http://dx.doi.org/10.7236/JIWIT.2012.12.4.119
JIWIT 2012-4-15

동시개발 방법을 적용한 단일화된 프로세스

Applying The Concurrent Development Approach To Unified Process

최명복*, 이상운**

Myeong-Bok Choi, Sang-Un Lee

요 약 최근 들어 소프트웨어 시스템은 점차적으로 복잡해지고 있으며, 고객(customer)은 보다 빠른 개발을 요구하고 있다. 전통적인(traditional) 순차적 접근법 (Sequential Approach)으로는 이러한 압력에 효과적으로 대처할 수 없어 대안으로 반복적 접근법 (Iterative Approach)이 적용되고 있다. 대표적인 반복적 접근법으로는 래쇼날의 단일화된 프로세스 (Rational Unified Process, RUP)가 있다. 그러나 RUP의 표준화된 수행방법은 단계, 반복과 활동들을 모두 순차적으로 수행하는 형태이다. 그 결과, 하나의 반복에서 수행된 하나의 활동은 다음 반복의 해당 활동이 수행될 때까지 기다려야 하는 인력낭비 현상이 발생한다. RUP를 수행하는 방법으로는 선형 접근법, 순차적 접근법, 중첩된 반복 접근법과 Time-boxed 반복 접근법이 제안되었다. 그러나 이들 방법은 인력낭비 현상 또는 적용시 프로젝트 관리의 어려움이라는 문제점을 갖고 있다. 본 논문은 활동들을 동시에 수행하는 방법을 제안하였다. 동시개발 접근법은 인력낭비 현상을 방지할 수 있으며, 프로젝트 관리의 어려움도 해결할 수 있는 장점을 갖고 있다.

Abstract Recently, the software system is getting complicating and the customers are requiring faster development. For the traditional sequential approach can't against this problem iterative approach is used instead. For the representative iterative approach, there is RUP (Rational's Unified Process). However, RUP standard practical methods are phase, iteration, and disciplines, sequentially. As a result, there's some waste of manpower when a discipline is executed in an iteration, it has to wait till the next same discipline is executed. There are linear approach, sequential approach, overlapped iteration approach, and time-boxed iteration for the efficient execution of RUP. However, they have some problems such as waste of manpower or difficulty in the project management. This paper suggests a method about how to execute the disciplines as a concurrent type. The concurrent approach prevents the waste of manpower and solves the difficulty of project management.

Key Words : Linear Approach, Sequential Approach, Overlapped Iteration Approach, Time-boxed Iteration Approach, Concurrent Approach

Ⅰ. 서 론

최근 들어 소프트웨어 시스템은 점차적으로 복잡해지고 있으며, 고객(customer)은 보다 빠른 개발과 개발된 제품을 보다 일찍 얻기를 원하고 있다. 전통적인 폭포수 프로세스 (Waterfall Process)는 이러한 요구사항을 만족

*종신회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
**정회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
접수일자 : 2012년 5월 10일, 수정완료 : 2012년 6월 25일
게재확정일자 : 2012년 8월 10일

Received: 10 May 2012 / Revised: 25 June 2012
Accepted: 10 August 2012
*Corresponding Author: cmb5859@gmail.com
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

시킬 수 없다. 따라서 이의 대안으로 반복 프로세스 (Iterative Process)가 점차 널리 적용되고 있다. 반복 프로세스의 대표적인 방법으로 레쇼날의 반복 프로세스 (Rational Unified Process, RUP)가 있다.[1]

RUP는 단계 (Phase), 반복 (Iteration)과 활동들 (Disciplines)로 분류되며, 하나의 반복은 폭포수 프로세스의 단계들로 구성되어 있어 미니 폭포수 프로세스 (Mini-Waterfall Process)라고도 한다. 따라서 하나의 반복이 종료되면 실행 가능한 서브 프로젝트가 고객에게 양도되어 고객의 추가적인 요구사항을 피드백 시킬 수 있다.[2]

RUP를 실제 적용할 경우, 활동들, 반복들을 어떤 방법으로 적용할 것인가가 문제로 발생한다. 래쇼날에서는 하나의 반복에서 Business Modeling, Requirements, Analysis&Design, Implementation, Test와 Deployment 활동들을 순차적으로 수행하여 해당 반복을 종료한다. 이어서 다음 반복에서도 동일한 순서로 각 반복들을 수행하는 "순차적 접근법 (Sequential Approach) 또는 캐스케이드 접근법 (Cascade Approach)"을 표준화된 적용방법으로 제시하고 있다.[3] 또한, 각 단계와 반복들을 종료한 시점에서 다음 단계나 반복을 진행할지 여부를 판단하는 명백한 이정표 (Milestone)를 적용하여 프로젝트 관리를 수행한다.

순차적 접근법으로 RUP를 적용할 경우, 하나의 반복에서 수행된 특정 활동이 종료된 후 다음 반복의 해당 활동이 수행될 때까지의 기간 (RUP에서는 하나의 반복을 2주에서 6주로 설정하고 있다.) 동안 아무런 활동을 수행하지 못해 인력낭비 현상을 초래한다. 이러한 인력낭비를 해소하는 최선의 방법은 개발팀 모두가 모든 활동들을 수행할 수 있는 전문가들로 구성되어 있어야만 한다. 그러나 소프트웨어 개발에 필요한 모든 활동들을 전문적으로 수행할 수 있는 인력은 현실적으로 거의 없는 실정이다. 또한, 대형 프로젝트의 경우 일반적으로 하나의 활동만을 수행하는 전문인력 들로 팀을 구성하기 때문에 인력낭비 현상은 더욱 큰 해결과제로 남는다.

RUP를 수행하는 방법으로는 폭포수 프로세스와 동일한 선형 접근법 (Linear Approach), RUP의 표준 접근법인 순차적 접근법, 중첩된 반복 접근법 (Overlapped Iteration Approach), Time-boxed 반복 접근법 (Time-boxed Iteration Approach)가 제안되었다.[3,4] 선형 접근법과 순차적 접근법은 프로젝트 관리는 용이하나 인력낭비 현상이 발생하며, 중첩된 반복 접근법과 Time-boxed 반복 접근법은 반복과 단계들을 명백히 구분하는 이정표 개념이 없이 미해결 문제들을 계속적으로 작업함으로 인해 프로젝트 관리의 어려움과 더불어, 다수의 선행 작업으로 인해 빈번한 요구사항 변경이 발생할 경우 재작업으로 인한 인력낭비 현상이 발생할 수 있다.

본 논문에서는 RUP를 적용함에 있어 인력낭비 현상을 최소화시키고, RUP의 근본 취지인 이정표 개념을 손상시키지 않는 개발 접근법을 제안한다. 2장에서는 RUP의 기본 개념과 더불어 단계, 반복과 활동들을 수행하는 방법을 살펴본다. 또한, RUP를 실제 수행하는 방법들에 대해 고찰해 보고 문제점을 제시한다. 3장에서는 동시개발 방법에 기반하여 RUP를 적용하는 기법을 제시한다.

II. 단일화된 프로세스의 적용

1. 단일화된 프로세스

최근 들어 소프트웨어 시스템은 점차적으로 복잡해지고 있으며, 고객은 보다 빠른 개발을 요구하고 있다. 이러한 환경 변화는 시스템 개발팀에게 압력을 증가시키고 있으며, 관리자는 개발비용과 개발 방법을 보다 신중하게 고려해야만 한다. 개발방법론 측면에서 볼 때, 모든 문제를 정의한 다음 모든 해법을 설계하고 소프트웨어를 코딩하여 마지막으로 제품을 시험하는 전통적인 순차적 접근법으로는 이러한 압력에 효과적으로 대처할 수 없다. 대안으로, 계속되는 정제 과정을 통해 문제에 대한 이해를 향상시키고 다수의 반복들을 통해 효율적인 해법을 점진적으로 성장시키는 반복적 접근법이 적용되고 있다.[5,6]

대표적인 반복적 접근법으로는 래쇼날의 단일화된 프로세스가 있다. RUP는 그림 1과 같이 가로축은 시간을 표현하며, 사이클, 단계들, 반복들과 이정표로 표현되는 프로세스의 동적 측면을 보이고 있다. 하나의 사이클은 제품에 대한 새로운 버전 생성으로, RUP는 하나의 개발 사이클에 대해 4개의 단계 (Inception, Elaboration, Construction과 Transition)를 거치면서 프로젝트를 개발한다. 각 단계 (Phase)는 다시 다수의 반복 (Iteration)들로 분할된다. 하나의 반복을 거치면 Mini-project (또는 컴포넌트)가 생성된다. 반복들은 계획된 시간이 경과하면 종료되는 Time-box 개념이지만, 단계들은 이정표 평

가기준을 만족할 때 종료되는 방식이다.[2]

세로축은 활동들, 산출물, 작업자와 작업 흐름을 어떻게 기술하는지에 대한 프로세스의 정적 측면을 표현하고 있다. 각 반복에서는 6개의 Core Disciplines (Business Modeling, Requirements, Analysis & Design, Implementation, Test와 Deployment)와 3개의 부수적인 Disciplines (Configuration & Change Management, Project Management와 Environment)들을 수행한다. 비록 각 반복에서는 대부분의 활동들 (Disciplines)을 포함하고 있지만 상대적인 노력과 주안점은 시간이 지남에 따라 변화한다.

Business Modeling은 Business Process를 Business Use Case와 Business Object Model로 기술하는 활동이며, Requirements는 시스템이 무엇을 수행하는지를 Vision Document, Use Case Model (Use Case Diagram과 Use Case Description)과 Supplementary Specification으로 표현하는 활동이다. Analysis는 요구사항 분석과 Architecture Design을, Design은 상세설계를, Implementation은 Coding, Unit Test와 Integration Test를 수행하는 활동이다. Test는 System Test를, Deployment는 소프트웨어 제품이 최종사용자에게 활용 가능함을 보이는 활동으로 Alpha와 Beta Test, Installation, 사용자 교육과 고객 수락시험 활동을 수행한다.

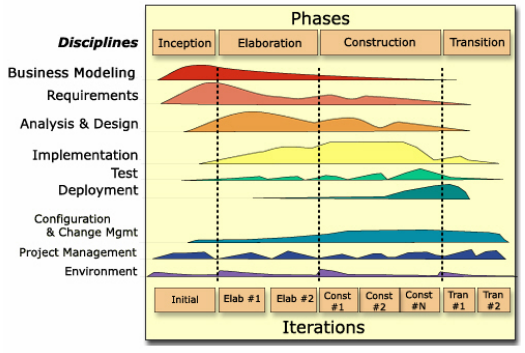


그림 1. 래쇼날의 단일화된 프로세스
Fig. 1. Rational Unified Process

RUP는 객체지향과 컴포넌트 기반의 소프트웨어 개발 분야에서 사실상의 표준화된 개발 프로세스 (또는 개발 방법론)가 되고 있다. RUP는 유스케이스 구동 (Use-case-driven), 아키텍쳐 중심(Architecture-centric)이며, 컴포넌트에 기반 (Component-based)하고 있으며, 반복적이고 점진적인 (iterative & incremental) 개발 프로세스이다.[4] RUP는 UML기반의 분석과 설계 표기법을 적용하고 있으며, Use-case Diagram과 Use-case Description을 이용하여 요구사항을 분석하고, 이에 기반하여 객체지향적으로 설계하는 방법으로 유스케이스 구동방식이라 한다. 프로젝트 개발은 먼저 개발 위험이 가장 큰 컴포넌트 또는 아키텍쳐를 먼저 개발하고 나머지 컴포넌트들을 개발하는 방식으로 아키텍쳐 중심이라 한다. 각 반복에서 핵심 개발 활동들을 반복적으로 적용되며, 다수의 반복을 거치면서 미니 프로젝트들이 통합되면서 점진적으로 시스템으로 성장하므로 "반복적이고 점진적인" 용어가 사용되었다.[7-9]

이 방법은 시스템을 서브시스템으로 분할하여 단계적으로 개발하고 납품하는 그림 2의 Staged Delivery Model[10-12]과 유사한 방법을 적용하고 있으나 다른 점은 프로젝트 초기단계부터 시스템을 서브시스템으로 분할하는 개념을 도입하고 있다. 여기서, 표현된 용어들은 전형적인 소프트웨어 개발단계들로 소프트웨어 개념정립 단계 (SC, Software Concepts), 요구사항분석단계 (RD, Requirement Analysis), 아키텍쳐 설계단계 (AD, Architecture Design), 상세설계단계 (DD, Detail Design), 구현단계 (I, Implementation 또는 Coding), 단위시험단계 (UT, Unit Test), 통합시험단계 (IT, Integration Test), 시스템시험단계 (ST, System Test)와 수락시험단계 (AT, Acceptance Test)를 마친 후 고객에게 납품 (Delivery)되는 단계를 의미한다.

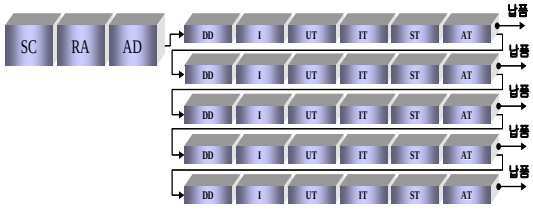


그림 2. 단계별 전달 모델
Fig. 2. Staged Delivery Model

Staged Delivery Model은 시스템의 모든 부분이 개발될 때까지 시스템의 어느 부분도 개발이 완료된 것을 가지지 못하는 Pure Waterfall Model의 문제점을 극복하기 위한 방법으로 RAD (Rapid Application Development)를 적용한 개념이다. 비록 초기 단계들은 Pure Waterfall Model을 따르지만 일단 아키텍쳐 설계 (AD)가 끝나면

납품 가능한 서브시스템으로 분할하고, 서브시스템에 대한 구현(상세설계, 코딩, 시험단계)과 납품이 순차적으로 이루어지는 관계로 점진적 구현이라고도 부른다. 하나의 서브시스템에 대한 개발(설계, 구현과 납품) 과정을 Stage라 한다. 서브시스템 개발 순서는 가장 높은 위험부분, 아키텍쳐의 근본 또는 가장 중요한 요구사항으로 선택된다. 다른 서브시스템들 간의 통합은 계속적으로 또는 개발의 마지막 부분에서 수행된다. Evolutionary Prototyping과 달리 구축하고자 하는 것과 언제까지 구축을 설정하고 언제까지 구축할 것인가를 정확히 알 수 있다. 이 방법은 정시에 납품하기 위해 해야 할 일이 무엇인지 모를 때, 제품 규모가 불확실 할 때, 개발 생산성이 불확실할 때, 개발상황 가시성과 통제가 필요할 때 적용하면 초기에 가시성을 제공할 수 있다.

대부분의 최근의 방법론들은 점진적과 반복적 프로세스를 거론하고 있지만 놀랍게도 실제로는 단지 몇 개의 프로젝트만이 이 방법으로 개발되고 있다.[13]

2. 단계, 반복과 활동 수행 방법

UP의 수행 계획은 개략적 계획 (Course-grained Plan)과 상세 계획 (Fine-Grained Plan)으로 구분하여 작성된다. 개략적 계획은 단계와 반복에 대한 수행 계획이며, 상세 계획은 하나의 반복에서 수행되는 활동들에 대한 구체적인 실행 계획이다.

RUP에서의 각 단계는 특정한 위험요소들에 초점을 맞춘다. 각 단계들에서 수행되는 활동들은 주로 위험을 감소시키는데 초점을 두고 있으며 프로젝트를 진행시킨다. 각 단계의 종료는 그림 3과 같이 이정표로 표현되며, 이는 결과를 평가하고, 프로젝트가 정상적으로 진행되고 있는지 확인하며, 프로젝트를 다음 단계로 진행시킬 것인지에 대한 결정을 하는 평가 지점으로 작용한다.[7] 하나의 단계가 종료된 후 이정표에 대한 평가를 수행하여 다음 단계로 이동할지 여부를 결정해야 하기 때문에 RUP에서는 단계들 간의 중첩 (Overlapping)은 허용하지 않는다.[14] 각 단계들의 목표, 산출물과 이정표에 대한 평가기준은 [5]을 참조하면 상세히 알 수 있으므로 여기서는 생략한다.

전형적으로 각 단계에 소요되는 개발기간과 노력, 각 단계에서 수행되는 활동들에 할당되는 노력의 비율은 표 1에 제시되어 있다.[15,16] Kruchten[15]은 각 활동들에 소요되는 노력의 양은 프로젝트에 따라 매우 다르지만 전형적으로 중형 규모의 프로젝트에 대한 초기 개발단계에서, 계획과 관리에 15%, 분석/요구사항에 10%, 설계/통합에 15%, 코딩/기능시험에 30%, 측정/평가/수락시험에 15%, 도구/환경/변경관리에 10%, 유지보수에 5%를 할당하는 방법을 제시하고 있다.

표 1. RUP의 개발기간과 노력 배분
Table 1. Development Period and Effort distribution of RUP

단계	개발 기간	노력	노력 배분[16]			
			관리자	설계자	개발자	평가자
Inception	10%	5%	50%	20%	20%	10%
Elaboration	30%	20%	10%	50%	20%	20%
Construction	50%	65%	10%	10%	50%	30%
Transition	10%	10%	10%	5%	35%	50%

다음으로 반복들을 수행하는 방법을 살펴보자. 각 단계들에서 수행되는 반복들의 수는 프로젝트의 규모와 특성에 따라 결정된다. 반복은 핵심 개발 활동들인 요구사항, 분석, 설계, 구현과 시험 과정을 거치는 사이클이 아니라 중요한 새로운 소프트웨어 버전을 생성하는 전형적으로 Time-boxed 미니 프로젝트이다.[7,8,17] 따라서, 하나의 반복은 미니 폭포수 프로세스 (Waterfall Process)로 해석될 수 있다. 각 반복의 종료 기준은 안정화되고 통합되고 시험된 실행 가능한 부분적으로 완전한 시스템을 납품 (내부적 또는 외부적)하는 것으로 결정된다. 반복은 전형적으로 2주에서 6주 사이로 기간을 결정한다.

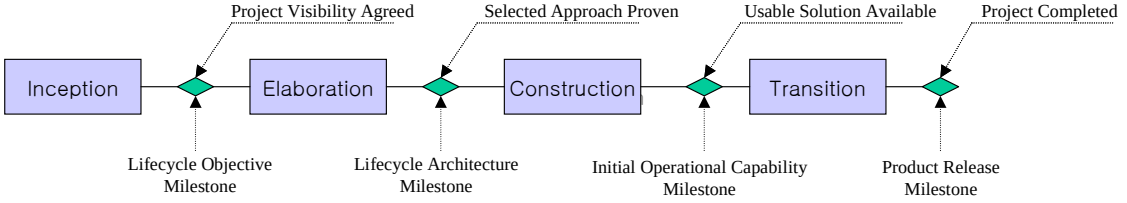


그림 3. 단계별 이정표
Fig. 3. Staged Milestone

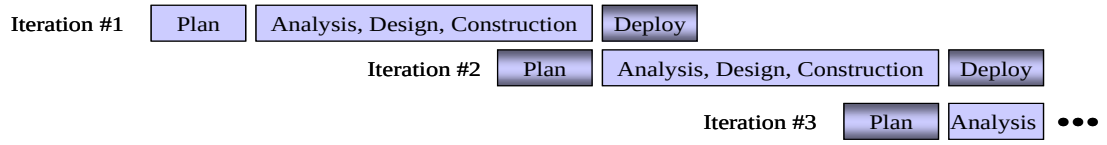


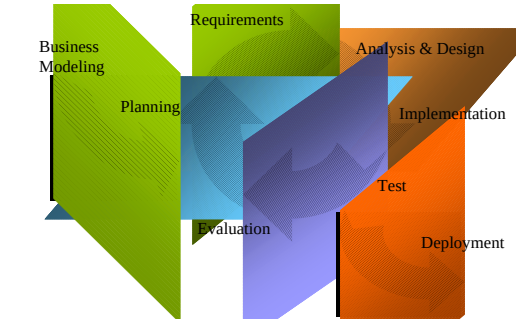
그림 4. 중첩된 반복 수행
Fig. 4. Overlapped Iteration Process

RUP의 공식적인 입장은 반복들 간에 중첩이나 동시 수행을 허용하지 않는다.[18] 그러면, 다수의 연속되는 반복들을 순차적으로 분리된 (이산적인) 방법으로만 수행해야 하는가? 아니면, 반복들 간에 중첩을 허용할 수 있는가? 이러한 문제에 대한 명확한 수행 지침은 프로젝트 수행 계획을 작성하고 실행하는데 중요한 기준이 될 수 있다. 또한, 프로젝트를 주어진 비용과 예산 범위 내에서 주어진 일정에 맞추어 되도록 빨리 프로젝트를 개발하기 위해 고려해야만 하는 중요한 문제이다.

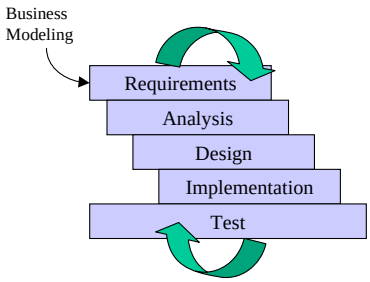
반복들 간에 순차적 또는 중첩 수행이 가능한가에 대한 연구 결과를 살펴보자. 먼저, 순차적으로 수행해야 한다는 의견으로는 Bittner[7], Larman[8], Eeles[17], Crain[18], Tassc[19], Cantor[20], Kroll[21]이 있다. 반면에, 그림 4와 같이 반복들을 약간은 중첩시킬 수 있다는 의견으로는 Wheaton[14], Kruchten[15]과 Rose[22]가 있다.

다음으로, 하나의 반복 내에서 수행되는 활동들의 수행 방법을 살펴보자. 하나의 반복 내에서 수행되는 활동들의 순서는 그림 5와 같이 다양하게 적용될 수 있다. RUP의 기본 방침은 하나의 반복 내에서의 활동들을 순차적으로 수행하는 것이다.[3,7,14,18] Ambler[23]과 Ambler[24]는 하나의 반복 내에서 수행되는 활동들은 하나의 활동이 종료된 후 다음 활동으로 넘어가기 전에 이전 활동들에서 수행된 미비점에 대해 재작업 후 다음 활동들을 수행하는 형태를 취하는 순차적 재작업 방법을 제안하고 있다. 활동들 간에 중첩을 허용한다는 의견으로는 Spence[9]와 Weaton[14]이 있다. 또한, 현실적으로는 동시에 수행될 수 있다는 의견으로는 Bittner[7], Tassc[19]과 Canter[20]가 있다.

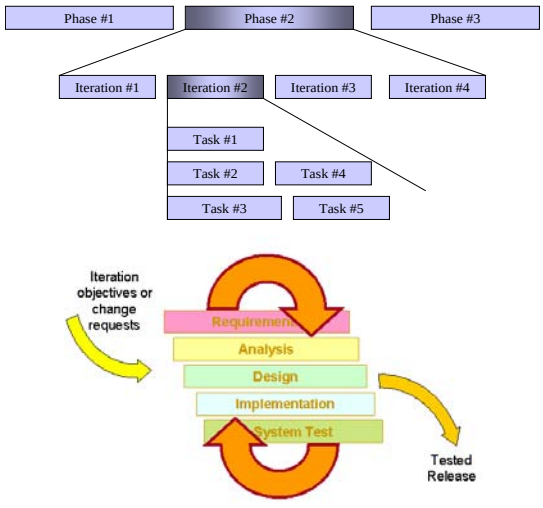
RUP를 적용하여 프로젝트를 개발할 경우, 단계 수행 방법에 대해서는 공통된 견해를 갖고 있지만, 반복과 활동들에 대해 이산적, 중첩 또는 동시 수행에 대한 다양한 의견이 공존하고 있어 프로젝트 수행 계획 작성에 애로가 발생하고 있다.



(a) 이산적인 순차적 수행



(b) 중첩된 순차적 수행



(c) 동시 수행

그림 5. 반복 내에서의 활동들 수행 방법
Fig. 5. Process Method of Disciplines within Iteration

3. RUP의 실제 수행 방법

RUP를 성공적으로 적용하려면, 프로젝트 전반에 걸친 책임이 있는 프로젝트 관리자 (Project Manager), 시스템과 아키텍쳐에 관한 책임을 지는 설계자 (Architect)와 UP를 적용할 책임이 있는 프로세스 공학자 (Process Engineer)가 협력과 대화를 통해 상호 균형을 이루어야 한다. 설계자는 시스템을 정의하고 프로세스 공학자는 시스템 납품에 요구되는 임무, 활동들과 산출물을 제안한다. 프로젝트 관리자는 시스템을 납품하기 위한 산출물에 요구되는 활동들을 실행하기 위한 자원을 적절히 할당하는 역할을 수행한다.

RUP의 단계, 반복과 활동들의 수행 방법들을 종합적으로 고려하여 실제 수행할 수 있는 방법을 고찰해보자. 이들 수행 방법에 대해 Crain[3]과 Alhir[4]가 있다. 먼저, Alhir[4]가 제안한 선형 접근법 (Linear Approach)으로 이는 폭포수 프로세스와 동일한 개념이다. 이 접근법은 그림 6과 같이 반복 개념은 없으며, 활동들이 순차적으로 적용된다. 이 경우는 기존의 폭포수 프로세스의 문제점들을 모두 갖고 있어 반복 개발의 이점이 전혀 없다. 이 접근법은 프로젝트 관리자가 너무 완고할 경우에 적용할 수 있다.

다음으로, RUP의 표준화된 방법은 반복들을 좌에서 우로 중첩 없이 순차적으로 수행하며, 각 반복 내에서의 활동들도 위에서 아래로 순차적으로 수행한다. 이는 그림 7과 같으며, Alhir[4]가 제안한 순차적 접근법 (Sequential Approach) 개념과 동일하다. 이 접근법은 설계자가 너무

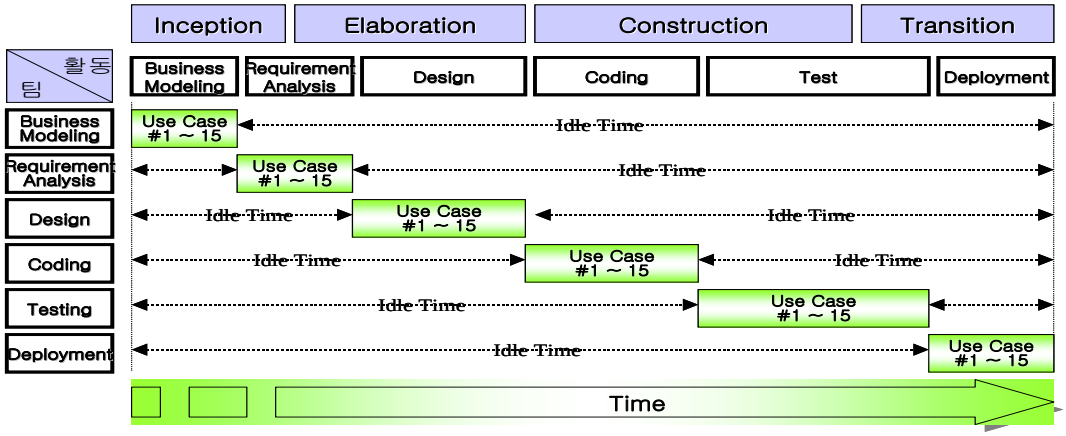


그림 6. RUP의 선형 접근법 적용
Fig. 6. Linear Approach of RUP

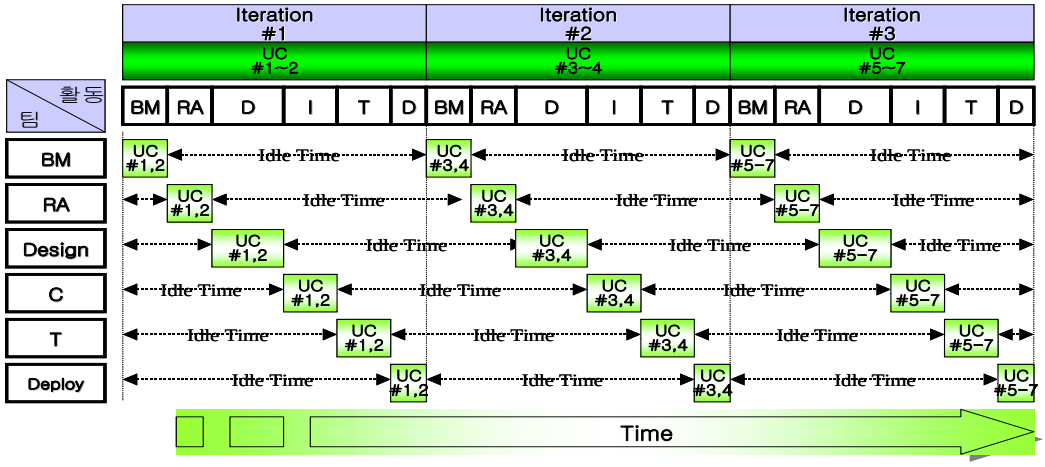


그림 7. RUP의 순차적 접근법 적용
Fig. 7. Sequential Approach of RUP

완고할 경우에 적용할 수 있다.

여기서 이해를 돕기 위해, 반복 #1에 Use Case (UC) #1과 #2가 할당되고, 반복 #2에서는 UC #3와 #4가 할당되었으며, 반복 #3에서는 UC #5~#7이 할당되었다고 가정한 경우이다. 이 접근법은 선형 접근법에 비해 다수의 미니 프로젝트를 조기에 고객에게 납품하여 문제점을 피드백 시키는 장점은 갖고 있다. 그러나 단점으로는 선형 접근법과 동일하게 비 활동시간 (Idle Time)이 과다하게 발생하여 자원 활용 (Resource Utilization)의 비효율성을 갖고 있다. 만약 개발팀을 요구사항 분석가, 설계자, 프로그래머, 시험 요원 등으로 구성한다면 N번째 반복의 해당 작업을 끝내고, N+1번째 해당 작업이 시작될 때까지 (약 2 ~ 6주) 활동을 하지 못하는 문제가 발생하여 비효율적인 업무의 수행이 된다.[3] 이러한 인력 낭비를 줄일 수 있는 유일한 방법은 요구사항 전문가, 설계 전문가, 프로그래머, 시험요원 등으로 개발팀을 구성하는 대신 개발자들이 프로젝트의 모든 활동들을 수행하는 만능인이 되는 것이다.[14,23,24] 그러나 모든 활동들을 완벽히 수행할 수 있는 만능 전문가는 거의 없는 실정으로 비현실적이라 할 수 있다.

Crain[3]은 RUP가 반복들을 중첩시키거나 선행 작업을 수행하지 않고 순차적 접근법을 고집하는 이유로 다음 사항들을 거론하고 있다.

(1) 자원 활용 위험 등 다양한 위험이 발생할 수 있다.
(2) 폐기와 재작업이 증가한다.
(3) 프로젝트 취소 비용이 증가한다.
(4) 빈번한 재작업과 폐기로 인해 근로의욕 (사기)가 낮아진다.
(5) 프로세스 향상에 영향을 미친다.
(6) 팀은 반복 또는 기능에 초점을 두지 않고 활동들에 초점을 둔다.
(7) 요구사항이 거의 완벽히 이해되지 않는 상태에 있는 미래의 요구사항들을 선행 작업하는 것은 현재 반복에서 필요한 것 이상으로 이들 선행 작업에 심도 있는 작업을 할당해야 한다.

RUP의 표준 접근법 (순차적 접근법)의 인력낭비 현상을 완화하기 위해 Crain[3]은 해당 반복에 할당된 작업을 완료하여 다음 팀에게 전달한 후 반복에 할당된 시간이 경과할 때까지 다음 반복들에 할당된 작업을 선행하여 수행하는 것이 타당하다고 판단하여 그림 8의 반복들을 중첩 (Overlapping) 시키는 접근법을 제안하였다.

RUP의 공식적인 입장은 이 접근법을 추천하지 않는다. 그러나 Crain[3]은 만약 면밀한 관리를 할 수 있다면 이 방법은 유용한 접근법이 될 수 있다고 언급하고 있다. 이 접근법은 한 팀이 하나의 반복에 할당된 작업을 완료하면 다음 팀에게 전달하고, 다음 반복에 할당된 작업을 계속적으로 진행하는 방식으로, 인력낭비 현상을 어느 정도는 완화시키는 장점을 갖고 있다. 그러나 반복들에 Time-boxed 개념이 도입되어 있지 않아 단계와 반복에 대한 시간 개념이 없어져 프로젝트 관리에 어려움이 발생하는 단점을 갖고 있다.

마지막으로, Crain[3]이 제안한 Time-boxed 반복 접근법이 있다. 이 접근법은 중첩된 반복 접근법이 가지고 있는 단점을 보완하고자, 그림 9와 같이 인력 낭비도 최소화 하면서, 반복에 Time-boxed 개념을 도입하였다. 즉, 각 반복에 할당된 시간이 경과할 때까지만 이후 반복에

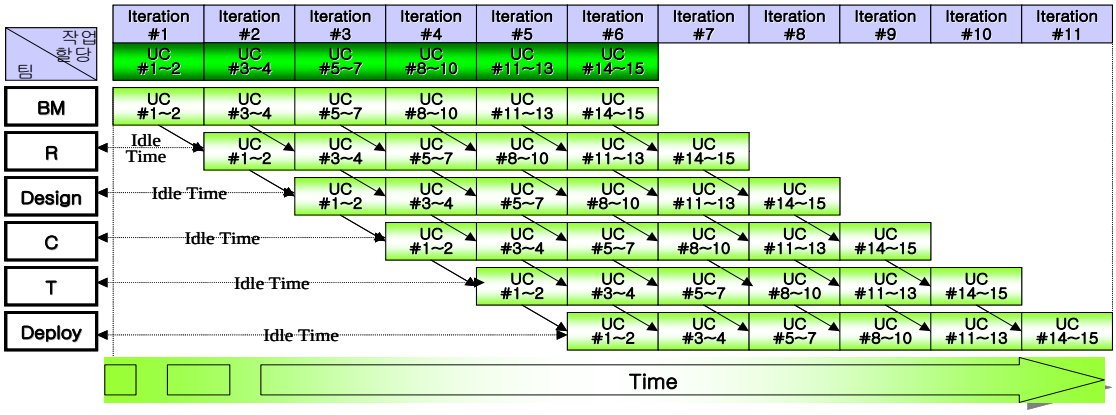


그림 8. RUP의 중첩된 반복 접근법 적용
Fig. 8. Overlapped Iteration Approach of RUP

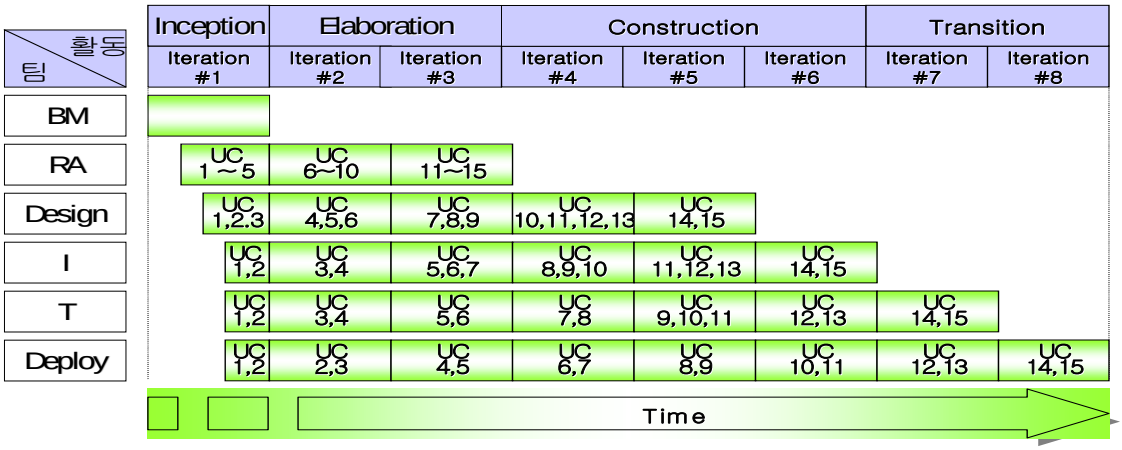


그림 9. RUP의 Time-boxed 반복 접근법 적용
Fig. 9. Time-boxed Iteration Approach of RUP

할당된 작업을 선행하여 수행하는 방식이다.

이 접근법은 프로젝트 관리자, 설계자와 프로세스 공학자자 상호 균형을 이룰 경우 적용할 수 있는 효과적인 방법이다. 또한, 반복들에 대한 Time-boxed 개념이 도입되어 반복의 종료 시점에서의 평가를 수행할 수 있는 장점을 갖고 있다. 그러나 단점으로는 RUP에서 단계들을 명확히 구분하는 이정표 개념이 없는 단점이 있다. 이로 인해, 정련단계에서 시스템의 아키텍처가 불안정한 상태에서도 구축단계가 수행됨으로 인해 선행된 작업에 대한 폐기와 재작업의 위험이 증가할 수 있다. 또한, 이 접근법은 한 팀이 하나의 반복에 주어진 작업을 모두 완료하고 다음 팀에게 전달하는 방식인지 아니면 하나의 요구사항에 대해 작업 후 바로 다음 팀에게 전달하는 방식인지가 명확하지 않는 의문이 제기된다.

Ⅲ. 동시개발 기반 단일화된 프로세스 적용

RUP를 적용함에 있어, 선형, 순차적 접근법은 인력의 비효율적 운영 문제점을 갖고 있어 Time-boxed 접근법이 가장 효율적인 것으로 판단된다. 그러나 Time-boxed 접근법은 단계가 종료되었을 때 다음 단계를 수행할 것인지에 대한 이정표 평가를 수행하기 어려우며, 반복이 종료되었을 때, 현재까지의 작업 진척도와 다음 반복에 대한 실행 계획을 작성하는데 어려움이 있다. 따라서 이러한 문제점을 해결하기 위해 본 장에서는 그림 10과 같이 동시 개발 접근법 (Concurrent Development Approach)을 제안한다.

본 제안 방법을 개략적으로 살펴보면, 각 단계들 간에는 명확한 분리가 이루어져 있어 하나의 단계에 할당된 요구사항에 대해서만 계속적으로 선행 작업을 할 수 있으며, 단계가 종료되면 이정표에서 평가가 이루어진다. 또한, 각 단계 내에서 수행되는 활동들은 동시개발 (Concurrent Development) 개념으로 볼 수 있다. 동시개발 개념은 실제로 오늘날 개발되는 소프트웨어의 시간 제약사항에 의해 영향을 받아 제기되었다. 동시개발은 소프트웨어가 팀 체제로 개발될 때 필연적으로 적용될 수 있다. 동시개발을 장려하는 주요 요인은 소프트웨어 개발하는데 사용되는 납품 기반 접근법 (Release-based Approach)이다. 전형적으로 하나의 소프트웨어는 납품의 연속으로 개발된다. 소프트웨어가 납품 기반 접근법으로 개발되어야 하는 이유는 경제적인 이유와 변화하는 사업 필요성에 적응하기 위함이다.[25-28] 납품을 RUP에서는 하나의 반복 수행 결과 얻는 Mini-project 또는 서브 프로젝트로 해석할 수 있다. 납품 기반 접근법은 이상적인 경우 하나의 소프트웨어 납품이 다른 납품 이후에 개발되는 개념이다. 그러나 현실적으로 소프트웨어 개발조직은 종종 동시에 2개 이상의 납품을 지원한다. 본 논문에서 제안하는 동시개발 개념은 이와 같이 다수의 납품을 동시에 개발하는 개념이 아닌 다수의 활동들을 동시에 수행하는 개념이다.

하나의 단계를 상세히 살펴보면 그림 11과 같이 요구사항 (Use Case) 1개 단위로 작업 후 다음 팀으로 전달하

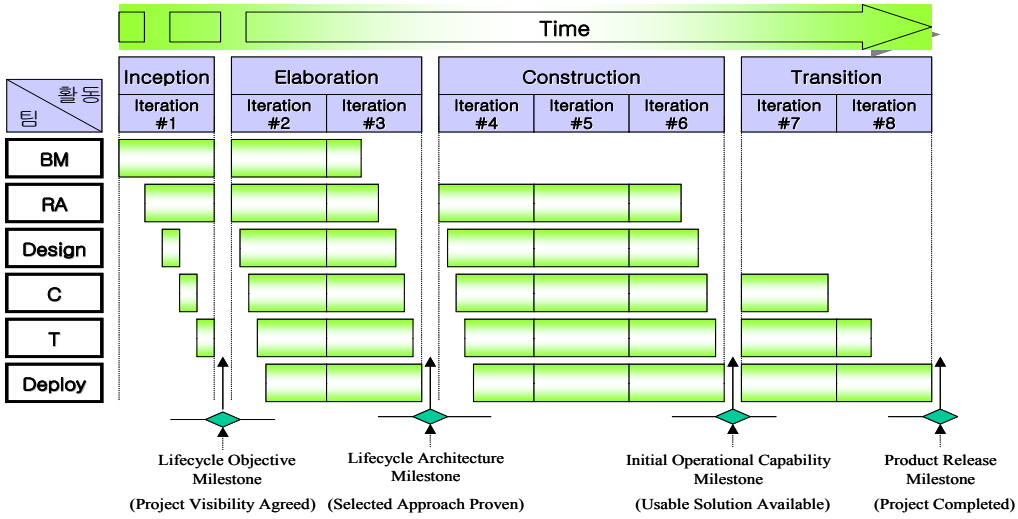


그림 10. RUP의 동시 개발 접근법 적용의 개략도
Fig. 10. Diagrammatic Concurrent Approach of RUP

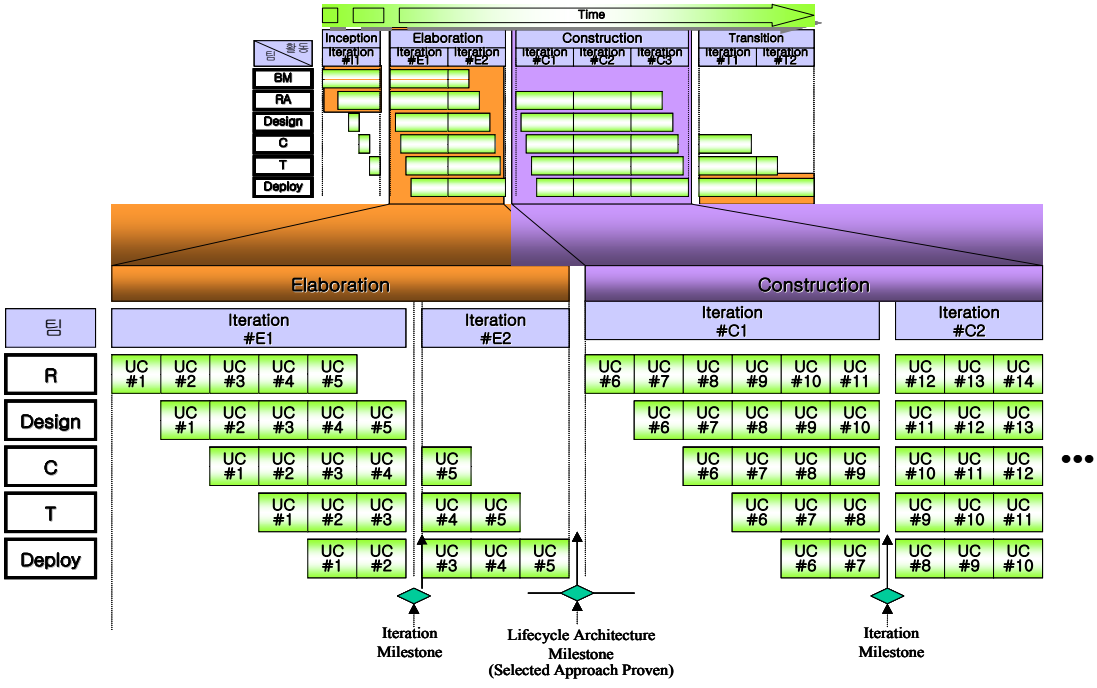


그림 11. RUP의 동시개발 개발 접근법 적용의 상세도
Fig. 11. Detail Concurrent Approach of RUP

는 동시개발 개념을 도입하고 있다. 도입단계에서는 대부분의 작업들이 Business Modeling과 Requirements에 집중되어 있으며, Design, Coding과 Test 활동은 Proof-of-concept에 대한 프로토타이핑과 관련된 작업이다. 또한, 전이단계는 대부분이 Deployment에 관련된 활동으로 Beta Test와 사용자 교육과 관련된 작업이 이루어지며, 이에 따라 발생되는 문제점을 해결하기 위한 Coding과 Test가 부수적으로 발생한다. 따라서 도입단계

와 전이단계에서는 엄밀히 말해 동시개발 접근법은 적용되지 않는다.

도입 단계 (Inception Phase)에서는 Vision이 제안되고, Use-case들의 목록이 작성되고, 개발 우선순위가 결정된다. 이들 Use-case들을 정련단계(Elaboration Phase)와 구축단계 (Construction Phase)의 각 반복들에 개발 우선순위에 따라 할당한다. 이 단계의 마지막 시점에서 Life-cycle Objective Milestone을 수행한다. 이 이정표는 도입단계에 대한 평가가 수행되고 정련단계를 진행할 지에 대한 합의를 도출한다. 정련단계에서는 할당된 Use-case들에 대해 요구사항팀이 UC #1을 분석한 후 설계팀에게 전달하고 UC #2의 요구사항을 분석한다. 설계팀은 UC #1을 설계한 후 프로그래머에게 전달한 후, 요구사항 팀에게서 받은 UC #2를 설계한다. 이와 같이 1개의 Use-Case 단위로 다음 팀에게 전달하는 방법을 취하며, 이 작업은 정련단계에 할당된 Use-case들이 종료될 때까지 계속된다. 정련단계 마지막에서는 Life-cycle Architecture Milestone을 수행한다.

이와 같은 방법으로 구축단계와 전이단계 (Transition) 단계를 수행한다. 따라서 본 논문에서 제안하는 동시개발 접근법은 유스케이스 기반 접근법(Use-case-based Approach)이라 할 수 있다. 또한, 정련단계와 구축단계의 각 반복에 할당된 시간이 경과하면 각 반복에서 최종적으로 개발이 완료된 부분에 대해 평가가 수행된다. 이는 RUP의 반복 완료에 따른 평가방법과 동일하다. 본 접근법은 다음과 같은 장점이 있다.

(1) Time-boxed 반복 접근법의 인력 활용 장점을 갖고 있다.
(2) 단계들 간에 명확한 구분으로 Time-boxed 반복 접근법의 문제점을 해소시킬 수 있다.
(3) 폐기와 재작업, 프로젝트 취소 비용 증가, 프로세스 향상에 영향 등 RUP에서 순차적 접근법을 고집하는 이유들을 어느 정도 해소시킬 수 있으며, 인력낭비 요인을 없앨 수 있다.
(4) 하나의 단계에서 동시 개발로 인해 RUP의 순차적 접근법 보다 개발기간을 단축시킬 수 있다.

Ⅳ. 결론 및 향후 연구과제

본 논문은 최근 들어 UML(Unified Modeling Language)과 Use-case에 기반한 객체지향 소프트웨어 개발에 널리 적용되고 있는 RUP에 대해 효율적으로 실제 수행하는 방법을 제시하였다.

먼저, RUP의 단계, 반복과 활동들을 세부적으로 수행하는 다양한 방법들을 살펴보고, 단계, 반복과 활동들을 결합시킨 통합된 관점에서 RUP를 수행하는 선형 접근법, 순차적 접근법, 중첩된 반복 접근법과 Time-boxed 반복 접근법들의 문제점을 살펴보았다.

기존의 방법들이 갖고 있는 인력낭비 현상이나 프로젝트 관리의 어려움 문제점을 해결하기 위해 본 논문은 동시개발 접근법을 적용하였다. 이 접근법은 하나의 단계에서 수행되는 반복들은 중첩되며, 각 활동들은 하나의 단계에 할당된 모든 작업에 대해 불연속 없이 계속적으로 수행하는 방법이다. 하나의 단계가 종료되면 해당 단계의 이정표를 수행한 후 다음 단계를 수행하는 형태로 RUP의 기본 취지인 이정표 개념을 여전히 갖고 있어 프로젝트 관리의 용이함이 있으며, 해당 단계에서의 인력낭비도 거의 없는 장점을 갖고 있다.

제안된 접근법은 이론상으로는 가장 좋은 방법으로 판단된다. 그러나 현실적으로 프로젝트 개발에 적용하여 실용성을 검증하지는 못하였다. 따라서 추후 현실적 적용으로 제안된 접근법의 효율성을 검증할 예정이다.

참 고 문 헌

[1] J. Bennett, "Software Development," http://homepage.ntlworld.com/jeremy.bennett/notes/index.htm, 2002.

[2] I. Spence and K. Bittner, "Managing Iterative Software Development with Use Cases," The Rational E-zine, http://www-128.ibm.com/developerworks/rational/library/5093.html, 2004.

[3] A. Crain, "Overlapping Iterations in a RUP-based Project," The RationalE-zine, http://www-128.ibm.com/developerworks rational/library/may05/crain/, 2005.

[4] S. S. Alhir, "Understanding the Unified Process (UP)," Methods & Tools, Marting & Associates, http://home.concast.net/~salhir/UnderstandingTheUP.PDF, 2002.

[5] Rational Software, "Rational Unified Process: Best

Practices for Software Development Teams," Rational Software White Paper, TP026B, Rational Software Corporation, 2001.

[6] G. Tattersall, "Supporting Iterative Development Through Requirements Management," The Rational E-zine, http://www-128.ibm.com/developerworks/rational/library/2830.html, 2002.

[7] K. Bittner, "Driving Iterative Development with Use Cases," The Rational E-zine, http://www-128.ibm.com/developerworks/rational/ library/4029.html, 2006.

[8] C. Larman, 'Agile and Iteration Development: A Manager's Guide," 2003.

[9] I. Spence and K. Bittner, "What is Iterative Development," The Rational E-zine, http://www-128.ibm.com/developerworks/rational/library/may05/bittner/index.html, 2005.

[10] A. Kushniruk, "Rapid Development: Lifecycle Planning," Department of Mathematics & Statistics, http://www.math.yorku.ca/~andrek/ITEC-4010/ITEC4010-8.ppt, 1996.

[11] C. Wallin and R. Land, "Software Development Lifecycle Models: The Basic Types," Research Methodology for Computer Science and Engineering, 2001.

[12] K. Curran, "Project Management: Project Lifecycle Planning," http://www.infm.ulst.ac.uk/kevin/com820/week3.ppt, University of Ulster, Magee Collage, N. Ireland, 2005.

[13] P. McBreen, "Incremental Requirements Capture," Xprogramming. com, 1999.

[14] P. Wheaton, "Overlapping of Phases in RUP," Java Rinch Big Moose Salon, 2004.

[15] P. Kruchten, "A Unified Development Process," Crosstalk, Vol. 9, No. 7, pp. 11-16, 1996.

[16] K. Holm, S. Larsen, and K. Røgenberg, "Project Management Artifacts: Rational Unified Process," Rational Software Corporation, 2004.

[17] P. Eeles, K. Houston, and W. Kozaczynski, "An Introduction to the Rational Unified Process: Building J2EE Application with the Rational Unified Process," Addison Wesley Professional, 2002.

[18] A. Crain, "RUP Iteration Planning," The Rational E-zine, http:// www-128.ibm.com/developerworks/rational/library/5335.html, 2004.

[19] Tassc ltd, "The ObjectMetrix Estimation Process Tassc," Tassc Ltd, 2001.

[20] M. Cantor, "Organizing RUP SE Projects," The Rational E-zine, http://www-128.ibm.com/developerworks/rational/library/814.html, 2003.

[21] P. Kroll, "Dr. Process: What Should the Coders do While Waiting for the Analysts to be Done?," The Rational E-zine, http://www-128.ibm.com/developerworks/rational/library/435.html, 2004.

[22] L. Rose, "Continuously Ensuring Quality: A Case Study," The RationalE-zine, http://www-128.ibm.com/developerworks/rational/library/dec04/rose/index.html, 2004.

[23] S. W. Ambler, J. Nalbone, and M. Vizdos, "The Enterprise Unified Process: Extending the Rational Unified Process," Prentice Hall PTR, 2005.

[24] S. W. Ambler, "A Manager's Introduction to the Rational Unified Process (RUP)," Ambysoft. com, 2005.

[25] M. Sayko, "Parallel Software Development is the New Normal," SOA Executive Forum, CM Crossroads, 2004.

[26] B. Prasad, "Concurrent Engineering Fundamentals," The International Institute of Concurrent Engineering, Prentice Hall, 1997.

[27] J. B. Park, H. S. Yang, "Quality Evaluation Method of Open Source Software," Journal of the Korea Academia-Industrial, cooperation Society, Vol. 13, No. 5 pp. 2353-2359, 2012.

[28] D. S. Kim, H. C. Kim, "The Study of Software Reliability Model from the Perspective of Learning Effects for Burr Distribution," Journal of the Korea Academia-Industrial Cooperation Society, Vol. 12, No. 10 pp. 4543-4549, 2011.

저자 소개

최 명 복(종신회원)

- 1994년 : 아주대학교 컴퓨터공학과(석사)
- 2001년 : 아주대학교 컴퓨터공학과(박사)
- 1997년 ~ 현재 : 강릉원주대학교 멀티미디어공학과 교수
- 2004.년 1월 ~ 현재 : 한국인터넷방송통신학회 이사

<주관심분야 : 지능형 정보검색, 소프트웨어 공학, 알고리즘>

- e-mail : cmb5859@gmail.com

이 상 운(정회원)

- 1995년 ~ 1997년 : 경상대학교 컴퓨터과학과 (석사)
- 2007.3 ~ 현재 : 강릉원주대학교 과학기술대학 멀티미디어공학과 부교수

<주관심분야 : 소프트웨어 척도, 분석과 설계 방법론, 소프트웨어 신뢰성, 그래프 알고리즘>

- e-mail : sulee@gwnu.ac.kr