※ 본 아티클은 CMP MEDIA LLC와의 라이선스 계약에 의해 국문으로 제공됩니다.

Gamasutra.com

재미를 포용하라

익스트림 프로그래밍이 게임 개발에 훌륭한 이유

Bill Schofield
2007년 3월 01일

http://www.gamasutra.com/features/20070301/schofield_01.shtml

익스트림 프로그래밍[1] (XP)은 변화를 예측하기 보다는 변화를 수용하는데 초점을 둔 애자일 소프트웨어 개발 방법론(Agile Software Development Methodology)이다. XP 방법론의 장점은 개발자들이 퍼블리셔와 경영진은 물론 개발자 자신의 다양한 요구를 충족시키면서 훌륭한 게임을 만들 수 있다는데 있다.

일반적으로 어느 하나에 신경 쓰다 보면 다른 것들에는 그만큼 무심해지는 것이 당연하다고 하지만 XP 는 개발팀이 이해관계자의 다양한 요구를 모두 충족시키면서 고품질의 게임을 개발하도록 이끈다.

XP 는 제대로 동작하는 소프트웨어의 빠른 릴리스를 강조하기 때문에 더욱 안정적으로 훌륭한 게임을 개발할 수 있다. 제대로 동작하는 소프트웨어의 빠른 릴리스란 특정 기능을 전체 시스템에 구현하지 않고도 이들 기능이 어떻게 동작하는지 게임 디자이너가 확인할 수 있다는 것을 의미한다. 이를 통해 게임 디자이너는 더욱 쉽고 빠르게 재미있는 게임 플레이를 찾을 수 있다.

그렇다면 XP 로 인해 게임 디자인에 더 많은 변화가 가해지는가? 이는 아마 대부분의 프로그래머가 가장 듣고 싶지 않은 말일 것이다. 특히 변경을 가하기 어려운 소프트웨어를 개발 중이라면 갑작스러운 변경만은 꼭 피하고 싶을 것이다. 하지만 현실에서 디자이너는 좀더 재미있는 게임 플레이를 찾기 위해 개발 중에 게임이 실제로 어떻게 동작하는지 확인하고 맘에 들지 않으면 계속해서 변경을 요구한다. 그리고 좀더 훌륭한 게임을 만들어야 할 책임이 있는 프로그래머는 가능한 많은 변화를 수용하고 처리해야 한다. 익스트림 프로그래밍은 프로젝트의 유연성을 유지하면서 결과를 더욱 빨리 전달하는데 개발 에너지를 집중한다.

이론상으로는 완벽하다. 하지만 과연 XP 는 어떤 방식으로 프로젝트가 쉽게 변화를 수용할 수 있도록 하는가? XP 는 상호보완적인 실천사항을 통해 최대한 단순하고 이상적인 코드를 유지한다. 즉 XP 는 개발팀에서 기능의 우선 순위를 정해 그 순서대로 작업하고 이들 기능을 구현하는데 필요한 코드만 작성하며 코드를 이상적으로 유지하기 위해 지속적으로 조금씩 디자인을 향상하도록 한다.

그럼 이제 테스트 주도적인 개발, 짝 프로그래밍(Pair Programming), 연속적인 디자인, 실제 고객 참여, 활동적인 작업 등 XP 의 다섯 가지 핵심 실천사항에 대해서 간략하게 살펴보자. 이들 다섯 개를 포함한 XP 의 여러 실천사항은 개발팀에서 더욱 빨리 눈에 보이는 결과를 만들어낼 수 있도록 돕는다. 그리고 이렇게 전달된 결과는 디자이너가 게임을 더욱 재미있게 만들 수 있도록 여러 가지 정보를 제공한다. 프로그래머는 코드를 유연하게 유지하기 때문에 새로운 변화를 수용하고 처리하기가 용이하고 이들 변화는 다음 사이클에서 확인할 수 있다.

이들 실천사항을 살펴보면 알겠지만 XP 는 자동 유닛 테스트나 주당 40 시간 근무, 또는 짝 프로그래밍이 아니라는 사실을 명심해야 한다. 대부분의 우수한 XP 팀에서 이런 모든 것들을 실천하고 있지만 XP 는 단순히 훌륭한 게임을 제작하는 것을 의미한다. 훌륭한 게임의 제작, 이것이야 말로 XP 의 진정한 의미이다.

테스트 주도적인 개발[2]

테스트 주도적인 개발(Test Driven Development, TDD)을 실천하려면 프로그래머는 프로그램 내에서 새로운 기능이 어떻게 작동하기를 바라는지를 설명하는 작은 코드를 작성한다. 이 코드를 "유닛 테스트(Unit Test)"라 부르는데 테스트에서 설명하는 기능이 아직 구현되지 않았기 때문에 처음 테스트는 실패한다. 이 프로세스가 가지고 있는 장점 중 큰 장점은 자동화된 유닛 테스트이다.

자동화된 유닛 테스트(Automated Unit Tests)에서는 소프트웨어의 각 부분이 프로그래머가 원하는 대로 움직이고 있는지 확인할 수 있다. 그리고 개발자는 간편하게 유닛 테스트를 실행해 새로운 변경 사항이 이미 구현된 기능을 방해하지는 않는지 쉽게 확인할 수 있다. 이들 테스트를 통해 프로젝트에 빠르고 안전하게 변경 사항을 적용할 수 있는 것이다.

회귀(Regression)는 개발팀이 넘어야 할 또 다른 큰 난관이다. 프로젝트 초기에는 지속적으로 작업해야 할 기능도 몇 개 없지만 프로젝트가 진행될수록 동시에 진행해야 할 기능의 수도 많아진다. 가끔은 어떤 기능들을 작업 중인지 파악하기조차 힘들어지기도 한다. 하지만 자동화된 테스트는 현재 개발팀에서 작업 중인 기능의 수를 보여주는 프레임워크를 제공하기 때문에 개발팀은 혹시라도 미처 파악하지 못한 회귀가 있지는 않을지 걱정하지 않고 새 기능을 구현할 수 있다.

그 외에도 테스트 주도적인 개발은 테스트 성공을 통해 프로그래밍 팀에 활기를 주기도 한다. 하나의 기능이 성공적으로 동작하기 전에 계속해서 작은 테스트를 진행하기 때문에 프로그래밍 팀에서 테스트 통과라는 기쁨도 덤으로 맛 볼 수 있는 것이다.

짝 프로그래밍[3]

짝 프로그래밍(Pair Programming)은 두 프로그래머가 하나의 컴퓨터에서 동시에 작업하는 프로세스이다. 일반적으로 둘 중 한 명은 실제로 키보드를 잡고 어떤 코드를 입력할지 결정해 타이핑하고 다른 한 명은 이를 지켜보고 방향을 인도한다.

옆에서 지켜보는 사람은 입력하는 사람이 어떤 전술상 실수를 저지르는지 살펴보기 보다는 전체적인 문맥에 집중한다. 두 프로그래머는 "지금 작성하는 새 코드가 기존 코드와 어떻게 어울리는가?" "우리가 방금 작성한 코드를 더욱 간단하지만 우수하게 만들 방법이 있는가?"와 같은 질문을 던진다.

두 명의 프로그래머가 함께 작업하면 실수가 훨씬 적어지기 때문에 따로 작업했을 때보다 훨씬 고품질의 단순화된 코드를 생성할 수 있다. "백지장도 맞들면 낫다."라는 속담이 이를 잘 표현한다. 그리고 이렇게 생성된 고품질의 코드는 혼자 작성한 코드보다 쉽고 안전하게 변경할 수 있다.

게임 프로그래머는 항상 디자인 상에 작은 변경을 가한다. 그리고 이 변경으로 인해 게임이 훨씬 재미있어지기도 하고 그 반대가 되기도 한다. 창의적인 두 명의 프로그래머는 몇 분간의 브레인스토밍을 통해 놀랄만한 결과를 만들어낼 수 있으며 짝 프로그래밍을 통해 복잡한 회의 과정이나 다른 사람의 방해 없이 빠른 의사 결정을 도출할 수 있다.

짝 프로그래밍은 두 프로그래머가 경험을 공유하고 상호작용을 높이며 실수는 줄이면서 서로간의 신뢰 하에 더욱 안정적이고 활기찬 팀을 만들도록 이끈다.

지속적인 디자인[4]

지속적인 디자인(Continuous Design)은 소프트웨어 디자인이 현시점에서 프로그램에 필수적인 사항과 조화를 이루도록 하는 기법이다. 이는 프로그램에 일반화(Generality)가 필요하기 전에는 굳이 일반화된 시스템을 만들 필요가 없다는 의미이기도 하다. 단, 시스템의 단순화된 버전이 훌륭한 디자인을 가지고 있어야 한다는 전제를 만족시켜야 한다. 코드를 적게 작성한다는 것은 중간 과정을 생략한다거나 편법을 사용한다는 의미가 아니다. 코드의 양이 적으면 코드 기반을 더욱 쉽게 변경할 수 있기 때문에 꼭 필요한 것만 구현할 수 있다는 것을 의미한다.

지속적인 디자인 프로세스에는 기능을 변경하지 않고 코드를 변경해 더욱 우수한 코드를 만드는 리팩토링(Refactoring)이 포함된다. 개발

중인 디자인을 더 잘 이해하거나 더 단순화된 디자인이 소프트웨어의 커져가는 요구에 부합하지 않을 경우, 코드의 구조를 바꾸면 프로그램이 더 우수해지거나 단순화될 수 있다. 리팩토링은 코드를 유연성 있게 유지하는 한 방법이다.

지속적인 디자인은 코드를 더욱 단순하게 만든다. 코드는 단순할수록 이해하기 쉽고 실패할 확률도 낮아지며 개발자들의 자신감을 높이고 안정을 보장해 즐겁고 활기찬 팀을 만든다.

실제 고객 참여[5]

실제 고객 참여(Real Customer Involvement)란 간단히 말해 되도록 빨리 실제 사용자에게 게임을 플레이 할 기회를 제공하는 것이다. 개발팀은 눈에 보이는 결과에 치중하므로 고객 참여는 되도록 개발 초기에 행해져야 한다. 실제 고객 참여는 개발팀에서 기능의 우선 순위를 정하고 향후 더욱 우수한 아이디어를 생각해낼 수 있도록 돕는다.

그 외에도 기능을 구현하는 동안 디자이너와 프로그래머가 매우 가깝게 작업하기 때문에 최고의 성능을 발휘하고, 균형을 이루며 디자이너가 이 정도면 지금은 충분하다고 말할 수 있는 프로그램 작성에 집중할 수 있다. 프로그래머가 빠른 시간 내에 눈에 보이는 결과를 제공하는데 집중하고 디자이너가 무엇이 재미있고 재미없는지 유용한 피드백을 제공할 수 있으면 프로그래머와 디자이너가 함께 더욱 재미있는 게임을 만들 수 있다.

최고의 결과를 얻기 위한 협동 작업은 고객과 개발자 모두를 만족시키며 협동 작업을 통한 기능 구현은 팀의 신뢰를 높인다.

활동적인 작업[6]

유지 가능한 페이스(Sustainable Pace) 또는 주 40 시간 근무(40 Hour Work-Week)로도 알려진 활동적인 작업은 생산성을 최대로 발휘할 수 있는 시간 동안만 작업하는 것을 가리킨다. 피곤하거나 몸이 안 좋은 상태에서도 작업을 계속하면 작업 속도가 떨어지고 실수도 많아진다. 그리고 몸은 더욱 피곤해지고 상태도 더욱 안 좋아진다.

프로그래밍이란 기본적으로 아이디어가 중요하기 때문에 적당한 시간에 좋은 아이디어를 생각해내기만 하면 짧은 시간에도 많은 작업을 처리할 수 있다. 그러나 좋은 아이디어란 쉽게 나오는 것이 아니어서 우리의 뇌를 100% 활용하지 못하면 최고의 해결책을 생각해낼 수 없다. 활동적인 작업은 우리의 뇌를 건강하고 즐거운 상태로 유지해 우리가 원하는 방향으로 작업하도록 만든다.

활동적인 작업은 다른 모든 XP 실천사항을 지원하고 프로그래머가 작업에 최선을 다해 더욱 우수한 코드와 재미있는 게임을 만들 수 있도록 돕는다.

결론

처음에 언급했듯이 XP 는 자동화된 유닛 테스트나 주 40 시간 근무, 짝 프로그래밍이 아니다. 대부분의 우수한 XP 팀에서 이것들을 실천하기는 하지만 XP 는 그저 훌륭한 게임을 제작하는 것을 의미할 뿐이다. 훌륭한 게임 제작, 이것이야 말로 XP 의 진정한 의미이다.

XP 방법론의 장점은 여러 이해관계자의 다양한 요구를 충족하면서 훌륭한 게임을 만들 수 있다는데 있다.

- 게임이 더욱 재미있고, 눈에 보이는 결과를 더욱 자주 확인할 수 있으며 마케팅 팀에서 변화를 요구했을 때 이 새로운 요구를 충족하도록 게임을 변경할 수 있기 때문에 퍼블리셔에서 만족한다.

- 팀의 의욕이 높고, 새로운 요구에 맞춰 게임을 변경할 수 있으며 눈에 보이는 프로세스와 피드백을 통해 팀이 계획이 맞춰 작업하는지 확인할 수 있기 때문에 경영진에서 만족한다.

- 대부분의 XP 실천사항에는 보상이 따르고 팀에 재미와 활기를 주기 때문에 팀원은 더욱 만족스럽다.

결론적으로 말하자면 모두들 훌륭한 게임을 만들기를 원하기 때문에 누구나 XP 를 사용해 성공할 수 있다.

XP에 대해 더욱 자세히 알아보기

앞서 살펴본 다섯 가지 실천사항 외에도 완벽한 팀(Whole Team), 게임 계획(Planning Game), 빈번한 릴리스(Small Releases), 고객 테스트(Customer Tests), 지속적인 통합(Continuous Integration), 코드 공동 소유권(Collective Code Ownership), 코딩 표준(Coding Standard), 메타포(Metaphor) 등도 눈에 보이는 결과 전달에 일조한다. XP 또는 XP의 실천사항을 소개하는 훌륭한 자료들이 많이 있지만 그 중 켄트 벡(Kent Beck)의 익스트림 프로그래밍(Extreme Programming Explained)은 XP를 가장 잘 소개한 책으로 손꼽힌다. 그 외에 아래 사이트에서도 자세한 정보를 확인할 수 있다.

- http://www.extremeprogramming.org/
- http://www.xprogramming.com/xpmag/whatisxp.htm

---

[1] Beck, Kent. Extreme Programming Explained, Embrace Change, Second Edition. Boston: Addison-Wesley, 2004.

번역서: 익스트림 프로그래밍: 변화를 포용하라, 2 판. 인사이트, 2006. 켄트 벡, 신시아 안드레스 지음. 김창준, 정지호 옮김.

[2] Beck, Kent. Test-Driven Development by Example. Boston: Addison-Wesley, 2003.

번역서: 테스트 주도 개발. 인사이트, 2005. 켄트 벡 지음, 김창준, 강규영 옮김.

[3] Williams, Laurie and Kessler, Robert. Pair Programming Illuminated. Boston: Addison-Wesley, 2003.

[4] Shore, Jim. Continuous Design. IEEE Software, vol. 21, no.1, pp. 20-22, Jan/Feb, 2004. Online

<http://www.martinfowler.com/ieeeSoftware/continuousDesign.pdf >

[5] Beck, Kent. Extreme Programming Explained, Embrace Change, Second Edition. Boston: Addison-Wesley, 2004. pp. 61-62

[6] Beck, Kent. Extreme Programming Explained, Embrace Change, Second Edition. Boston: Addison-Wesley, 2004. pp. 41-42