MaRMI(Magic and Robust Methodology Integrated)와 RUP(Rational Unified Process) 개발방법론 비교 분석

The Comparative analysis of MaRMI(Magic and Robust Methodology Integrated) & RUP(Rational Unified Process)

김재열, 송미영
한국한의학연구원 한의학정보화사업단

Kim jae-yeol, Song mi-young
Korea Institute of Oriental Medicine

요약

시스템 개발방법론은 소프트웨어 생성을 위한 개발 단계를 정의하고, 활동, 산출물, 검증 절차, 각 단계의 완결 조건을 명시하는 체계적인 방법으로 정의할 수 있다. 정보시스템의 대형화, 복잡화, 분산화 추세에 따라서 표준화된 개발방법론에 대한 관심이 국내에서도 급속히 증가하고 있다. 정보기술의 발전에 따른 신기술의 계속적인 수용과 사용자 요구사항의 변화의 수용, 시스템 개발의 생산성과 품질의 보증을 위해서는 개발방법론과 개발도구의 활용이 필수적이다. 최근 많은 관심을 보이고 있는 마르미(Magic and Robust Methodology Integrated)와 RUP(Rational Unified Process)을 비교 분석하고자 한다.

I. 서 론

객체지향에 기반한 정보시스템의 개발은 단순한 개발도구의 변경을 의미하는 것이 아니라 개발될 시스템을 바라보고 기본적인 개념, 패러다임의 변화를 필요로 한다. 따라서 현재 많이 사용되고 있는 구조적 방법론에 입각한 SSADM(Structured System Analysis and Design Method), 정보공학방법론, 메소드/1 등으로 객체지향 시스템의 개발을 지원하는데는 한계를 갖는다.

이러한 문제의식에서 80년대말부터 객체지향 개발방법론에 대한 연구들이 진행되었으며, 90년대 초에는OOD(Object-Oriented Design), OMT(Object Modeling Technique), OOSE(Object-Oriented Software Engineering) 등의 20여 개의 개발 방법론들이 제시되어 적용되었다. 다수의 객체지향 개발방법론의 출현은 객체지향 개발방법론의 보급을 저해하였고, 도구간의 상호작용성(Interoperability), 모형화 산출물간의 변환 등의 문제를 야기하였다. 이에 따라 90년대 중순부터 표준화된 객체지향방법론에 대한 논의가 활발히 대두되었다. 그 결과로 1997 년 가을에 OMG(Object Management Group)에서 UML(Unified Modeling Language)을 표준화된 객체지향 모형화 언어로 채택하게 되었으며, 이로 인해서 객체지향 개발방법론의 실제적 적용은 급격히 늘어날 것으로 예상된다[14].

본 논문에서 소개하고자 하는 마르미(MaRMI, Magic and Robust Methodology Integrated)는 국내의 개발 여건을 반영하여 국내 기술진에 의해서 개발된 한국형 객체지향 개발방법이다.

마르미에서는 국내의 개발 여건, 즉 객체지향 방법론의 대형 프로젝트에 본격적으로 적용되기 시작하는 현재의 상황을 반영하여, 쉽게 객체지향 방법론을 기업에 적용할 수 있도록 실제적이고 상세화된 방안을 제시하고자 한다. 이를 위해서 Rational 사의 RUP(Rational Unified Process)를 제시하고 상세한 수준의 개발 절차, 구체적인 개발 기법과 지침을 비교 분석하였다.

II. 마르미 개발 방법론

한국전자통신연구원에서 만든 컴포넌트 기반 소프트웨어 개발 방법론이다. 마르미는 컴포넌트의 개발 및 컴포넌트 기반의 시스템 개발에 필요한 작업과 작업 수행에 필요한 기법 및 작업별 산출물을 정의하고, 작업에 따른 상세한 개발 절차와 지침을 제공한다

1. 상세화되고 실제적인(Concrete and Practical) 방법론

국내에서는 객체지향 개발방법론이 실험적인 시스템에 주로 적용되었으며, 실제적으로 대형 시스템의 개발에 적용되기 시작한 것은 최근의 일이다. 따라서 국내의 대부분의 개발업체들은 객체지향 지식과 경험을 충분히 가지고 있는 인력을 확보

하고 있지 못하다. 이러한 국내 여건을 반영하여 타 방법론보다 훨씬 상세한 수준의 개발 절차를 제시하여 방법론 사용자가 쉽게 적용할 수 있도록 하였다. 또한 마르미에서는 개발 기법을 구체적으로 제시하고 이와 관련된 지침(guidelines)을 체계화하여 시행 착오를 가능한 줄이면서 프로젝트를 진행할 수 있도록 하였다. 또한 개발 공정뿐만 아니라 관리 및 지원 공정을 포괄적으로 지원하여 별도로 이들을 고려해야 하는 부담을 줄여주도록 하였다.

2. UML 기반

Rumbaugh, Booch, Jacobson 등이 함께 참여하고 있는 Rational 사에서 개발한 UML(Unified Modeling Language)은 개발 초기부터 객체지향 방법론 시장의 빅(Big) 3가 통합하여 관심을 집중시켰다. 1997년 가을 OMG에서 UML을 객체지향 모형화 언어의 표준으로 채택하면서 향후 UML이 표준 언어로써 확고한 위치를 점할 것으로 보인다. UML은 객체지향 모형화 언어로서 개발 공정, 도구에 독립적인 특징을 갖는다. 따라서 마르미에서는 UML을 모형화 언어로 사용하면서 객체지향 시스템을 개발하기 위한 절차, 지침, 도구를 제공하고자 한다. 또한 모형화 구조물(construct)의 확장이 필요한 경우에는 UML이 제공하는 확장 매카니즘을 활용하므로써 UML 의 표준을 준수하도록 하였다.

3. 사용사례(Use Case) 중심

Jacobson 의 OOSE 에서 제시되었던 사용사례 개념은 Objectory나 Fusion과 같은 대표적인 객체지향 방법론들에서 요구분석의 중심 기법으로 활용되고 있다. 사용사례는 사용자 입장에서 가시적인 시스템의 기능적(functional) 요구사항을 의미하며, 사용자와 시스템간 일련의 상호작용의 개념적 묶음이다. 사용사례 개념은 개념 자체의 단순함으로 인해서 사용자들이 쉽게 이해할 수 있기 때문에, 사용자와 개발자간의 유용한 의사소통 도구로 활용되고 있다. 마르미에서는 사용사례를 요구관리의 기본 단위로 뿐만 아니라 반복 개발의 단위로 사용하므로써 개발 과정의 가시성을 높이는데 활용하고 있다.

4. 아키텍처 중심

최근의 소프트웨어 개발에서 견고한 시스템 아키텍처를 개발 초기 단계에 정립하는 것이 프로젝트 성패에 중요한 요인으로 인식되고 있다. 개발 초기에 확고한 아키텍처를 정립하면 재작업을 줄이고, 재사용성, 확장성, 시스템 품질 측면에서 많은 장점을 얻을 수 있다. 마르미의 개발 절차에서는 개발 초기에 견고한 아키텍처의 정립을 유도하고 있다. 또한 정립된 아키텍처에 대하여 시제품 개발을 통해서 아키텍처를 검증하여 실행가능하고(feasible) 상세화된 아키텍척를 개발 초기에 획득할 수 있도록 한다.

5. 반복적, 점진적 개발

최근의 기업 환경이 급변하면서 정보시스템 개발에 있어서 지속적으로 사용자 요구사항의 변화를 반영할 수 있는 능력이 필요하다. 반복적이고 점진적 개발에서는 개발 단위를 작게 분할하고 시스템을 점진적으로 개발함으로 과거의 폭포수 모형(waterfall model)이 요구사항의 변경에 제대로 대처하지 못했던 경직성을 극복한다. 즉, 반복적이고 점진적 개발은 사용자와의 지속적인 상호작용을 통해서 상황의 동적인 변화들을 반영할 수 있도록 한다. 마르미에서는 미니프로젝트(Mini Project) 개념을 도입하여, 개발 프로젝트를 반복적으로 수행하는 다수의 미니프로젝트로 분할하여 수행하므로 반복적, 점진적 개발이 가능하도록하고 있다. 이러한 접근법은 사용자의 요구사항에 부합되는 시스템의 개발을 가능하게 하고, 병행적 개발을 통해서 개발기간을 단축할 수 있으며 위험 관리 측면에서도 장점을 갖는다.

6. 위험 관리

대표적인 개발 절차 모형 중의 하나인 폭포수 모형의 단점 중에 하나가 위험 관리가 취약하다는 것이다. 즉 시스템 개발과 관련된 위험들이 개발 과정에서 해결되지 않은 상태에서 그대로 유지되다가 개발이 완료되는 시점에서 프로젝트의 실패 또는 성공으로 나타난다는 것이다. 따라서 시스템 개발의 체계적인 위험 관리를 위해서는 시스템의 개발과 관련된 위험들을 가능한 개발 초기에 해소하고, 시스템 개발에 관련된 모든 위험들이 체계적으로 규명, 분석, 감시, 통제되어야 한다. 마르미에서는 시제품 개발을 통해서 기술적 위험을 초기에 감소시키도록 하였고, 반복적 절차를 통해서 위험을 가장 많이 해소시킬 수 있는 사용사례를 우선적으로 선정하여 개발하므로 개발 과정에서 위험들을 계속적으로 줄여가도록 하고 있다. 또한 위험분석서를 통해서 시스템 개발과 관련된 위험들이 계속적으로 점검되고 체계적으로 관리되도록 하였다.

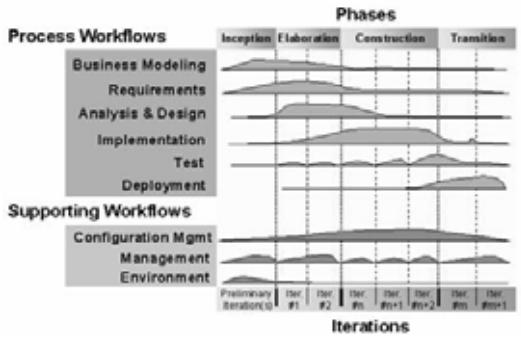
7. 마르미와의 일관성 유지

마르미는 기존의 마르미 방법론이 가지고 있는 메타모형, 용어, 산출물, 프로젝트 관리 활동 등 상당 부분을 공유하고 있다. 이는 마르미 방법론 개발의 효율성을 높이기 위한 측면과 함께 기존의 마르미 사용자들이 자연스럽게 마르미의 객체지향 버전인 마르미를 채택하여 사용할 수 있도록 하기 위함이다.

III. RUP 개발 방법론

RUP(Rational Unified Process)는 소프트웨어 개발의 전체 생명주기를 지원하는 프로세스 프레임웍이다. 즉, 바로 적용하여 사용할 수 있는 방법론이라기보다는 다양한 유형의 소프트웨어 시스템, 도메인, 그리고 조직을 위한 프로세스의 프레임웍을 제공해준다. 따라서, 소프트웨어 개발 조직의 특정 요구사항에 맞게끔 프로세스를 조정할 수 있으며, 이것은 해당 프로젝트를 위한 Development Case에 문서화된다. RUP는 Objectory, OMT, Booch, 그리고 Rational의 접근법 등, 90년대 초반에 등장하였던 다양한 객체지향 방법론들을 통합하여 발전해왔는데, UML을 창안한 Jacobson, Rumbaugh, 그리고 Booch가 RUP 개발에 참여하였음을 고려해보면 이것은 결코 우연이 아니다.

1. RUP의 구조



▶▶ 그림 1. RUP 구조

"시간"의 흐름을 나타내는 가로축과 특정 시간대에 중점적으로 수행해야 할 "활동"을 보여주는 세로축으로 구성되는 2차원 구조를 가지고 있다. 먼저 가로축은 소프트웨어 생명주기를 시간의 흐름에 따라 4개의 단계로 나누어 구성하고 있으며, 이것이 하나의 개발 주기를 이룬다.

각각의 단계는 일련의 반복을 포함할 수 있으며(inception) 단계에서는 프로젝트의 범위를 설정하고, 핵심적인 요구사항 파악 및 후보 아키텍처의 윤곽을 제시하며, 전체 프로젝트의 비용과 일정을 견적하고, 잠재적인 위험 요소를 식별한다. 이어지는 정련(elaboration) 단계에서는 대부분의 기능적 요구사항이 정의되고, 안정적인 아키텍처가 설계되며, 그리고 프로젝트 계획이 수립된다. 대부분의 구현과 테스트 작업은 다음에 이어지는 구축(construction) 단계에서 수행된다. 마지막으로 전이(transition) 단계에서는 베타 테스트가 수행되고, 사용자 교육이 시행되며, 소프트웨어를 사용자에게 인계하게 된다.

반면에 세로축은 전통적인 소프트웨어 개발 접근법에서의 절차와유사한 9개의 핵심 워크플로우(workflow, 최근에는 discipline이라는 개념으로 바뀌었다)를 보여주고 있다. 각각의 반복은 정의된 모든 워크플로우를 포함하지만, 앞서 언급한 것처럼 특정 반복이 강조하게 되는 워크플로우는 단계가 진행함에 따라서 변하게 된다. 초기의 반복에서는 주로 요구사항을 정의하는데 대부분의 노력을 들이지만, 다음 반복에서는 점차 분석 및 설계, 그리고 구현쪽에 중점을 두게 된다. 그리고, 최종적으로는 테스트와 배포에 더 많은 비중을 두게 된다.

RUP는 유스케이스, 아키텍쳐, 패턴, 그리고 컴포넌트 등 현대 소프트웨어 공학이 일반적으로 다루고 있는 거의 모든 기법과 산출물들을 포괄하고 있는 기능이 매우 풍부한 방법론이다. 더욱이 RUP는 반복적(iterative)이고 점진적인(incremental) 방법으로 이 산출물들을 정제하는 활동을 강조하고 있다. 이것은 위험요소를 조기에 발견하여 사전에 위험을 최소화시키고, 요구사항의 변화를 효과적으로 수용함으로써 소프트웨어의 품질을 향상시키게 된다. 무엇보다도 RUP의 가장 큰 특징은 소프트웨어 개발에 있어서 그 동안의 성공적인 경험을 바탕으로 한 6가지의 베스트 프렉티스(best practices)를 강조하였으며, 개발 프로세스를 자동화 시켜주는 다양한 도구와 이것을 방법론에 적용하기 위한 방안(Tool mentors)을 제공하고 있다는 점이다.

방법론의 기능이 풍부하다는 것은 RUP의 강점이 될 수도 있지만, 오히려 약점으로 작용할 수도 있다. RUP는 최근 소프트웨어 공학의 거의 모든 베스트 프렉티스를 포함하고 있지만, 이들 사이에서 필연적으로 발생하게 되는 긴장 상태를 어떻게 해결할 지에 대한 구체적인 지침이 부족하다. 예를 들어, RUP는 "유스케이스 중심적(기능지향적)"임과 동시에 "아키텍쳐 중심적(객체지향적)"적이지만, 이 둘 사이에 충돌이 발생했을 경우에 어느 것을 우선시해야 할지에 대해서는 명확히 설명하고 있지 않다. 또한, 방법론이 비록 수행할 작업을 작은 반복으로 나누어 놓고 있기는 하지만, 대부분의 반복이 모든 핵심 워크플로우를 수행하는 것은 아니다. 그러므로, 시스템에 대한 전반적인 분석과 설계 활동이 수행되기 전까지는 코딩이나 테스트 같은 구체적인 구현 활동들은 여전히 행해지지 않는다. 결국, 폭포수형 모델의 기본 개념이 그 배경에 깔려있으며, 산출물과 프로세스 절차 구성에 큰 영향을 미치고 있다. 물론 RUP는 소프트웨어 컴포넌트의 개념을 충분히 지원하고 있지만, 방법론을 주의 깊게 살펴보면 몇 가지 문제점을 안고 있다.

RUP가 컴포넌트 기반 개발을 지원할 수 있는 방법은 외부의 이벤트를 시작점으로 관련 있는 기능들을 하나로 묶어 이것을 하나의 컴포넌트로 정의하는 방법, 클래스간에 서로 긴밀하게 협력하는 클래스 그룹을 식별하고 이들의 결합 정도를 고려하여 컴포넌트를 정의하는 방법, 그리고 아키텍처 수준에서 식별된 서브시스템을 기준으로 컴포넌트를 정의하는 방법 등이 있다.

첫번째와 두번째 방법은 초기 정련 단계에서 수행하게 되며, 마지막 세번째 방법은 구축 단계에서 수행된다. 컴포넌트는 클래스와 패키지의 속성을 모두 갖는다는 점에서 UML의 서브시스템의 개념과 매우 유사하지만, 이것은 자칫 컴포넌트를 전체 개발 과정 중 구현과 배포 단계에서만 중요한 역할을 갖게 되는 것으로 취급할 수 있다. 이것은 RUP가 전체 생명주기에 걸쳐 컴포넌트의 개념을 충분히 지원하지 못하고 있음을 의미한다. 또한, UML Reference Manual에는 “서브시스템은 자신의 행위를 갖지 않는다”라고 기술되어 있다. 즉, 서브시스템이 자신과 관련한 행위를 갖더라도, 이것은 자신의 고유한 행위가 아니라 서브시스템을 구성하고 있는 모델 요소들의 행위를 포함한다는 것을 의미한다. 결국 서브시스템은 다른 요소들에 의해서 구현되어져야 하는 행위를 한데 모으고 보여주기 위한 컨테이너로서 사용되며, 실제 시스템 구현에는 영향을 주지 않는다. 그럼에도 불구하고, RUP는 최근의 객체지향 소프트웨어 공학 방법론들의 수많은 구성 요소들을 이용하기 위한 유용한 프레임웍을 제공하였으며, 어떻게 이것들을 함께 적용할 수 있는지 보여주었다. 또한, 폭 넓은 사용자 층과 수많은 프로젝트를 통해 만들어진 방대한 자료는 여전히 우리가 RUP에 매력을 느끼기에 충분하다.

IV. 결 론

본 논문에서는 현재 널리 알려져 있는 주요 CBD 방법론 2개를 비교해 보았다. 각각의 방법론마다 적용 가능한 대상이 있고 장단점이 다르기 때문에 특정 회사나 조직에 맞는 방법론을 분류할 수는 있으나 절대적으로 다른 방법론에 비해 효과적인 방법론을 단정짓기는 어렵다. 하지만 주요 방법론의 비교를 통해서 각 방법론의 특징을 파악함으로써 회사 및 프로젝트의 성격에 맞는 CBD 방법론을 선택하는데 도움을 줄 수 있다. 또한 회사나 조직에서 사용하는 방법론과도 비교함으로써 해당 방법론의 향상에 도움을 줄 수 있을 것이다.

향후 방법론의 추세는 특정 방법론을 선정하여 모든 프로젝트에 같은 방법론을 획일적으로 적용하는 것이 아니라, 필수적인 요소로 기본적인 방법론을 표준화하고, 각 프로젝트의 특성에 따라 맞춤화(Customizing)하는 방향으로 진행되고 있다. 이는 과거에 비해 개발하는 시스템의 종류가 다양해지고, 단순한 방법론의 적용으로 많은 문서를 도출해내는데 만족하지 않고, 프로젝트의 규모, 인원, 기간, 목적, 제한 조건 등을 반영하여 효율적인 소프트웨어 개발을 하고자 하는 경향이 진행되고 있음을 알 수 있다.

▌참 고 문 헌▐

[1] 전상욱 외, CBD 방법론 비교 분석, 2004.
[2] 조진희, 마르미 개요, 2004
[3] Rational company, RUP소개, 2006
[4] 김수동, 소프트웨어공학, 2004
[5] Booch, Grady, Object-Oriented Design with Application, Benjamin-Cummings, 1991.
[6] Clements, Paul C. and Northrop, Linda M., "Software Architecture: An Executive Overview," Technical Report, Software Engineering Institute, Carnegie Mellon University, 1996.
[7] Coleman, Derek, et al., Object-Oriented Development: The Fusion Method, Prentice Hall, 1994.
[8] Graham, Ian, Henderson-Sellers, Brian, and Younessi, Houman, The OPEN Process Specification, ACM Press, Harlow, England, 1997.
[9] ISO/IEC, "ISO/IEC 12207 Information Technology - Software Life Cycle Processes," ISO/IEC, 1995.
[10] Jacobson, Ivar, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.
[11] Jacobson, Ivar, The Object Advantage: Business Process Reengineering with Object Technology, ACM Press, 1995.
[12] Kruchten, Philippe, "A Rational Development Process," White Paper, Rational Software Corp., 1997.
[13] Martin, James, Information Engineering Book II: Planning and Analysis, Prentice Hall, 1991.
[14] Rational, Unified Modeling Language(Version 1.1), Rational Software Corp., 1997.
[15] Rational, "The Rational Approach," White Paper, Rational Software Corp., 1997.
[16] Rumbaugh James, et al., Object-Oriented Modeling and Design, Prentice Hall, 1991.
[17] Van Scoy, Roger L, "Software Development Risk: Opportunity, Not Problem," Technical Report,
Software Engineering Institute, Carnegie Mellon University, 1992.