을 줄이는 방법을 보여줄 겁니다. CI 시스템을 만드는 과정에서, 이 같은 개발 프로세스 중 상당 부분이 자동화되며, 개발 중인 소프트웨어가 변경될 때마다 실행됩니다.

지속적인 통합은 무엇일까요?

소프트웨어를 통합하는 과정은 새삼스런 문제가 아닙니다. 소프트웨어 통합은 외부 시스템 의존성이 낮은 개인 프로젝트라면 그리 문제되지 않을지 모르지만, 프로젝트의 복잡도가 증가하게 되면(단 한 명만 더 투입한다고 해도), 소프트웨어 컴포넌트를 (초기에, 그리고 자주) 통합하고 이렇게 통합한 컴포넌트가 잘 작동하는지 확인해야 합니다. 프로젝트 막판까지 통합을 미루면 온갖 종류의 소프트웨어 품질 문제를 겪게 되는데, 이는 비용면에서 소모적일 뿐 아니라 프로젝트 지연으로 이어지기도 합니다. 지속적인 통합은 이러한 위험을 보다 빨리, 그리고 점진적으로 해소해 줍니다.

마틴 파울러는 유명한 기사 "Continous Integration"[1]에서 지속적인 통합을 이렇게 묘사했습니다.

지속적인 통합은 팀 구성원들이 자신이 한 일을 자주 통합하는 소프트웨어 개발 실천 방법인데, 보통 각자 하루에 적어도 한 번 꼴로 통합합니다-결국 하루에 여러 차례 통합하게 됩니다. 각 통합은 자동화된 빌드(테스트를 포함해서)가 검증하여 최대한 빨리 통합 오류를 탐지해냅니다. 이런 접근 방법을 사용하면 통합 문제를 상당히 줄일 수 있고 응집력 있는 소프트웨어를 개발할 수 있다는 사실을 많은 팀이 알게 됐습니다.

저의 경험에 따르면, 이는 다음과 같은 의미가 있습니다.

- 개발자 모두가 버전 관리 저장소에 코드를 커밋해 넣기 전에 자신의 워크스테이션에서 개인 빌드private build[2]를 돌려서 자기가 변경한 내역이 통합 빌드를 깨먹지 않는지 확인합니다.
- 개발자는 적어도 하루에 한 번 버전 관리 저장소에 코드를 커밋해 넣습니다.
- 통합 빌드가 빌드 컴퓨터 여러 대에서 하루에 예닐곱 번 정도 수행됩니다.
- 매 빌드마다 테스트를 100% 통과해야 합니다.
- 기능적인 면을 테스트할 수 있는 제품(예, WAR, 어셈블리, 실행 파일 등)을 생성해냅니다.
- 깨진 빌드를 고치는 일이 가장 우선순위가 높습니다.
- 개발자 몇몇은 빌드 과정에서 생성된 코딩 표준 및 의존성 분석 보고서 같은 보고서를 읽고 개선할 여지가 없는지 검토합니다.

계속 반복되고 오류가 발생하기 쉬운 프로세스를 자동화하면 여러 혜택을 볼 수 있기 때문에 이 책은 지속적인 통합의 자동화된 측면을 논의합니다. 하지만 마틴 파울러가 밝혔듯이, 지속적인 통합은 작업을 자주 통합하는 과정을 뜻하며, 자동화된 프로세스만이 지속적인 통합의 자격이 있는 건 아닙니다. 지속적인 통합을 자동화된 프로세스로써 지원하는 훌륭한 도구가 많이 있으니, 지속적인 통합 서버를 사용해서 CI 실천 방법을 자동화하는 게 효과적인 접근 방법이라고 우리는 확신합니다. 그럼에도 불구하고, (자동화된 빌드를 활용해서) 사람이 직접 통합을 수행하는 접근 방법이 여러분의 팀에 잘 맞을 수도 있습니다.

1) www.martinfowler.com/articles/continuousIntegration.html를 참고하세요. 황상철씨의 번역글: http://moai.tistory.com/224.

2) 개인 (시스템) 빌드와 통합 빌드 패턴은 Stephen P. Berczuk 와 Brad Appleton가 쓴 『Software Configuration Management Patterns』에서 다룹니다.

☆ 신속한 피드백

지속적인 통합을 사용하면 피드백을 받을 기회가 늘어납니다. 지속적인 통합을 수행하면, 프로젝트의 상태에 대해 하루에도 여러 차례 배우게 됩니다. 지속적인 통합은 결함이 발생하는 시점과 고쳐지는 시점의 간격을 줄이고, 그럼으로써 전체적인 소프트웨어 품질을 향상시키는 용도로 써도 됩니다.

개발 팀은 지속적인 통합 시스템이 자동화되었다고 해서 통합 문제로부터 안전할 거라 믿어선 안 됩니다. 개발 팀이 자동화된 도구를 소스 코드 컴파일 용도로만 사용할지라도-어떤 이는 이를 두고 '빌드'라고 부르지만 사실은 그렇지 않습니다(1장을 참고하세요)-이는 마찬가지입니다. CI를 효과적으로 실천하려면 도구 이상의 것이 필요합니다.이 책에서 윤곽을 그릴 버전 관리 저장소로의 잦은 커밋, 깨진 빌드 즉시 고치기, 그리고 별도의 통합 빌드 머신 사용하기와 같은 실천 방법이 이에 해당합니다.

CI 실천 방법은 피드백을 보다 빨리 받게 해줍니다. CI 실천 방법을 효과적으로 활용하면, 여러분은 개발 중인 소프트웨어의 전체적인 건강 상태를 하루에도 예닐곱 번은 알게 될 겁니다. 더욱이, 지속적인 통합은 리팩토링과 테스트 주도 개발과 같은 실천 방법과도 잘 어울립니다. 왜냐하면 이런 실천 방법이 작은 변경을 가한다는 개념에 중심을 두기 때문입니다. 지속적인 통합은 본질적으로, 변경 내역이 소프트웨어의 나머지 부분과 제대로 작동한다는 걸 확인하는 안전망입니다. 좀더 높은 수준에서 보자면, 지속적인 통합은 팀 공동의 자신감을 증가시키고 프로젝트에 필요한 인적 활동의 양을 줄여줍니다. 지속적인 통합은 소프트웨어가 변경될 때마다 자동으로 작동하는 프로세스이기 때문입니다.

☆ '지속적인'이란

우리가 이 책에 '지속적인'이란 용어를 사용하긴 하지만, 그 사용법은 기술적으로 봤을 때 정확하지 않습니다. '지속적인'이란 말에는 한번 시작하면 절대 멈추지 않는다, 라는 의미가 내포되어 있습니다. 이는 프로세스가 끊임없이 통합한다는 걸 암시하지만, 가장 격렬한 CI 환경에서조차도 그렇지는 않습니다. 그러니, 우리가 이 책에서 묘사하는 것은 '빈번한 통합'에 보다 가깝습니다.

누가 이 책을 읽어야 할까요?

우리의 경험에 따르면, 소프트웨어 개발을 업무(job)로 취급하는 사람과 전문적인 직업(profession)으로 취급하는 사람 사이에는 뚜렷한 차이가 있습니다. 이 책은 직업인으로서 일하면서 자신들이 프로젝트에서 똑같은 과정을 되풀이해 수행하고 있다고 느끼는 사람을 위한 책입니다(반복되는 과정을 수행하는 줄 모르고 있다면, 우리가 여러분을 도와 똑같은 일을 얼마나 자주 하는지 깨닫게 해드리겠습니다). 우리는 지속적인 통합을 실천하는 방법과 그 장점을 설명하고 이런 실천 방법을 적용하기 위한 지식을 전달하여, 여러분의 시간과 전문 지식을 보다 중요하고 도전적인 이슈로 돌릴 수 있도록 하겠습니다.

이 책은 지속적인 통합과 관련된 주요 주제를 다루는데, 지속적인 피드백, 테스트, 배포, 검사, 그리고 데이터베이스 통합을 이용해 지속적인 통합을 구현하는 방법을 포함합니다. 소프트웨어 개발에서 어떤 역할을 하든지, 지속적인 통합을 소프트웨어 개발 프로세스에 도입할 수 있습니다. 만약 여러분이 점차 유능해지고자 하는 소프트웨어 전문가라면, 더 많은 일을 점점 더 믿음직스럽게 해나가고자 하는 사람이라면, 이 책에서 많은 걸 가져갈 수 있을 겁니다.

개발자

사용자를 위한 소프트웨어를 개발하는 데 집중하지 못하고 소프트웨어 통합 문제 때문에 씨름하고 있다면, 이 책이 여러분이 겪을 거라고 생각하는 '통합의 난항'에서 어려움 없이 빠져나올 수 있도록 도울 것입니다. 이 책은 여러분에게 통합하는 데 시간을 더 쓰라고 요구하진 않습니다. 이 책이 바로 소프트웨어 통합 문제를 별 것 아닌 일로 만들어주기 때문입니다. 따라서 여러분이 가장 좋아하는 일, 바로 소프트웨어를 개발하는 일에 집중할 수 있게 해줍니다. 이 책에서 소개하는 실천 방법이나 예제 상당수는 효과적인 CI 시스템을 어떻게 구현하는지 실지로 보여줍니다.

빌드/형상/릴리즈 관리

여러분이 하는 일이 소프트웨어를 세상 밖으로 내놓는 것이라면, 버전 관리 저장소에 변경 내역이 적용될 때마다 프로세스들을 돌림으로써, 응집력 있고 잘 작동하는 소프트웨어를 생성해낼 수 있다는 걸 우리가 보여주면 이 책이 아주 흥미롭다고 생

각할 겁니다. 여러 사람이 프로젝트 내에서 개발 같은 다른 역할을 이행하면서 빌드를 관리하고 있습니다. CI는 여러분을 위해 어느 정도 '생각'을 대신해주고, 개발 생명주기의 끝까지 기다리지 않아도, 배포 가능하고 테스트 가능한 소프트웨어를 하루에도 여러 번 만들어냅니다.

테스터

지속적인 통합은 '버그 수정'이 적용된 후에도 결함이 다시 발생하는 전통적인 고통을 거의 제거함으로써 소프트웨어 개발에 신속한 피드백 접근 방법을 제공합니다. CI를 사용하면 테스터는 좀더 만족감을 얻게 되고 프로젝트에서 자신이 맡은 역할에 더욱 관심을 갖게 되기 마련입니다. 테스트할 소프트웨어가 더 자주 준비될 뿐만 아니라 그 범위도 조금씩 커지기 때문입니다. 개발 생명주기에 CI 시스템을 갖춰놓는다면, 뒤늦게 테스트를 하거나 아예 테스트를 하지 않는 전형적인 '모 아니면 도' 시나리오를 따르지 않고, 줄곧 테스트를 해나갈 수 있습니다.

관리자

만약 팀이 작동하는 소프트웨어를 견실히 그리고 되풀이하여 전달할 수 있다는 자신감을 키우고 싶다면, 이 책이 큰 영향을 미칠 것입니다. 여러분은 시간, 비용, 그리고 품질을 훨씬 더 효과적으로 관리할 수 있게 될 텐데, 이는 단순히 프로젝트 일정상의 작업 항목에만 의존하지 않고, 작동하는 소프트웨어와 더불어 실제적인 피드백과 메트릭에 기반을 두고 의사결정을 하게 될 것이기 때문입니다.

이 책의 구성

이 책은 두 부분으로 나뉩니다. 1부는 CI를 소개하고 그 개념과 실천 방법을 밑바닥부터 점검합니다. 1부는 CI의 핵심 실천 방법에 친숙하지 않은 독자에게 맞췄습니다. 하지만 테스트, 검사, 배포, 그리고 피드백과 같은 CI의 핵심 개념을 CI 시스템이 수행하는 다른 효과적인 프로세스로 자연스럽게 확장해주는 2부가 없다면, CI의 실천 방법이 완전하다고 생각하지는 않습니다

1부: 지속적인 통합의 배경–원칙과 실천 방법

1장 '시작하기'는 지속적인 통합을 활용해서 소프트웨어를 지속적으로 빌드하는 고수준의 사례를 제시함으로써 곧바로 적용할 수 있게끔 해줍니다.

2장 '지속적인 통합 도입하기'는 공통적인 실천 방법과 우리가 CI를 도입했던 방식에 친숙해지게 해줍니다.

3장 '지속적인 통합을 사용해서 위험 줄이기'는 시나리오에 기반한 사례를 사용하여 CI가 완화시켜 줄 수 있는 핵심 위험 요소를 알아봅니다.

4장 '코드가 변경될 때마다 소프트웨어를 빌드하기'는 자동화된 빌드를 활용하여 변경 내역마다 소프트웨어를 통합하는 실천 방법을 탐구합니다.

2부: 완전한 기능을 갖춘 지속적인 통합 시스템 만들기

5장 '지속적인 데이터베이스 통합'은 데이터베이스를 다시 빌드하고 테스트 데이터를 모든 통합 빌드의 일부로써 적용하는 프로세스와 관련된 보다 발전한 개념으로 이어갑니다.

6장 '지속적인 테스트'는 통합 빌드를 할 때마다 소프트웨어를 테스트하는 개념과 전략을 다룹니다.

7장 '지속적인 검사'는 다른 도구와 기법을 사용하여 자동화되고 지속적인 검사(정적 및 동적 분석)를 하는 방법을 소개합니다.

8장 '지속적인 배포'는 지속적인 통합 시스템을 사용하여 소프트웨어를 배포하여, 소프트웨어를 기능적으로 테스트할 수 있게 하는 프로세스를 탐구합니다.

9장 '지속적인 피드백'은 지속적인 피드백 장치(이메일, RSS, X10 그리고 앰비언트 오브와 같은)를 사용하여, 빌드가 끝나면 성공 또는 실패 여부를 통지 받을 수 있는 방법을 설명하고 보여줍니다.

'후기'는 지속적인 통합의 향후 가능성을 탐구합니다.

부록

부록 A '지속적인 통합 관련 리소스'엔 지속적인 통합과 관련된 웹 주소, 도구, 그리고 문서가 있습니다.

부록 B '지속적인 통합 도구 평가하기'는 시장에 나와 있는 다양한 지속적인 통합 서버와 관련된 도구를 평가하고, 책에서 설명한 실천 방법에 적용할 수 있는지 여부

를 논의하고, 각각의 장점과 단점을 알아보며, 보다 흥미로운 특징 중 일부를 사용하는 법을 설명합니다.

그 밖의 특징

이 책은 본문에서 설명한 내용을 여러분이 더 잘 배우고 적용할 수 있게 도와주는 특징을 담고 있습니다.

- 실천 방법 – 이 책에는 40가지가 넘는 CI와 관련된 실천 방법을 다룹니다. 여러 장(Chapter)의 작은 표제는 실천 방법을 담고 있습니다. 대부분의 장 도입부에 제시된 그림은 어떤 실천 방법을 다루는지 보여주고, 흥미를 끄는 영역을 찾을 수 있게 해줍니다. 예를 들어, "전용 통합 빌드 머신을 사용하라"와 "코드를 자주 커밋하라"는 이 책에서 논의되는 실천 방법의 사례라 하겠습니다.
- 예제 – 다양한 언어와 플랫폼으로 구성된 다양한 예제를 사용함으로써 이런 실천 방법을 어떻게 적용하면 되는지 보여줍니다.
- 질문 – 각 장은 프로젝트의 CI 실천 방법을 응용할 때 도움이 될 만한 질문 목록을 제시하고 끝납니다.
- 웹 사이트 – 이 책의 자매 웹 사이트 www.integratebutton.com에는 책의 최신 정보, 코드 예제, 그밖의 자료가 있습니다.

무엇을 배우게 되는가

이 책을 읽음으로써, 응집력 있고 잘 작동하는 소프트웨어를 하루에도 여러 번 만들어낼 수 있는 개념과 실천 방법을 배우게 될 겁니다. 우리는 실천 방법을 먼저 제시하고, 이런 실천 방법을 응용하는 방법이 뒤따르도록 하고, 가능하면 실제로 사용할 수 있는 예제를 포함하려고 집중했습니다. 예제는 자바, 마이크로소프트 닷넷, 그리고 심지어 루비와 같은 서로 다른 개발 플랫폼을 사용합니다. CruiseControl(자바와 닷넷 버전)은 줄곧 주요 CI 서버로 사용됩니다. 하지만 자매 웹사이트(www.integratebutton.com)와 부록 B에서 소개한 다른 서버와 도구를 사용해서도 비슷한 예제를 만들어봤습니다.

이 책을 따라가다 보면, 다음과 같은 통찰을 얻게 됩니다.

- CI를 구현하면 어떤 식으로 개발 생명주기의 매 단계마다 배포 가능한 소프트웨어를 생산해내게 되는지.
- CI가 어떤 식으로 결함이 발생하는 시점과 결함이 탐지되는 시점의 간격을 줄이고, 그럼으로써 결함을 고치는 비용을 줄이는지.
- 어떻게 하면 개발 주기 막판까지 기다리지 않고 소프트웨어를 자주 빌드함으로써 소프트웨어에 품질을 구축해 넣을 수 있는지.

이 책이 다루지 않는 것

이 책은 지속적인 통합 시스템을 이루는 모든 도구-빌드 스케줄링, 프로그래밍 환경, 버전 관리 등-를 다루진 않습니다. 효율적인 CI 시스템을 개발하기 위한 CI 실천 방법을 구현하는 데 초점을 둡니다. CI 실천 방법을 우선 논의합니다. 만약 예시된 특정 도구를 더 이상 사용하지 않는다든가 그 도구가 특정 요구에 부합하지 않는다면, 다른 도구를 통해 실천 방법을 적용하여 똑같은 효과를 달성하면 됩니다.

이뿐만 아니라 CI 시스템에 사용되는 모든 종류의 테스트, 피드백 메커니즘, 자동화된 검사기, 그리고 배포 유형을 다루기란 가능하지 않고 실용적이지도 않습니다. 이 책에서 내내 언급하듯, 자매 웹 사이트인 www.integratebutton.com 에는 책에서 다루지 않는 다른 도구와 프로그래밍 언어를 사용한 예제가 있습니다.

작가 소개

이 책은 세 명의 공저자와 한 명의 기고자가 썼습니다. 많은 장을 제가 썼고 스티브 마티야스는 4, 5, 7, 8장, 부록 A, 이와 더불어 책에 수록된 몇몇 예제를 작성하는 데 기여했습니다. 앤디 글로버는 6, 7, 8장을 썼으며 예제를 제공하고, 다른 부분에도 기여했습니다. 에릭 타블라는 부록 B를 썼습니다. 그러니 일인칭 대명사를 쓰는 문장이 있어서 누가 쓴 글인지 헷갈릴 때, 이 점을 고려하면 누가 무슨 말을 했는지 명확해질 겁니다.

책 표지에 대해 [3]

우리가 쓴 책이 명성 높은 마틴 파울러 서명 시리즈의 일부가 된다는 걸 알고 저는 무척 흥분했습니다. 그때 저는 책 표지로 다리를 선택해야겠다고 마음 먹었습니다. 공저자들과 저는 워싱턴 D.C에서 자란 희귀종들입니다. 워싱턴 D.C 출신이 아닌 여러분이 이해할 수 있게 말하자면, 그곳은 공허한 지역입니다. 좀더 구체적으로 말하자면, 우리는 노던 버지니아 출신이어서 버지니아 주의 내추럴 브리지(Natural Bridge)를 선택한다면 고향에 알맞은 헌정을 하는 셈이라고 생각했습니다. 저는 책 표지로 그 다리를 선택한 직후인 2007년 초가 되어서야 그 다리에 처음 가봤습니다. 그 다리는 매우 흥미로운 역사를 간직하고 있었고, 매일 같이 자동차가 그 위를 지나다니는 기능적인 다리라는 사실을 알고 경이로움을 느꼈습니다(물론, 제 자신도 차를 몰고 다리 위를 여러 번 오갔습니다). 여러분이 이 책을 읽고 나서 CI를 다음 번 소프트웨어 개발 프로젝트의 자연스런 일부로 받아들여줬으면 합니다.

3) 원서 표지에 대한 설명입니다.

감사의 말

얼마나 많이 감사의 말을 읽었는지, 그리고 그때마다 작가들이 "~가 없었다면 이 일을 결코 해내지 못했을 것이다"라는 식의 말을 써놓았는지 모릅니다. 저는 항상 속으로 이렇게 생각했습니다. "지나치게 겸손한 거겠지." 하지만 제가 제대로 헛짚었습니다. 이 책은 제가 여기서 열거한 고마운 사람들이 엄청나게 도와준 결과물입니다.

출판사인 애디슨 웨슬리에 감사를 전하고 싶습니다. 특히, 책임 편집자인 Chris Guzikowski에게 이 진 빠지는 작업 내내 저와 함께 일해준 것에 대해 고마움을 표현하고 싶습니다. 그의 경험, 통찰, 그리고 격려는 대단했습니다. 그리고 여러 교정 및 편집 주기 내내 탄탄한 충고를 해준 개발 편집자인 Chris Zahn에게도 감사를 전합니다. K Karen Gettman, Michelle Housley, Jessica D'Amico, Julie Nahil, Rebecca Greenberg 그리고 마지막으로 누구보다 첫 번째 책임 편집자였던 Mary O'Brien에게 감사를 전하고 싶습니다.

Rich Mills는 책을 위해 CVS 서버를 호스팅해줬으며 브레인스토밍 세션 동안에 멋진 아이디어를 제안해줬습니다. 2002년에 직업적 글쓰기로 나를 이끌어주고 글쓰는 과정 내내 유난히 상세한 검토를 해준 저의 멘터이자 친구인 Rob Daly에게 감사를 전하고 싶습니다. John Steven은 이 책을 쓰는 과정을 시작할 때 도움을 줬습니다.

공동 저자, 편집자, 공헌 저자들에게도 사의를 표하고 싶습니다. 스티브 마티야스와 저는 여러분이 오늘 읽고 있는 이 책을 만들어내느라 여러 밤을 잠 못 이루고 견뎌냈습니다. 앤디 글로버는 자신이 프로젝트에서 쌓아온 상당한 개발자 테스트 경험을 제공해준 대단한 글쓴이였습니다.

공헌 편집자인 리사 포터는 지치지 않고 주요 교정판을 모두 자세히 검토하고 책의 품질을 높이는 데 도움이 된 교정과 충고를 제공해줬습니다. 지속적인 통합 도구 부록을 써준 에릭 타블라와 부록 B에서 Maven2에 대한 자신의 경험을 제공해준 Levent Gurses 에게도 감사를 전합니다.

이 프로젝트 내내 훌륭한 피드백을 제공해준 개인적인 기술 검토자들이 많았습니다. Tom Copeland, Rob Daly, Sally Duvall, Casper Hornstrup, Joe Hunt, Erin Jackson, Joe Konior, Rich Mills, Leslie Power, David Sisk, Carl Tallis, Eric Tavela, Dan Taylor, 그리고 Sajit Vasudevan 등이 그들이었습니다.

Charles Murray와 Cristalle Belonia, 그리고 Unbancode의 Maciej Zawadzki와 Eric Minick에게도 도와줘서 고맙다고 전하고 싶습니다.

Stelligent사에서 매일 나에게 영감을 준 Burke Cox, Mandy Owens, David Wood, 그리고 Ron Wright 을 비롯한 여러 훌륭한 사람들의 도움에도 사의를 표합니다. Rich Campbell, David Fado, Mike Fraser, Brent Gendleman, Jon Hughes, Jeff Hwang, Sherry Hwang, Sandi Kyle, Brian Lyons, Susan Mason, Brian Messer, Sandy Miller, John Newman, Marcus Owen, Chris Painter, Paulette Rogers, Mark Simonik, Joe Stusnick, 그리고 Mike Trail 등은 여러 해에 걸쳐 나의 일에 영감을 주었습니다.

Scott Ambler, Brad Appleton, Jon Eaves, Martin Fowler, Paul Holser, Paul Julius, Kirk Knoernschild, Mike Melia, Julian Simpson, Andy Trigg, Bas Vodde, Michael Ward, 그리고 Jason Yip으로 구성된 애디슨 웨슬리 기술 검토 팀에서 준 철저한 피드백에도 감사를 전합니다.

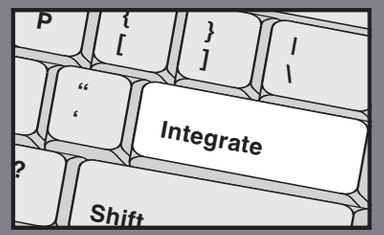
2006년 시카고에서 열린 지속적인 통합과 테스팅 컨퍼런스(CITCON)에 참여한 사람들에게도 지속적인 통합과 테스트에 대한 자신의 경험을 우리 모두에게 나눠준 것에 감사를 전합니다. 컨퍼런스를 조직해준 폴 줄리어스와 제프리 프레데릭과 이 행사에 참여한 모든 사람에게 감사합니다. 마지막으로 이 책을 만드느라 우여곡절을 겪는 내내 함께 있어주고 꾸준히 지원해준 아내 Jenn에게도 감사를 전하고 싶습니다.

폴 M. 듀발
버지니아주 페어팩스에서
2007년 3월

공 헌 자 소 개

리사 포터는 미국 정부에 네트워크 보안 솔루션을 제공하는 컨설팅 회사의 선임 테크니컬 라이터(Technical Writer)입니다. 리사는 이 책을 펴내기에 앞서 기술 편집을 해줬습니다. 초창기에는 여러 개의 애플리케이션을 개발하는 대규모 소프트웨어 개발 프로젝트를 지원했는데, 거기서 그녀는 요구 사항 결정과 프로젝트 성숙도/역량 활동에 대해 많은 걸 배웠습니다. 리사는 외국어 번역과 건축학/공학 산업 분야에 기술적 글쓰기의 원칙들을 적용해왔습니다. 리사는 2002년부터 책과 온라인 출판물을 편집해왔습니다.

에릭 타블라는 5AM Solutions사의 아키텍트 책임자인데, 이 곳은 생명과학연구 커뮤니티를 위해 소프트웨어 엔지니어링의 최고의 실천 방법들을 적용하는 데 초점을 맞춘 소프트웨어 개발 회사입니다. 에릭은 주로 자바/J2EE 애플리케이션을 설계하고 구현하고 객체 지향 소프트웨어 개발과 UML 모델링 분야에서 개발자들을 멘토링해왔습니다.



part 1

지속적 통합의 배경: 원칙과 실천 방법

chapter 1 시작하기

■ 변경할 때마다 소프트웨어 빌드하기

먼저, 기초부터 통달하라

– 래리 버드(Larry Bird, 미국 프로 농구 선수)

Javaranch.com의 설립자인 케이시 시에라Kathy Sierra는 자신의 블로그에서 이렇게 말했습니다. " '매일 사과 하나를 먹어라' 라고 말하는 것과 실제로 사과를 먹는 것 사이엔 커다란 차이가 있다."[1] 소프트웨어 프로젝트에 적용해야 할 기초적인 실천 방법을 따를 때도 마찬가지입니다. "테스트해봐야 효율적이지 않아" 라든가 "코드 검토는 시간 낭비야" 라든가 잦은 소프트웨어 빌드는 따르기엔 좋지 않은 실천 방법이라고 말하는 사람은 드뭅니다. 하지만 겉보기엔 기초적인 것 같은 이런 실천 방법도 설교보다는 실천하는 데 보다 중점을 둬야 합니다. 이런 실천 방법이 프로젝트에 적용되는 빈도가 비참할 정도로 낮기 때문입니다.

통합 빌드를 자주 돌리고, 그렇게 해서 통합 빌드를 대수롭지 않은 일로 만들고 싶다면, 지속적인 통합(Continuous Integration, CI)이 많은 도움이 될 것입니다. 지속적인 통합은 컴파일 및 데이터베이스 재빌드, 자동화된 테스트와 검사, 소프트웨어 배포, 사용자 피드백 받기 등의 과정을 포함합니다. 1장에서는 이 같은 기초적인 소프트웨어 실천 방법을 토대로 하는 CI 시스템에서 공통적으로 볼 수 있는 특징에 대해 알아볼 겁니다.

1) 출처: http://headrush.typepad.com

지속적인 통합의 기초는 이해하기 무척 쉬우므로, 이러한 기초적인 실천방법들을 금방 적용할 수 있을 겁니다.

변경할 때마다 소프트웨어를 빌드하기

책을 읽을 때, 저는 먼저 예제를 보고 뒤이어 '왜 그런지'를 배우고 싶어합니다. 왜냐하면 예제를 보면 글의 문맥에 따라 '왜 그런지' 이해할 수 있기 때문입니다. 이 책에서도 전형적인 구현 방식에 근거를 두고 CI 시나리오를 설명합니다. 독자분들은 CI 시스템을 구축하는 방법이 상당히 다양하다는 것에 놀랄지 모르지만, 이를 통해 전형적인 CI 시스템의 여러 요소를 더 잘 이해할 수 있게 될 것이라고 믿습니다.

☆ 빌드란?

빌드는 컴파일(또는 동적 언어에서와 같은 변형된 형태)보다 훨씬 많은 걸 의미합니다. 빌드에는 컴파일과 테스트, 검사, 배포 등의 과정들이 포함될 수 있습니다. 빌드라 함은 소스 코드를 한 곳에 모아보고, 소프트웨어가 응집력 있는 하나의 단위로써 작동하는지 확인하는 과정이라 하겠습니다.

CI 시나리오는 개발자가 소스 코드 저장소에 코드를 커밋할 때 시작됩니다. 보통의 프로젝트라면, 프로젝트에서 제각기 다른 역할을 맡은 사람들이 변경 사항을 커밋할 테고, 그때 CI 주기가 시작됩니다. 이는 개발자가 소스 코드를 변경하고, 데이터베이스 관리자(DBA)가 테이블 정의를 고치고, 빌드 팀이나 배포 팀에서 설정 파일을 변경하고, 인터페이스 팀이 DTD/XSD 명세를 변경하는 것 등을 모두 포함합니다.

☆ 최신 예제 보기

책에 '실제' 예제를 써넣으면 위험이 따르는데, 금세 시대에 뒤진 예제가 될 수 있기 때문입니다. 지속적인 통합과 같이 동적인 주제라면 특히 그렇습니다. 이 책이 출판된 후 발생할지 모르는 차이를 줄이려고, 우리는 자매 웹 사이트인 www.integratebutton.com에 CruiseControl과 Ant 뿐만 아니라 다른 CI 서버와 도구에 관한 예제도 갱신해 넣을 것입니다.

지속적인 통합 시나리오라면 일반적인 각 단계는 다음과 같습니다.

1. 개발자가 버전 관리 저장소에 코드를 커밋해 넣습니다. 그 사이, 통합 빌드 컴퓨터에 설치된 CI 서버가 변경 사항이 없는지 이 저장소를 계속 확인합니다. 이를 테면, 몇 분마다 확인(폴링: polling)할 수 있습니다.
2. 커밋이 되면 곧이어, CI 서버가 버전 관리 저장소에서 변경 내역을 감지해냅니다. 그리고 CI 서버는 저장소에서 최신 소스 코드 복사본을 가져와서 빌드 스크립트를 실행하고 소프트웨어를 통합합니다.
3. CI 서버는 지정된 프로젝트 구성원들에게 이메일로 빌드 결과를 보냄으로써 피드백을 줍니다.
4. CI 서버는 버전 관리 저장소에 변경 사항이 들어왔는지 계속해서 폴링합니다.

그림 1-1은 CI 시스템의 이런 측면을 묘사합니다.

그림 1-1에 나와있는 도구와 참가자들에 대해서는 이어지는 글에서 자세히 설명합니다.

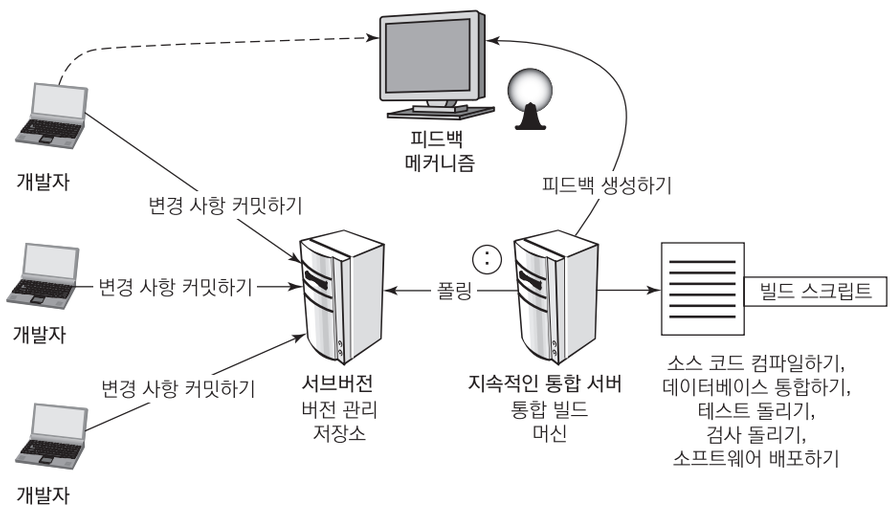


그림 1-1 | CI 시스템의 컴포넌트

개발자

작업 내용과 관련된 수정 작업이 끝나면, 개발자는 개인 빌드(private build, 개인 빌드는 팀의 다른 개발자들이 작업한 최신 내용을 작업 PC에 다운받아 컴파일하는 것을 의미합니다)를 수행하고, 그러고 나서 변경 내역을 버전 관리 저장소에 커밋합니다. 이 과정은 언제라도 발생할 수 있으며, CI 프로세스의 이후 과정엔 영향을 주지 않습니다. 통합 빌드는 변경사항이 버전 관리 저장소에 적용될 때 비로소 시작됩니다.

목록 1-1은 명령줄에서 Ant 빌드 스크립트를 호출하여 개인 빌드를 실행시키는 예제입니다. 이 스크립트가 서브버전 버전 관리 저장소에서 최신 업데이트를 가져오는 걸 눈여겨 보시기 바랍니다.

★ 자주 빌드하여 문제를 보다 이른 시기에 찾아내세요

일단 빌드가 자동화되어 있고, 명령어 하나로 빌드를 실행시킬 수 있게 됐다면, CI를 수행할 준비가 되었다고 할 수 있습니다. 프로젝트의 버전 관리 시스템에 변경 사항이 커밋될 때마다 자동화된 빌드를 돌리면, 다음과 같은 질문에 팀이 대답할 수 있게 됩니다.

- 모든 소프트웨어 컴포넌트가 함께 작동합니까?
- 코드 복잡도는 얼마나 됩니까?
- 팀이 정해진 코딩 표준을 잘 지킵니까?
- 자동화된 테스트가 적용되는 코드가 얼마나 됩니까?
- 가장 최근의 변경 사항을 반영한 후에 모든 테스트가 성공했습니까?
- 애플리케이션이 성능 요구사항을 여전히 충족시킵니까?
- 마지막 배포에 아무런 문제가 없었습니까?

마지막 변경 사항이 적용된 후에 소프트웨어가 성공적으로 '빌드됐다'는 사실을 아는 것도 도움이 되지만, 소프트웨어가 올바르게 빌드됐다는 걸 안다면 더할 나위 없을 겁니다. 소프트웨어 결함은 의심할 여지 없이 어느 시점엔가 코드 베이스에 슬그머니 들어올 것이기 때문이죠. 빌드를 지속적으로 하려는 것도 피드백을 신속히 받으면 개발 생명주기 내내 문제를 발견하고 해결할 수 있기 때문입니다.

목 록 1-1 | Ant를 사용해서 개인 빌드 실행시키기

```
> ant integrate
Buildfile: build.xml
clean:
svn-update:
all:
compile-src:
compile-tests:
integrate-database:
run-tests:
run-inspections:
package:
deploy:
BUILD SUCCESSFUL
Total time: 3 minutes 13 seconds
```

개인 빌드가 성공하면, 이제 새로운 파일이나 변경된 파일을 저장소에 체크인할 수 있습니다. 목록 1-2가 보여주듯, 대부분의 버전 관리 시스템은 이런 과정을 위한 간단한 명령어를 제공합니다.

목 록 1-2 | 변경 사항을 서브버전 저장소에 커밋해 넣기

```
> svn commit –m "Added CRUD capabilities to DAO"
Sending src\BeerDaoImpl.java
Transmitting file data .

Committed revision 52.
```

통합 개발 환경(IDE)을 쓰더라도 빌드 스크립트를 실행시키고 저장소에 변경 사항을 커밋해 넣을 수 있습니다. 단지 명령줄에서도 두 가지 일(빌드 스크립트를 실행시키고 저장소에 변경 사항을 커밋해 넣는)을 수행할 수 있게끔 해둬야 합니다. 그래야 IDE나 버전 관리 시스템에 지나치게 의존하지 않는다는 걸 확신할 수 있습니다.

버전 관리 저장소

단언하건대, 지속적인 통합을 수행하려면 버전 관리 저장소를 반드시 사용해야 합니다. 사실, CI를 쓰지 않는다 해도, 버전 관리 저장소는 프로젝트의 표준이 되어야 합니다. 버전 관리 저장소의 목적은 접근 제어형 저장소를 사용하여 소스 코드와 다른 소프트웨어 자산에 가해지는 변경 사항을 관리하는 것입니다. 버전 관리 저장소는 '단일 출처'를 제공하므로 한 곳에서 모든 소스 코드를 가져올 수 있습니다. 버전

관리 저장소는 시간을 거슬러 올라가 다른 버전의 소스 코드와 기타 파일을 가져올 수 있게 해줍니다.

버전 관리 저장소의 주흐름mainline(예를 들어, CVS와 서브버전 같은 시스템에서 말하는 헤드와 트렁크Head/Trunk에 대해 지속적인 통합을 돌리게 됩니다. 세상에는 다양한 버전 관리 시스템이 있습니다. 이 책에 쓰인 대부분의 예제에서 서브버전을 사용하는데, 그 기능 집합이 뛰어날 뿐만 아니라 무료로 쓸 수 있기 때문입니다. 다른 소프트웨어 형상 관리(Software Configuration Management, SCM)/버전 관리 도구에는 CVS, Perforce, PVCS, ClearCase, MKS, Visual SourceSafe가 있습니다. 소프트웨어 형상 관리를 효과적으로 수행하는 기법을 배우고 싶다면, Stephen Berczuk와 Brad Appleton이 쓴 『Software Configuration Management Patterns』을 참고하세요.

지속적인 통합 서버

지속적인 통합 서버는 버전 관리 저장소에 변경 사항이 커밋되어 들어올 때마다 통합 빌드를 돌립니다. 대체로, 버전 관리 저장소에 변경 사항이 들어왔는지 몇 분 간격으로 확인하도록 CI 서버를 설정하게 됩니다. 지속적인 통합 서버는 소스 파일을 가져와서 빌드 스크립트나 다른 스크립트를 실행시킬 겁니다. 지속적인 통합 서버가 매 시간마다 또는 일정 주기로 빌드하도록 일정을 잡을 수도 있습니다(하지만 이런 게 지속적인 통합이라고 생각하진 마세요). 이뿐만 아니라 지속적인 통합 서버는 보통 빌드 결과를 게시하는 편리한 대시보드를 제공합니다. 비록 권장되기는 하지만, 지속적인 통합 서버가 반드시 지속적인 통합을 수행해야 하는 건 아닙니다. 스스로 자신만의 맞춤형 스크립트를 작성할 수 있습니다. 더 나아가, 변경 사항이 저장소에 적용될 때마다 통합 빌드를 직접 돌릴 수도 있습니다. 지속적인 통합 서버를 사용하면[2] 사람이 직접 작성해야 할 맞춤형 스크립트의 개수를 줄일 수 있습니다. 여러 지속적인 통합 서버가 무료이고 오픈소스입니다. 목록 1-3은 CruiseControl의 설정 파일 config.xml을 사용해서 서브버전 저장소에 변경 사항이 들어왔는지 확인하는 예제입니다.

목 록 1-3 | 서브버전 저장소를 폴링하는 CruiseControl의 설정 파일 config.xml

```
<project name="brewery" >
  <listeners>
    <currentbuildstatuslistener file="logs/${project.name}/status.txt"/>
  </listeners>
  <modificationset quietperiod="30">
    <svn RepositoryLocation="http://build.ib.com/trunk/brewery"
      username="bfranklin"
      password="G0Fly@Kite"/>
  </modificationset>
  <schedule interval="300">
    <ant anthome="apache-ant-1.6.5" buildfile="bld-{project.name}.xml"/>
  </schedule>
  <log dir="logs/${project.name}">
    <merge dir="projects/${project.name}/impl/logs/junit"/>
    <merge dir="projects/${project.name}/impl/logs/cobertura"/>
  </log>
  <publishers>
    <artifactspublisher dir="projects/${project.name}/impl/logs"
dest="artifacts/${project.name}"/>
  <artifactspublisher dir="projects/${project.name}/impl/logs"
dest="artifacts/${project.name}"/>
  </publishers>
</project>
```

2) 지속적인 통합 서버에 대한 정보를 더 얻고 싶다면, 부록 B를 보세요.

목록 1-3을 보면, `schedule` 태스크task의 `interval` 속성attribute이 서브버전 저장소에 변경 사항이 들어왔는지, CruiseControl이 얼마나 자주 변경 사항을 확인해야 하는지 지시합니다(이 예제에서는 300초입니다). 만약 CruiseControl이 뭔가 변경됐다는 걸 알아내면, 빌드를 위임(목록 1-3의 `buildfile` 속성에서)합니다. 위임한 빌드(보이진 않지만)는 최신의 소스 코드를 저장소에서 가져와서 프로젝트 빌드 파일을 실행시킵니다. 목록 1-3의 예제처럼 말이죠. 다른 CI 서버라면 관리하기 위해 웹 기반의 설정 인터페이스 또는 다른 용도의 인터페이스를 사용할지도 모릅니다. CruiseControl엔 웹 애플리케이션이 딸려있어서 최근의 빌드 결과와 빌드 보고서를 볼 수 있습니다(테스트 보고서나 검사 보고서 같은). 그림 1-2는 CruiseControl 빌드 결과에 대한 예제입니다.

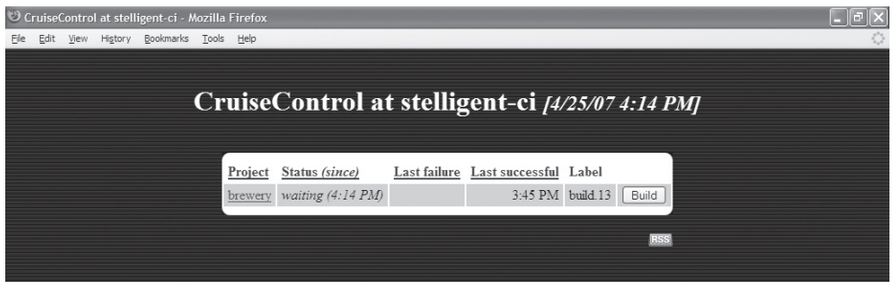


그림 1-2 | 최근의 빌드 상태를 보여주는 CruiseControl 대시보드

빌드 스크립트

빌드 스크립트는 코드 컴파일 및 검사, 배포 등의 작업들을 자동화하는 배치 스크립트를 말합니다. 물론 지속적인 통합 시스템이 아니더라도 빌드 스크립트를 사용할 수 있습니다. 이런 자동화 빌드 스크립트 도구로는 Ant나 NAnt, make, MSBuild, Rake 등이 있지만, 이 도구들 자체가 CI를 구현해 주지는 않습니다. 물론 IDE로 빌드하는 사람도 많습니다. 하지만 지속적인 통합은 '사람이 개입되지 않는hands-off' 자동화된 과정이어야 하므로, IDE 기반의 빌드를 이용해서는 진정한 지속적인 통합이라 할 수 없습니다. 가능하다면, IDE 없이 빌드할 수 있어야 합니다. 목록 1-4에 일반적으로 개인 빌드에 사용되는 Ant 스크립트의 예가 나와있습니다.[3)]

목 록 1-4 | 빌드를 수행하는 Ant 스크립트 껍데기

```
<?xml version="1.0" encoding="iso-8859-1"?>
<project name="brewery" default="all" basedir=".">
   <target name="clean" />
   <target name="svn-update" />
   <target name="all" depends="clean,svn-update"/>
   <target name="compile-src" />
   <target name="compile-tests" />
   <target name="integrate-database" />
   <target name="run-tests" />
   <target name="run-inspections" />
   <target name="package" />
   <target name="deploy" />
</project>
```

피드백 메커니즘

지속적인 통합을 사용하는 주 목적 중 하나는 통합 빌드 시, 이에 대한 피드백을 즉시 주는 것입니다. 최신 빌드에 뭔가 문제가 있다면 최대한 빨리 그 사실을 알아야 하니까요. 피드백을 일찍 받으면 더 신속하게 문제를 해결할 수 있습니다. 그림 1-3에서는 이메일로 피드백을 줍니다. 9장에선 좀더 다양한 피드백 장치에 대해 살펴볼 것입니다. 문자 메시지(SMS)나 RSS를 통해 피드백을 줄 수도 있습니다.

3) 보다 구체적인 예제는 www.integratebutton.com에 있습니다.

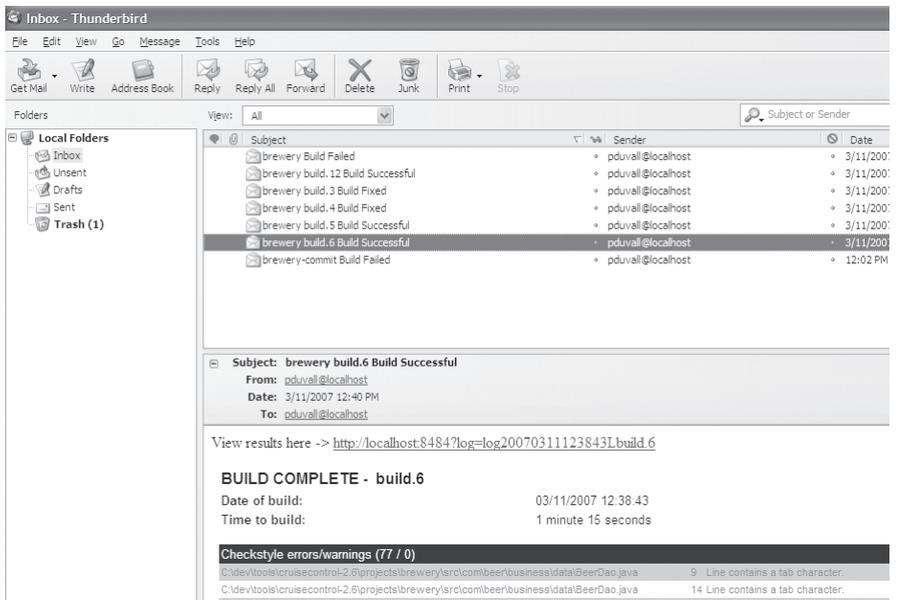


그림 1–3 | CI 서버가 보낸 이메일 메시지

목록 1-5는 CruiseControl CI 서버를 활용해서 프로젝트 구성원들에게 이메일을 보내는 예제입니다.

목 록 1-5 | 이메일을 보내도록 설정한 CruiseControl 빌드 파일 config.xml

```
<project>
   …
   <publishers>
      <htmlemail
         css="./webapps/cruisecontrol/css/cruisecontrol.css"
         mailhost="localhost"
         xsldir="./webapps/cruisecontrol/xsl"
         returnaddress="pduvall@localhost"
         buildresultsurl="http://localhost:8080"
         mailport="225"
         username="pduvall"
         password="password"
         reportsuccess="always"
         spamwhilebroken="true">
         <always address="pduvall@localhost"/>
         <always address="aglover@localhost"/>
      </htmlemail>
   </publishers>
   …
</project>
```

통합 빌드 머신

통합 빌드 머신은 소프트웨어를 빌드하는 것을 그 유일한 존재 목적으로 삼는 별도의 컴퓨터를 말합니다. 통합 빌드 컴퓨터에서 CI 서버가 돌아가고, CI 서버는 버전 관리 저장소를 폴링합니다.

지속적인 통합의 특징

이제 빌드할 때 쓸 수 있는 예제가 있으니, 지속적인 통합의 특징을 깊이 파고들 수 있습니다. 지속적인 통합이 요구하는 컴포넌트는 단 네 가지뿐입니다.

- 버전 관리 저장소로의 연결
- 빌드 스크립트
- 몇 종류의 피드백 메커니즘 (이메일 같은)
- 소스 코드 변경 내역을 통합하기 위한 프로세스 (수작업으로 하든, CI 서버를 활용하든)

이것이야말로 효과적인 CI 시스템의 핵심이라 하겠습니다. 버전 관리 시스템에 변경 내역이 들어올 때마다 자동화된 빌드가 작동하도록 일단 만들어놓고 나면, CI 시스템에 다른 기능을 넣을 수가 있습니다.

데이터베이스 통합, 테스트, 검사, 배포, 피드백을 자동화하고 지속적으로 수행함으로써, CI 시스템은 프로젝트에서 흔히 발생하는 일반적인 위험 요소를 줄여줍니다. 또 이렇게 함으로써 프로젝트 수행에 대한 자신감을 키워주며, 의사소통이 더 잘 이뤄지게 해줍니다. CI의 일부 특징은 다른 특징에 의존적인데, 예를 들면 자동화된 테스트는 소스 코드 컴파일에 의존적입니다.

이렇게 반복 가능한 프로세스는 개발 생명 주기를 통틀어 발생할 법한 위험을 경감시키는 데 도움이 될 수 있습니다. 이러한 하부 프로세스는 뒤이어 자세히 설명합니다.

소스 코드 컴파일

지속적인 소스 코드 컴파일은 CI의 가장 기본적이고 일반적인 특징 중 하나입니다. 사실, 지속적인 소스 코드 컴파일은 아주 흔하게 하는 일이라 CI와 동의어로 취급받

★ 통합 버튼

통합 버튼(그림 1-4를 보세요)은 완벽하게 작동하는 자동화된 통합 빌드(빌드를 별 것 아닌 일로 만드는)를 시각화한 것입니다. 여기엔 소프트웨어가 의도한 바대로 작동하게 해주는 여러 과정이 포함됩니다. 여러분은 컴파일하고, 테스트 데이터로 데이터베이스를 재구축하고, 테스트를 돌리고, 검사하고, 배포하고, 피드백을 제공할 수 있습니다. 빌드를 자동화하면, 버튼 하나만 눌러도 이 많은 과정을 수행할 수 있게 됩니다.

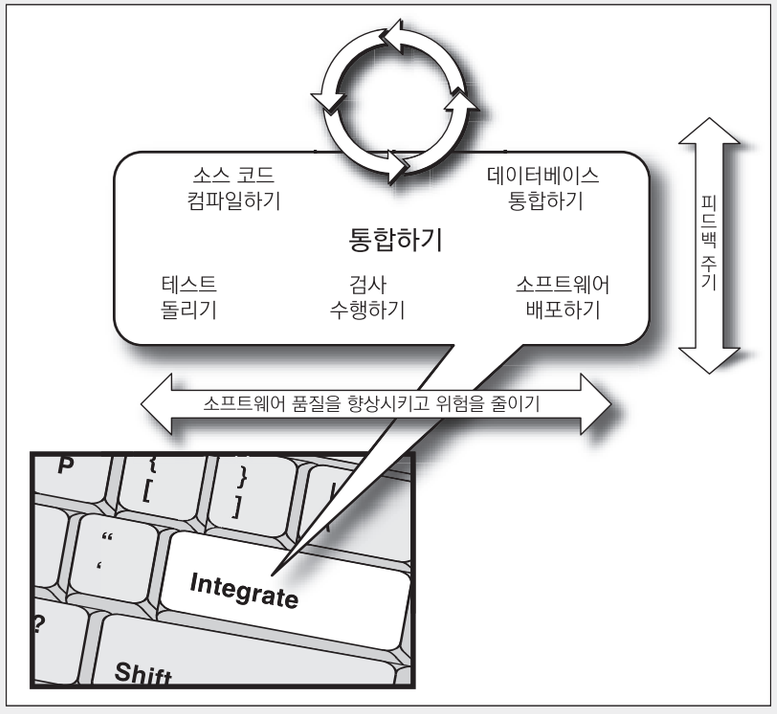


그림 1-4 | 통합 버튼의 시각화

을 정도입니다. 컴파일이란 인간이 읽을 수 있는 소스 코드로부터 실행 가능한 코드를 만들어내는 일과 관련이 있습니다. 그럼에도 불구하고 지속적인 통합은 소스 코드 컴파일보다 훨씬 더 나아간 개념입니다. 동적 언어-파이썬, PHP, 루비 등등-가 널리 쓰이게 되면서, 이런 환경에서의 컴파일은 미묘하게 다른 개념이 됐습니다. 비록 동적 언어를 사용해서 바이너리를 생성하진 않더라도, 여러 동적 언어가 엄격한

점검(strict check)을 수행하는 기능을 제공하므로, 이런 언어의 관점에서는 엄격한 점검을 컴파일로 간주할 수 있습니다. 이런 미묘함에도 불구하고, 동적 언어 환경은 지속적인 통합 빌드 과정에서 실행되는 다른 활동으로부터 혜택을 받습니다.

데이터베이스 통합

소스 코드 통합과 데이터베이스 통합을-언제나 서로 다른 부서가 수행하는-완전히 별개의 업무로 생각하는 사람도 있습니다. 이는 상당히 잘못된 생각인데, 왜냐하면 데이터베이스는 대부분의 애플리케이션에 있어 없어서는 안될 우주의 중심이라고 할 수 있기 때문입니다. 그러므로 CI 시스템에 데이터베이스 통합은 반드시 포함되어야 하며 이를 통해 단일 소스, 즉 동일한 버전 관리 저장소를 통해 데이터베이스 역시 통합되어 확실한 결과를 얻을 수 있어야 합니다.

그림 1-5는 CI 시스템의 빌드 시 데이터베이스 역시 통합되고 있는 것을 보여줍니다. 데이터베이스 소스 코드-데이터 정의 언어(DDL) 스크립트, 데이터 조작 언어(DML) 스크립트, 저장 프로시저, 분할 등등-역시 다른 소스 코드와 똑같이 취급합니다. 예를 들어, 누군가(개발자나 DB 관리자) 데이터베이스 스크립트를 수정하고 그것을 버전 관리 시스템에 커밋한다면, 빌드 스크립트가 이를 이용하여 DB를 생성하고 데이터도 빌드합니다.

목록 1-6에서는 Ant의 SQL 작업을 이용하여 MySQL DB를 삭제(drop)하고 생성(create)합니다. DB와 테스트 데이터를 다시 빌드하려면 좀더 해야 할 일이 많습니다. 이 예제에서는 설명을 위해 일부러 매개변수들을 하드 코딩해 넣었습니다.

목 록 1-6 | MySQL과 Ant

```
<target name="db:create-database”>
   <sql driver="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost:3306/"
      userid="root"
      password="sa"
      classpathref="db.lib.path"
      delimiter=";">
      <fileset file="${database.dir}/drop-database.sql"/>
      <fileset file="${database.dir}/create-database.sql "/>
   </sql>
</target>
```

5장에선 데이터베이스 통합 예제와 접근 방법, 그리고 그 장점을 보여주겠습니다.

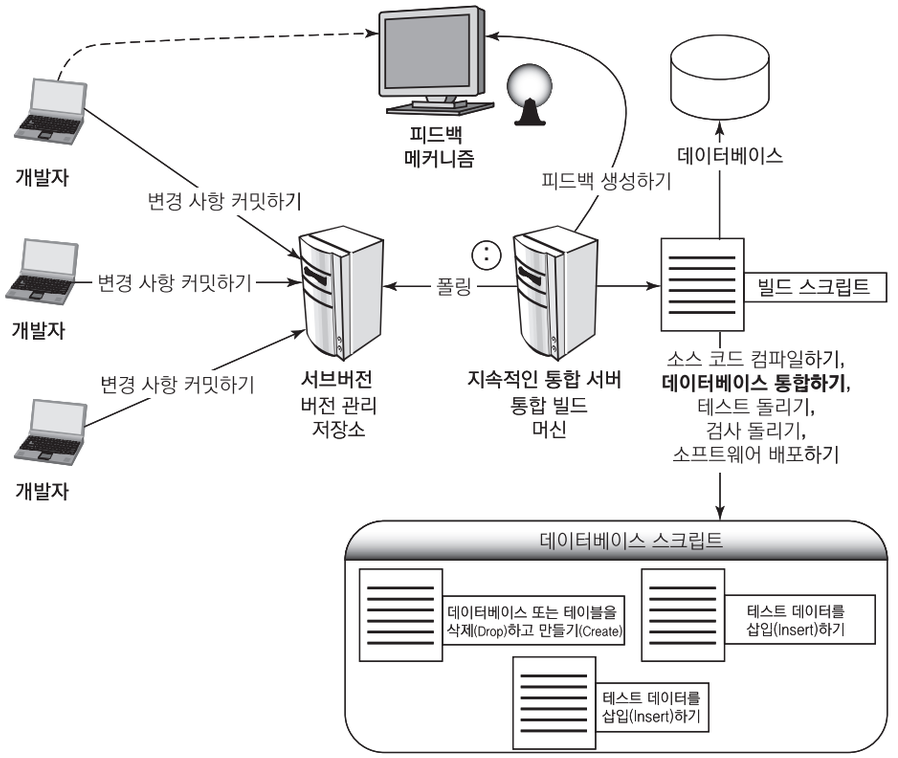


그림 1–5 | 데이터베이스 통합 설계

테스트

자동화되고 지속적인 테스트가 구비되어 있지 않은 CI는 CI가 아니라고 생각하는 사람이 많습니다. 전적으로 동의합니다. 자동화된 테스트가 없다면, 개발자나 다른 프로젝트 인원들이 자신감을 갖고 소프트웨어를 수정하기 어렵습니다. 그렇기 때문에, 대부분의 CI 시스템을 이용하는 프로젝트에서는 JUnit이나 NUnit 또는 기타 xUnit 프레임워크의 단위 테스트 도구를 사용하여 테스트를 돌립니다. 물론 테스트를 여러 범주로 나눠서 번갈아가며 돌리는 식으로 빌드 속도를 높여도 됩니다. 이런 테스트들에는 단위, 컴포넌트, 시스템, 부하 및 성능, 보안 테스트 등을 포함할 수 있습니다. 6장에서 이에 대해 자세히 논의하고자 합니다. 그림 1-6에 CruiseControl에 사용되는 JUnit의 테스트 보고서가 나와 있습니다.

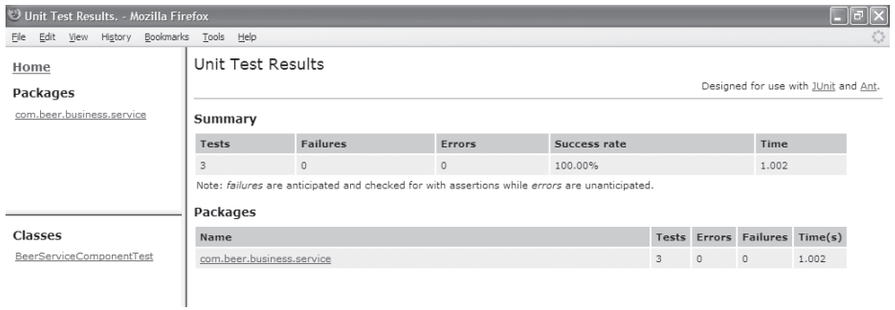


그림 1.6 | JUnit을 사용한 단위 테스트 회귀 보고서

목록 1-7은 JUnit 테스트 배치를 돌리고, Ant 태스크를 사용해서 그림 1-6의 보고서를 생성하는 예제를 보여줍니다.

목 록 1-7 | Ant와 JUnit

```
<?xml version="1.0" encoding="iso-8859-1"?>
   <target name="run-tests">
      <mkdir dir="${logs.junit.dir}" />
      <junit fork="yes" haltonfailure="true" dir="${basedir}"
         printsummary="yes">
        <classpath refid="test.class.path" />
        <classpath refid="project.class.path"/>
        <formatter type="plain" usefile="true" />
        <formatter type="xml" usefile="true" />
        <batchtest fork="yes" todir="${logs.junit.dir}">
          <fileset dir="${test.unit.dir}">
            <patternset refid="test.sources.pattern"/>
          </fileset>
        </batchtest>
      </junit>
      <mkdir dir="${reports.junit.dir}" />
      <junitreport todir="${reports.junit.dir}">
        <fileset dir="${logs.junit.dir}">
          <include name="TEST-*.xml" />
          <include name="TEST-*.txt" />
        </fileset>
        <report format="frames" todir="${reports.junit.dir}" />
      </junitreport>
   </target>
</project>
```


검사 (Inspection)

자동화된 코드 검사(이를테면, 정적 분석이나 동적 분석)는 여러 코딩 규칙을 강제함으로써 소프트웨어의 품질을 높이는 데 사용할 수 있습니다. 예를 들어, 어떤 프로젝트는 주석이 안 달린 코드가 300줄을 넘는 클래스가 있어선 안 된다는 규칙이 있을지 모릅니다. CI 시스템을 써서 코드 베이스에 대해 이런 규칙이 자동으로 지켜지게 할 수 있습니다. 7장에서는 다양한 도구와 기법을 논의하고 보여줄 것입니다.

그림 1-7의 소프트웨어 검사 보고서 예제는 자바 코드를 검사하는 Checkstyle을 이용해 생성했습니다. 이 같은 보고서를 사용하면 코딩 표준과 품질 메트릭을 지속적으로 모니터링할 수 있습니다.

목록 1-8은 Checkstyle 같은 정적 코드 분석 도구를 Ant와 함께 사용하는 예제를 보여줍니다. 이 예제는 그림 1-7의 보고서를 생성합니다.

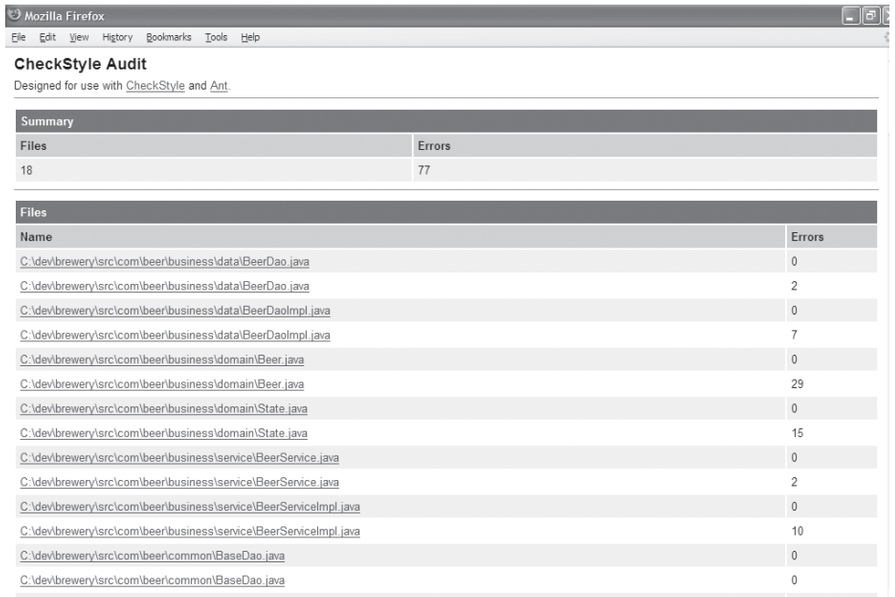


그림 1–7 | Checkstyle를 사용하여 자동 생성한 검사 보고서

목 록 1-8 | Ant를 사용한 Checkstyle 예제 [4]

```
<target name="run-inspections">
   <taskdef resource="checkstyletask.properties"
classpath="${checkstyle.jar}"/>
   <checkstyle config="${basedir}/checkstyle-rules.xml"
failOnViolation="false">
      <formatter toFile="${checkstyle.data.file}" type="xml" />
      <fileset dir="${src.dir}" includes="**/*.java" />
   </checkstyle>
   <xslt taskname="checkstyle"
      in="${checkstyle.data.file}"
      out="${checkstyle.report.file}"
      style="${checkstyle.xsl.file}" />
</target>
```

배포

상당수 빌드 과정은 일종의 배포 과정이라 할 수 있습니다. 사실, 이 장에서 논의된 각 작업 과정 역시 배포 과정의 일부라고 할 수 있습니다. 배포를 지속적으로 하면 어느 순간이라도 실제로 작동하는 소프트웨어를 배포할 수 있습니다. CI 시스템의 핵심 목표는 항상 최근 코드를 가지고 소프트웨어를 빌드하고 테스트하는 것입니다.

무엇보다도, 버전 관리 저장소에서 소스 파일을 체크아웃하고, 빌드를 수행하고, 테스트와 검사를 전부 성공시키고, 릴리즈에 꼬리표를 붙이며, 배포 파일을 잘 꾸며야 합니다.

CI를 이용하면 그림 1-8처럼 파일을 적절한 환경에 자동으로 배포하거나 설치할 수도 있습니다. 더욱이, 배포에는 배포할 때 적용된 모든 변경 사항을 자동으로 되돌리는 능력이 있어야 합니다. 주의하세요. 개발할 때의 운영 환경(그림 1-8에 묘사된 Jetty를 예로 들자면)은 통합 환경이나 테스트 환경(Tomcat)과는 약간 다를지도 모릅니다. 여하튼, 매개변수만 약간 다를 뿐인 자동화된 빌드가 운영 환경이나 통합 및 테스트 환경에서 실행됩니다. 우리는 8장에서 이 같은 전략을 논의합니다.

4) 폴 듀발이 2006년 8월자 IBM developerWorks에 발표한 '사람을 위한 자동화: Continuous Inspection'이 출처입니다. http://www.ibm.com/developerworks/kr/library/j-ap08016/

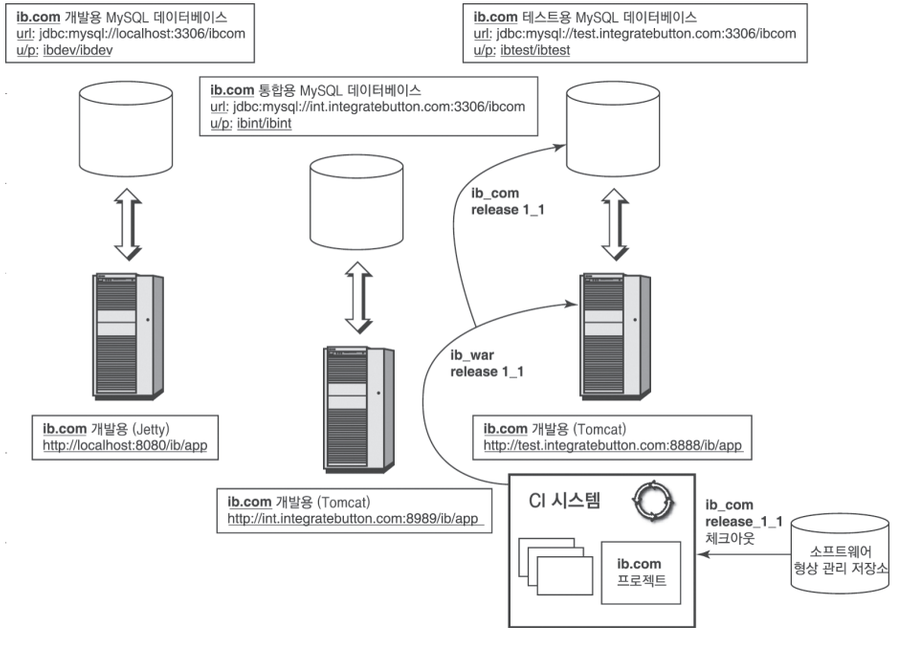


그림 1-8 | 배포 환경

목록 1-9는 Ant와 웹 컨테이너 간의 인터페이스를 제공하는 Cargo란 도구를 사용하는 걸 보여줍니다. 이 경우에는 Tomcat 서버에 배포를 합니다. Cargo는 시장에 나온 유명한 웹 컨테이너 다수를 지원합니다.

목 록 1-9 | Ant와 Cargo를 이용해서 Tomcat에 배포한다

```
<target name="deploy">
   <cargo containerId="tomcat5x" action="start"
      wait="false" id="${tomcat-refid}">
      <zipurlinstaller installurl="${tomcat-installer-url}"/>
      <configuration type="standalone" home="${tomcatdir}">
         <property name="cargo.remote.username" value="admin"/>
         <property name="cargo.remote.password" value=""/>
         <deployable type="war" file="${wardir}/${warfile}"/>
   </configuration>
   </cargo>
</target>
```

문서화와 피드백

많은 개발자들이 문서는 소스 코드에 있는 법이라고 굳게 믿으며 일하는데, 사실 잘 선택한 클래스, 변수, 메소드 이름으로 무장한 깔끔하고 간명한 코드야말로 최고의 문서라 하겠습니다. 지속적인 통합 시스템은 고생하지 않아도 문서화의 이점을 제공해줄 수 있습니다. Maven, Javadoc, NDoc 같은 도구를 이용하면 문서를 생성할 수 있습니다. 이뿐만 아니라 클래스 다이어그램과 기타 정보를 생성할 수 있는 도구도 있는데, 모두가 버전 관리 저장소에 커밋된 소스 코드를 기반으로 삼습니다. 여러분은 지속적인 통합 시스템을 활용하여 소스 코드와 프로젝트 상태 문서화를 실시간에 획득했을 때 얻게 되는 어마어마한 이점을 알게 될 겁니다. 사람에 따라선 문서 산출물을 지속적으로 생성하기보단 주기적으로 생성하기로 결정하는 경우도 있을 겁니다.

좋은 CI 시스템의 핵심적인 특징은 속도입니다. 지속적인 통합 시스템의 정수는 개발자와 프로젝트 이해관계자들에게 시의 적절하게 피드백을 제공하는 겁니다. 지속적인 통합 시스템에 이것저것 끼워 넣다보면(완벽하게 하려고), 한 주기를 끝마치는 데 말도 안 되는 시간이 걸리기 쉽습니다. 그러니 결과를 신속하게 제공해야 할 필요성과 지속적인 통합 프로세스의 폭과 깊이 사이에서 균형을 잡아야 합니다. 이것은 지속적인 테스트를 활용할 때 특히 중요합니다. 우리는 4장과 6장에서 빠른 빌드를 만드는 기법에 대해 논의합니다.

요약

이 장에서는 지속적인 통합의 특징을 개략적으로 알아봤습니다. 그리고 포괄적인 데이터베이스 통합, 테스트, 검사, 배포, 피드백과 같은 추가적인 프로세스를 지속적인 통합 시스템에 집어넣는 방법도 살펴보았습니다. 이 책의 나머지 부분에서는 지속적인 통합을 사용한 소프트웨어 개발과 연관 지어 이러한 프로세스를 하나씩 구체적으로 들여다 볼 것입니다.

질문

지속적인 통합을 올바르게 수행하고 있는지 어떻게 알 수 있을까요? 다음과 같은 질문을 통해 프로젝트에 무엇이 부족한지 파악할 수 있습니다.

- ☑ 버전 관리 저장소(소프트웨어 형상 관리 도구)를 사용합니까?
- ☑ 빌드 과정이 자동화되고 똑같이 반복될 수 있습니까? 사람이 개입하지 않아도 빌드 과정이 완전히 작동합니까?
- ☑ 자동화된 테스트를 작성해서 돌립니까?
- ☑ 빌드 과정의 일부로써 테스트를 실행합니까?
- ☑ 코딩 및 설계 표준을 어떻게 강제합니까?
- ☑ 자동화시킨 피드백 메커니즘에는 어떤 게 있습니까?
- ☑ 소프트웨어를 빌드할 때 별도의 통합 빌드 머신을 사용합니까?