# 소규모 프로젝트용 Rational Unified Process 사용:

# XP(eXtreme Programming) 확장

Gary Pollice

Rational Software 백서

TP 183, 3/01

Rational®
the software development company

# 목차

## *요약*

Rational Unified Process  또는 RUP  제품은 바로 사용할 수 있는 여러 인스턴스와 함께 제공되는 완전한 소프트웨어 개발 프로세스 프레임워크입니다. RUP 에서 파생된 프로세스는 제품 주기가 짧은 소규모 프로젝트의 요구를 처리하는 간단한 프로세스에서 분산된 대규모 프로젝트 팀의 폭넓은 요구를 처리하는 보다 포괄적인 프로세스에 이르기까지 다양합니다. 모든 유형 및 크기의 프로젝트는 성공적으로 RUP 를 사용했습니다. 이 백서는 소규모 프로젝트에 RUP 를 간단한 방식으로 적용하는 방법을 설명합니다. 완전한 프로젝트의 보다 광범위한 컨텍스트 내에서 XP(eXtreme Programming) 기법을 효과적으로 적용하는 방법을 설명합니다.

## *소개*

### 사례

어느날 아침 관리자가 와서 몇 주 동안 회사에서 새로 시작한 신사업에 사용할 간단한 정보 시스템을 구축할 수 있는지 물었습니다. 마침 현재 프로젝트에 싫증을 느끼고 새로운 업무를 기대하고 있었으므로 제의를 수락했습니다. 업무도 빨리 시작할 수 있었으며 기존에 일하던 대규모 조직의 관료주의나 절차상의 번거러움 없이 훌륭한 시스템을 새로 개발할 수 있었습니다.

시작은 아주 순조로웠습니다. 처음 6 개월 동안은 긴 시간이지만 대부분 즐겁게 일했습니다. 업무 생산성이 우수하여 경력에 도움이 될 훌륭한 성과도 거둘 수 있었습니다. 개발 주기도 빨라 일반적으로 몇 주 단위로 주요 시스템 파트를 새로 개발했습니다. 사용자와의 상호작용은 단순하고 직접적이어서 완벽한 팀웍을 보여주었으며 따라서 형식성이나 문서화의 필요성을 전혀 느끼지 못했습니다. 디자인 측면의 형식성도 거의 없어 코드가 곧 디자인이고 디자인이 곧 코드였습니다. 말그대로 모든 것이 완벽한 것처럼 보였습니다.

그러나 완벽한 시간은 오래 가지 못했습니다. 시스템 규모가 확장됨에 따라 작업량 또한 증가했습니다. 문제점이 변경될 때마다 기존 코드를 수정해야 했으며 수행할 작업의 개념도 정제되었습니다. 또한 개발 업무를 지원할 인력을 채용했습니다. 단일 팀으로서의 팀웍은 우수했으며 종종 두 명이 한 조가 되어 문제점 파트를 해결했습니다. 원활한 커뮤니케이션을 통해 역시 형식성의 필요성은 느끼지 못했습니다.

일 년이 지났습니다.

그 동안 계속 인력을 충원하여 팀원이 세 명, 다섯 명, 일곱 명으로 늘어났습니다. 새 인력이 투입될 때마다 상당 기간의 학습 시간이 필요했으며 해당 기간 동안에는 실무 경험을 축적할 수 없었습니다. 시간이 갈수록 전체 시스템을 이해하고 설명하기가 어려워졌으며 개요를 파악하기도 어려웠습니다. 이러한 문제를 해결하기 위해 전체 시스템 구조와 주요 개념 및 인터페이스를 형식화하여 나타내는 화이트보드 다이어그램을 캡처하기 시작했습니다.

시스템이 필요한 기능을 수행하고 있는지 확인하기 위한 기본 수단으로 계속 테스트를 사용했습니다. 또한 새로운 "사용자"가 많이 추가되면서 프로젝트 초기에 효과적이었던 비정규 요구사항 및 개인적인 관계만으로는 충분하지 않게 되었습니다. 또한 빌드할 대상을 파악하는 데도 많은 시간이 소요되었습니다. 결과적으로, 회의 내용을 계속 서면으로 작성했으며 이로 인해 결정사항을 지속적으로 상기할 필요가 없어졌습니다. 또한 요구사항 및 사용 시나리오를 설명함으로써 시스템을 새로 사용하는 사용자를 교육시키는 데 도움이 된다는 사실을 발견했습니다.

시스템 규모가 커지고 복잡도가 증가함에 따라 예상치 못한 일이 발생했고 시스템 아키텍처에 대한 특별한 주의가 필요하게 되었습니다. 초기에는 아키텍처의 윤곽을 머리 속에 그릴 수 있었지만 나중에는 약간의 정리되지 않은 참고 또는 간단한 차트로 표시했습니다. 게다가 프로젝트 구성원이 증가함에 따라 아키텍처 제어는 더 어려운 과제가 되었습니다. 처음부터 프로젝트에 참여하지 않은 사람은 아키텍처의 특정 변경에 따른 의미를 파악할 수 없었으며 따라서 시스템의 아키텍처 제한조건을 보다 정확한 용어로 정의해야 했습니다. 아키텍처에 영향을 줄 수 있는 변경사항은 팀의 동의와 궁극적으로 책임자의 승인이 필요했습니다. 이는 어려운 과정이었으며 아키텍처의 중요성을 실제로 인정하기 전에 어려운 문제가 발생하기도 했습니다.

이는 실제 사례이며 이 프로젝트의 힘든 경험 중 일부만을 설명한 것입니다. 이 사례의 특징은 구성원 중 일부가 수년 동안 프로젝트 초기에서 종료 시점까지 참여했다는 것입니다. 일반적으로 프로젝트는 인력 이동이 빈번하여 자신의 행동에 따른 다운스트림 영향을 알지 못합니다.

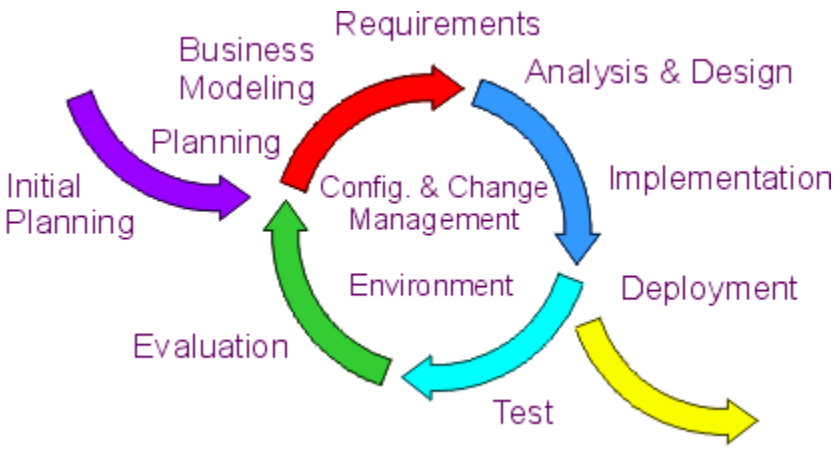
이 프로젝트는 약간의 프로세스로도 도움을 받을 수 있는 상황이었습니다. 프로세스가 너무 많은 경우에도 방해가 되지만 부족한 경우 또 다른 위험성이 있습니다. 고수익만 바라보고 위험성이 높은 주식에 투자하는 사람과 같이 프로젝트 환경의 핵심 위험성을 무시하고 프로세스를 너무 적게 사용하는 그룹은 "최대 성과를 바라면서 최악의 경우에는 대비하지 않는 것과 같습니다."

## 개요

이 문서는 앞에서 설명한 예와 같이 프로젝트에 프로세스를 적용하는 방법에 대해 설명합니다. 우선 프로세스의 "올바른 레벨"에 도달하는 데 초점을 맞춥니다. 개발 팀이 직면한 도전 과제와 해당 비즈니스 환경을 이해함으로써 프로세스의 올바른 형식성 레벨을 도출할 수 있습니다. 이러한 도전 과제를 이해하면 위험성을 완화시킬 수 있는 프로세스를 제공합니다. 어떤 경우에나 적용할 수 있는 완벽한 프로세스는 없으며, 간단한 프로세스 또는 다른 경우 모두 마찬가지 입니다. 이후 섹션은 올바른 프로세스 레벨과 위험성의 관계에 대해 검토합니다.

널리 사용되는 두 가지 방법, Rational Unified Process(RUP)와 XP(eXtreme Programming)를 사용하여 올바른 프로세스 레벨을 달성하기 위한 방법에 초점을 맞춥니다. RUP 를 소규모 프로젝트에 맞게 사용자 조정하는 방법과 XP 에서 다루지 않는 많은 영역을 RUP 에서 처리하는 방법을 보여줍니다. RUP 와 XP 를 함께 설명함으로써 프로젝트 팀에서 위험성을 완화하고 소프트웨어 제품을 제공하는 목적을 달성하는 데 필요한 안내를 얻을 수 있습니다.

RUP 는 Rational Software 에서 개발한 프로세스 프레임워크입니다. RUP 는 업계에서 입증된 여섯 가지 우수 사례를 기반으로 하는 반복적 개발 방법론입니다(RUP 부록 참조). RUP 기반 프로젝트는 시간의 경과에 따라 도입/인식(Inception), 정제(Elaboration), 구현/구축(Construction) 및 전이(Transition)의 네 단계로 수행됩니다. 각 단계에는 하나 이상의 반복이 포함됩니다. 각 반복에서는 요구사항, 분석 및 디자인, 테스트 등 여러 원칙 각각에 다양한 정도의 노력을 투자합니다. RUP 의 핵심 구동 요소는 *위험성 완화*입니다. RUP 는 수천 명의 Rational 고객 및 파트너가 참여한 수천 가지 프로젝트를 통해 정제되었습니다. 다음 다이어그램은 일반 반복의 플로우 예를 보여줍니다.




**일반 반복 플로우**

위험성이 프로세스에 미치는 영향을 설명하기 위해, 비즈니스를 모델링해야 하는지 여부를 물을 수 있습니다. 비즈니스를 이해하지 못해 잘못된 시스템을 빌드하게 될 심각한 위험성이 존재하는 경우 일반적으로 비즈니스

모델링을 수행해야 합니다. 모델링 노력을 형식화해야 하는지 여부는 프로젝트 관계자에 따라 결정됩니다. 즉, 소규모 팀에서 해당 결과를 비공식적으로 사용하는 경우 약간의 비공식 자료만 작성하면 됩니다. 다른 조직 구성원이 결과를 사용하거나 검토하려는 경우 일반적으로 약간의 노력을 더 기울여 프리젠테이션의 정확성과 이해 용이성에 보다 초점을 맞추어야 합니다.

RUP 는 거의 모든 프로젝트의 요구에 맞게 사용자 정의할 수 있습니다. 바로 사용할 수 있는 프로세스 또는 로드맵이 사용자의 특정 요구에 맞지 않는 경우 사용자가 직접 쉽게 로드맵을 생성할 수 있습니다. 로드맵은 프로젝트에서 프로세스 사용 계획을 수립하는 방법을 설명하며 해당 프로젝트의 특정 프로세스 인스턴스를 나타냅니다. 즉, RUP 는 필요에 따라 간단하거나 복잡할 수 있으며 해당 내용은 이 문서에서 설명합니다.

XP 는 소규모 프로젝트에 적합한 간단한 코드 중심 프로세스입니다(XP 부록 참조). XP 는 Kent Beck 이 개발한 기법으로서 1997 년 Chrysler Corporation 의 C3 급여 프로젝트에서 소프트웨어 업계의 주목을 받기 시작했습니다. XP 는 RUP 와 같이 소규모 릴리스, 단순 디자인, 테스트 및 지속적 통합과 같은 여러 사례를 구현하는 반복을 기반으로 합니다. XP 는 해당 프로젝트 및 상황에 효과적인 여러 기법을 사용하지만 숨겨진 가정, 활동 및 역할이 있습니다.

RUP 와 XP 는 서로 근본 철학이 다릅니다. RUP 는 특정 소프트웨어 프로젝트에 적용할 수 있는 프로세스 컴포넌트, 메소드 및 기법의 프레임워크로서, 사용자가 직접 특수화할 수 있습니다. 반면에 XP 는 보다 제한적인 프로세스로서 완전한 개발 프로젝트를 위해서는 추가사항이 필요합니다. 이러한 차이를 통해 소프트웨어 개발 커뮤니티의 전반적인 인식을 파악할 수 있습니다. 즉, RUP 는 대규모 시스템의 문제점을 해결할 수 있으며 XP 는 소규모 시스템의 문제점을 해결할 수 있다는 것입니다. 경험상으로 볼 때 대부분의 소프트웨어 프로젝트에서는 해당 상황에 적합한 프로세스 레벨을 달성하기 위해 노력하지만 그렇다고 프로세스에만 집중하는 것은 아닙니다.

RUP 범위에 일부 XP 기법을 결합함으로써 모든 프로젝트 구성원이 긍정하는 올바른 프로세스 규모를 달성하고 주요 프로젝트 위험성을 모두 처리할 수 있습니다. 상대적으로 높은 신뢰 수준의 환경에서 작업하는 소규모 프로젝트 팀의 경우, 사용자가 팀의 핵심 파트이므로 XP 가 효과적입니다. 그러나 팀이 분산되고 코드 베이스가 증가하거나 아키텍처가 올바르게 정의되지 않은 경우 다른 조치가 필요합니다. 즉, 사용자와 "계약" 스타일로 상호작용하는 프로젝트의 경우 XP 만으로는 부족합니다. RUP 는 필요 시 보다 견고한 기법 세트로 XP 를 확장할 수 있는 프레임워크입니다.

이 문서의 나머지 부분에서는 RUP 의 네 가지 단계를 기반으로 하는 소규모 프로세스에 대해 설명합니다. 각 단계별로, 생성된 활동과 아티팩트를 식별합니다.[1] RUP 와 XP 는 다른 역할과 책임을 식별하지만 이 차이점은 이 문서에서 다루지 않습니다. 어떤 조직 또는 프로젝트에서나 실제 프로젝트 구성원은 프로세스의 올바른 역할과 연관되어야 합니다.

## *프로젝트 시작 — 도입/인식(Inception)*

도입/인식은 새 개발 노력에 있어 중요한 단계입니다. 이러한 경우 중요한 비즈니스 및 요구사항 위험성을 처리해야 프로젝트를 진행할 수 있습니다. 기존 시스템의 개선에 중점을 두는 프로젝트의 경우 도입/인식 단계가 더 짧지만 이 때도 역시 프로젝트의 수행 가치 및 가능성을 확인하는 데 초점을 맞추어야 합니다.

도입/인식 단계에서는 소프트웨어 빌드를 위한 비즈니스 사례를 작성합니다. *비전*은 도입/인식 단계에서 생성되는 핵심 아티팩트로서, 시스템에 대한 상위 레벨 설명입니다. 비전에는 해당 시스템의 정의, 시스템 사용자, 시스템의 사용 이유, 필수 기능 및 제한조건이 명시됩니다. *비전*은 일반적으로 하나 또는 두 개의 단락으로 짧게 작성합니다. *비전*에는 소프트웨어가 고객에게 제공해야 하는 중요 기능이 포함되기도 합니다.

다음은 Rational 외부 웹 사이트를 리엔지니어링하는 프로젝트에 대해 작성된 간단한 *비전* 예제입니다.

> *Rational 은 방문자에게 셀프 서비스, 지원 및 대상에 맞는 컨텐츠를 제공하는 개인화된 동적 웹 구현을 통해 고객 관계를 강화함으로써 전세계 e-development(도구, 서비스 및 우수 사례) 분야의 선두 기업이*

[1] XP 는 탐색(Exploration), 확약(Commitment) 및 조정(Steering)의 세 가지 단계를 정의합니다. 이 단계는 RUP 단계에 정확히 맵핑되지 않으므로 네 가지 RUP 단계를 사용하여 프로세스를 설명합니다.

*됩니다. 새 프로세스와 우수한 기법을 적용하여 컨텐츠 제공자가 양질의 컨텐츠를 신속하게 제공할 수 있는 간편하고 자동화된 솔루션을 제공합니다.*

RUP 에 지정된 네 가지 필수 도입/인식(Inception) 활동은 다음과 같습니다.

- **프로젝트의 범위 공식화** — 시스템을 생성하는 경우 시스템의 정의와 이해 당사자(stakeholder)를 만족시킬 수 있는 방법을 정확히 이해해야 합니다. 이 활동에서는 컨텍스트와 가장 중요한 요구사항을 자세히 캡처하여 제품의 적합성 기준을 도출합니다.
- **비즈니스 사례 계획 및 준비** — *비전*을 토대로 위험성 완화 전략을 정의하고 초기 프로젝트 계획을 개발하며 알려진 비용, 스케줄 및 수익성의 절충을 식별합니다.
- **후보 아키텍처 종합** —고려 대상 시스템이 기존 시스템과 유사하거나 아키텍처를 쉽게 파악할 수 있는 경우 이 단계를 건너뛸 수 있습니다. 고객의 요구사항을 파악하는 즉시 잠재적인 후보 아키텍처를 조사하기 위한 시간을 할당합니다. 신기술을 채택하는 경우 소프트웨어 문제점에 대한 새로운 솔루션 또는 개선된 솔루션을 제공할 수 있습니다. 프로세스 초기에 기술을 선택하고 초기 프로토타입을 개발하는 것 이외에 구매와 빌드 간의 절충을 평가함으로써 프로젝트의 주요 위험성을 줄일 수 있습니다.
- **프로젝트 환경 준비** — 모든 프로젝트에는 프로젝트 환경이 필요합니다. 짝 프로그래밍과 같은 XP 기법 또는 보다 전통적인 기법 사용 여부에 관계 없이 실제 자원, 소프트웨어 도구 및 팀에서 수행하는 프로시저를 결정해야 합니다.

소규모 프로젝트의 경우 도입/인식(Inception)을 수행하기 위한 "프로세스 시간"은 그리 길지 않습니다. 도입/인식 단계는 일반적으로 이틀 안에 완료할 수 있습니다. 다음 섹션은 이 단계의 예상 아티팩트와 *비전*에 대해 설명합니다.

### 승인된 비즈니스 사례

이해 당사자(stakeholder)에게는 비즈니스 관점에서 프로젝트의 유효성에 동의할 수 있는 기회가 제공됩니다. RUP 와 XP 모두 잘못된 프로젝트에 가치있는 자원을 낭비하는 것보다 프로젝트의 유효성을 조기에 판단하는 것이 바람직하다는 데 동의합니다. XP(*Planning Extreme Programming 참조*)[1]의 경우 프로젝트 구현 방법과 필요한 역할에 대한 정의(기존 비즈니스 또는 시스템 컨텍스트에 가장 명확하게 정의)가 명확하지 않지만 탐색(Exploration) 단계에서 RUP 의 해당 도입/인식(Inception) 아티팩트를 처리합니다.

XP 와 같이 비즈니스 문제를 비공식적으로 처리하거나, RUP 와 같이 비즈니스 사례를 최우선 프로젝트 아티팩트로 작성하는지에 관계없이 이를 고려해야 합니다.

### 위험성 목록

프로젝트 전체에서 위험성 목록을 유지보수합니다. 이는 완화 전략 계획이 포함된 간단한 위험성 목록일 수 있습니다. 각 위험성에는 우선순위가 부여됩니다. 프로젝트 관련자는 누구나 원하는 시점에 위험성 처리 방법과 해당 위험성을 확인할 수 있습니다. Kent Beck 은 XP 에서 처리하는 위험성 세트와 해당 처리 방법을 설명하지만 위험성 처리를 위한 일반적인 접근 방식은 제공하지 않습니다.[2]

---

[1]현재 XP(eXtreme Programming)에 대해 출간되어 있는 세 권의 서적 중 하나입니다.

[2] Kent Beck 은 *eXtreme Programming eXplained* 에서 여덟 가지 위험성과 XP 의 위험성 처리 방법에 대해 설명하고 있습니다(페이지 3 – 5). 독자는 이 위험성 목록을 확인하고 적합성 여부에 대한 피드백을 제공할 수 있습니다. 이 밖에도 여러 가지 다른 위험성이 존재할 수 있으며 모든 프로세스에는 일반적인 위험성 처리 전략이 필요합니다.

### 예비 프로젝트 계획

이 계획에는 자원 예상, 범위 및 단계 계획이 포함됩니다. 이러한 예상은 모든 프로젝트에서 계속 변경되므로 지속적인 모니터링이 필요합니다.

### 프로젝트 적합성 계획

정규 계획의 존재 여부는 프로젝트 유형에 따라 다릅니다. 고객이 프로젝트 성공을 평가하는 방법을 결정해야 하며 XP 프로젝트의 경우, 이는 고객이 작성하는 적합성 테스트 양식을 취합니다. 보다 일반적인 프로세스의 경우, 고객이 실제로 테스트를 구성하지는 않더라도 적합성 기준은 고객이 직접 또는 시스템 분석가와 같이 고객과 직접 상호작용하는 역할을 통해 적용해야 합니다. 일반 사용자 문서 및 도움말 생성과 같이 XP 에서 다루지 않는 다른 적합성 기준도 있습니다.

### 초기 정제(Elaboration) 반복 계획

RUP 기반 프로젝트에서는 이전 반복 종료 시점에서 각 반복에 대한 세부 계획을 수립합니다. 반복 종료 시점에서는 해당 반복 시작 시 지정된 기준에 따라 진행상태를 평가합니다. XP 는 반복 성공을 효과적으로 모니터하고 측정할 수 있는 몇 가지 기법을 제공합니다. 메트릭은 간단하여 반복 계획 및 평가 기준에 쉽게 통합할 수 있습니다.

## *정제(Elaboration)*

정제 단계의 목적은 구현/구축(Construction) 단계에서의 대부분의 디자인 및 구현 노력에 대한 안정적인 기초를 제공할 수 있도록 시스템 아키텍처의 기준선을 작성하는 것입니다. 아키텍처는 가장 중요한 요구사항(시스템의 아키텍처에 많은 영향을 미치는 사항)에 대한 고려사항 및 위험성 평가를 통해 발전합니다. 아키텍처의 안정성은 하나 이상의 아키텍처 프로토타입을 통해 평가됩니다.

RUP 의 디자인 활동은 시스템 아키텍처와 소프트웨어 아키텍처(소프트웨어 집약 시스템의 경우) 개념을 중심으로 수행됩니다. 컴포넌트 아키텍처 사용은 RUP 에서 제공하는 여섯 가지 소프트웨어 개발 우수 사례 중 하나입니다. RUP 에서는 아키텍처 개발 및 유지보수에 시간을 할애하도록 권장합니다. 이러한 노력에 시간을 할애함으로써, 불안정하고 경직된 시스템과 관련된 위험성을 완화시킬 수 있습니다.

XP 에서는 아키텍처 대신 "메타포"라는 개념을 사용합니다. 메타포는 아키텍처의 특정 파트를 캡처하며 나머지 아키텍처 파트는 코드 개발에 따라 발전됩니다. XP 에서는 가장 간단한 디자인을 생성한 후 지속적으로 코드를 리팩토링함으로써 아키텍처가 생성되는 것으로 가정합니다.

RUP 의 아키텍처는 메타포 이상의 개념입니다. 정제(Elaboration)에서 실행 가능한 아키텍처를 구성하며 이러한 아키텍처를 통해 성능, 신뢰성 및 견고성과 같은 비기능적 요구사항 충족과 연관된 많은 위험성을 줄일 수 있습니다. XP 관련 서적을 통해 RUP 에서의 정제 단계에 대한 일부 규정 내용(예: XP 에서 말하는 하부 구조에 대한 과도한 집중)이 불필요한 노력임을 추론할 수 있습니다. XP 에서는 하부 구조 빌드 노력을 너무 일찍 수행하는 경우 솔루션이 지나치게 복잡해지고 고객에게 아무런 가치도 제공하지 않는 솔루션이 생성된다고 주장합니다. RUP 에서는 아키텍처와 하부 구조가 서로 다른 개념입니다.

RUP 와 XP 의 아키텍처에 대한 접근 방식은 매우 다릅니다. RUP 에서는 아키텍처에 집중함으로써 시간 경과에 따른 범위 확장, 프로젝트 규모 확대 및 신기술 추가와 연관된 위험성을 예방할 수 있다고 조언합니다. XP 에서는 기존 아키텍처를 가정하거나, 아키텍처가 단순하거나 쉽게 이해할 수 있어 코드와 함께 발전할 수 있는 것으로 가정합니다. 또한 XP 에서는 내일을 위한 디자인이 아닌 오늘을 위한 구현을 강조합니다. 즉, 디자인을 최대한 단순하게 유지함으로써 미래에 문제가 발생하지 않는다는 것입니다. RUP 에서는 이와 같은 제안에 따른 위험성을 평가하도록 권장합니다. XP 에서는 시스템 전체 또는 그 일부를 미래에 다시 작성하는 것이 가능성을 전제로 미리 계획을 수립하는 것보다 효율적이며 비용도 적게 소요된다고 주장합니다. 일부 시스템의 경우에는 이러한 주장이 적용되지만 RUP 를 사용하는 경우 정제(Elaboration) 단계에서 위험성을 고려함으로써 이러한

결론을 얻을 수 있습니다. RUP 에는 이를 모든 시스템에 적용하지 않으며 경험상으로 볼 때 복잡한 대규모 시스템과 완전히 새로운 시스템의 경우 심각한 문제가 발생할 수 있습니다.

발생하지 않을 수도 있는 향후의 가능성에 대해 지나치게 집착하는 것은 낭비이며 미래에 대한 적절하면서도 신중한 대비 자세가 필요합니다. 코드를 계속 다시 작성하거나 리팩토링할 수 있는 회사는 많지 않습니다.

모든 프로젝트의 정제(Elaboration) 단계에서는 최소한 다음 세 가지 활동을 수행해야 합니다.

- **아키텍처의 정의, 유효성 검증 및 기준선 작성** — 위험성 목록을 사용하여 도입/인식(Inception) 단계에서 후보 아키텍처를 개발합니다. 또한 해당 소프트웨어의 실현 가능성 여부를 확인해야 합니다. 선택한 기술이 기존 기술과 유사하거나 시스템이 복잡하지 않은 경우 이 타스크를 수행하는 데 많은 시간이 소요되지 않습니다. 기존 시스템에 추가할 때 기존 아키텍처를 변경하지 않아도 되는 경우 이 타스크가 필요하지 않습니다. 아키텍처 관련 위험성이 실제로 존재하는 경우, 아키텍처가 변경되지 않도록 합니다.

  이 활동의 일부로서, 특정 컴포넌트를 선택하고 구입/빌드/재사용 결정을 할 수 있습니다. 이와 관련하여 많은 노력이 필요한 경우, 별도 활동으로 분할할 수 있습니다.

- **비전 정제** — 도입/인식 단계에서 비전을 개발했습니다. 프로젝트의 실현 가능성을 판별할 때 이해 당사자(stakeholder)가 시스템을 검토하고 의견을 표시할 경우 비전 문서 및 요구사항이 변경될 수 있습니다. 비전 및 요구사항 개정은 일반적으로 정제 단계에서 발생합니다. 정제 종료 시점에서는 아키텍처 및 계획 결정사항과 관련된 가장 중요한 유스 케이스를 완벽히 이해해야 합니다. 이해 당사자는 현재 아키텍처 컨텍스트에서 완전한 시스템 개발을 위한 현재 계획을 실행함으로써 현재 비전을 달성할 수 있음에 동의해야 합니다. 후속 반복에서는 변경 작업이 감소해야 하지만 요구사항 관리를 위해 각 반복에 일정 시간을 할당할 수 있습니다.

- **구현/구축(Construction) 단계의 반복 계획 작성 및 기준선 작성** —계획 세부사항을 작성합니다. 각 구현/구축 반복의 종료 시점에서 계획을 재검토하고 필요에 따라 조정합니다. 조정 작업은 일반적으로 노력을 잘못 예상했거나 비즈니스 환경 또는 요구사항이 변경된 경우 수행됩니다. 유스 케이스, 시나리오 및 기술 노력의 우선순위를 결정한 후 반복에 지정합니다. 각 반복의 종료 시점에서는 이해 당사자에게 가치를 제공하는 중간 산출물 계획을 수립합니다.

정제(Elaboration) 단계에서 이 밖에 다른 활동을 수행할 수도 있습니다. 예를 들어, 테스트 환경을 구축하거나 테스트 개발을 시작하도록 권장합니다. 세부 코드가 없는 경우라도 통합 테스트를 디자인하고 구현할 수 있습니다. 프로그래머는 유닛 테스트 개발 준비를 완료한 상태에서 프로젝트에 선택된 테스트 도구 사용 방법을 숙지해야 합니다. XP 에서는 코드보다 먼저 테스트를 작성하도록 권장합니다. 이는 특히 기존 코드에 추가할 때 바람직한 방법입니다. 테스트 수행 방법에 관계 없이 정제 단계에서 정규 테스트 계획을 수립합니다.

RUP 에서 설명하는 정제(Elaboration) 단계에는 XP 의 탐색(Exploration) 및 확약(Commitment) 단계의 요소가 포함됩니다. 기술적 위험성(새로운 시스템, 복잡도) 처리를 위한 XP 의 접근 방식은 노력을 예상하기 위해 일정 실험 시간을 할애하는 "스파이크" 솔루션입니다. 이 기법은 여러 경우에 효과적인 기법으로서, 단일 유스 케이스 또는 스토리에 큰 위험성이 없는 경우 시스템 성공 및 정확한 노력 예상을 위한 약간의 노력이 더 필요합니다.

정제(Elaboration) 단계에서는 일반적으로 요구사항 및 위험성 목록과 같은 도입/인식(Inception) 단계의 아티팩트를 갱신합니다. 정제 단계에서 나타날 수 있는 아티팩트는 다음과 같습니다.

- **소프트웨어 아키텍처 문서(SAD)** — SAD 는 프로젝트 전체에서 기술 정보의 단일 소스를 제공하는 컴포지트 아티팩트입니다. 정제 종료 시점에서는 구조적으로 중요한 유스 케이스에 대한 세부 설명과 핵심 메커니즘 및 디자인 요소에 대한 식별 정보가 포함됩니다. 프로젝트를 통해 기존 시스템을 강화하는 경우 이전 SAD 를 사용하거나, 문서를 작성하지 않는 데 따른 위험성이 거의 없는 것으로 판단할 수 있습니다. 모든 경우에 있어, 문서 작성을 위한 사고 프로세스를 수행해야 합니다.

- **구현/구축(Construction) 반복 계획** —정제 단계의 구현/구축 반복 횟수를 계획합니다. 각 반복에는 특정 유스 케이스, 시나리오 및 해당 반복에 지정된 기타 작업 항목이 있습니다. 이 정보는 반복 계획에서 캡처하고 기준선을 작성합니다. 정제 종료 기준의 일부로 계획을 검토하고 승인합니다.

간단한 소규모 프로젝트의 경우, 정제(Elaboration) 반복과 도입/인식(Inception) 및 구현/구축(Construction)을 병합할 수 있습니다. 필수 활동은 계속 수행되지만 반복 계획 및 검토에 필요한 자원은 감소합니다.

### 초기 유스 케이스 모델

이는 매우 형식적이고 부담감을 느낄 수도 있지만 상당히 간단합니다. 유스 케이스는 XP 에서 고객이 작성하는 "스토리"에 해당됩니다. 차이점은 유스 케이스가 실제 값을 제공하는 시스템 외부의 관계자, 요소 또는 액터에 의해 시작된 전체 조치 세트라는 것입니다. 유스 케이스에는 여러 XP 스토리가 포함될 수 있습니다. RUP 에서는 프로젝트 범위를 정의하기 위해 도입/인식(Inception) 단계에서 유스 케이스 및 액터를 식별하도록 권장합니다. 사용자의 관점에서 전체 조치 세트에 초점을 맞춤으로써 값을 제공하는 파트로 시스템을 분할할 수 있습니다. 또한 이를 통해 적합한 구현 기능을 결정함으로써 각 반복 종료 시점에 고객에게 전달할 수 있는 결과물이 생성됩니다(초기 도입/인식 및 정제(Elaboration) 반복 제외).

RUP 및 XP 모두 전체 시스템 중 80%가 완성된 상태에 있지 않음을 확인할 수 있지만 인도물 형태로 완성된 것은 없습니다. 고객에게 가치를 제공할 수 있는 시스템을 릴리스해야 합니다.

유스 케이스 모델은 이 시점에서 지원 세부사항이 거의 또는 전혀 없는 상태에서 유스 케이스 및 액터를 식별합니다. 유스 케이스 모델은 손으로 직접 그리거나 그리기 도구로 작성한 UML(Unified Modeling Language) 다이어그램 또는 간단한 텍스트입니다. 이 모델을 통해 이해 당사자(stakeholder)에게 필요한 올바른 기능이 포함되었는지, 누락된 기능은 없는지 확인할 수 있으며 또한 전체 시스템을 쉽게 파악할 수 있습니다. 유스 케이스는 위험성, 고객에 대한 중요성 및 기술적 난이도와 같은 여러 요인을 기반으로 우선순위가 결정됩니다.

도입/인식(Inception) 단계의 아티팩트는 지나치게 형식적이거나 대규모가 아니어도 됩니다. 필요에 따라 단순하게 또는 정규 아티팩트로 작성할 수 있습니다. XP 는 계획 및 시스템 적합성에 대한 안내를 제공하며 RUP 는 프로젝트 초기에서 약간의 내용을 더 추가합니다. RUP 의 경우 이를 통해 보다 완벽한 위험성 세트를 처리할 수 있습니다.

## *구현/구축(Construction)*

구현/구축의 목적은 시스템 개발을 완료하는 것입니다. 구현/구축 단계는 어떤 의미에서 제조 프로세스와 같습니다. 즉, 자원 관리와, 비용, 스케줄 및 품질 최적화 오퍼레이션 제어를 강조합니다. 이런 의미에서 볼 때 관리 방식은 도입/인식 및 정제 단계에서의 지적 자산 개발에서 구현/구축 및 전이(Transition) 단계의 배치 가능한 제품 개발로 전이됩니다.

XP 는 구현/구축(Construction)을 특히 강조합니다. 구현/구축 단계에서는 코드를 생성합니다. XP 단계는 계획을 목적으로 하지만 XP 의 초점은 코드 빌드에 있습니다.

각 구현/구축(Construction) 반복에는 다음과 같은 세 가지 필수 활동이 있습니다.

- **자원 관리 및 프로세스 제어** — 모든 구성원이 자신이 앞으로 수행할 작업과 해당 시기를 이해해야 합니다. 워크로드가 자신의 능력을 초과하지 않도록 해야 하며 스케줄에 따라 작업이 진행되어야 합니다.
- **컴포넌트 개발 및 테스트** — 유스 케이스, 시나리오 및 기타 반복 기능성을 충족시키는 데 필요한 컴포넌트를 빌드합니다. 해당 컴포넌트는 유닛 및 통합 테스트를 통해 테스트합니다.
- **반복 평가** — 반복이 완료되면 반복 목적의 달성 여부를 판별해야 합니다. 목적을 달성하지 못한 경우 우선순위를 다시 결정하고 전달 날짜에 맞게 해당 범위를 관리해야 합니다.

시스템 유형에 따라 다른 기법이 필요합니다. RUP 는 소프트웨어 엔지니어가 올바른 컴포넌트를 빌드할 수 있도록 다른 가이드라인을 제공합니다. 요구사항은 유스 케이스 및 보충(비기능적) 요구사항과 같은 양식으로 엔지니어가 작업을 수행할 수 있도록 세부적으로 작성됩니다. RUP 의 여러 활동이 다양한 유형의 컴포넌트 디자인, 구현 및 테스트를 위한 안내를 제공합니다. 숙련된 소프트웨어 엔지니어는 이러한 활동을 자세히

살펴보지 않아도 되지만 경험이 부족한 엔지니어는 우수 사례에 대한 많은 정보를 얻을 수 있습니다. 각 팀 구성원의 프로세스 기여도는 필요에 따라 다릅니다. 그러나 구성원 모두 단일 프로세스 지식 기반을 참조합니다.

XP 에서는 스토리를 통해 구현이 수행됩니다. *Extreme Programming Installed*(Jeffries 외 공동 집필)에서는 스토리를 프로그래머와의 "대화 약속"으로 표현합니다.[1] 올바른 커뮤니케이션은 효과적이면서도 지속적인 커뮤니케이션입니다. 설명해야 할 세부사항은 항상 존재하지만 프로그래머가 해당 작업을 대부분 수행할 수 있을 정도로 스토리가 세부화되지 않으면 작업을 수행할 수 없습니다. 유스 케이스는 프로그래머가 구현할 수 있을 정도로 세부화되어야 합니다. 많은 경우에 있어 프로그래머가 유스 케이스의 기술 세부사항을 작성하는 데 도움을 줄 수 있습니다. Jeffries 및 다른 저자는 또한 대화가 문서화되어 스토리에 첨부되는 것으로 제안합니다. RUP 또한 유스 케이스 스펙의 경우를 제외하고는 이 제안에 동의합니다. 유스 케이스 스펙은 필요에 따라 비정규 형태를 나타낼 수 있습니다. 대화 결과물의 캡처 및 관리는 관리 대상 타스크입니다.

XP 의 핵심은 구현/구축(Construction)입니다. 대부분의 팀에 유용한 중요 정보와 안내가 제공됩니다. 가장 주목할만한 XP 사례는 다음과 같습니다.

- **테스트** — 프로그래머가 해당 코드 실행을 위한 테스트를 지속적으로 작성합니다. 테스트에는 스토리가 반영됩니다. XP 에서는 테스트를 먼저 작성해야 합니다. 이러한 경우 스토리를 완벽하게 이해하고 필요에 따라 많은 질문을 해야 하므로 효과적인 방법입니다. 테스트는 코드와의 순서에 관계 없이 반드시 작성해야 합니다. 작성한 테스트는 테스트 스위트에 추가하고 코드가 변경될 때마다 실행해야 합니다.
- **리팩토링** — 해당 동작은 변경하지 않고 시스템을 지속적으로 재구성함으로써 시스템을 단순하게 작성하거나 유연성을 추가합니다. 이 사례가 해당 팀에 적합한 사례인지 판단해야 합니다. 특정 팀에 간단한 작업이 다른 팀에게는 복잡한 작업이 될 수 있기 때문입니다. 예를 들어, 같은 프로젝트를 수행하는 두 명의 유능한 엔지니어가 매일 저녁 또 다른 엔지니어의 복잡한 코드를 재작성하는 경우를 가정할 수 있습니다. 부산물로서, 이들은 팀의 나머지 구성원을 위해 다음날이면 또 빌드를 해체합니다. 이러한 테스트가 도움이 되기는 하지만 과도한 코드 작업이 팀에게는 더 부담이 될 수 있습니다.
- **짝 프로그래밍** — XP 에서는 짝 프로그래밍으로 짧은 시간 안에 보다 우수한 코드를 생성할 수 있다고 주장합니다. 실제 예도 있습니다.[2] 이 사례를 구현하는 경우 여러 가지 인적 요인과 환경 요인을 고려해야 합니다. 즉, 프로그래머가 해당 작업을 기꺼이 수행하려고 하는지, 두 명의 프로그래머가 단일 워크스테이션에서 효과적으로 작업을 수행할 수 있는 공간과 같은 실제 환경이 제공되는지, 두 명의 프로그래머가 무선 통신을 사용하거나 다른 위치에서 작업을 수행하는 경우 필요한 조치 등을 고려해야 합니다.
- **지속적 통합** — 시스템을 자주 통합하고 빌드합니다(일반적으로 하루에 두 번 이상). 이 방법을 사용하는 경우 코드의 구조적 무결성을 보장할 수 있으며 통합 테스트를 통해 효과적이고 지속적인 모니터링 작업을 수행할 수 있습니다.
- **공동 소유권** — 누구나 언제라도 원하는 코드를 변경할 수 있습니다. XP 에서는 올바른 유닛 테스트 세트로 이 사례의 위험성을 줄일 수 있다고 주장합니다. 모든 사람이 모든 코드의 수정 권한을 갖는 데 따른 효과를 얻을 수 있는 한계(1 만 개, 2 만 개, 5 만 개 미만의 코드 행 등)가 있습니다.
- **단순 디자인** — 리팩토링에서와 같이, 시스템 디자인을 지속적으로 수정함으로써 복잡도를 제거할 수 있습니다. 이 사례 역시 정상적으로 작업이 가능한 작업 규모를 결정해야 합니다. 정제(Elaboration) 단계에서 아키텍처 디자인에 시간을 할애하면 단순 디자인에서 시작하여 곧 안정화됩니다.
- **코딩 표준** — 항상 올바른 사례입니다. 모든 사람이 사용에 동의하는 표준이 올바른 표준입니다.

RUP 와 XP 모두 반복을 관리(또는 조정)해야 하는 데 동의합니다. 메트릭은 팀에 가장 적합한 선택을 하는 데 유용한 올바른 계획 정보를 제공할 수 있습니다. 시간, 크기 및 결함을 측정해야 하며 이를 통해 모든 유형의

[1] 이 설명은 Allistair Cockburn 의 정의입니다.

[2] *Strengthening the Case for Pair Programming*(IEEE Software, 2000 년 7 월/8 월).

유용한 통계 정보를 얻을 수 있습니다. XP 는 진행상태를 판별하고 성과를 예측하기 위해 사용하는 단순 메트릭을 제공합니다. 이러한 메트릭은 완료된 스토리 수, 패스한 테스트 수 및 이러한 통계의 동향에 집중합니다. 많은 데이터를 참조한다고 프로젝트 성공 가능성이 높아지는 것은 아니므로 XP 에서는 최소 메트릭을 사용합니다. RUP 는 측정 대상 및 측정 방법에 대한 가이드라인과 메트릭 예제를 제공합니다. 모든 경우에 있어 메트릭은 단순하고 객관적이며 수집하기 쉽고 해석하기 쉬우며 오해할 여지가 적어야 합니다.

구현/구축(Construction) 반복에서는 전반부 또는 후반부 여부에 따라 다음 아티팩트를 작성할 수 있습니다.

- **컴포넌트** — 컴포넌트는 소프트웨어 코드(소스, 2 진 또는 실행 파일), 또는 정보(예: 시작 파일 또는 Read Me 파일)를 포함하는 파일을 나타냅니다. 컴포넌트는 여러 실행 파일을 구성하는 응용프로그램과 같은 다른 컴포넌트의 집계일 수도 있습니다.
- **훈련 자료** — 시스템에 강력한 사용자 인터페이스 측면이 있는 경우, 유스 케이스를 기반으로 가능한 빨리 사용자 매뉴얼 및 기타 훈련 자료의 예비 초안을 작성합니다.
- **배치 계획** — 고객에게 시스템이 필요합니다. 배치 계획은 제품을 설치, 테스트하여 사용자 커뮤니티로 효과적으로 전이하는 데 필요한 타스크 세트에 대해 설명합니다. 웹 중심 시스템의 경우, 배치 계획이 보다 중요합니다.
- **전이 단계 반복 계획** — 소프트웨어를 사용자에게 배치하기 전에 전이 단계 반복 계획을 완료하고 검토합니다.

### 코드와 디자인

RUP 와 XP 는 아키텍처상의 차이점 이외에 다른 차이점도 있습니다. 그 중에 하나가 디자인 정보를 제공하는 방식입니다. XP 는 코드와 디자인을 동일하게 여깁니다. 코드는 항상 코드 자체와 일치합니다. 코드 이외에, 디자인을 캡처하여 알리기 위한 다른 노력에는 많은 시간이 필요합니다. 다음은 이러한 사실을 잘 나타내는 한 사례입니다.

한 엔지니어가 디자인이 코드에 포함되어 있어 코드에서만 디자인 정보를 얻을 수 있는 두 소프트웨어 프로젝트를 수행했습니다. 이 두 가지 프로젝트는 모두 컴파일러 관련 프로젝트로서, 하나는 Ada 컴파일러의 최적화 프로그램을 유지보수하는 프로젝트였고 다른 하나는 컴파일러의 프론트 엔드를 새 플랫폼에 연결하고 써드파티 코드 생성기를 링크하는 프로젝트였습니다.

컴파일러 기술은 복잡하지만 잘 알려져 있는 기술입니다. 엔지니어는 두 가지 프로젝트에서 모두 컴파일러(또는 최적화 프로그램)의 디자인 및 구현을 전체적으로 파악하려고 했습니다. 각 프로젝트 모두 두께가 몇 인치나 되는 아주 많은 분량의 소스 코드 목록을 받아 "분석"해야 했습니다. 잘 구성된 다이어그램 몇 개와 일부 지원 텍스트를 받기도 했습니다. 최적화 프로그램 프로젝트는 완료되지 않았습니다. 컴파일러 프로젝트는 성공적으로 완료되었으며 코드와 함께 개발된 광범위한 분량의 테스트 세트로 인해 코드 품질이 우수했습니다. 엔지니어는 디버그 프로그램을 통해 코드를 분석함으로써 코드 기능을 이해하는 데 몇 일이 소요되었습니다. 그러나 엔지니어의 개인적인 노고와 팀 비용을 고려할 때 그만한 가치는 없는 작업이었습니다. XP 에서 제안하는 40 시간의 최대 작업 시간은 적용되지 않았으며 작업 완료를 위해 많은 노력을 기울였습니다.

코드에만 의존하는 경우 주된 문제점은 아무리 올바르게 문서화된 경우라도 코드만으로 해당 코드가 해결하는 문제점을 알 수 없다는 것입니다. 즉, 문제점의 솔루션만 제공합니다. 일부 요구사항 문서화 내용은 원래 사용자와 개발자가 프로젝트 팀에서 떠난 후에라도 오랫동안 원래 목적을 설명할 수 있습니다. 시스템을 유지보수하기 위해서는 일반적으로 원래 프로젝트 팀의 의도를 알고 있어야 합니다. 일부 상위 레벨 디자인 문서는 유사합니다. 종종 코드는 추상 레벨이 너무 낮아 전체로서의 시스템이 수행하려는 사항을 실제로 나타낼 수 없습니다. 이는 특히 클래스를 검토하는 것만으로는 실행 스레드를 따르기 어렵거나 불가능한 객체 지향 시스템의 경우에 해당됩니다. 디자인 문서를 참조하면 나중에 문제점이 발생할 때 검토해야 할 부분을 알 수 있습니다. 문제점은 *반드시* 발생하기 마련입니다.

이 예로 얻을 수 있는 교훈은 디자인 문서를 캡처하고 유지보수하는 작업이 실제로 도움이 된다는 것입니다. 즉, 오해의 위험성을 줄이고 개발 속도를 가속화할 수 있습니다. XP 접근 방식은 디자인을 스케치하거나 CRC

카드를 사용하는 데 몇 분의 시간을 투자하는 것입니다.[1] 팀에서는 이 카드를 유지보수하지 않지만 코드 작업을 수행합니다. 또한 타스크가 단순하여 이미 진행 방법을 알고 있는 것으로 암묵적으로 가정합니다. 이렇게 하더라도 다음 작업자가 쉽게 작업을 수행하지 못하는 경우도 있습니다. RUP 에서는 이러한 디자인 아티팩트를 캡처하고 유지보수하는 데 좀더 많은 시간을 투자하도록 제안합니다.

## *전이(Transition)*

전이 단계의 초점은 일반 사용자가 소프트웨어를 사용할 수 있도록 하는 것입니다. 전이 단계에는 릴리스 준비를 위한 제품 테스트와 사용자 피드백 기반의 사소한 조정이 포함됩니다. 이 라이프사이클 시점에서는 주로 제품의 세부 조정, 구성, 설치 및 사용성 문제에 대한 사용자 피드백이 필요합니다.

릴리스는 초기에 또한 자주할수록 좋습니다. 그러나 릴리스의 정확한 의미를 정의할 필요가 있습니다. XP 에서는 릴리스에 대한 정의가 명확하지 않으며 상용 소프트웨어 릴리스에 필요한 제조 문제를 다루지 않습니다. 내부 프로젝트에 대한 일부 문제를 참조할 수도 있지만 이러한 경우라도 문서화, 훈련 등이 필요합니다. 또한 지원 및 변경 관리가 현장 고객이 제어할 수 있을 정도로 현실적인지도 확인해야 합니다. Bruce Conrad 는 XP 에 대한 InfoWorld 검토에서[2] 고객은 계속 변경되는 소프트웨어를 원하지 않을 수도 있다고 지적합니다. 변경사항을 고객에게 빠르게 전달하는 데 따른 이점과 변경 및 가능한 불안정에 따른 불이익에 대한 가중치를 적용해야 합니다.

릴리스가 결정되면 일반 사용자에게 코드 이상의 것을 제공해야 합니다. 전이(Transition) 활동 및 아티팩트에서 소프트웨어 개발 프로세스의 이 파트를 처리합니다. 활동은 고객에게 사용 가능한 제품을 제공하는 데 중점을 두며 필수 전이 활동은 다음과 같습니다.

- **일반 사용자 지원 자료 완성** — 이 활동은 목록 항목을 검사하는 것 만큼 단순할 수 있지만 조직이 고객을 지원할 수 있는 준비가 되어 있는지 확인해야 합니다.
- **고객 환경에서 제품 인도물 테스트** —현재 위치에서 가능한 경우 고객 환경을 시뮬레이션합니다. 불가능한 경우, 고객을 직접 찾아가 소프트웨어를 설치하고 올바르게 작동하는지 확인합니다. 고객에게 "저희 시스템에서는 올바르게 작동했었습니다"라고 말하는 상황이 발생해서는 안됩니다.
- **고객 피드백을 기반으로 제품 세부 조정** — 가능한 경우, 제한된 수의 고객에게 소프트웨어를 전달하는 하나 이상의 베타 테스트 기간을 계획합니다. 이러한 경우, 베타 테스트 기간을 관리하고 "최종 제품"에 고객 피드백을 고려해야 합니다.
- **일반 사용자에게 최종 제품 전달** — 소프트웨어 제품 및 릴리스 유형에 따라, 패키징, 제조 및 처리해야 할 기타 프로덕션 문제에 대한 많은 세부사항이 존재합니다. 단지 소프트웨어를 디렉토리에 넣어 두고 고객 커뮤니티에 해당 사실을 통보하는 우편을 보내기만하는 경우는 거의 없습니다.

다른 대부분의 단계에서와 같이, 프로세스의 형식성과 복잡도도 다양합니다. 그러나 배치 세부사항에 주의를 기울이지 않으면 몇 개월 또는 몇 주 간의 개발 노력에도 불구하고 제품이 목표 시장에서 실패할 수 있습니다.

전이 단계에서는 여러 아티팩트를 생성할 수 있습니다. 제품의 향후 릴리스까지 고려해야 하는 경우,다음 릴리스의 기능과 결함 수정사항을 식별하기 시작해야 합니다. 모든 프로젝트의 필수 아티팩트는 다음과 같습니다.

- **배치 계획** — 구현/구축(Construction) 단계에서 시작한 배치 계획을 마무리하고 고객 전달을 위한 로드맵으로 사용합니다.

---

[1] CRC(Class, Responsibility, and Collaboration) 카드는 Kent Beck 및 Ward Cunningham 이 개발했으며 종사자(practitioner)에게 객체 지향 디자인의 원칙을 알려줍니다.

[2] http://www.infoworld.com/articles/mt/xml/00/07/24/000724mtextreme.xml

- **릴리스 정보** — 일반 사용자에게 최종 지시사항을 제공하지 않는 드문 소프트웨어 제품입니다. 해당 계획을 세우고 필요한 정보를 사용 가능하고 일관된 형식으로 작성해야 합니다.
- **훈련 자료 및 문서** — 이러한 자료는 다양한 양식으로 제공됩니다. 예를 들어, 온라인 또는 학습서 양식으로 제공될 수 있습니다. 또한 제품 도움말을 사용 가능한 상태로 완성해야 합니다. 지원 자료는 개발자가 아닌 고객 관점에서 작성해야 하며 제품의 성공은 이러한 고객 지원에 달려 있습니다.

## *요약*

소프트웨어 빌드는 코드 작성 이상의 작업입니다. 소프트웨어 개발 프로세스는 고객에게 고품질의 소프트웨어를 제공하는 데 필요한 모든 활동에 초점을 맞추어야 합니다. 완벽한 프로세스가 복잡한 프로세스를 의미하는 것은 아닙니다. 지금까지 프로젝트의 필수 활동 및 아티팩트에 초점을 맞춤으로써 간단하면서도 완벽한 프로세스를 작성할 수 있음을 설명했습니다. 프로젝트 위험성을 완화시킬 수 있는 활동을 수행하거나 아티팩트를 작성해야 합니다. 프로젝트 팀과 조직에 필요한 적정 프로세스 및 형식성을 사용하는 것이 중요합니다.

RUP 와 XP 가 항상 배타적인 것만은 아닙니다. 두 방법의 기법을 통합함으로서 현재보다 나은 품질의 소프트웨어를 빠르게 전달할 수 있는 프로세스를 구축할 수 있습니다. Robert Martin 은 프로세스를 RUP 관점을 따르는 *dX 프로세스* [1] 로 설명합니다. 이 프로세스는 RUP 프레임워크를 기반으로 빌드된 프로세스 인스턴스입니다.

올바른 소프트웨어 프로세스에는 업계에서 입증된 우수 사례가 반영되어 있습니다. 우수 사례는 실제 소프트웨어 개발 조직에서 장기간에 걸쳐 테스트된 사례입니다. XP 는 오늘날 많은 주목을 받고 있는 방법으로서, 코드 중심의 최소 프로세스 오버헤드 및 최대 생산성 제공을 강조합니다. XP 의 많은 기법은 올바른 상황에 대한 고려와 채택을 보장합니다.

XP 는 스토리, 테스트 및 코드에 초점을 맞춥니다. 계획의 중요성도 어느 정도 인정하지만 계획의 캡처는 중요하게 다루지 않습니다. XP 에서는 사용자가 "몇몇 카드로 CRC 디자인을 수행하거나 일부 UML 을 스케치" 또는 "사용되지 않는 기타 아티팩트 또는 문서를 생성하지 않음"과 같은 기타 사항을 생성할 수 있지만 이들을 패스된 것으로 간주합니다. RUP 에서는 개발 계획을 작성하고 갱신할 때 필요한 유용한 요소만 생성하도록 제안하며 해당 요소를 식별합니다.

RUP 는 전체 소프트웨어 개발 라이프사이클에 적용되는 프로세스로서, 수천 가지 프로젝트에 걸쳐 발전된 우수 사례에 초점을 맞춥니다. 우수 사례를 실현할 수 있는 새로운 기법을 연구하고 고안해야 하며 새로운 우수 사례가 나타나면 RUP 에 통합하는 것이 좋습니다.

## *부록 A: Rational Unified Process*

Rational Unified Process 또는 RUP 는 소프트웨어 개발을 위한 원칙을 사용한 접근 방식을 제공합니다.RUP 는 Rational Software 에서 개발하고 관리하는 *프로세스 제품으로서, 다양한 소프트웨어 프로젝트 유형에 대한 여러 가지 기본 로드맵을 제공합니다. RUP 는 또한 Rational 의 다른 소프트웨어 개발 도구 사용에 유용한 정보를 제공하지만 효과적인 조직 적용을 위한 Rational 도구는 필요하지 않습니다. 다른 벤더 제품과의 통합이 가능하기 때문입니다.*

RUP 는 소프트웨어 프로젝트의 모든 측면에 대한 안내를 제공하며 특정 활동을 수행하지 않거나 특정 아티팩트를 생성하지 않아도 됩니다. RUP 는 조직의 특수 사항을 결정할 수 있는 정보와 가이드라인을 제공합니다. 또한 기본 로드맵이 프로젝트 또는 조직에 적합하지 않은 경우 프로세스를 사용자 조정하는 데 유용한 가이드라인을 제공합니다.

RUP 는 새 소프트웨어 개발과 관련된 *위험성을 줄이기 위한 한 가지 방법으로* 최신 소프트웨어 개발에 대한 특정 *우수 사례 채택을 강조합니다. 이러한 우수 사례는 다음과 같습니다.*

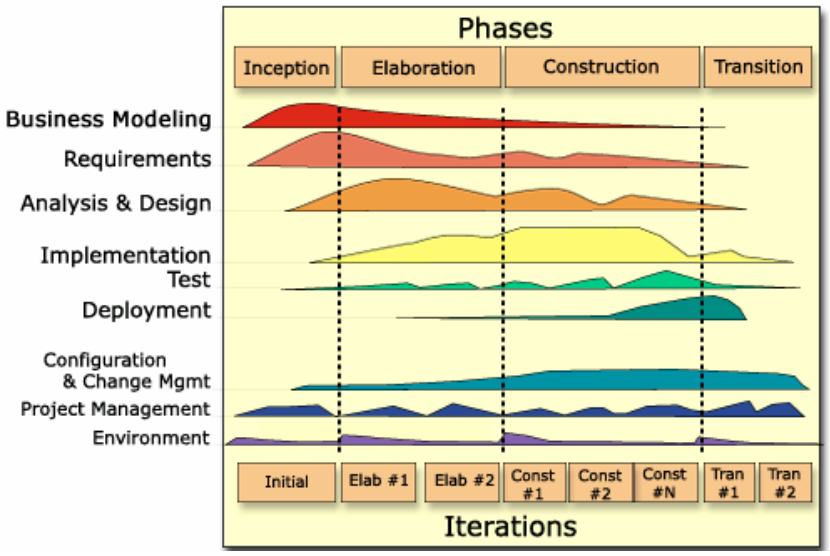
[1] http://www.objectmentor.com/publications/RUPvsXP.pdf 아직 발행되지 않은 Martin, Booch 및 Newkirk 의 서적 내용 중 일부입니다.

1. 반복적으로 개발
2. 요구사항 관리
3. 컴포넌트 기반 아키텍처 사용
4. 시각적 모델링
5. 지속적인 품질 검증
6. 변경 제어

Rational Unified Process 정의에는 다음과 같은 우수 사례가 통합됩니다.

- *역할* — 수행된 타스크와 소유한 아티팩트의 세트
- *원칙* — 요구사항, 분석 및 디자인, 구현, 테스트와 같은 소프트웨어 엔지니어링 노력 영역에 초점을 맞춥니다.
- *타스크* — 아티팩트를 생성하고 평가하는 방식에 대한 정의
- *아티팩트* — 타스크 성능에서 사용, 생성 또는 수정되는 중간 산출물

RUP 는 소프트웨어 개발 프로젝트의 네 단계를 식별하는 반복 프로세스입니다. 프로젝트는 시간의 경과에 따라 도입/인식(Inception), 정제(Elaboration), 구현/구축(Construction) 및 전이(Transition) 단계로 수행됩니다. 각 단계에는 실행 가능하지만 불완전한 시스템을 생성하는 하나 이상의 반복이 포함됩니다(도입/인식 단계 제외). 각 반복에서는 여러 원칙의 활동을 다양한 세부사항 레벨로 수행합니다. 다음은 RUP 의 개요 다이어그램입니다.




RUP 개요 다이어그램

*The Rational Unified Process, An Introduction, Second Edition* 은 RUP 에 대한 올바른 개요 정보를 제공합니다. RUP 에 대한 추가 정보와 평가는 Rational Software 웹 사이트, www.rational.com을 참조하십시오.

## *부록 B: XP(eXtreme Programming)*

XP(eXtreme Programming)는 1996 년 Kent Beck 이 개발한 소프트웨어 개발 원칙으로서, 커뮤니케이션, 단순성, 피드백 및 용기의 네 가지 가치를 기반으로 합니다. XP 는 개발 진행 과정에 현장 고객을 참여시킴으로써 고객과 개발 팀 구성원 간의 지속적인 커뮤니케이션을 강조합니다. 현장 고객은 빌드 내용과 해당 순서를 결정합니다. 코드를 지속적으로 리팩토링하고 코드 이외의 아티팩트 세트를 최소화함으로써 단순성을 추구할 수 있습니다. 여러 가지 간단한 릴리스와 지속적인 유닛 테스트가 피드백 메커니즘입니다. 용기는 다른 사람이 하지 않더라도 올바른 일을 수행하는 것을 의미합니다. 즉, 자신이 할 수 있는 일과 할 수 없는 일을 정직하게 인정하는 것을 의미합니다.

다음과 같은 12 가지 XP 사례가 네 가지 가치를 지원합니다.

- **계획 게임** — 우선순위가 결정된 스토리와 기술 예상을 조합함으로써 다음 릴리스의 기능을 결정합니다.
- **소규모 릴리스** — 소규모 소프트웨어 릴리스를 자주 점진적으로 고객에게 제공합니다.
- **메타포** — 메타포는 시스템 작동 원리에 대한 간단한 공유 스토리 또는 설명입니다.
- **단순 디자인** — 코드와 디자인을 단순하게 유지합니다. 코드 복잡도를 지속적으로 검토하여 제거해야 합니다.
- **테스트** — 고객은 스토리를 테스트하기 위한 테스트를 작성합니다. 프로그래머는 코드를 확인할 수 있는 모든 내용을 테스트하기 위한 테스트를 작성합니다. 테스트는 코드를 작성하기 전에 작성합니다.
- **리팩토링** — 중복 코드와 코드의 복잡도를 제거하는 단순화 기법입니다.
- **짝 프로그래밍** — 두 명의 프로그래머가 단일 컴퓨터를 사용하는 팀 구성으로 모든 코드를 개발합니다. 한 프로그래머는 코드를 작성하거나 *구동시키고* 동시에 다른 프로그래머는 코드의 정확도와 이해 가능성을 검토합니다.
- **공동 소유권** — 모든 사람이 모든 코드를 공유합니다. 이는 모든 사람이 언제라도 원하는 코드를 변경할 수 있음을 의미합니다.
- **지속적인 통합** — 구현 타스크가 완료될 때마다 하루에 여러 번 시스템을 빌드하고 통합합니다.
- **주당 40 시간 작업** — 피곤한 프로그래머는 최대의 업무 효율성을 나타낼 수 없습니다. 2 주 연속 연장 근무를 해서는 안됩니다.
- **현장 고객** — 실제 고객이 개발 환경에 공식적으로 참여함으로써 시스템 정의, 테스트 작성 및 질의 응답 등의 지원을 제공합니다.
- **코딩 표준** — 프로그래머는 일관적인 코딩 표준을 채택합니다.

현재 사용 가능한 XP 에 대한 서적은 다음과 같은 세 가지가 있습니다.

1. eXtreme Programming Explained
2. Extreme Programming Installed
3. Planning Extreme Programming

여러 웹 사이트에서 XP 에 대한 추가 정보를 얻을 수 있습니다.

Rational
the software development company

본사 안내:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
전화번호: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
전화번호: (781) 676-2400

수신자 부담 전화번호: (800) 728-1212
전자 우편: info@rational.com
웹: www.rational.com
전세계 지사 안내: www.rational.com/worldwide