RUP 및 XP 비교

John Smith

Rational Software 백서

TP 167, 5/01

Rational®
the software development company

목차

소개

이 문서는 현재 소규모에서 대규모 프로젝트에 이르기까지 다양한 소프트웨어 프로젝트에서 널리 사용되고 있는 프로세스 프레임워크인 Rational Software 의 대표 솔루션, Rational Unified Process(RUP)와 요구사항이 변화하는 환경에서 소규모 시스템을 효과적으로 빌드할 수 있는 방법으로 인정받고 있는 소프트웨어 개발 접근 방식인 XP(Extreme Programming)를 비교합니다.

중점 비교 대상은 프로세스 설명을 위한 각 접근 방식의 특성 및 스타일과, 기본 구조, 가정 및 프리젠테이션입니다. 이 문서에서는 또한 각 방식의 잠재 대상과 방식 적용의 결과를 점검합니다.

대부분의 프로세스에는 조직적인 비교를 가능하게 하는 일부 공통 요소가 포함되어 있습니다. 이러한 요소에는 일반적으로 개인 또는 팀 단위로 작업하는 사용자가 아티팩트 또는 중간 산출물을 생성하는 역할로 수행되는 일련의 타스크 또는 타스크 그룹이 필요합니다. 해당 아티팩트 또는 중간 산출물은 일부 또는 모두 고객에게 전달됩니다. 대부분의 프로세스에서는 또한 시작과 종료를 포함한 프로세스 인스턴스의 시간 차원과 함께, 중요 활동 또는 활동 클러스터의 완료와 연관 아티팩트 생성을 나타내는 중요 중간 이정표를 고려합니다. 따라서 이 문서에서는 다음과 같은 프로세스 차원을 탐색하고 해당 차원에 따라 비교합니다.

- **시간 및 노력 할당** — 일정 기간 각 프로세스가 배열되는 방식과 인력 구성 노력 할당의 비교 방식에 대해 설명합니다.
- **아티팩트** —XP 및 RUP 기반 프로젝트 과정에서 생성되는 *중간 산출물*을 비교합니다.
- **타스크** —각 프로세스에서 해당 아티팩트를 생성하는 방식에 대해 설명합니다.
- **원칙** — XP 와 RUP 가 소프트웨어 엔지니어링의 주요 관심 영역을 나타내는 방식을 비교합니다.
- **역할** — RUP 에서 타스크를 수행하는 역할과 XP 에서 팀 내 직위와 유사한 역할 간의 차이점을 분석합니다.

이 문서의 목적은 프로세스 범위에서 RUP 와 XP 의 상대적 위치를 명확히 함으로써 각 방식이 제공할 수 있는 가치를 확인하고 XP 가 *복잡한* RUP 에 비해 *간단하고* 바람직한 대안이라는 견해에 반론을 제기하는 것입니다.

이 문서를 작성하는 데 사용된 XP 정보는 주로 다음과 같은 세 권의 Addison-Wesley XP 시리즈 서적에서 참조한 것입니다.

- Extreme Programming Explained [Beck00]
- Extreme Programming Installed [Jeffries01]
- Planning Extreme Programming [Beck01]

이 책들은 XP 에 대한 정확한 정보를 가장 많이 제공하여 많은 관심있는 독자와 잠재적 XP 사용자가 주로 사용하기 때문에 선택했습니다. 또한 이 문서 작성 당시 아직 출간되지 않았던 다른 책들도 참조할 수 있습니다. RUP 정보는 Rational Software Corporation 에서 제공한 RUP 제품을 직접 참조했습니다.

이 문서는 XP 또는 RUP 모두에 있어 학습서용으로 작성된 것이 아니므로, 이러한 접근 방식에 대해 어느 정도의 지식 또는 경험을 갖고 있거나 일부 참고 자료(예: XP 의 경우 [Beck00], RUP 의 경우 [Kruchten00])에 액세스할 수 있는 경우 보다 쉽게 이해할 수 있습니다.

시간 및 노력 할당

RUP 는 소프트웨어 개발 프로젝트를 처리하며 해당 수명은 *도입/인식(Inception), 정제(Elaboration), 구현/구축(Construction)* 및 *전이(Transition)* 단계로 구분됩니다. 각 단계는 다시 반복으로 분할되며 각 반복에는 여러 빌드가 필요할 수 있습니다.

대부분의 RUP 프로젝트의 경우 반복 지속 기간은 2 주 – 6 개월입니다.[1]반복 횟수는 전체 프로젝트 수명에서 3 – 9 회입니다. 따라서 이 기본 설정에서 RUP 는 지속 기간이 6 주 – 54 개월인 프로젝트를 처리합니다.

XP 의 반복 지속 기간은 약 2 주이며 릴리스 지속 기간은 2 개월 또는 그보다 약간 더 소요됩니다. 릴리스 지속 기간은 고객이 직접 정의합니다. XP 의 릴리스 지속 기간은 XP 에 적합한 프로젝트의 경우 최소한 RUP 에서 정제(Elaboration) 또는 구현/구축(Construction)[2] 의 반복 지속 시간과 같습니다. 이러한 프로젝트의 최대 팀 규모는 10 명[3]이며 정해진 아키텍처 기준선을 기반으로 합니다.[4]

이를 입증하려면 XP 의 최대 팀 규모를 10 으로 가정하고 해당 팀이 수행할 수 있는 최대 규모의 프로젝트를 찾아 RUP 가 이 규모 이하의 프로젝트를 처리하는 방법을 확인합니다. COCOMO II[5] 를 예상 모델로 사용하는 경우 10 명으로 구성되는 팀은 일반적으로 최대 지속 기간이 약 15 개월인 프로젝트와 연관되는 것으로 가정합니다. 프로젝트가 15 개월 이상 지속되는 경우 일반적으로 10 명 이상으로 구성되는 팀을 선택합니다. 스케줄을 연장할 수 있는 경우 10 명의 인력만으로도 대규모 시스템을 빌드할 수는 있지만, 그러한 경우라도 효과적으로 할 수만 있다면 인력을 더 추가할 만큼 스케줄은 중요합니다. 이 프로젝트 예에서는 처음부터 약 40,000 – 50,000 개의 소스 코드 행 수(SLOC) 또는 Java 의 경우 800 – 1000 점의 기능 점수[6] 를 제공합니다.

이 모델에 따르면 이 프로젝트는 XP 규모 팀에서 즉시 시도할 수 있는 최대 프로젝트입니다. 소규모 팀에서 대규모 시스템을 효과적으로 빌드할 수 없는 또 다른 이유로 예를 들어, 시간이 지나면서 이전에 빌드된 코드를 잘 이해하지 못하게 되는 경우를 들 수 있습니다. 대규모 팀에서 병렬로 수행할 수 있는 일을 소규모 팀에서는 연속적으로 수행해야 하므로 통합을 테스트하고 디버깅해야 하는 경우 소규모 팀에서는 상당히 오래 전에 작성한 코드를 재검토해야 합니다.

이러한 규모의 프로젝트 세트를 볼 때 XP 의 릴리스와 RUP 의 반복 간에 맵핑되는 부분을 발견할 수 있습니다. 다음 예제는 RUP 에서 이러한 프로젝트를 계획하는 방법을 보여줍니다. 표의 숫자는 참고용이지만 이러한 규모의 프로젝트에 합당한 수치입니다. 또한 해당 숫자는 사용자 안내서, 설치 및 운영 정보 등과 같이 코드 이외에 전달되는 모든 필수 RUP 아티팩트를 준비하는 데 소요되는 노력과 시간을 포함합니다.

---

[1] e-business 개발 프로젝트의 예상 반복 지속 기간은 일반 프로젝트 기간보다 짧을 수 있고(일반적으로 2 – 6 주)

[2] 도입/인식(Inception)의 반복은 일반적으로 더 짧습니다. 또한 RUP 의 단계 모델은 그 모양이 다양하므로, 장기 전이(Transition) 단계에서는 2 개월 간격으로 각각 고객에 대한 소프트웨어 인도를 완료하는 일련의 반복을 예상할 수 있습니다. 그러나 전이 단계로 진입할 경우 아키텍처는 완전히 안정된 상태가 되고 대부분 조정, 향상 및 수정을 위한 변경이 수행됨을 의미합니다.

[3] [Beck00]을 참조하십시오. 여기서는 XP 프로젝트를 20 명의 프로그래머로는 실행할 수 없고 10 명의 프로그래머로는 "실행할 수 있는 것"으로 설명합니다.

[4] XP 는 고객에게 직접적인 비즈니스 가치를 기능으로 제공하는 것이 목적이므로 아키텍처를 직접 고려하지는 않습니다. 이러한 경우 기능 추가에 따라 소프트웨어에 대한 일련의 리팩토링이 발생할 수 있습니다. 솔루션 아키텍처가 아직 잘 확립되지 않은 경우, 이전 가정 및 임시 솔루션을 무효화하고 로컬 리팩토링을 따르지 않는 문제점을 발생시키는 기능이 추가됨에 따라 이로 인해 일련의 심각한 장애가 발생할 수 있는 위험성이 있습니다.

[5] COCOMO II 는 Barry Boehm 박사가 최초로 개발한 클래식 COCOMO 소프트웨어 비용 예상 모델의 수정 모델입니다. 이 모델은 소스 코드 행 수(SLOC)가 약 2,000 개인 소규모 프로젝트에 맞게 조정되었습니다. [Boehm00]을 참조하십시오.

[6] 기능 점수는 소스 코드와 관계 없이 소프트웨어 크기를 측정하는 수단으로서, 사용자에게 제공되는 기능성을 정량화하여 평가하며 논리 디자인 및 기능적 스펙만을 기반으로 합니다. 이 정의는 International Function Point Users Group 웹 사이트(http://www.ifpug.org/)에서 인용한 것입니다.

XP 를 따르는 프로젝트 예제

예제 1 은 5,000 개의 새 Java 코드 행으로 구성되며 7 개월의 경과 시간 동안 12 인월(person-months)의 인력이 필요한 소규모 프로젝트를 나타냅니다.[7]

예제 1

	도입/인식(Inception)	정제(Elaboration)	구현/구축(Construction)	전이(Transition)
인력	1	1.5	2	2
지속 기간(주)	3	6	18	3
반복 횟수(및 각 지속 기간(주))	1(3)	1(6)	3(6)	1(3)

예제 2 는 10,000 개의 새 Java 코드 행으로 구성되며 8 개월의 경과 시간 동안 약 27 인월(person-months)의 인력이 필요한 소규모 프로젝트를 나타냅니다.

예제 2

	도입/인식(Inception)	정제(Elaboration)	구현/구축(Construction)	전이(Transition)
인력	1.5	2.5	4	3
지속 기간(주)	4	7	20	4
반복 횟수(및 각 지속 기간(주))	1(4)	1(7)	3(7)	1(4)

예제 3 은 40,000 개의 새 Java 코드 행으로 구성되며 15 개월의 경과 시간 동안 115 인월(person-months)의 인력이 필요한 중간 규모 프로젝트를 나타냅니다.

예제 3

	도입/인식(Inception)	정제(Elaboration)	구현/구축(Construction)	전이(Transition)
인력	3	5	10	8
지속 기간(주)	6	16	36	6
반복 횟수(및 각 지속 기간(주))	1(6)	2(8)	4(9)	1(6)

개발 기간이 긴 이러한 규모의 프로젝트에서는 해당 의도 및 지속 기간 측면 모두에서 RUP 의 반복과 XP 의 릴리스가 매우 밀접하게 맵핑됩니다.

[7] 이 예제는 지나치게 긴 것처럼 보일 수도 있지만 아이디어만 있고 필요한 인력과 요구사항도 정의되지 않은 처음 시작부터 프로젝트 승인 및 완료까지 전체 라이프사이클을 나타낸다는 점을 고려해야 합니다. 또한 이 예제는 COCOMO II 모델의 결과물로서, 스케줄을 더 단축할 수도 있지만 그러한 경우 비용과 위험성이 증가합니다. 또한 표의 스케줄을 따르면 프로젝트 시작 후 3 개월 반 정도 되는 시점(첫 번째 구현/구축(Construction) 반복 종료 시점)에 유용한 기능을 사용할 수 있습니다.

XP 에 적합하지 않은 대규모 시스템 개발

대규모 개발 프로젝트는 RUP 로 처리하며 XP 에는 적합하지 않습니다. 또한 이러한 경우 RUP 의 반복 지속 기간이 매우 깁니다. 예제 4 는 1,500,000 개의 새 Java 코드 행으로 구성되며 45 개월의 경과 시간 동안 4,600 인월(person-months)의 인력이 필요한 프로젝트를 나타냅니다.[8]

예제 4

	도입/인식(Inception)	정제(Elaboration)	구현/구축(Construction)	전이(Transition)
인력	35	70	140	100
지속 기간(주)	20	50	100	20
반복 횟수(및 각 지속 기간(주))	2(10)	2(25)	3(33)	2(10)

이 예제의 인력은 하나의 단일 팀으로 구성되지 않았습니다. 즉, 이 프로젝트는 각각 서브시스템을 담당하는 여러 팀으로 구성됩니다. 좀 더 낮은 레벨에서 XP 로 처리할 수 있는 규모의 프로젝트와 유사한 규모의 계획을 수립할 수 있도록 해당 서브시스템을 하위 서브시스템으로 구성할 수 있습니다. 그러나 이 프로젝트는 통합 시스템을 나타내야 하므로 최상위 레벨에서는 XP 수준보다 높은 규모(33 주)의 반복에 대한 계획이 필요합니다. 이러한 규모의 통합 프로젝트는 계획 작업도 지연되므로 해당 반복은 이 지속 기간에 수행됩니다. RUP 의 반복 내 계획은 시스템 및 서브시스템 레벨에 존재하는 통합 빌드 계획을 통해 수행되며 이러한 계획은 특히 정제(Elaboration) 및 구현/구축(Construction) 후반부에서 XP 릴리스와 유사한 규모(대규모 시스템의 경우)와, 약 2 주의 시간이 필요한 XP 반복과 유사한 규모의 최하위 레벨에서 구성됩니다.

소규모 시스템 예제로 돌아와, RUP 의 반복은 XP 의 릴리스에 해당되며 RUP 의 빌드는 XP 의 반복에 해당됩니다.

XP 에서 RUP 단계의 맵핑 대상

언뜻 보면 RUP 단계는 XP 주기에서 잘 묘사되지 않은 측면과 맵핑됩니다. XP 는 비즈니스 주기에 따라, 반복을 포함하는 일정한 릴리스 주기에 맞게 구성되거나 최소한 기술된 것처럼 보입니다. 자력으로 생성되어야 하는 유형의 *프로젝트*가 있고([Beck01]의 "프로젝트 범위 지정" 참조) 참조) 현 시점에서 생성된 *대규모 계획*이 릴리스로 분할되고 이 릴리스가 다시 반복으로 분할된다는 인식도 있습니다. [Beck01]에서는 대규모 계획의 장점 중 하나를 프로젝트 투자의 정당성을 확인시켜주는 것으로 설명합니다.

따라서 XP 에서는 릴리스 및 반복 주기가 시작되기 전에 프로젝트의 실현 가능성 여부와 해당 비용을 결정하는 작업을 수행해야 하며 이를 위해 필수 기능성에 대한 개략적인 아이디어를 구성해야 합니다. 이를 RUP 에서는 도입/인식(Inception) 단계라고 합니다. XP 의 경우 이러한 작업은 매우 신속하게 수행되어야 합니다. 반면에 RUP 에서는 내재된 위험성에 따라 최대한 신중하게 수행됩니다. XP 의 경우에도 소규모 팀에서 이 시점에 수행해야 하는 다음 작업에 유의해야 합니다.

- 요구사항 도출("화제" 작성)
- 계획
- 고객과의 협상(기대 설정)
- 비즈니스 제한조건 고려

[8] 프로젝트를 이런 방식으로 구성해서는 안된다는 주장이 제기될 수도 있습니다. 즉 이러한 규모의 프로젝트는 각각 XP 로 처리할 수 있는 보다 소규모의 많은 프로젝트로 분할해야 한다고 주장할 수도 있습니다. 물론 이러한 규모의 프로젝트는 서브시스템의 복합 시스템으로 구성되거나 이보다 더 큰 규모의 프로젝트의 경우 복합 시스템으로 구성되어야 합니다. 그러나 이러한 서브시스템 또는 시스템을 단일 제품으로 통합해야 하는 경우 아키텍처와, 특히 소프트웨어 집계와 해당 집계를 생성하는 팀 간의 인터페이스를 고려해야 합니다. 현재 XP 양식으로는 이러한 문제를 처리할 수 없습니다.

처음 시작부터 합의된 비전(RUP 에서 "화제"를 일컫는 표현) 및 비즈니스 사례(프로젝트를 정당화하는 예산 및 투자 수익(ROI))의 완성과 소프트웨어 개발 계획의 처음 시작까지 몇 일이 소요될 수 있습니다.

RUP 에서는 XP 에서와 같이 프로젝트 시작부터 충분한 탐색 및 계획을 수행함으로써 높은 성공 가능성을 입증하여 계속 진행할 수 있습니다. 앞의 예제 2 와 같이, RUP 계획은 약 $250,000 의 지출을 정당화시키기 위해 도입/인식(Inception)에서 약 $12,000 의 투자 노력을 제안합니다. 이 금액은 경우에 따라 다릅니다. 예를 들어, 잘 해결된 문제점 영역의 경우 이 정도로 수행할 필요가 없으며 처음부터 시작하지는 않을 수 있습니다. 또는 특정 연구 및 개발이 필요한 익숙하지 않은 도메인의 경우 처음부터 많은 비용을 지출하게 됩니다.

XP 의 경우 도입/인식(Inception) 후에는 첫 번째 릴리스 계획 단계로 진행됩니다. RUP 에서는 도입/인식 다음에 정제(Elaboration) 단계가 진행되며 정제 반복에서는 솔루션 아키텍처가 안정화됩니다. 이와 달리 XP 에서는 기능에 따라 릴리스 계획이 수행되어 모든 릴리스가 고객에게 비즈니스 가치를 제공합니다. XP 는 하부 구조 계획을 중요하게 여기지 않으며 해당 릴리스에 대해 선택된 기능성을 지원하기만 하면 됩니다. 추가 하부 구조는 나중에 필요에 따라 추가되며, 나중에 기능이 추가되어 이전의 하부 구조 결정사항이 무효화됨으로써 시스템이 중단되면 코드를 리팩토링하여 시스템을 실행합니다.[9] 아이디어는 고객에게 가치를 제공하지 않는 요소를 빌드하는 데 많은 시간을 소비하지 않는 것입니다.

반대로 RUP 에서는 다음과 같은 사항을 고려해야 합니다.

- *RUP 의 아키텍처 개념은 단순한 하부 구조가 아닙니다.*

  RUP 에서, "특정 시점의 소프트웨어 시스템 아키텍처는 인터페이스를 통해 상호작용하는 중요한 시스템 컴포넌트의 조직 또는 구조이며 이 컴포넌트는 연속적으로 더 작은 컴포넌트 및 인터페이스로 구성됩니다."

  따라서 아키텍처는 특정 관점에서 볼 때 적절한 추상 레벨에서 —하부 구조뿐만 아니라 —전체 솔루션에 적용됩니다.

- *RUP 의 실행 가능 아키텍처에 대한 개념은 고객에게 비즈니스 가치를 제공합니다.*

  고객의 비기능적 요구사항(프로세스 내 일부 핵심 기능적 요구사항도 가능)을 충족시키는 솔루션을 조기에 보여줄 수 있습니다.

  이러한 방식으로 위험성을 줄이는 것은 그 자체로 고객에게 상당한 비즈니스 가치를 제공하는 것입니다. 또한 비기능적 요구사항은 일반적으로 위험성을 갖고 있습니다.

이러한 방식의 프로세스 설명이 필요한 이유에 대해 생각해 보아야 합니다. RUP 는 위험성을 효과적으로 제거할 수 있으며, 또한 코드 리팩토링으로 아키텍처를 생성할 수 있는 XP 접근 방식을 준수하지 않는 여러 시스템 및 문제점이 존재합니다. 이러한 시스템은 일반적으로 복잡한 대규모 시스템입니다. 한 가지 위험성은 로컬 리팩토링 변경을 수행하는 데 있어(해당 범위는 제한됨), 전체적으로 차선의 솔루션을 생성하는 로컬 최적화가 이루어진다는 것입니다. Kent Beck 또한 [Beck00]의 *Four Values* 장에서 *Courage* 라는 주제로 이러한 내용을 주장했습니다. 문제는 XP 가 구조적으로 중요한 변경사항을 수행한 원인에 대해 설명하지 않으므로 해당 변경사항을 적용하려면[10] 용기가 필요하다는 것입니다.

또한 리팩토링이 어려운 변경사항도 있습니다. 예를 들어, 단일 사용자, 단일 시스템을 다중 사용자, 멀티프로세서 시스템으로 만들려면 많은 요소를 다시 디자인해야 하며 이러한 경우에도 해당 시스템에는 여전히 많은 제한조건이 존재하게 됩니다.

---

[9] 그러나 리팩토링은 로컬 요소만 개선하고 전체 프레임워크에 적용되지 않습니다. 전체 프레임워크가 잘못된 문제점을 해결하는 경우 리팩토링은 올바른 솔루션을 생성하지 않습니다. 유일한 솔루션은 과감한 변경 작업을 수행하거나 극단적인 경우 처음부터 다시 시작하는 것입니다. 그러나 이러한 경우 예산 및 스케줄 위험성을 감수해야 합니다.

[10] 물론 용기는 기본적으로 "좋은 사람"들이 갖고 있는 유용한 가치이지만 가장 용기있는 사람이라도 경우에 따라서 어려운 문제점을 해결할 수 있는 프레임워크와 조직적인 작업 방식이 필요합니다.

아키텍처에 대한 RUP 의 강조사항은 초기 접근 방식이 잘못되어 완전히 다시 고려해야 하는 상황을 처리하는 것입니다. 소규모 시스템의 경우에는 이러한 위험성이 적습니다. 즉, 리팩토링으로 시스템 전체에 광범위하게 영향을 줄 수 있으며 초기 아키텍처가 잘못되었을 가능성도 적습니다.

규모, 신기술, 경험 부족, 복잡도, 기타 성능 요건, 신뢰성, 안전 등과 같은 인지된 위험성이 거의 없고 쉽게 이해할 수 있는 기존 프레임워크로 솔루션을 전달할 수 있는 경우, RUP 정제(Elaboration) 단계(아키텍처 문제를 처리하는 단계)가 매우 간단하며 중요한 기능적 요구사항의 도출, 정제 및 시연과 많은 관련이 있습니다. 즉, RUP 라이프사이클은 XP 라이프사이클을 따르는 대상에는 적합하지 않습니다. XP 로 문제점을 해결할 수 있는 경우 RUP 개발 사례 또한 완전히 같지는 않더라도 유사하게 '간단한' 결과물을 생성합니다.

RUP 의 구현/구축(Construction) 단계는 XP 에서 일련의 릴리스에 해당합니다. 단, RUP 를 따르는 경우 각 구현/구축 반복의 결과를 고객에게 전달해야 XP 와 동일한 효과를 얻을 수 있습니다. XP 에는 실제로 RUP 의 전이(Transition) 단계에 해당하는 단계가 없습니다. 각 XP 릴리스가 이미 고객에게 전달된 상태이기 때문입니다. XP 프로젝트 종료 시점에는 [Beck00]에서 프로젝트 완료(해당 정보는 뒷 부분의 *XP 라이프사이클 표제 부분 참조)로 칭하는 단계가 있습니다. 여기서는 일부 동일한 프로젝트 완결 활동이 수행됩니다.*

RUP 에서 XP 단계의 맵핑 대상

XP 의 단계(XP 릴리스 및 반복에 모두 적용)는 탐색, 확약 및 조정(steering)입니다. 릴리스 레벨의 경우 이러한 단계는 RUP 프로젝트 관리 원칙의 특정 타스크(RUP 의 활동) 콜렉션과 조정됩니다. 이는 다음 반복 계획(탐색 및 확약으로 맵핑)과 프로젝트 모니터 및 제어(조정(steering)으로 맵핑)입니다. 이 내용은 다음 섹션에서 설명합니다.

XP 단계

[Beck00]에는 이상적인 XP 프로젝트의 라이프사이클에 대해 설명하는 장이 있으며 첫 번째 레벨 표제는 **탐색**, **계획**, **첫 번째 릴리스 반복**, **'프로덕션'** , **유지보수** 및 **완료**입니다. 이러한 단계는 최상위 레벨 단계로 보일 수도 있지만 정확한 단계가 아닙니다. XP 프로젝트는 *초기* 탐색 단계에서 시작하여 프로젝트 완결 시 "완료"로 끝납니다. 그러나 주요 주기는 릴리스이며 *각 릴리스마다 탐색 단계가 있습니다.* 따라서 프로젝트에 적용되고 첫 번째 릴리스로 연결되는 탐색 단계는 프로젝트 지속 여부에 대해 결정하고 패스해야 하는 첫 번째 관문([Beck01]의 *프로젝트 범위 지정 참조)이 있는 특별한 단계입니다. XP 릴리스 및 반복은 모두 세 가지 단계, 탐색, 확약 및 조정(steering)으로 구성됩니다. "유지보수"는 실제로 첫 번째 릴리스 이후 XP 프로젝트의 특성을 나타냅니다.*

XP 라이프사이클

전반적으로, 15 개월 동안 일곱 개의 릴리스로 진행되는 프로젝트의 XP 라이프사이클은 그림 1 과 같습니다.

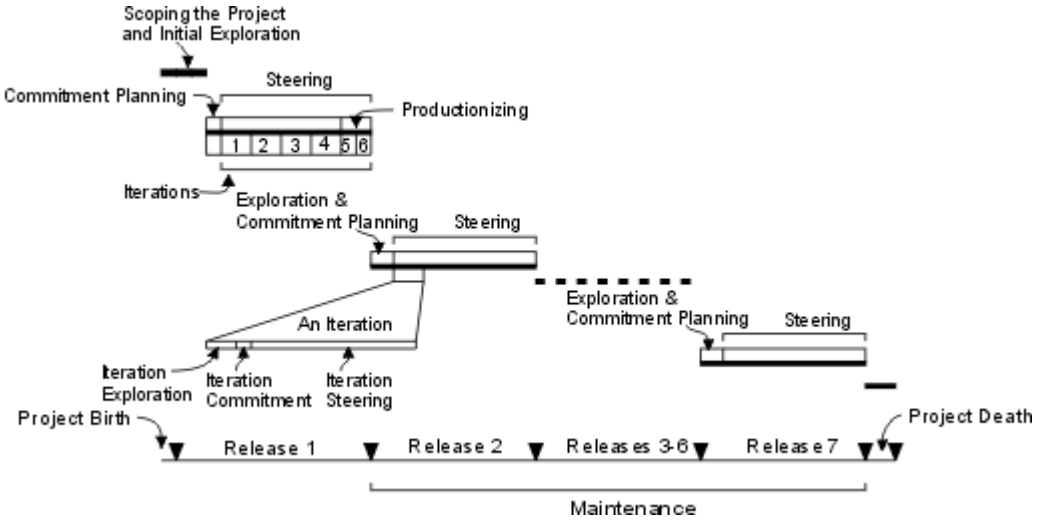

그림 1

첫 번째 릴리스(3 개월) 이후 각 릴리스의 지속 기간은 약 2 개월이며 각 반복은 약 2 주 동안 진행됩니다. 단, 반복 속도가 증가하는 마지막 릴리스 단계([Beck00]에서 설명하는 "프로덕션")의 경우는 예외입니다.

이 라이프사이클이 계획 메트릭 관점에서 올바른 라이프사이클인지 검토해 보아야 합니다. 이 가상 예제는 해당 릴리스의 지속 기간을 약 2 개월로 설정하고 첫 번째 릴리스의 지속 기간을 2 – 6 개월로 설정한 XP 의 규정 가이드라인을 따르며 팀 규모는 10 명 미만으로 유지합니다. 이 프로젝트가 앞에서 설명한 예제 3 과 같다면 모두 합쳐 약 40,000 개의 Java 코드 행 또는 800 점의 기능 점수(COCOMO II 모델의 변환 요인을 승인하는 경우)를 제공합니다. 또한 이 프로젝트를 릴리스 간에 균일하게 나눈다면 각 릴리스는 약 115 점의 기능 점수를 제공합니다.

불확실한 파트는 첫 번째 릴리스입니다. 즉, 한 명의 인력만 지정되고 요구사항도 캡처되지 않고 범위도 지정되지 않은 처음 시작에서 3 개월 안에 고객에게 최종 제품과 유사한 품질의 제품을 제공하는 것은 매우 어려운 일입니다. 생산성을 높일 수 있는 경우라도, 즉 매월 인력당 12 점의 기능 점수를 제공할 수 있는 경우라도(The David Consulting Group 의 산업 데이터 기반, http://davidconsultinggroup.com/indata.htm 참조) 임의로 인력 비율을 늘릴 수는 없습니다. 여기서 다루는 문제점 규모(기능 점수 115 점)는 매우 작으므로, 임의로 더 작은 범위로 나눠 대규모 팀이 수행할 수 있도록 할 수 없습니다.

그러나 가능한 경우, 필요한 평균 팀 규모는 네 명이며 프로젝트의 첫 번째 릴리스가 종료될 때 팀 규모는 일곱 명입니다. 프로젝트의 지속 기간 동안 한 명의 인력이 팀에 더 추가되면 나머지 프로젝트 기간은 XP 에서 정상 모드(유지보수)를 유지하며 팀 규모는 여덟 명(XP 의 안전 규모)입니다. 제공된 규모가 증가하면 일반적으로 생산성은 약간 저하되지만 릴리스 간격(2 개월)이 줄어들어 증가한 인력 수를 상쇄합니다. 결과적으로 각 릴리스는 첫 번째 릴리스와 동일한 기능적 영향을 추가로 제공할 수 있습니다. 전체 프로젝트 노력은 앞의 예제 3 보다 약간 적은 108 인월(staff-months)이 됩니다.

기본 RUP 라이프사이클

대조적으로, 유사한 규모의 RUP 프로젝트에 대한 *기본* 라이프사이클은 다음과 같습니다.

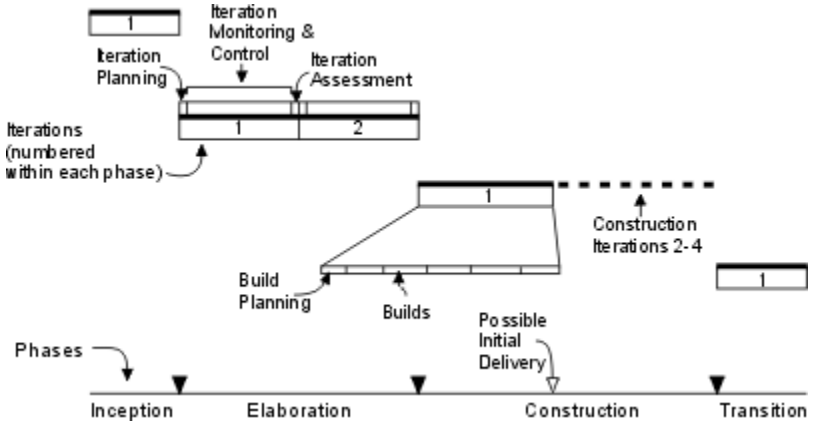

그림 2

이 계획 역시 예제 3 을 기반으로 하며 RUP 반복은 각 단계에 숫자로 표시됩니다. 기본 RUP 라이프사이클을 사용하는 경우, 고객에게 최종 제품과 유사한 품질의 제품을 *제공*할 수 있는 최초의 기회는 일반적으로 첫 번째 구현/구축(Construction) 반복의 종료 시점입니다. 이 반복은 프로젝트 시작 후 7 개월 시점에 수행됩니다. 그러나 이 시점에서 RUP 가 제공할 수 있는 기능 수준은 200 점의 기능 점수 또는 총 기능의 1/4 정도의 수준입니다(XP 의 경우 3 개월 후 기능 점수는 115 점입니다). 일반적으로 고객에게 처음 제품을 제공하는 시점은 구현/구축 종료 시점(이 경우 프로젝트 시작 후 13 개월 시점)이며 본질적으로 모든 기능이 제공됩니다. 그러나 이 시점에서 처음 고객이 제품을 보게 된다는 의미는 아닙니다. RUP 에서는 고객이 반복 평가에 초대되어 발전 단계에서 테스트 중인 제품을 확인할 수 있습니다.

RUP 와 XP 간의 차이점은 특히 이러한 기본 경우에 있어, 고객에게 필요한 기능성을 빌드하기 전에 정제(Elaboration)에서 두 가지 반복을 통해 안정화시킬 수 있는 솔루션 아키텍처에 위험성이 존재하는 것으로 가정하기 때문에 발생합니다. 이러한 위험성의 예로는 해당 아키텍처가 처음 작성된 아키텍처이거나 신기술을 이용하거나 비기능적 요구사항에 따른 부담이 존재하는 경우를 들 수 있습니다. 이는 정제 종료 시점까지 구조적으로 중요한 요구사항만 탐색하고 구현됨을 의미합니다.

XP 에서는 이러한 작업을 수행하지 않습니다. XP 는 아키텍처가 릴리스 간에 손상되지 않거나 리팩토링으로 해당 손상을 복구할 수 있다는 가정하에, 순차적으로 완료된 일련의 솔루션을 각 릴리스에서 제공합니다. 이러한 경우 기능을 알고 있는 기존 아키텍처에서 빌드할 수 있는 시스템으로 올바른 XP 적용을 제한하게 됩니다.

실제 RUP 라이프사이클

이러한 시스템의 경우, 실제 RUP 라이프사이클은 약간 다르게 나타납니다. 즉, 정제(Elaboration) 단계가 많이 짧아지고 도입/인식(Inception) 단계 역시 짧아집니다. 특정 정제 반복을 구현/구축 반복과 교환하는 경우 첫 번째 구현/구축 반복의 종료 시점이 몇 개월 정도 빨라집니다. 즉, 7 개월 대신 5 개월이 소요됩니다.

이러한 경우 기본 예제의 기능 점수 200 점보다 낮은 점수를 제공하지만(이전 구현/구축(Construction) 반복에서 작업하는 인력의 수가 더 적으므로) 이는 XP 스케줄의 민첩성에 접근하고 위험성 또한 현저히 완화시킬 수 있습니다. 극단적인 경우, 소규모 시스템(XP 예제에서 115 점의 기능 점수)의 초기 전달 시점에 확립된 RUP 기반 접근 방식을 수행한 후 여러 가지 발전적인 전체 RUP 주기(도입/인식(Inception), 정제(Elaboration), 구현/구축 및 전이(Transition) 시퀀스)를 수행함으로써 나머지 시스템을 XP 와 유사한 유지보수 모드로 전달하는 것으로 가정할 수 있습니다.

약 6 개월 정도 진행된 시점에서 다시 초기 전달이 XP 에 권장되는 시간 범위의 한계에 도달할 수 있지만 그러한 경우 RUP 에서 인식하여 릴리스 배치와 같이 XP 에서 다루는 내용에 대한 계획을 수립하고 시간 및 노력을 할당하도록 요구합니다.

이러한 내용을 바탕으로, XP 가 적합한 프로젝트의 특성에 대한 기타 제한사항을 추론할 수 있습니다. XP 에서는 고객과 개발 팀 간의 친밀한 관계가 매우 중요하므로 항상 고객이 현장에 존재해야 하며 스토리를 작성하고 릴리스를 정의해야 합니다. XP 에서는 또한 고객이 팀 내 실제 역할입니다. 즉, 역할을 수행하는 개인 또는 그룹은 별도 조달 회사에 속하거나 속하지 않을 수 있지만, 소프트웨어를 조달하는 주체를 대변할 수 있는 권한을 갖고 있습니다. 또한 본질적으로 요구사항을 갖고 있습니다.

이러한 유형의 관계는 일반적으로 외부 클라이언트와 공식 개발 계획을 맺은 경우보다 내부 개발 프로젝트에 적용됩니다. 이러한 환경에서는 2 개월의 릴리스 주기 또한 적합합니다. 2 개월 간격으로 회사 외부에 새 릴리스(결함 수정사항과 새 기능 모두 포함)를 배치하는 경우 물류와 관련된 많은 오버헤드가 발생됩니다.

이러한 유형의 완화(소규모 팀, 이전에 확립된 아키텍처, 형식적인 의식(ceremony) 배치를 줄인 내부 스타일 개발)를 통해 올바르게 사용자 조정된 하나 이상의 RUP 주기가 XP 개발과 유사한 영향력을 갖게 됩니다. 초기 릴리스 지속 기간은 XP 최적값보다는 약간 길지만 위험성이 낮습니다.

반대의 경우(대규모 팀, 처음 작성된 아키텍처, 전달 및 배치 제한조건을 적용하여 공식 계약을 체결한 개발 프로젝트), XP 의 확장 방법을 확인하기 어렵습니다. 보다 정확히 말해 이 문서의 소스에서 이러한 경우 XP 를 적용할 수 있다고 주장하지 않습니다. 반면에 RUP 는 이러한 프로젝트에 수반되는 형식성과 의식(ceremony)에 대처할 수 있도록 디자인되었습니다. 일반적으로는 XP 와 유사한 접근 방식(즉, 짝 프로그래밍 및 리팩토링과 같은 XP 기법 사용)을 대규모 RUP 프로젝트 *내부*에 임베드할 수 있지만 우선 아키텍처가 안정화되어야 합니다.

아티팩트

RUP 에는 100 개 넘는 아티팩트가 포함되며 이로 인해 소규모 프로젝트의 경우 무리한 사무 부담을 준다는 비판을 받고 있습니다. 그러나 이러한 견해는 다음과 같은 네 가지 관점에서 잘못된 견해입니다.

- 프로젝트는 모든 아티팩트를 생성하지 *않아도 됩니다.* 즉, 아티팩트를 선택하고 사용자 조정하는 것이 프로세스의 필수 파트입니다. RUP 는 생략할 수 있는 아티팩트와 사용자 조정할 수 있는 아티팩트에 대한 가이드라인을 제공합니다.
- 아티팩트(RUP 의 경우 아티팩트 *설명*)는 스펙 엔티티로서, 타스크로 생성될 정보를 지정합니다. 또한 이를 시스템적으로 수행하기 위해 아티팩트의 추상 *양식*을 설명합니다. 아티팩트의 RUP 특성을 모델, 모델 요소 또는 문서로 나타내는 이유는(RUP 아티팩트의 약 50%가 문서로 분류됨) 특정 *실현(realization)을 나타내기 위한 것은 아닙니다.*

  UML 모델은 목적 빌드 도구(예: Rational Rose)를 사용하여 캡처하고 나타낼 수 있습니다. 또는 일반적인 경우는 아니지만 단순 그래픽 도구를 사용하여 작성된 다이어그램이나 화이트보드의 스케치도 포함됩니다. 화이트보드의 경우 분실의 위험성이 있기는 하지만 RUP 관점에서 계속 아티팩트로 간주합니다.

  "문서"라는 용어는 기존 의미에 따라 특정 필수 세트의 단락 표제가 있는 별도 서면 기반의 중간 산출물을 연상시킵니다. 문서를 완성하려면 이러한 단락 표제를 모두 텍스트와 다이어그램으로 작성해야 합니다. 그러나 이는 RUP 문서의 한 가지 실현(realization)일뿐입니다.

  RUP 문서는 텍스트 및 다이어그램으로 구성되는 정보 세트입니다. RUP 는 유용한 정보 세트를 작성하기 위해 필요한 정보의 특성을 지정합니다. 이러한 작업을 간편하고 시스템적으로 수행하려면 고려할 항목 및 문제를 나열하고 설명하는 템플리트를 연결해야 합니다. 이러한 템플리트를 직접적이고 공식적으로 사용함으로써 일반적으로 아티팩트를 문서 양식(전자 또는 서면)으로 실현할 수 있습니다. 많은 프로젝트에서 이 작업을 수행해야 하므로 이러한 방식으로 어느 정도 직접적으로 사용할 수 있는 템플리트 세트를 제공합니다. 그러나 이 작업은 필수 작업은 아닙니다. 즉, 템플리트에 제시된 모든 요소가 특정 프로젝트와 반드시 연관되어야 하는 것은 아닙니다. 또한 RUP 프로젝트는 아티팩트에 대한 적합한 실현(realization) 및 전달 메커니즘을 선택할 수 있습니다. 중요한 것은 아티팩트에 포함된 정보입니다.
- 프로세스의 모든 잠재적 중간 산출물은 프로세스 엔지니어와 프로젝트 관리자가 고려할 수 있도록 명확히 식별되어야 합니다. 따라서 해당 중간 산출물을 생략 또는 사용자 조정하는 의사 결정은 생략에 따른 결과를 정확히 이해하고 신중하게 내려야 합니다. 아티팩트 목록을 이러한 방식으로 완전하고 명확하게 작성함으로써 프로세스를 객관적으로 구성할 수 있습니다. 또한 초보자는 간과할 수 있지만 경험이 풍부한 관리자라면 언급되지 않은 경우라도 "고려할" 내용을 경험이 부족한 프로젝트 관리자가 파악할 수 있도록 함으로써 이들을 지원합니다. 또한 잘 정의된 문서 세트를 작성함으로써 고객 및 프로젝트 관리자가 전달될 내용과 관련 양식에 대해 조기에 동의하여, 종종 프로젝트 정지로 연결되는 불쾌한 논쟁을 피할 수 있습니다.
- RUP 는 여러 컴포지트 아티팩트에 대해 설명하며 또한 해당 컴포넌트를 아티팩트로 설명합니다. 이를 통해 필요한 경우 타스크 입력 및 출력을 자세히 설명할 수 있습니다. 해당 설명 내용에서, RUP 는 컴포지트(예: 디자인 모델)에서 중지할 수 있으며 클래스를 독립적인 아티팩트가 아닌 디자인 모델의 파트로 참조했습니다. 해당 파트를 아티팩트로 식별하는 경우 용도를 정확하게 설명할 수 있지만 동시에 전체 아티팩트 수가 증가하는 것을 감수하고 프로세스 메타 모델을 준수해야 합니다.

XP 는 과도한 아티팩트에 따른 문제가 없는 것처럼 보이지만 다음과 같은 두 가지 이유를 볼 때 이는 지나치게 단순화된 것입니다.

- XP 는 특정 유형의 개발에 맞게 *이미 사용자 조정*되어 있습니다. 즉, RUP 원칙의 서브세트를 특정 규모로 처리하므로 보다 적은 아티팩트를 필요로 합니다.
- XP 는 유저 스토리와 코드의 중요성을 강조하지만 기타 중간 산출물은 프로세스를 설명할 때 언급한 내용이므로 아티팩트 수는 생각보다 적지 않습니다.

XP 접근 방식을 고려할 때, 이러한 비교를 위해 소스 자료로 참조한 세 권의 책(프로세스에 대한 중요 자료)에서 "아티팩트"와 "중간 산출물"이 색인에 나타나지 않는다는 것은 그다지 놀라운 일은 아닙니다. 그러나 텍스트를 자세히 검토하면 아티팩트에 대한 내용을 확인할 수 있습니다. 다음은 몇 가지 예제입니다.

- 스토리
- 제한조건
- 타스크
- 기술 타스크
- 승인 테스트
- 소프트웨어 —코드
- 릴리스
- 메타포
- 디자인 —CRC, UML 스케치
- 디자인 문서 —프로젝트 종료 시점에 생성
- 코딩 표준
- 유닛 테스트
- 작업공간(개발 및 기타 기능)
- 릴리스 계획
- 반복 계획
- 회의 보고서 및 노트
- 전체 계획 —예산
- 진행상태 보고서
- 예상 스토리
- 예상 타스크
- 결함(및 연관된 데이터)
- 대화에서 추가되는 문서
- 지원 문서
- 테스트 데이터
- 테스트 프레임워크 도구
- 코드 관리 도구
- 테스트 결과
- 스파이크(솔루션)
- 타스크 작업 시간 레코드
- 메트릭 데이터
    - 자원
    - 범위
    - 품질

  - 시간
- 기타 메트릭
- 추적 결과

약 30 개의 아티팩트가 있으며 그 중 일부는 컴포지트입니다. 이 목록은 참조용이며 전체 목록은 아닙니다. XP 서적에서는 서적이 작성된 레벨로 이러한 아티팩트 대부분에 대해 설명하는 데 많은 시간을 할애하지 않으며, 이를 기대하지도 않습니다. 그러나 프로젝트에서 아티팩트를 실현해야 하는 경우 해당 컨텐츠 및 양식의 세부사항이 필요합니다. 프로젝트에서는 해당 작업을 즉시 수행할 수 있지만 실제 작업에서는 시간이 소요됩니다. 반대로 RUP 는 선행 지침을 제공함으로써 프로젝트 시간을 절약합니다.

XP 에서 모든 RUP 아티팩트가 필요하지 않은 이유

한 가지 이유는 XP 에 RUP 범위가 없기 때문입니다. 이는 다분히 계획적인 것입니다. 즉, XP 는 비즈니스 요구를 충족시키기 위한 프로그래밍 도구입니다. 비즈니스 요구의 발생 원인과 모델링, 캡처 및 합리화 방법은 XP 의 주요 관심사항이 아닙니다. XP 팀 구성원 중 고객은 요구사항의 스토리 처리를 XP 개발 팀에 제공한다는 점에서 비즈니스 가치의 중재자이기도 합니다. 스토리를 해당 양식으로 표현하는 방법은 XP 의 관심 사항이 아닙니다. 따라서 예를 들어, RUP 가 비즈니스 모델링 원칙에서 설명하는 내용은 XP 의 범위에 포함되지 않습니다(RUP 의 비즈니스 모델링에는 최대 14 개의 아티팩트가 포함됩니다). XP 는 고객에 대한 정규 릴리스를 갖는 프로세스에 대해 설명합니다. 이러한 릴리스 배치 물류는 개발 관련 관심사항이 아니므로 RUP 배치 원칙은 XP 의 범위에서 완전히 벗어나는 내용입니다(RUP 의 배치에는 최대 9 개의 아티팩트가 포함됩니다). 따라서 XP 를 준수하는 소규모 프로젝트의 경우 RUP 사용자 조정 시 최대 23 개의 아티팩트를 생략할 수 있습니다.

또 다른 이유로, XP 에서는 요구사항 및 디자인 캡처를 비교적 간단하게 수행할 수 있다고 알려져 있습니다. 요구사항은 유저 스토리로 캡처됩니다. 이 유저 스토리는 경우에 따라 분할되어야 하고 나중에 타스크로 분해됩니다. 또한 타스크는 본질적으로 디자인이므로 시스템에 대한 메타포를 염두에 두어야 합니다. 이 내용이 모든 시스템에 적용되는 것은 절대 아니지만 일부 시스템에는 확실히 적용됩니다. XP 의 경우 모든 시스템에 적용되지는 않습니다. XP 작성자 또한 이러한 주장을 쉽게 하지는 못했을 것입니다.

복잡한 대규모 시스템의 바람직한 동작은 유스 케이스와 같은 시스템적인 접근 방식 없이는 정확하게 나타내기 어렵습니다. 사람의 기억력은 정확하지 못하므로 고객과 개발자 간의 대화에 의존하여 복잡한 유저 스토리를 지속적으로 정제할 수도 없습니다. 또한 대규모 시스템의 복잡한 동작을 *실현하는 구조를 개발할 때 해당 시스템 아키텍처에 대한 다양한 추상적 보기를 구성하고 추론할 수 있는 기능을 활용할 수 있습니다. RUP 에서 요구사항(최대 14 개 아티팩트)과 분석 및 디자인 원칙(17 개 아티팩트)에서 설명하는 아티팩트를 이용하면 이러한 시스템의 다양성, 규모 및 복잡도에 대처할 수 있습니다.*

RUP 의 소규모 프로젝트 로드맵에서는 컴포지트 아티팩트를 참조하는 것만으로도 요구사항과 분석 및 디자인에 대한 아티팩트 수가 일곱 개로 감소합니다. 이는 속임수도 아니며 XP 와 동일한 방식으로 아티팩트를 처리합니다. 예를 들어, 소규모 프로젝트에서 실현될 수 있는 방식으로 인해 RUP 는 디자인 모델에 대해 다음과 같이 설명합니다.

> "디자인 모델은 개발자가 CRC 카드와 손으로 직접 작성한 다이어그램을 사용하여 디자인을 탐색하고 캡처하는 일련의 브레인스토밍 세션을 통해 발전됩니다. 디자인 모델은 개발자가 유용하다고 판단하는 경우에만 유지보수됩니다. 또한 디자인 모델은 구현과 일치되도록 유지되지 않고 참조용으로만 보관됩니다."

마지막으로, RUP 에서는 필요한 경우와 일부 하청 계약에 있어 프로젝트 관리의 높은 형식성(및 "의식(ceremony)")을 허용합니다.RUP 의 많은 프로젝트 관리 아티팩트(프로젝트 관리 원칙의 경우 15 개)는 컴포지트 아티팩트의 일부이며, 프로젝트의 형식성에서 요구하는 경우에만 별도 문서로 실현되어야 합니다. 예를 들어, RUP 의 소프트웨어 개발 계획에는 위험성 관리 계획 및 제품 적합성 계획과 같은 아티팩트가 "포함"됩니다. 보다 덜 정형화된 소규모 프로젝트에서는 소프트웨어 개발 계획의 하나 또는 두 개의

단락만으로도 이러한 계획에서 처리하는 문제를 처리할 수 있습니다. RUP 의 소규모 프로젝트 로드맵에서 프로젝트 관리 아티팩트의 수는 여섯 개로 감소합니다.

소규모 프로젝트의 아티팩트 비교

이제 소규모 프로젝트에 맞게 RUP 를 사용자 조정하고 아티팩트 요구사항 또한 사용자 조정하는 경우 일반적인 결과에 대해 알아봅니다. RUP 의 소규모 프로젝트 로드맵에서 아티팩트 수는 30 개(일부 배치를 생략하는 경우 26 개)로서 많은 수가 아닙니다. RUP 는 XP 가 명확하게 처리하지 못하는 내용을 명확히 나타낼 수 있으며 필요한 내용을 결정할 수 있도록 해주고 선택 방법을 알려줍니다. 그 세분성은 양쪽 모두 다르지만 중요한 점은 XP 가 처리할 수 있는 소규모 프로젝트 유형에 대한 RUP 의 아티팩트 수가 XP 의 경우와 비슷한 수준이라는 것입니다.

개략적인 XP 아티팩트와 RUP 아티팩트의 맵핑

XP	RUP 소규모 프로젝트 로드맵
스토리 대화에서 추가되는 문서	비전 용어집 유스 케이스 모델
제한조건	보충 스펙
승인 테스트 유닛 테스트 테스트 데이터 테스트 결과	테스트 모델
소프트웨어 —코드	구현 모델
릴리스	제품 릴리스 정보
메타포	소프트웨어 아키텍처 문서
디자인 —CRC, UML 스케치 타스크 기술 타스크 디자인 문서 —프로젝트 종료 시점에 생성 지원 문서	디자인 모델
코딩 표준	디자인 가이드라인 프로그래밍 가이드라인
작업공간(개발 및 기타 기능) 테스트 프레임워크 도구	 도구
릴리스 계획 예상 스토리 예상 타스크 반복 계획	소프트웨어 개발 계획 반복 계획
전체 계획 —예산	비즈니스 사례 위험성 목록 제품 적합성 계획
진행상태 보고서 타스크 작업 시간 레코드 메트릭 데이터: 자원, 범위, 품질, 시간 기타 메트릭 추적 결과 회의에 대한 보고서 및 노트	상태 평가
결함 및 연관 데이터	변경 요청
코드 관리 도구	형상 관리 계획 프로젝트 저장소

	작업공간
스파이크	프로토타입
	개발 사례 프로젝트 특정 템플리트

가이드라인

RUP 는 아티팩트와 연관지어 해당 아티팩트에 대한 자세한 정보인 가이드라인을 제공합니다. 이 가이드라인에는 해당 의미, 표시, 다른 아티팩트와의 관계, 컨텐츠 및 사용 방법이 포함됩니다. 또한 이 가이드라인은 실제 수백 페이지의 내용을 담고 있어 모두 인쇄하는 경우 부담이 따르므로 필요한 아티팩트에 대한 내용만 참조합니다.

XP 서적은 또한 해당 관계를 정확하게 모델링하지 않고 사례 및 아티팩트에 대한 많은 안내 정보를 제공합니다. XP 에 대한 자료도 그다지 적지는 않습니다. 작성 당시 총 600 페이지 분량의 서적 세 권이 나와 있는 상태였으며 700 페이지 정도 분량의 서적 두 권이 곧 출간될 예정입니다. 또한 리팩토링에 대한 서적 [Fowler99]이 400 페이지 이상의 분량으로 나와 있습니다.

이 밖에도 여러 웹 사이트에서도 XP 에 대한 내용을 다루고 있습니다. 해당 내용들은 각기 다른 관점에서의 XP 점검에서 경험 베이스에 대한 추가까지 중복되는 경향을 보입니다. 실제로 이를 통해 RUP 와 XP 간의 또 다른 차이점이 부각됩니다. 즉, RUP 는 제품이고 XP 는 제품이 아니라는 점입니다. XP 의 경우 서적이 출간되어 있기는 하지만 단일 소스가 없습니다. XP 를 가장 빨리 "습득"할 수 있는 방법은 상업적으로 이용이 가능한 영향력이 있는 전문가로부터 필요한 훈련을 받는 것입니다.[11]

[11] RUP 는 때때로 "자산"으로 묘사되기도 하지만 저작권 및 라이센스 계약을 준수하는 한 누구나 RUP 를 구매하여 해당 아이디어를 이용하고 추가할 수 있으며 해당 조직 또는 프로젝트와 관련이 없는 파트는 삭제할 수 있습니다. XP 의 서적 내용 또한 작성자 및 발행자가 소유하는 지적 재산권입니다. 차이점은 해당 소스에서 각각을 파악할 수 있는 정도입니다. 제품으로서의 RUP 는 완벽을 추구하지만 최신 XP 서적의 경우 집중 훈련 또는 컨설팅을 통한 보충 설명이 필요합니다.

타스크

RUP 는 입력 아티팩트를 사용 및 변환하고, 변경 또는 새 출력 아티팩트를 생성함으로써 역할이 수행하는 작업으로 "타스크"라는 용어를 정의합니다. RUP 는 이러한 타스크를 나열하여 RUP 의 정의에 따른 프로젝트의 "원칙" 또는 주요 "관심 영역"에 따라 분류합니다. 해당 원칙은 다음과 같습니다.

- 비즈니스 모델링
- 요구사항
- 분석 및 디자인
- 구현
- 테스트
- 배치
- 형상 및 변경 관리
- 프로젝트 관리
- 환경

타스크는 타스크로 생성하고 소비하는 아티팩트를 통해 시간과 관련이 있습니다. 즉, 타스크는 해당 입력을 사용할 수 있을 때 또한 해당 성숙 상태로 논리적으로 시작할 수 있습니다. 즉, 아티팩트 상태에서 허용되는 경우 제작자-소비자 타스크 쌍이 시간적으로 겹칠 수 있음을 의미합니다. 해당 시퀀스는 그다지 중요하지 않습니다.

RUP 의 타스크는 아티팩트 생성을 위한 올바른 프로세스를 명확하게 만들기 위한 것으로서, 생성 방법에 대한 정확한 안내 정보를 제공합니다. 타스크는 또한 프로젝트 관리자의 계획을 돕는 데 사용할 수 있습니다. RUP 에서 라이프사이클, 아티팩트 및 타스크를 통해 설명한 내용이 "우수 사례"입니다. 이는 예측 가능한 스케줄 및 예산으로 고품질의 소프트웨어를 빌드하는 것으로 증명된 소프트웨어 엔지니어링 원칙입니다.

RUP 는 타스크 및 타스크 관련 아티팩트를 통해 이러한 우수 사례를 지원하고 실현합니다. 또한 우수 사례는 RUP 를 통해 실행되는 주제입니다. XP 에서도 "사례" 개념을 이용하지만 RUP 의 우수 사례 개념과 정확히 일치하지는 않습니다.

XP([Beck00], 9 장 참조)는 소프트웨어 개발 보기를 특정 지원 사례에 따라 사용할 수 있고 구조화되는 네 가지 기본 타스크를 갖는 단순 보기로 나타냅니다.

- 코딩
- 테스트
- 청취
- 디자인

실제로, XP 의 타스크는 그 범위에서 RUP 의 타스크 보다 RUP 의 원칙에 가까우며 코딩, 테스트, 청취 및 디자인 이외에 XP 프로젝트에서 발생하는 상당 부분이 해당 사례의 정제와 적용에서 비롯됩니다.

RUP 타스크에 해당하는 XP 요소

XP 에도 RUP 와 같은 "타스크"가 있지만 정식으로 식별 또는 설명되어 있지는 않습니다. 예를 들어, [Jeffries01]의 4 장, *User Stories* 에는 "*Define requirements with stories, written on cards* " 표제가 있으며 이 장 전체에 유저 스토리의 개념과 생성 방법 및 생성 주체에 대한 안내 정보와 프로세스 설명이 함께 나와 있습니다. 또한 이 책에서는 XP 를 주요 표제([Jeffries01]의 경우 아티팩트와 활동에 초점을 맞춘 표제)별로 설명하며 다양한 규정 및 세부사항에 따라 "완료 요소"와 "생성 요소"를 설명합니다.

RUP 의 명확한 규정은 타스크와 해당 입출력(I/O)을 시스템적으로 처리하는 방법에 대한 완전성 및 높은 형식성을 기반으로 합니다. XP 의 경우 규정은 부족하지 *않지만* "간단한" 상태를 유지하기 위해 형식성 및

세부사항은 생략됩니다. RUP 의 세부사항은 XP 를 프로젝트에서 구현할 때 추가되어야 합니다. 특수성의 부족은 강점도 약점도 아니지만 XP 에서 세부 정보가 부족한 것을 단순성과 혼동해서는 안됩니다. 프로젝트 참여자는 특정 시점에서 수행 작업을 파악해야 하며 이러한 경우 세부사항이 필요합니다.

원칙

RUP 의 원칙은 소프트웨어 개발과 관련된 중요 측면 또는 관심사항을 나타내는 특정 아티팩트 세트를 생성하는 타스크(및 연관 개념)의 콜렉션입니다. 이러한 사용법은 지식 또는 학습의 일환으로서 원칙의 사전적 정의와 잘 부합됩니다.

앞에서 설명한 대로, RUP 의 원칙에는 비즈니스 모델링, 요구사항, 분석 및 디자인, 구현, 테스트, 배치, 형상 및 변경 관리, 프로젝트 관리와 환경이 포함됩니다. 그러나 여기에는 조직 또는 비즈니스에서 일반적인 소프트웨어 시스템을 개발, 배치, 운영, 지원, 판매, 마케팅하거나 해당 시스템을 다룰 사람을 채용할 때 수행해야 하는 작업의 모든 측면이 포함되지는 않습니다. RUP 는 현재 예를 들어, 시스템 엔지니어링을 처리하지 않습니다. 또한 ISO 15504(예를 들어, 소프트웨어 획득 및 인적 자원 관리와 관련된 측면)와 같은 일부 국제 소프트웨어 프로세스 표준의 일부 요구사항만 처리합니다. 이는 선택의 문제로서 이러한 측면은 중요하기는 하지만 RUP 의 엔지니어링 초점에는 벗어나는 영역입니다.

XP 는 자체 영역을 보다 명시적으로 제한합니다. 즉, XP 는 네 가지 기본 활동인 코딩, 테스트, 청취 및 디자인(앞에서 RUP 원칙과 밀접한 연관이 있는 것으로 설명)으로 구성됩니다. 이러한 활동은 여러 사례를 사용하여 수행되며 이를 위해서는 RUP 의 일부 다른 원칙에 맵핑되는 다른 활동을 수행해야 합니다. XP 의 사례는 다음과 같습니다([Beck00]의 내용 인용).

- *"계획 게임*—비즈니스 우선순위와 기술적 예상을 결합시켜 다음 릴리스의 범위를 신속히 판별합니다. 계획이 현실에 미치지 못하면 계획을 갱신하십시오.
- *소규모 릴리스*—간단한 시스템을 프로덕션에 빨리 투입한 후 아주 짧은 주기로 새 버전을 릴리스합니다.
- *메타포*—전체 시스템의 작동 방법에 대한 간단한 공유 스토리를 통해 모든 개발을 안내합니다.
- *단순 디자인*—시스템은 언제든지 가능한 단순하게 디자인되어야 합니다. 초과된 복잡도는 발견되자마자 제거됩니다.
- *테스트*—프로그래머는 유닛 테스트를 계속 작성해야 하며 이 테스트는 지속적인 개발을 위해 완벽하게 실행되어야 합니다. 고객은 기능이 완료되었음을 예시하는 테스트를 작성합니다.
- *리팩토링*—프로그래머는 중복을 제거하고, 통신을 개선하며, 단순화 또는 유연성 추가를 위해 동작을 변경하지 않고 시스템을 재구성합니다.
- *짝 프로그래밍*—하나의 시스템에서 두 명의 프로그래머가 함께 전체 프로덕션 코드를 작성합니다.
- *공동 소유권*—누구나 언제든지 어디에서나 시스템의 모든 코드를 변경할 수 있습니다.
- *지속적 통합*—타스크가 완료될 때마다 하루에 여러 번 시스템을 통합하고 빌드합니다.
- *주 40 시간*—일반적으로 한 주에 40 시간 이하 동안 작업합니다. 두 주를 연속해서 초과 근무하지 않습니다.
- *현장 고객* —언제나 질문에 응답할 수 있는 실제 작업 중인 사용자를 팀에 포함시킵니다.
- *코딩 표준*—프로그래머는 코드를 통한 커뮤니케이션을 강조하는 원칙에 따라 모든 코드를 작성합니다."

단순 디자인 주제와 관련하여 유의해야 할 사항은 "추가 복잡도"라는 용어가 다소 주관적이므로 숙련된 사용자 이외에는 정의하기가 어렵다는 점입니다.

예를 들어, *계획 게임* 사례의 결과로 수행된 활동은 주로 RUP 의 프로젝트 관리 원칙에 맵핑됩니다. 그러나 비즈니스 모델링과 같은 일부 주제는 XP 의 범위에 포함되지 않습니다. 요구사항 *도출*은 XP 의 범위에서 크게 벗어납니다. 즉, XP 에서는 현장 고객이 팀 구성원으로서 스토리 양식으로 요구사항을 정의하고 제공합니다. 릴리스된 소프트웨어의 배치 또한 XP 의 범위에서 벗어납니다. 해당 배치의 규모와 유형으로 인해 XP 는 환경

원칙과 형상 및 변경 관리 원칙의 문제를 *매우* 가볍게 다룰 수 있습니다. 반면에 RUP 에서는 이를 매우 자세히 다룹니다.

RUP 에서 XP 사례 사용

XP 와 RUP 가 겹치는 원칙에서는 XP 에서 설명하는 특정 사례를 RUP 에서 채택할 수 있으며 일부는 이미 채택되어 있습니다. 예를 들어, 다음과 같은 경우입니다.

- *쌍 프로그래밍:* XP 에서는 단일 워크스테이션에서 함께 작업하는 두 명의 프로그래머가 프로덕션 코드를 생성해야 하며 이로써 코드 품질에 긍정적인 영향을 미치는 것으로 알려져 있습니다. 또한 해당 스킬을 확보함으로써 보다 긍정적인 영향을 미칠 수 있습니다. RUP 는 코드 생성 기술을 이처럼 세부적으로 설명하지 않으며 RUP 기반 프로세스에서 짝 프로그래밍을 사용할 수 있습니다. RUP 에서는 현재 이 사례와 *테스트 우선 디자인 및 리팩토링*(아래 참조)에 대한 일부 정보를 백서 양식으로 제공합니다. RUP 에서 반드시 이 사례를 사용해야 하는 것은 아니며 이를 명시할 필요도 없습니다.

  *산업적 관점*에서의 이점에 대한 증거도 미약하며 관련 사례도 많습니다. [Nosek98] 및 [Williams00]의 연구에서는 혼자 작업하는 개인과 짝 프로그래밍을 비교했지만 올바른 팀 작업 방식은 아닙니다. 개방적인 커뮤니케이션 문화를 갖고 있는 팀 환경에서는 개인이 동료(및 팀 리더 또는 관리자)에게 자유롭게 질문하고 정의된 프로세스에 따라 작업을 수행할 수 있으므로 총 라이프사이클 비용에 대한 영향 관점에서 짝 프로그래밍의 이점을 분간하기 어려울 것으로 추측할 수 있습니다. 올바르게 운영되고 있는 팀에서는 강요하지 않아도 사람들이 함께 모여 매우 자연스럽게 문제점을 논의하고 해결하기 때문입니다.

  [Beck00]에서는 "짝 프로그래밍의 또 다른 강력한 기능이 일부 사례의 경우 반드시 짝 프로그래밍 방법을 사용해야 한다는 점에서 사람과 프로세스에 대해 약간 부정적인 시각을 나타냅니다. 즉, 작업자가 스트레스를 받으면 테스트 작성을 건너뛰고 리팩토링을 연기합니다." 이는 일반적으로 보다 나은 결과로 연결되는 사례이므로 이를 거부하는 사람은 없습니다.

  올바른 프로세스의 인스턴스화는 방해 작업이 아니며 "세부" 레벨에서 강제 실행되어야 한다는 의견은 바람직하지 않습니다. 그러나 짝을 이루어 하는 작업은 다음과 같이 경우에 따라 상대방에게 서로 이점을 제공할 수 있습니다.

  - 사람들이 친해지기 시작하는 팀 구성 초기
  - 새 기술에 서투른 팀
  - 숙련된 작업자와 초보자가 혼합된 팀

- *테스트 우선 디자인 및 리팩토링:* RUP 의 구현 원칙에 적용할 수 있는 우수한 기법입니다. 테스트 우선 디자인은 특히 요구사항을 세부적으로 명시하는 데 있어 효과적인 방법입니다.
- *현장 고객:* 많은 RUP 타스크는 현장 고객을 팀 구성원으로 포함시킴으로써 효율성 증대 관점에서 많은 이점을 얻을 수 있습니다. 고객을 이러한 방식으로 참여시킴으로써 많은 중간 인도물(특히 문서)에 대한 요구를 줄일 수 있습니다.

  XP 에서는 코드 이외의 캡처 대상에 대한 언급을 피하며 바람직한 커뮤니케이션 매체로 대화를 강조합니다. 이를 위해서는 지속성과 친숙성이 반드시 필요합니다. 소규모 시스템의 경우라도 시스템을 전이해야 하는 경우 다른 요소도 생성해야 합니다.

  XP 에서는 결국 추가 정보 형태로 이를 허용합니다. 예를 들어, 프로젝트 종료 시 디자인 문서가 생성됩니다. 실제로, XP 에서는 문서 또는 기타 아티팩트 생성을 금지하지 않으며 단지 언급하지 않을 뿐입니다. 즉, 실제로 필요하고 사용되는 대상만 생성하도록 명시합니다. RUP 도 마찬가지입니다. 다만 지속성 및 친숙성이 이상적이지 않은 경우 필요*할 수* 있는 내용을 나열하고 설명합니다.

- *코딩 표준:* RUP 에는 일반적으로 필수로 간주되는 아티팩트(프로그래밍 가이드라인)가 있습니다(사용자 조정의 주요 구동 요소가 되는 대부분의 프로젝트 위험성 프로파일이 이렇게 수행됩니다).
- *지속적인 통합:* RUP 는 반복에서 서브시스템 및 시스템 레벨의 빌드를 통해 이를 지원합니다. 유닛 테스트를 마친 컴포넌트는 새 시스템 컨텍스트에서 통합되고 테스트됩니다.

확장되지 않는 XP 사례

그러나 일부 사례는 확장되지 않으므로(XP 에서는 확장 가능성을 주장하지 않음) 해당 사례 사용 시 RUP 의 이 조건을 적용합니다. 예를 들어, 다음과 같습니다.

- *공동 소유권:* 소규모 팀에서는 소규모 시스템 또는 대규모 시스템의 서브시스템을 담당하는 구성원이 해당 코드를 모두 잘 알고 있는 것이 좋습니다. 모든 팀 구성원에게 코드를 변경할 수 있는 권한을 동등하게 부여할지 여부는 시스템 또는 서브시스템의 특성과 복잡도에 따라 결정됩니다. 일반적으로 현재 코드 세그먼트 작업을 수행하는 개인(또는 짝)이 수정하도록 하는 것이 보다 빠르고 안전합니다.

  올바르게 작성된 코드라도 특히 알고리즘이 복잡한 경우 시간이 경과함에 따라 이해하기가 어려워집니다.

- *리팩토링:* 대규모 시스템의 경우, 잦은 리팩토링으로 아키텍처의 부족을 보완할 수 없습니다. [Beck00]에서는 XP 의 디자인 전략이 등산(hill-climbing) 알고리즘과 유사하다고 설명하고 있습니다. 단순한 디자인을 가져와 이 디자인을 조금 더 복잡하게 만든 후 조금 더 단순하게 만들고 또 다시 조금 더 복잡하게 만듭니다. 등산(hill-climbing) 알고리즘의 문제점은 소규모 변경이 아닌 대규모 변경만으로 상황을 개선할 수 있는 로컬 최적 조건에 도달한다는 것입니다."

  RUP 에서 아키텍처는 "높은 산"에 대한 보기 및 액세스를 제공하여 복잡한 대규모 시스템을 추적할 수 있습니다.

- *메타포:* 복잡한 대규모 시스템의 경우 단순히 메타포로서의 아키텍처는 충분하지 않습니다. RUP 는 [Beck00]의 설명대로 단지 "큰 상자와 연결"이 아닌, 아키텍처에 대한 보다 자세한 설명 프레임워크를 제공합니다.
- *빠른 릴리스:* 고객이 새 릴리스를 승인하고 배치할 수 있는 속도는 일반적으로 비즈니스 영향과 상관된 여러 요인(예: 시스템의 크기)에 따라 결정됩니다. 일부 시스템 유형의 경우 2 개월 주기는 너무 짧을 수 있습니다. 예를 들어, 배치 계획에서는 이를 금지할 수 있습니다.

또한 처음에는 RUP 에서 잠재적으로 사용할 수 있는 올바른 사례로 보이는 경우라도 실제 적용 시에는 일반적으로 다음과 같은 약간의 정제(Elaboration) 및 주의가 필요합니다.

- *단순 디자인:* XP 는 상당히 기능을 기반으로 합니다. 즉, 유저 스토리를 선택하고 타스크로 분해한 후 구현합니다. [Beck00]에 따르면 올바른 소프트웨어 디자인의 필수 특성은 다음과 같습니다.

  1. 모든 테스트를 실행합니다.
  2. 중복 로직이 없습니다.…
  3. 프로그래머에게 중요한 모든 의도를 나타냅니다.
  4. 클래스와 메소드를 최소 규모로 유지합니다."

  XP 에서는 현재 필요하지 않은 요소는 추가하지 않습니다. 따라서 RUP 에서 비기능적 요구사항으로 분류되는 유형을 처리하는 데 있어 로컬 최적화 문제점과 유사한 문제점이 발생합니다. 이러한 요구사항은 고객에게 비즈니스 가치를 제공하지만 스토리로 표현하기에는 어렵습니다. XP 의 제한조건이 이 카테고리에 포함됩니다.

  RUP 는 디자인 시 필요 이상으로 설명하는 방법을 지원하지 않고 아키텍처 모델(비기능적 요구사항을 충족시키기 위한 핵심 중 하나인 아키텍처 모델)을 사용하여 디자인하는 방법은 지원합니다.

  RUP 또한 "단순 디자인"이 모든 테스트를 실행해야 한다는 점에서 XP 와 같습니다. 그러나 이러한 경우 소프트웨어가 비기능적 요구사항을 충족시키는지 여부를 확인하기 위한 테스트가 포함됩니다. 즉, 시스템 크기와 복잡도가 증가하거나 새 아키텍처가 생성되거나 비기능적 요구사항이 부담이 될 때 주요

문제로 대두됩니다. 예를 들어, 이기종 분산 환경에서의 작동을 위한 데이터 마샬링(marshalling)에 대한 요구는 코드를 지나치게 복잡하게 만들지만 이 요구는 프로그램 전체에서 여전히 필요합니다.

- *주 40 시간:* XP 와 같이, RUP 에서도 작업 시간이 만성적으로 초과되지 않도록 제안합니다. XP 는 작업 시간에 대한 여러 허용 오차를 인정하여 엄격한 40 시간 제한을 제시하지 않습니다. 소프트웨어 엔지니어는 추가 보상 없이 단지 특정 작업의 완성에 따른 만족감을 얻기 위해 장시간 동안 일하는 것으로 유명하므로 관리자가 임의로 이를 중지시킬 필요는 없습니다.

  그러나 관리자가 이를 이용하거나 강요해서는 안됩니다. 또한 관리자는 보상은 하지 않더라도 *실제* 작업 시간에 대한 메트릭을 항상 수집해야 합니다. 특정 작업자에 대해 기록한 작업 시간이 장기간 너무 긴 경우 조사가 필요합니다. 그러나 이러한 문제가 발생하는 경우 나머지 팀 구성원이 가질 수 있는 우려를 고려하여 관리자와 해당 개인이 해결해야 합니다. 40 시간은 강제 조항이 아닌 안내 조항일 뿐입니다.

역할

RUP 에서는 역할에 의해 타스크가 수행됩니다.[12] 또한 역할은 특정 아티팩트에 대한 책임을 집니다. 책임을 맡은 역할은 일반적으로 아티팩트를 작성하고 다른 역할(허용된 경우)이 작성한 변경사항으로 아티팩트가 손상되지 않도록 합니다. RUP 의 역할은 개인 또는 그룹에 의해 수행될 수 있습니다. 또한 개인 또는 그룹은 여러 역할을 수행할 수 있습니다. 역할은 조직에서 단일 위치 또는 "슬롯"으로 맵핑되지 않아도 되며 조직 단위에 대한 역할 맵핑 또한 다 대 다 맵핑이 가능합니다.

RUP 역할

RUP 는 총 30 개의 역할을 정의합니다. 그러나 하나의 역할(예: 소프트웨어 설계자)이 반드시 하나의 원칙에 한정될 필요는 없고 원칙이 주로 속하는 역할만 나타내므로 역할과 원칙이 정확히 일치하지는 않습니다.

- 비즈니스 모델링에는 세 가지 역할이 있습니다.
- 요구사항에는 다섯 가지 역할이 있습니다.
- 분석 및 디자인에는 여섯 가지 역할이 있습니다.
- 구현에는 세 가지 역할이 있습니다.
- 테스트에는 두 가지 역할이 있습니다.
- 배치에는 네 가지 역할이 있습니다.
- 형상 및 변경 관리에는 두 가지 역할이 있습니다.
- 프로젝트 관리에는 두 가지 역할이 있습니다.
- 환경에는 세 가지 역할이 있습니다.

RUP 의 역할이 많은 이유

RUP 의 역할은 타스크를 분할하고, 역할을 수행하는 데 필요한 스킬과 역량을 세분화하기 위해 사용되며 이를 통해 역할을 수행할 인력을 선택할 수 있습니다. 이러한 분할 레벨은 또한 역할의 중요성이 변경될 때 새로운 조직 직무를 쉽게 식별할 수 있도록 해줍니다. 예를 들어, 소규모 비정규 프로젝트의 경우, 한 사람이 프로젝트 관리자와 형상 관리자의 역할(RUP 역할)을 잘 수행할 수 있습니다. 반면에 대규모 정규 군사 항공 우주 프로젝트의 경우, 형상 관리자 작업은 매우 전문화되고 번거로우므로 소규모 팀을 배정할 수 있습니다. RUP 는 역할을 이러한 방식으로 설명함으로써 이 맵핑을 용이하게 합니다.

XP 역할

[Beck00]에서는 XP 에 적용 가능한 일곱 가지 역할을 식별한 후 역할의 책임, 스킬 및 해당 역할을 수행할 사람에게 필요한 특성에 대해 설명합니다. 이러한 역할의 예는 다음과 같습니다.

- 프로그래머
- 고객
- 테스터
- 추적자
- 코치
- 컨설턴트
- 총 책임자

[12] 보다 정확히 말하면 역할을 *수행하는 팀 또는 개인에 의해 타스크가 수행됩니다.*.

이 역할은 일부 다른 XP 서적에서 역할이 수행하는 타스크에 대한 정제와 관련하여 언급되어 있습니다.

XP 와 RUP 의 역할 수 차이는 쉽게 설명할 수 있습니다.

- XP 는 RUP 원칙 중 일부에만 적용됩니다.
- XP 역할은 실제로 RUP 역할보다는 직무에 가깝습니다. 예를 들어, XP 의 프로그래머는 실제로 여러 RUP 역할(구현자, 통합자 및 코드 검토자 역할)을 수행하며 이러한 역할은 각각 조금씩 다른 역량을 필요로 합니다.

RUP 역할을 소규모 프로젝트로 맵핑하는 경우(RUP 의 소규모 프로젝트 로드맵 참조), 맵핑되는 XP 와 유사한 역할(직무)의 수는 30 개보다 훨씬 줄어듭니다. 소규모 프로젝트 로드맵에서 직무의 수는 다섯 개입니다(다음 표 참조).

ABC 회사의 소규모 프로젝트에 맵핑되는 RUP 역할

ABC 회사의 직책	RUP 역할
프로젝트 관리자	프로젝트 관리자 프로세스 엔지니어 배치 관리자 요구사항 검토자 아키텍처 검토자
ABC 회사 임원	프로젝트 검토자 이해 당사자(stakeholder) 요구사항 검토자
책임 프로그래머	시스템 분석가 요구사항 지정자 사용자 인터페이스 디자이너 소프트웨어 설계자 디자인 검토자 프로세스 엔지니어 도구 전문가 형상 관리자 변경 제어 관리자 *이 밖에* 프로그래머와 동일한 역할
프로그래머	디자이너 구현자 코드 검토자 통합자 테스트 디자이너 테스터
보조 관리자	*책임을 맡은 업무:* • *"소규모 프로젝트" 웹 사이트 유지보수* • 프로젝트 *관리자의 활동 계획 또는 스케줄링 지원* • *아티팩트 변경사항 제어와 관련한* 변경 제어 관리자 *역할 지원*

	• *이 밖에 필요에 따라 다른 역할에 대한 지원을 제공할 수 있습니다.*

결론

XP 는 실제로 RUP 와 같은 유형이 아닙니다. RUP 는 특정 프로세스가 구성된 후 인스턴스화될 수 있는 프로세스 프레임워크입니다. RUP 는 *반드시* 구성되어야 하며 실제로 RUP 에 정의된 필수 단계입니다. 엄밀히 말해, XP 는 사용자 조정된 RUP 버전과 비교해야 합니다. 즉, XP 가 명시적으로 설정(및 추론 가능)하는 프로젝트 특성에 맞게 사용자 조정된 RUP 와 비교해야 합니다. 이렇게 사용자 조정된 RUP 프로세스는 일부 XP 사례(예: 짝 프로그래밍, 테스트 우선 디자인 및 리팩토링)를 수용할 수는 있지만, 아키텍처, 추상(모델링 시), 위험성 및 시간에 따라 다른 구조(단계 및 반복)의 중요성을 강조한다는 점에서 XP 와 다릅니다.

RUP 는 프로세스 구현/구축(Construction) 시 그 규모와 유형에서 XP 범위를 벗어나는 프로젝트를 수용할 수 있도록 허용합니다. RUP 는 구현 시 필요에 따라 아티팩트, 인도물, 형식성, 규정, 의식(ceremony) 또는 기타 "가중치" 측정에서 복잡하거나 간단할 수 있는 프로세스 집합의 전체 *설명*이라는 점에서만 복잡하다고 할 수 있습니다. XP 는 간단한 프로세스의 구현을 추구한다는 점에서는 간단하다고 할 수 있지만 XP 의 설명은(최소한 책의 내용으로는) 충분하지 않습니다. 따라서 XP 구현에서는 항상 발견, 고안 또는 정의해야 하는 내용이 있으며 RUP 와 비교할 때 설명 자료 또한 충분하지 않습니다.

그러나 이러한 상황은 곧 변하게 될 것이며 실제로 두 권의 책이 이미 발행되었습니다. 그 중 [Succi01]은 512 페이지에 달합니다. 그러나 현재로는 두 접근 방식 간의 채택 노력 프로파일이 다릅니다. 즉, RUP 는 훈련 요구사항 및 프로세스 사용자 조정 모두에 있어 많은 선행 노력을 기울입니다. 조직 또한 일반적으로 프로젝트의 특정 유형 및 규모에 있어 조직 전체에 RUP 를 적용할 수 있도록 사용자 조정하며 여러 프로젝트에 해당 결과를 사용합니다. XP 의 경우 일부 선행 훈련이 필요하지만 나머지 채택 노력은 XP 작업에 필요한 것으로 입증된 모든 해당 보조 사항을 확장하고 캡처함으로써 프로젝트에 분산됩니다. XP 는 "회사 메모리" 캡처에 대한 명확한 동기를 부여하지 않아 채택 조직의 직원 교체에 따른 취약성에 노출되어 있습니다(해당 프로세스 경험을 저장하지 않는 경우).

추가 규정 없이 RUP 를 중량, XP 를 경량으로 레이블링하면 각각의 개념과 의도된 수행 작업이 모호해져서 둘 모두에 해가 됩니다. 또한 평가 절하하기 위해 이를 사용하는 경우 이는 의미없는 태도일 뿐입니다. 이는 "복잡" 또는 "간단"한 프로세스로서의 구현이며 환경 요구에 따라 복잡 또는 간단하게 구현되어야 합니다.

XP 는 원칙을 따르는 자유 양식이 아닙니다. 따라서 XP 는 소프트웨어 개발의 특정 측면과 값을 전달하는 방식에만 초점을 맞추며 이를 달성할 방식을 세부적으로 규정합니다.

RUP 의 적용 범위는 이보다 훨씬 넓으며 그 깊이는 정확한 "크기"를 나타내는 정도입니다. 그러나 세부 프로세스 레벨에서 볼 때 RUP 는 XP 와 달리 때때로 똑같이 유효한 대안을 허용하고 제공합니다(예: 짝 프로그래밍 사례). 이는 XP 를 비난하기 위한 의도가 아니라, 이름이 의미하듯 XP 의 초점이 얼마나 좁혀졌는지를 보여줄 뿐입니다.

참조

[Beck00] *Extreme Programming Explained,* Kent Beck, Addison-Wesley, 2000

[Beck01] *Planning Extreme Programming,* Kent Beck, Martin Fowler, Addison-Wesley, 2001

[Boehm00] *Software Cost Estimation with COCOMO II,* Barry W. Boehm et al, Prentice Hall PTR, 2000

[Fowler99] *Refactoring: Improving the Design of Existing Code,* Martin Fowler et al, Addison-Wesley, 1999

[Jeffries01] *Extreme Programming Installed,* Ron Jeffries, Ann Anderson, Chet Hendrickson, Addison-Wesley, 2001

[Kruchten00] *The Rational Unified Process, An Introduction, Second Edition,* Philippe Kruchten, Addison-Wesley, 2000

[Martin01] *Extreme Programming in Practice,* Robert C. Martin, James W. Newkirk, Addison-Wesley, 2001(현재 미출간)

[Nosek98] *The Case for Collaborative Programming,* John T. Nosek, *Comm. ACM,* Vol. 41, No. 3, 1998, pp. 105-108

[Succi01] *Extreme Programming Examined,* Giancarlo Succi, Michele Marchesi, Addison-Wesley, 2001(현재 미출간)

[Williams00] *Strengthening the Case for Pair Programming,* Laurie Williams, Robert R. Kessler, Ward Cunningham, Ron Jeffries, IEEE Software, Vol. 17, No. 4, 2000, pp. 19-25

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