# 소프트웨어 개발 프로세스와 XP

한국전자통신연구원 이강우
(주)이비즈온 송태국
충북대학교 조은선

## 1. 서 론

1980년대 이후 대두된 소프트웨어 위기(software crisis)에 대응하기 위해 다양한 소프트웨어 공학적 기법들이 출현하였다. 이 중 소프트웨어의 질적 향상을 꾀하기 위한 개발의 지침으로 여러 가지 소프트웨어 개발 프로세스(process)들이 제시되고 있다. 이러한 프로세스들이 중점을 두고 있는 목표는 소프트웨어 질과 개발 시간에 관련한 개인 또는 조직의 자질 향상이라고 볼 수 있다. 그리고 프로세스 별로 다소 차이는 있으나, 사용자의 요구 사항을 명확히 설정하고 이에 맞추어 개발되고 있는지에 대한 반복적인 검증을 진행하는 것을 그 기본 틀로 하는 것이 일반적이다.

그러나 이러한 접근 방법이 가지고 있는 맹점 중 하나는, 사용자의 요구가 개발 초기에 완전히 정리되어 개발자에게 전달된다는 가정에 지나치게 의존되어 있다는 점이다. 컴퓨터의 응용 분야가 확대됨에 따라, 사용자의 요구를 처음부터 정확히 파악하기도 힘들 뿐 아니라 사용자 스스로도 자신들의 요구를 개발 초기에 정리/전달해 주지 못하는 등, 사용자의 요구가 개발 과정 동안에 수정되어야 하는 상황은 상당히 빈번하게 발생되고 있다.

많은 소프트웨어 개발 프로세스들이 가지는 또 다른 문제점으로는 개발자에게 과도한 문서 작성 등의 관리 관련 업무가 부과되는 것이다. 본래 소프트웨어 개발을 위한 문서 작업은 검증 작업의 근거가 됨과 동시에 개발 담당자의 부재 시에도 소프트웨어의 유지 보수를 조직 차원에서 가능하게 한다는 목적으로 각 개발 프로세스에 도입되었다. 그러나 불필요한 다량의 문서 작성으로 인해 개발자의 개발 시간을 줄어들게 하고 개발 의욕을 저하시키기는 역효과도 있다.

본 논문에서 소개하는 XP(eXtreme Programming) 는 중소형 개발팀을 대상으로 Kent Beck과 그의 동료들에 의해 정의된 소프트웨어 개발 프로세스이다. XP의 가장 큰 특징 중 하나는, 사용자 요구 사항이 모호하거나 개발 도중 동적으로 변화되는 경우를 기본적으로 가정하고 짧은 주기의 반복적 절차를 통한 탄력적인 개발 일정 관리를 가능하게 하여 실패 위험도를 낮추는 것이다. 또 XP는 철저하게 개발자 중심의 프로세스로서 기존의 문서화에 기반한 정형화된 프로세스들이 가지는 틀에서 탈피하여, 개발팀의 개인적 집단적 역량이나 체계적인 테스트 과정 등에 많은 부분을 의존하여 효과를 얻도록 구성되어 있다. XP라는 이름은 좋은 소프트웨어 개발을 위한 미덕으로 알려진 각종 지침 (타인에 의한 코드 검토, 철저한 테스트, 단순 명확한 설계 및 코딩, 통합 테스트 등)들을 극단적으로 적용하자는 데에 기인한다[1].

다음 장에서는 XP에서 정의하고 있는 소프트웨어 개발을 위해 정의하고 있는 기본 활동들에 대해 간단히 소개하고, 3 장에서는 이를 따르기 위해 구체적으로 취해야하는 지침들을 나열한다. 4 장에서는 XP를 실제 적용한 사례, 특히 한국에서의 현황에 관해 소개한다. 5 장에서는 결론을 맺는다.

## 2. XP의 기본 활동(Activities)

앞서 언급했듯이 XP는 기존의 전통적인 소프트웨어 개발 프로세스들과는 매우 큰 차이를 가지고 있다. 이에 대한 이해를 돕기 위하여, 본 장에서는 XP를 구성하는 기본 활동에 관해 개념적으로 살펴본다.

XP에서 소프트웨어의 개발 주기는 짧게 반복되며 따라서 프로토타입이 일찍 그리고 자주 만들어지게 된다. 이로써 개발자들은 사용자로부터 구체적인 피드백을 빨리, 지속적으로 받도록 한다. 그리고 개발 계획은 초기에 완성되는 것이 아니라 프로젝트 진행 동

안 계속 변경되도록 한다. 따라서 변화하는 필요성에 따라서 구현 일정을 신축적으로 운용할 수 있게 된다.

또, XP는 프로그래머에게 중요한 일을 제외한 불필요한 작업을 과다하게 부과하지 않는다. 또, 협동 작업을 기초로 하여, 프로그래머가 개발 시 어려운 문제를 해결하는 데에 있어 팀 내에서 상호 도움을 주도록 권장하며, 전체 프로그램에 대한 권리와 의무를 팀의 개발자들이 함께 공유하게 한다.

사용자나 관리자의 입장에서 XP는 매 주단위로 결과를 확인할 수 있게 해주며, 몇 주단위로는 원하는 목표에 도달했는지에 대한 검증을 할 수 있도록 한다. 결과적으로는 프로젝트의 방향을 바꾸는 것에 대한 비용을 최소화 시킨다는 장점을 가진다.

XP가 정하고 있는 기본 활동을 소프트웨어 개발 과정의 각 단계에 따라 정리하면 다음과 같다.

- **설계(design)** : 시스템의 설계는 계속 검토되며 결과적으로 진화하게 된다. 이러한 설계의 변경은 시스템이 존재하는 한 영속적으로 이어진다. 따라서 사용자 요구와 상황 등이 변함과 함께 설계부터 적절하게 변화함으로써 설계의 질이 보장되게 된다.
- **코딩(coding)** : 매일 코딩 종료 시에 프로그램이 하나씩 완성되어야 한다. 또한, 일반적인 능력을 가진 개발자들 간의 긴밀한 협동 작업을 극도로 중시한다.
- **테스트(test)** : 활동 수행 여부 결정은 테스트에 전적으로 의존한다. 예를 들어, 프로그램의 일정 부분을 완성한다는 것은 해당 부분의 완성도를 검증할 수 있는 자동화된 테스트 프로그램까지 함께 작성하여 통과시켜야한다는 것이 된다. 이 때 테스트 프로그램의 작성은 담당 프로그래머 뿐 아니라 사용자도 함께 참여할 수 있다. 이렇게 면밀히 준비된 테스트는 개발의 진척도를 파악할 때, 또는 제품을 변경시키는 경우에도 유용하게 사용될 수 있고, 무엇보다도 제품의 결함을 일찍 알아낼 수 있다는 장점을 가지고 있다. 결과적으로 안정적인 제품을 만드는 데에 기여한다.
- **요구 변경 경청(listening)** : 완성될 제품에 대한 사용자의 요구 사항이나 현재 개발 중인 제품에 관한 사용자의 피드백을 자주 받는다. 생명 주기가 짧으므로 주어진 피드백에 대한 반영 속도도 빨라지게 된다.

## 3. XP의 실행 지침(Practice)

본 장에서는 XP를 따르기 위해 정의된 구체적인 실행 지침들을 살펴봄으로써 XP에 대한 보다 명확한 개념 정립을 꾀한다. 다음은 총 12가지의 지침들에 관한 간략한 설명이다.

지침 1. 개발 계획 수립

개발 계획 수립은 개발 의뢰를 맡긴 사용자 측과 프로젝트를 수행하는 개발자 측, 두 그룹이 모여 수행한다. 개발 계획 수립의 목표는 개발될 소프트웨어의 주요 배포(release) 일정을 선정하는 작업과, 각 배포에 포함된 기능을 결정하는 것이다. 계획 수립 과정에서 대부분의 주요 결정은 사용자 측이 맡고, 개발자 측은 사용자 측이 현명한 계획을 수립할 수 있도록 지원하는 역할을 맡는다.

개발 계획 과정은 먼저, 개발될 소프트웨어가 갖추어야 할 여러 가지 기능들을 사용자가 여러 장의 스토리 카드(story card, 그림 1 참조)라는 작업 카드에 기록하는 것으로 시작된다. 개발자 측에서는, 작성된 한 장의 스토리 카드 내용이 너무 방대한 경우 여러 장의 스토리 카드에 나누어 작성하도록 유도하고, 반대로 한 스토리 카드의 내용이 적은 경우에는 여러 카드의 내용을 통합하여 한 장의 카드로 작성하도록 유도한다.

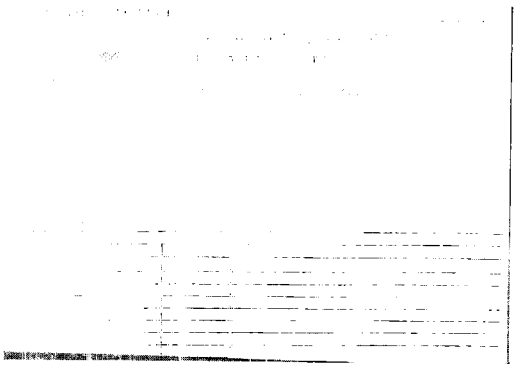


그림 1 스토리 카드의 예[1]

개발자 측의 도움으로 사용자 측이 필요한 모든 기능을 스토리 카드에 기록하면, 다음 작업으로 배포 횟수를 결정하고, 각 배포에서 구현되어야 할 스토리 카드들을 선정하게 한다. 이 작업 역시 개발자 측과의 협의 하에 수행된다.

지침 2. 짧은 배포 주기

앞서 여러 차례 언급 되었 듯이 XP에서는 사용자의 요구 사항 변화에 대처하는 것에 많은 초점을 두고 있다. 이를 위해 가치가 있다고 생각되는 최소한의 기능이 제품에 추가되면 신속히 사용자에게 제공함으로써, 사용자가 보다 자주 빨리 제품의 상태 및 기능들을 파악하고 피드백을 줄 수 있도록 한다. 일반적으로 XP에서 권장되는 배포 주기는 한 두 달 정도이다.

지침 3. 시스템 메타포(metaphor)

시스템 메타포는 최종적으로 개발되어야 할 시스템의 구조를 기술한 문장으로서, 일반적인 소프트웨어 개발 시 그려지는 시스템 아키텍처(architecture)에 해당하는 역할을 담당한다. 단, 그림 형태로 그려지는 아키텍처와는 달리, 시스템 메타포는 그 구조를 문장으로 기술하게 되어있다.

시스템 메타포는 개발자 측과 사용자 측이 서로 다른 시스템을 목표로 하여 프로젝트가 수행되는 상황을 방지하고, 양측간의 의사 교환 시 시스템 메타포에 기술된 표준 용어를 기반으로 함으로써 동일 개념에 대해 상이한 용어를 사용하는 데서 발생되는 여러 문제점을 해결하기 위해 필요하다. 뿐만 아니라, 개발 팀 내에서 시스템에 새로운 기능을 제안하는 경우에도 이러한 메타포를 근간으로 하도록 하여 원활한 의사 소통을 돕는다.

지침 4. 단순한 설계

'오늘 문제는 오늘 해결하고 내일 문제는 내일 해결하라'는 논리에 입각해서 설계를 당장 필요한 것에만 초점을 맞추어 단순화 한다. 단순화된 설계란 현재 개발자에게 꼭 필요한 모든 것들은 포함되어 있으면서도, 프로그램 수행 로직에 중복이 없으며 클래스나 메소드들을 최소화하는 설계를 말한다. 따라서, 단순 설계 지침을 반하는 요인(특히 성능 최적화 등)은 제품이 완성되어 작동되고 난 뒤로 미루도록 한다.

이로써 개발자들은 적지 않은 시간을 절약할 수 있게 된다. 실험에 따르면 향후 기능 확장을 예측하고 설계에서 고려된 추가 사항들 중, 많아야 10% 정도만 실제 추후에 필요했던 것으로 밝혀지고 있다 [3].

지침 5. 테스트

모든 프로그램은 자동화된 테스트 프로그램과 함께 개발 된다. 이러한 자동적인 테스트는 프로그램 개발 종료 여부를 명확히 알려준다는 장점이 있으며, 프로그램이 생명 주기 동안의 변화에 좀 더 잘 적응할 수 있게 하고, 코드가 보다 안정적으로 개발되는 동시에, 개발 기간 자체도 줄어드는 결과를 가져올 수 있다.

테스트에는 단위(unit) 테스트와 기능(functionality) 테스트의 두 가지가 종류가 있다. 단위 테스트는, 개발자에 의해 작성되고 코드 보다 먼저 개발된다. 객체지향 개발인 경우 각 주요 메소드 마다 하나씩 만들어지며, 100%가 수행되어야 완성된 것으로 보기 때문에 개발자 스스로가 코드에 확신을 가지게 된다는 잇점이 있다. 단위 테스트는 해당 메소드의 완성 여부 판단 자체에 사용되는 경우 외에도, 프로그램의 생명 주기 동안 진화되는 것에 따른 회귀(regression) 테스트나 여러 모듈을 합하여 전체 시스템을 구성할 때 사용되는 통합 테스트 등에도 사용된다.

기능 테스트는 적합도 테스트라고도 하며, 사용자에 의해 기안되어 구체적인 것은 개발자 측 담당자의 도움을 받아 작성된다. 스토리 카드를 기반으로 작성되며, 내부 구조를 몰라도 테스트가 되는 블랙 박스 테스트이다. 사용자들을 확신시키기 위해 존재하며, 기능 테스트가 100% 모두 만족된다는 것은 결국 개발 완료를 의미한다.

지침 6. 재구성(refactoring)

재구성이란 코드의 기능에는 전혀 지장을 주지 않고 코드를 변경하는 것을 말한다. 코드 자체를 깨끗하고 읽기 좋게 만드는 작업을 의미하므로, 앞서 언급한 설계 단순화를 위해 사용되는 가장 중요한 도구 중 하나이다. 재구성하는 시점은 주관적이긴 하나, 메소드 길이가 길어지는 등 코드가 다소 짜임새를 잃었을 때 실시하도록 한다. 재구성에 대한 보다 자세한 내용은 참고 문헌 [2] 등에서 별도 주제로 다루어지고 있다. 그림 2에서는 두 개 이상의 독립된 기능이 한 메소드 내에 존재하므로 이를 분리시키는 재구성 기법을 보여주고 있다.

지침 7. 페어 프로그래밍(pair programming)

XP의 가장 독특한 특징 중 하나는 모든 코드를 두 명의 개발자가 함께 개발하는 페어 프로그래밍을 한다는 점이다. 이로써 언제나 두 명 이상의 개발자가 설계에 참여하게 되며, 코드는 항상 재검토되므로 소프트웨어의 질적 향상을 가져오는 효과가 있다. 또, 페어 프로그래밍의 파트너는 계속 바뀌는 것을 가정하고 있으므로, 모든 개발자들이 시스템의 모든 부분에 대해 잘 알게 되고, 장기적으로 각 개발자들이 개인적으로 보유한 지식이나 기술이 전체 개발팀에 전파될 수 있다는 장점이 있다.

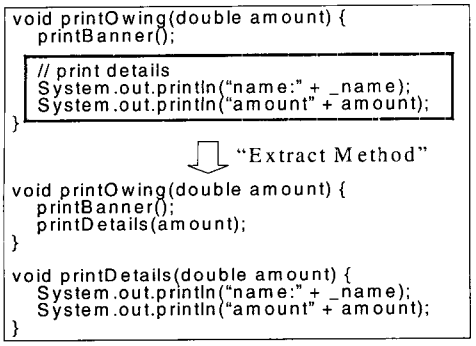



그림 2 재구성의 예[2]

개발 인원이 두 배로 들게 되므로, 직관적으로는 동일 인원이 참여 할 경우의 개발 시간이 두 배가 되는 것을 우려할 수도 있다. 그러나, 실험 결과에 따르면 페어 프로그래밍은 개발 속도도 향상시키기 때문에, 참여 개발자들이 각기 동시에 다른 프로그램을 했을 때 걸리는 시간에 비해 단지 10% 정도가 더 드는 것으로 조사되어 있다[3]. 페어 프로그래밍이 가져오는 여러 효과들을 고려할 때 큰 수치가 아니라고 볼 수 있다.

지침 8. 공동 소유/책임(collective ownership)

XP의 또 다른 특성 중 하나로 모든 개발자가 모든 코드에 권리와 책임을 가지는 공동 소유 개념이 있다. 모든 코드를 모든 개발자가 관리하게 되므로, 기능을 추가하거나 오류를 찾아 수정하는 일 및 재구성을 하는 것에 있어서도 보다 나은 결과를 얻을 수 있다.

공동 소유/책임은 모든 개발자가 모든 코드에 대해 친숙하도록 하는 것을 전제로 하므로 페어 프로그래밍을 하는 경우 유리하다. 공동 소유 개념은 작성자가 아닌 다른 사람이 임의로 코드를 수정할 수 있다는 측면에서 다소 문제가 있을 수도 있다. 그러나, 많은 경우 이에 따른 위험 및 부담은, 페어 프로그램을 통한 2 인 이상이 동의하는 경우 수정이 이루어진다는 것과 원래의 코드 작성자에 의해 만들어진 단위 테스트를 100% 통과해야 한다는 규칙에 의해 보완된다.

지침 9. 지속적인 통합(continuous integration)

통합테스트는 모든 단위 테스트를 거친 후 이루어지게 되는데, XP에서는 하루에도 여러 차례 통합을 권장한다. 그 결과 얻는 가장 큰 장점으로 통합 시 발견하는 오류를 일찍 쉽게 발견할 수 있다는 점이다. 따라서, 일반적인 타 개발 과정의 최종 통합 단계에서 흔히 나타나는 큰 실패 위험과 부담이 거의 없으며, 통합에 걸리는 시간도 전통적인 개발 단계에서와 달리 단지 몇 초만에 끝나는 경우가 많게 된다. 통합을 위한 컴퓨터는 별도로 한 대를 준비하여 사용하도록 함으로써 다수 팀이 동시에 통합하는 과정에서 발생되는 일관성 유지 문제를 예방하게 된다.

지침 10. 1주일 40시간 작업

XP는 초과 근무를 기피 대상으로 하고 있다. 대신 매일 아침 새롭고 적극적인 자세로 프로그램에 임할 수 있도록 하고, 낮에 열심히 일한 결과 밤에는 다소 피곤하나 만족스러운 기분으로 일을 마칠 것을 권하고 있다. 주말은 재충전의 시간으로 삼아 월요일에는 다시 생기 있게 일을 시작하는 것이 생산성 향상에 오히려 도움을 준다고 여기고 있다.

만일 불가피하게 장기간 초과 근무를 해야 하는 상황이 생기면, 프로젝트 진행에 심각한 문제가 있음을 반증하는 현상으로 여기고 개발 계획을 수정해야 하는 시점으로 간주한다.

지침 11. 개발 장소로 사용자 파견

실제 사용자가 개발팀 일원으로 파견될 것을 말하고 있다. 이렇게 파견된 사용자는 발자들의 질문에 답을 주거나, 개발팀 내의 분분한 의견이 있으면 바로 잡고, 작은 단위의 일들에 대해 범위나 우선 순위를 결정해 주거나, 기능 테스트를 작성하는 등의 임무를 진다. 결과적으로 사용자로부터의 최대한 빠르고 구체적인 피드백이 가능하게 된다.

지침 12. 코딩 표준

앞서 언급되었듯이 XP에서의 페어 프로그래밍은 파트너를 바꾸어 가면서 코딩을 할 것을 가정한다. 또, 시스템을 구성하는 코드는 다른 사람이 작성한 경우라도 필요에 따라 오류 수정이나 재구성을 할 수 있게 되어 있다. 이러한 상호 의사 교환의 필요성 때문에, 코딩은 공통된 표준을 따라서 할 것을 강력히 권고하고 있다. 이 때의 공통된 표준은 작성된 코드가 가져야할 단순한 포맷 설정 외에도 코딩 패턴(coding pattern)[4] 등 의미론적인 것까지 포함한다.

## 4. XP의 적용 사례 및 국내 현황

앞에서 언급했듯이 지금까지의 개발 프로세스가 주로 조직 중심의 개발 프로세스였다면 XP는 철저히 개발자 중심의 개발 프로세스이다. 본 장에서는 개발자의 관점에서 국내에서 수행되는 일반적인 프로젝트의 특징을 살펴보고, XP가 적용될 수 있는 프로젝트

유형을 찾아 본다. 그 다음 XP를 적용한 간단한 사례와 도입을 위해 필요한 것들에 대해서 알아본다.

### 4.1 [illegible]

국내에서 수행되는 소프트웨어 개발 프로젝트의 특징 중의 하나는 프로젝트 시작 단계에서는 여유가 있다가 프로젝트가 끝나갈 무렵에는 미처 테스트도 제대로 해보지 못하고 기능 구현에 급급한 경우가 많다는 것이다. 이러한 현상은 너무도 보편적이기 때문에, 그 원인을 단순히 개발자들의 프로그램 작성 수준 탓으로 돌리기는 어렵다.

그보다는 이것은 초기에 사용자의 요구 사항을 제대로 파악해 내지 못하는 데에서 오는 현상으로 해석되는 것이 적절하다. 일반적으로 사용자들은 소프트웨어 개발 프로젝트를 통해 구축하고자 하는 미래(To-be) 시스템의 모습에 대해서 지극히 추상화된 아이디어를 가지고 있기 때문에 개발팀에게 구체적이고도 정확한 요구 사항을 제시하는 경우가 드물다. 이것은 서론에서 언급된 바와 같이, 사용자가 실제로 어떤 시스템을 만들어야 할지를 정확히 모르고 있거나, 소프트웨어 시스템과 그 시스템이 만들어지는 과정에 대해서 잘 모르고 있기 때문이기도 하다.

이런 상황에서 사용자의 정확한 요구 사항을 이끌어 내기 위해서는 다양한 방법이 필요하며, 이것은 프로젝트 승패의 첫 갈림길이 될 정도로 중요한 일이기도 하다. 하지만, 일반적으로는 개발팀이 사용자에게 요구 사항을 도출하기를 강요하는 것이 전부인 경우가 대부분이다. 그러나 이 경우, 요구 사항을 명확히 제시할 책임이 있는 사용자 역시, 미래의 시스템에 대해서 명확하게 그림이 그려지지 않는 안타까운 상황에 놓일 수 밖에 없다.

이러한 문제를 해결할 수 있는 가장 가까운 방법은 프로토타입 시스템으로 볼 수 있다. 즉, 스토리보드 형식이나 실제로 실행될 수 있는 간단한 프로그램 형식 등을 빌어, 고객의 눈앞에 무엇인가를 펼쳐놓고, 사용자의 상상 속에 매우 추상적으로 자리 잡고 있는 개발될 시스템에 대한 아이디어를 구체적인 서술로 끌어내는 것은, 요구 사항 파악에 최선의 방법이 될 수 있다. 프로토타입 시스템은 사용자와 함께 보고 토의할 수 있는 대상으로서의 가치와 사용자의 피드백을 받아낼 수 있는, 그리고 검토할 수 있는 시스템으로서의 가치를 가진다. 다시 말하면 사용자의 요구사항을 가장 정확히 이끌어 낼 수 있는 방법은, 가능한 한 사용자가 시스템을 사용해 볼 기회를 많이 제공해 주고 사용자가 시스템 사용 후 제공하는 피드백을 다음 단계의 개발에 반영하는 것이다.

앞 장에서 살펴 본 바에 의하면, XP는 빠른 반복 주기를 가지고, 심지어는 하루에도 몇 차례씩 통합을 하게 된다. 이것은 곧 개발자의 입장에서 통합에 따르는 문제를 조기에 발견할 수 있다는 것을 의미한다. 마찬가지로 사용자의 입장에서는 개발되는 시스템을 사용해 볼 수 있는 기회를 보다 자주 얻고, 필요할 경우 변경 요구 사항을 낼 수 있다는 것을 의미한다. 따라서 사용자가 개발될 시스템에 대해 명확한 요구 사항을 제시하지 못할 때, XP 에서 제시하고 있는 빠른 반복 주기와 프로젝트 현장에 사용자 참여 등을 통하여 빠른 시일 내에 사용자가 원하는 시스템을 정확히 파악할 수 있게 된다.

### 4.2 국내 적용 현황 및 사례

국내 XP 도입 현황을 간단히 살펴보면, 자체 솔루션을 기획하고 개발하는 일부 소프트웨어 개발회사들이 XP를 자체 개발 방법론으로 실험적으로 도입하여, 본 글에서 언급되고 있는 XP의 여러 장점들을 얻고자 하고 있다. 특히 이 회사들은 급변하는 시장의 요구사항에 맞추어서 개발의 방향을 재빠르게 재설정할 수 있기 때문으로 풀이된다. 현재는 도입 결과로 많은 성과를 얻은 일부 중소규모 소프트웨어 개발업체와 동호인들을 중심으로 XP가 아주 느린 속도로 전파되고 있는 단계이다.

하지만 중대형 SI 회사 중심의 국내 SI 시장에서는 XP가 차지하고 들어설 틈이 아직은 없는 것으로 보인다. 이러한 프로젝트에서는 프로세스와 프로젝트 산출물이 정확하게 연관되어 있어 정형화된 산출물을 중심으로 개발 프로세스 진행하게 된다. 따라서 이러한 경우 XP를 도입하게 되면, XP의 프로세스 순발력이나 철저한 실용주의적인 관점보다는 XP에서 정의하고 있는 산출물을 일차적으로 검토하기 때문에, XP 산출물의 빈약함과 프로세스의 형식이 체계적이지 못하다는 이유로 외면을 당하는 것이 현실이다.

이런 가운데서도 중소규모 SI 프로젝트에서 XP를 전체 프로세스의 일부분에 도입을 하거나 또는 전체 프로젝트 일정이 너무 촉박할 경우 제한된 기간동안 도입하여 효과를 보고 있는 사례가 간간이 보고되고

있다. 성공적인 XP의 활용을 보고하는 프로젝트 팀의 사례를 살펴보면 다음과 같은 몇 가지 특징을 발견할 수 있다.

- 프로젝트 책임자가 XP에 대해 정확히 이해하고 강력하게 이끌음
- 프로젝트를 수행하는 개발 팀이 빠른 통합 주기를 따라 올 수 있을 정도로 기술 수준이 높음
- 페어 프로그래밍을 적절히 잘 활용하여 개발실에 활력을 주고 소스 코드의 품질을 향상 시킴
- 페어 프로그래밍을 못하고 혼자서 개발하기를 고집하는 개발자들을 적절하게 통제
- 주 40시간 근무와 같은 XP의 현실 인식에 기반을 둔 지침을 철저히 따름으로써 개발자에게 동기를 부여하고, 개발자의 생산성을 높임

국내의 프로젝트 수행 문화에서, 아직은 실용적인 목적을 위해 기존의 정형화된 개발 프로세스를 버리고 전적으로 XP를 채용하기에는 이른 감이 있다. 더우기 XP는 큰 프로젝트를 수용하지 못한다는 단점이 확대 해석되기 쉬운 면도 존재한다. 따라서 기존의 안정적인 개발 프로세스를 기반으로 XP의 좋은 점들을 하나씩 수용하는 것이 성공적인 프로젝트 수행을 위한 최선의 방법이 되리라고 본다.

또, XP는 개발자 중심의 프로세스이므로 도입을 하기 위해서는 우선 개발자를 설득하는 것이 가장 중요하다. 페어 프로그래밍과 공동 소유 개념과 같은 지침은 XP의 근간을 이루고 있는 중요한 부분인 반면, 자칫 잘못하면 자부심으로 가득 찬 개발자들에게 부정적인 인식을 심어줄 수 있게 된다. 따라서 철저한 교육과 이해, 그리고 실험적인 적용을 통해서 그 장점을 확실히 인식한 다음, 서서히 도입하는 것이 바람직하다.

## 5. 결 론

XP 는 사용자의 요구가 지속적으로 변화한다는 것을 기본 가정으로 하여 구성된, 개발자 중심의 대표적인 개발 프로세스이다. 그 특징은 다음과 같이 요약된다.

- 빠르고 구체적이고 지속적인 피드백을 사용자로부터 얻는다.
- 개발 도중에도 점진적으로 계획을 개선시켜나감으로써 생명 주기 동안 제품이 변경되거나 개선되는 경우 유리하다.
- 신축성 있는 일정 운용을 가능하게 한다.
- 개발자 대 사용자 뿐 아니라 개발자 대 개발자 간의 상호 의사 교환도 매우 중요시하며 개발 과정에서 도출되는 의견들을 개발되는 제품에 자유로이 반영한다.
- 설계 자체도 제품의 생명 주기 동안 지속적으로 반복, 보정된다.
- 엄격한 테스트를 통하여 품질을 보장한다.
- 통합 테스트를 일찍 함으로써 개발 위험성을 낮춘다.

결과적으로 XP는 소프트웨어의 품질 향상을 꾀하고 예측된 시간에 제품을 완성할 수 있도록 하며 기타 비용절감도 절감되는 효과를 가지고 있다.

XP가 보급되어 실효를 거두기 위해서는, XP를 적용하고자 하는 프로젝트 책임자의 이해 및 지도 역량과 개발자들의 기술 수준이나 XP에 대한 인식 등이 전제 조건이 된다. 현재 국내 실정으로는 철저한 교육과 이해 및 실험을 바탕으로 하는 점진적인 도입을 최선의 방법으로 보고 있다.

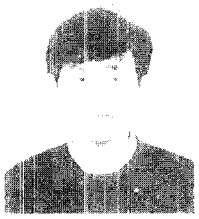
## 참고문헌

[1] Kent Beck, "eXtreme Programming Explained" – Embrace Change, Addison Wesley, 1999.

[2] Martin Fowler, "Refactoring: Improving the Design of Existing Code," Addison Wesley, 1999.

[3] http://www.extremeprogramming.org/lessons.html, "What We Have Learned About Extreme Programming".

[4] Kent Beck, "Smalltalk Best Practice Patterns", Prentice Hall ,1996.



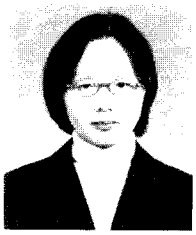
이 강 우

1991. 2 서울대학교 계산통계학과 졸업
1993. 2 서울대학교 전산과학과 석사
2000. 2 서울대학교 전산과학과 박사
2000.1 ~ 현재 한국전자통신연구원 선임연구원
E-mail:kwlee@etri.re.kr

송 태 국

삼성전자 소프트웨어 개발실 연구원
래셔널 코리아 선임 컨설턴트
PTC 코리아 책임 컨설턴트
현재 (주) 이비스온 대표이사
E mail:tsong@e bizon.com

조 은 선



1988.8 서울대학교 전산과학과 박사
1999.1~2000.4 한국과학기술원 연구원
2000.5~2002.2 아주대학교 정보통신전문대학원 조교수 대우
2002.3~현재 충북대학교 전기전자컴퓨터공학부
E mail:cmohan@hitel.net