SPRi Issue Report
2015. 12. 23. (2015-012호)

MDD(모델 주도 개발) 유용성 논의와 사례 분석

진회승
(hschin@spri.kr)
김태호
(teokim@spri.kr)

SPRi 소프트웨어정책연구소

○ 본 보고서는 「미래창조과학부 정보통신진흥기금」을 지원받아 제작한 것으로 미래창조과학부의 공식의견과 다를 수 있습니다.

○ 본 보고서의 내용은 연구진의 개인 견해이며, 본 보고서와 관련한 의문사항 또는 수정·보완할 필요가 있는 경우에는 아래 연락처로 연락해 주시기 바랍니다.

- 소프트웨어정책연구소 진회승 선임연구원(hschin@spri.kr)

《 Executive Summary 》

MDD(Model Driven Development)는 모델에 중점을 둔 개발방법론이다. 개발과정에서 목표시스템을 단순화한 모델을 활용하여, 사용자는 시스템을 쉽게 이해할 수 있고 개발자는 개발을 용이하게 할 수 있다.

프로젝트 실패 주요 원인인 요구사항 불명확성을 개선하고, 급변하는 경영 환경과 기술 환경 하에서 원하는 시기에, 원하는 비용으로, 사용자에게 원하는 소프트웨어를 제공할 수 있는 소프트웨어 개발방법론의 하나로 MDD를 검토한다.

MDD 도입 시 모델의 추상성으로 인하여 도메인 기술이 축적되고, MDD 설계 산출물과 소스코드의 일치로 일어나는 여러 학술적인 장점에도 불구하고, 실제 현업에서는 어느 정도 쓰이고 얼마나 장점이 있는 지에 대해 공감대 형성이 되어 있지 않은 상태이다.

이에 MDD를 이용한 국내외 업체의 개발사례를 MDD 프로세스 적용 방식, 인력, 도구, 도입 효과 측면에서 살펴보고, 학술적인 이론과 국내외 현실 차이에 대해 검토한다.

결론적으로 MDD는 요구사항 명확화를 통해 프로젝트 성공률을 높이고, 어플리케이션에 도메인 기술을 축적하여 소프트웨어 재활용이 가능하도록 하는 유용성이 있는 것으로 보인다. 또한 국내 소프트웨어 개발 인력 구조 문제 해결에 도움이 되고, 나아가 소프트웨어 산업 발전에 중요한 계기가 되리라 예측된다. 그러나 MDD를 이용하여 효과적인 소프트웨어를 개발하고, 국가 소프트웨어 산업 발전에 기여하기 위해서는 MDD 도구의 개발, 조직적 지원, 참여자의 적극적인 수용 등 해결해야 하는 과제도 많다.

《 목 차 》

1. MDD(Model Driven Development, 모델 주도 개발)[1] 의의

(1) MDD의 개념과 특징

MDD는 모델에 중점을 둔 개발 방법론으로 모델을 이용하여 목표 시스템을 단순화함으로써, 사용자는 시스템을 쉽게 이해할 수 있고 개발자는 개발을 용이하게 하는 것을 목적으로 한 개발방법론임

□ MDD는 모델 개발에 중점을 둔 개발방법론

○ 모델은 목표 개발물의 특징을 추출하여 단순화하는 표현 방식

- 모델은 업무, 프로세스, 도메인(금융, 제조, 정부, 의료 등), 정보 & 데이터, 시스템(하드웨어, 플랫폼)의 정보를 가진 생성물로 구분하여 정의함[2]
- 업무, 프로세스, 도메인 모델은 컴퓨터 하드웨어 및 소프트웨어 아키텍처 기술을 숨겨서 소프트웨어 개발 시 업무 프로세스 개발에 집중하게 함
- 모델은 표현 범위 및 내용에 따라 상위·하위 모델로 구분하며, 하위 모델로 갈수록 구체적임

  * 상위 모델은 목표 시스템의 세부정보는 생략하고 특징적인 요소만을 표현

  * 상위 모델은 모델 변환 기술(예: 모델 → 소스코드)에 의해 하위 모델 또는 프로그램 코드로 변경되며, 하위모델이나 프로그램 코드로 변경될수록 IT 기술과 연관된 내용이 포함됨

○ 모델은 목표 시스템에 대하여 사용자와 IT기술자가 소통할 수 있는 언어로 표현됨

- 일반적인 모델링 언어로는 UML(Unified modeling Language), SQL[3], BPMN[4], E/R 다이어그램[5] 등이 있음

1) MDD, MDE(Model Driven Engineering), MDSE(Model Driven Software Engineering) 등이 혼용되어 사용되고 통일된 정의가 아직 정립되지 않아 본 연구서에서는 CMU SEI(Model-Driven Engineering : Automatic Code Generation and Beyond, CMU SEI, 2015.03)를 참고하여, MDD로 명칭하여 사용. MDA(Model Driven Architecture)는 MDD에 대한 OMG에서 정의한 표준

2) Object Management Group Model Driven Architecture(MDA) MDA Guide rev. 2.0, 2014.06

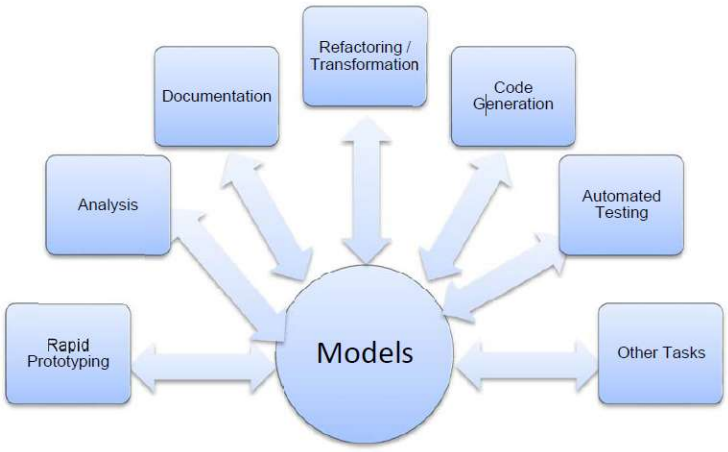
3) Structured query language, 데이터베이스를 구축하고 활용하기 위해 사용하는 데이터 모델링 언어

* UML은 객체관련 표준화 기구인 OMG[6]에서 1997년 11월 객체모델링 기술을 이용하여 만든 모델링 언어. 객체 지향적 분석, 설계 방법론의 표준지정을 목표로, 요구분석, 시스템 설계, 시스템 구현의 과정에서 생길 수 있는 개발자간의 의사소통을 불일치를 해소하고자 개발. 유스케이스 다이어그램, 클래스 다이어그램 등 8개의 다이어그램을 기반으로 분석, 설계 장치를 제공

- 도메인 특화 언어인 DSL(Domain Specific Language)을 이용하여 모델을 구현하기도 함

○ MDD는 이러한 모델을 중심으로 분석, 설계를 수행하고 이에 대한 결과물로써 소스코드 및 산출물을 자동 생성하는 소프트웨어 개발방법론임

[그림 1] MDD 모델 역할[7]



- 기존 코드 중심 개발 방식과 달리 사용자의 요구사항을 모델로 생성하는 과정이 중심인 개발방법론임

4) Business Process Model and Notation, 비즈니스 프로세스를 표현하기 위한 그래픽 표현으로 조직 내에서 업무에 대한 커뮤니케이션에 이용함
5) Entity/Relation 다이어그램, 구조화된 데이터에 대한 일련의 표현, 데이터의 구조와 두 개 이상의 데이터의 관계를 표현
6) Object Management Group, 1989년 소프트웨어 판매업자, 사용자, 학계, 정부 등이 연합하여 만든 비영리 기구. UML, MDA(Model Driven Architecture) 등 모델링 표준을 정의하여 소프트웨어 설계, 개발, 유지보수에 활용하고자 함
7) Model-Driven Engineering : Automatic Code Generation and Beyond, CMU SEI 2015.03 재인용, Brambilla, Marco. *Model-Driven Software Engineering in Practice - Chapter 1 - Introduction.* Morgan & Claypool, 2012

- MDD는 추상적인 형태의 모델과 모델간 변환 기술로 구성되어 있음. 이러한 추상성은 시스템을 간단히 하고 정규화(표준화)하여 소스코드 생성, 문서제작, 테스팅 자동화가 가능해 짐[8)]
- MDD 도구의 핵심은 "모델 변환"이며 상위모델에서 하위모델로 변환, 하위모델에서 프로그램 코드간의 변환을 자동화함으로써 소프트웨어 개발 생산성을 높임

○ MDD는 단순화된 모델을 통해, IT기술과 업무를 분리함

- 소프트웨어 개발 시 모델을 활용하여 도메인전문가와 IT 기술자간의 커뮤니케이션이 용이
- 업무와 IT 기술이 변경되더라도 전체 소프트웨어 변경이 아닌 필요한 모델만 변경하여 소프트웨어 개발이 가능함

□ MDD의 소프트웨어 개발 방법은 개발플랫폼에 종속되지 않은 모델(메타모델)을 정의하고, 소프트웨어 아키텍처를 세분화함

○ MDD 표준의 하나인 OMG의 MDA(Model driven Architecture)[9)]의 경우, 개발플랫폼에 종속되지 않은 소프트웨어를 개발을 위해 소프트웨어와 하위 기술 플랫폼을 분리하여 개발

○ MDA는 CIM, PIM, PSM을 정의하고 모델을 변환을 통해 소프트웨어를 생성

- CIM (Computation Independent Model)은 하위 소프트웨어 구조와 관계없이 업무, 도메인에 대한 기능만을 정의

  * UML은 일반 업무 정의 언어, MOF(MetaObject Facility)는 메타모델 정의 언어임. 메타모델을 이용해 이 기종 어플리케이션의 공통 정보를 저장함.

- PIM (Platform Independent Model)은 구체적인 플랫폼과는 관계없이 업무프로세스를 정의. 변환 규칙이 생성되면, 여러 플랫폼에서 재사용 가능

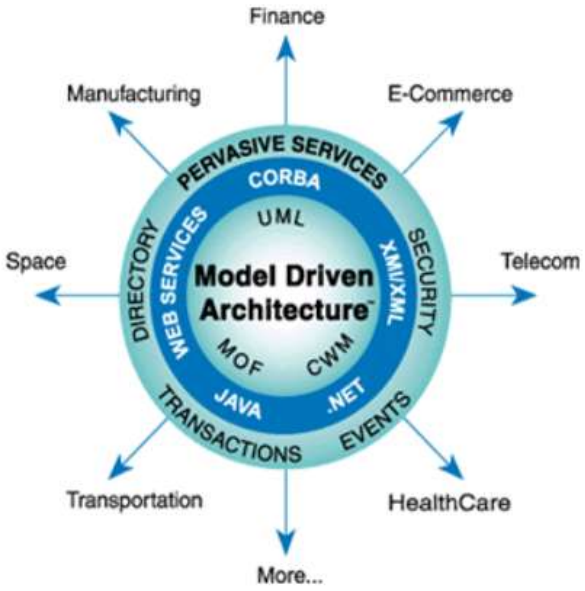
  * 플랫폼이란 배치, 웹, java, .net 등 소프트웨어 개발 시 구체적인 소프트웨어 구동을 실현하는 환경임

8) IBM Systems Journal - Model-driven software development, Model-driven development : The good, the bad, and the ugly, B.hailpern, P,Tarr, 2006.07
9) Object Management Group Model Driven Architecture(MDA) MDA Guide rev. 2.0, 2014.06

- PSM (Platform Specific Model)은 PIM에서 자동 변환되는 IT 기술에 종속되는 모델임

  * 보안, 디렉토리 서비스, 트랜잭션 등 플랫폼에 관련된 사항이 포함됨

[그림 2] Model Driven Architecture 구성도



자료: OMG 홈페이지

□ MDD의 성공적인 도입을 위해서는 모델간의 변환, 모델과 소스 간의 변환을 구현하는 MDD 도구가 지원되어야 함

○ MDD에 대한 연구는 2000년 초부터 시작하여, 학술적으로는 많은 성과가 있으나, 산업계에서 MDD 도입 성과를 확인하기 어려운 이유 중의 하나가 MDD 도구로 기인한 것임

- MDD의 수많은 장점에도 불구하고, MDD의 이론을 적용하여 자유로운 모델과 변환 툴을 만드는 것은 쉬운 일이 아님

  * 도구가 하위 모델과 밀접하게 연관되면 모델의 범위가 좁아지나, 자동 코드 생산율이 높아지며, 반대의 경우는 모델의 범위는 넓어지나 자동 코드 생산율이 낮아짐

- MDD 도구는 도메인 범위, 코드 자동생산 정도, 개발프레임의 종속성 정도에 따라 다양함

○ MDD 도구 선정은 사용자 요구사항에 따라 기능, 품질, 가격, 기간 등을 고려해야함[10)]

- 기본적인 도구선정 요건은 도구의 안정성, 완전성, 명료성, 실행 가능임
- 설계, 개발, 테스트의 개발 프로세스에서 각각의 도구 요구사항에 대해서 확인이 필요

  * 설계: 기능 구현이 가능하고, 사용용이성, 외부시스템과 연결성, 성능, 테스트 가능성, 하드웨어에 대한 제한 등을 확인

  * 개발: 개발표준 구현이 가능하고, 개발자가 툴을 학습하기 쉬우며, 모델 변경이 가능해야 함

  * 테스트: 코드생성과 테스트 가능 여부 및 다른 시스템과 연동 여부 확인

- 생성된 코드는 신뢰성, 안정성, 보안, 유지보수성을 보장해야 함
- 프로젝트 참여자는 MDD도구를 사용하여 프로젝트를 수행하고, 그 결과가 프로젝트 관리 도구에 연동되어야 함

□ MDD의 개발 참여자는 기존프로젝트와 다른 역할이 부여됨

○ 기존 프로젝트와 다르게 MDD에서 새로운 역할을 하는 담당자는 다음과 같음

- 도메인 전문가 : MDD에서 도메인 전문가는 DSL의 정의, 기본 설계를 수행함

  * 기존 프로젝트에서는 도메인 전문가와 설계자, 개발자를 따로 분리하지 않았으며, 도메인에 대한 지식을 가진 전문가가 설계와 개발을 맡음

- MDD 도구 개발자 : DSL이 정의되면, 모델 간 변환을 위한 툴 개발을 담당할 MDD 도구 개발자가 모델 간 변환, 모델과 소스코드 변환을 수행하는 MDD 도구를 개발함
- 프레임워크 담당자(소프트웨어 아키텍트가 수행) : 하위 시스템에 맞도록 MDD 도구를 수정하고, 교육하는 역할을 함

  * 기존 프로젝트에서 프레임워크 담당자는 프로젝트 수행 시 공통으로 사용가능한 라이브러리 및 메타코드를 생성하는 역할을 수행

10) Model-Driven Engineering : Automatic Code Generation and Beyond, CMU SEI, 2015.03에서 MDD 도구 선택 기준 발표

○ 대형 SI[11] 프로젝트의 경우, 관리조직은 그대로 유지되나 모델 생성을 위한 모델러가 필요하고, 개발자의 비중보다 설계자의 비중이 커짐

- 프로젝트 수행단계를 분석, 설계, 개발, 테스트, 이행이라고 나눌 때, MDD에서는 설계와 분석 단계의 구분이 어려우며, 설계와 동시에 개발, 테스트가 진행됨
- 모델 생성을 함으로써 기존에 무시되기 쉬웠던 상세 설계의 중요성이 커지고 소프트웨어 개발 후에나 사용자가 확인 가능했던 요구사항과 소프트웨어의 검증이 설계단계에서 이뤄짐
- 프로젝트 수행 경험이 늘어나고 도메인 기술이 축적됨에 따라 상세 설계 기간을 줄임으로써 프로젝트 기간을 줄이고 생산성 향상을 가져올 수 있음

○ MDD에서 코드 자동생성을 구현하기 위해 MDD 도구 개발자의 역할이 중요함

- MDD 도구 개발자는 프레임워크 담당자와 MDD도구를 수정하는 역할을 담당함
- 설계자 및 개발자는 프로젝트 참가 전에 MDD 도구 사용 및 소스 생성 후 오류 검증에 대한 학습이 필요함

11) System Integration, 정보 시스템을 기획부터 구축, 운용까지 필요한 업무를 일괄하여 제공하는 서비스

(2) MDD 출현의 배경

□ (역사적 배경) MDD 출현 이전 유사한 개념으로 객체지향방법론, 컴포넌트 기반 방법론(CBD)이 있었음

○ 객체지향, 컴포넌트 기반 방법론의 기본 단위인 객체, 컴포넌트가 추상화된 개념이기 보다는 코드 자체라는 측면에서 MDD와는 구분됨

○ 소프트웨어공학의 소프트웨어 개발방법론은 하드웨어, 소프트웨어 아키텍처 등 시스템의 하위구조를 숨겨 사용자의 시스템의 대한 이해를 돕고 개발자가 개발이 용이한 방향으로 점차 발전

- 시스템 하위구조를 숨기는 방법은 그래픽을 이용한 방법, 프로그램 라이브러리를 이용한 방법 등이 있음

○ 1980년대 CASE[12]는 그림으로 프로그램 할 수 있도록 도구를 제공하여 수동 코딩으로 인한 프로그램 상 오류를 제거하고 자동화를 추구하였으나 제한적으로 사용됨

- 실질적으로 널리 쓰이는 못한 이유는 기술적, 경제적 한계 때문임[13]

* 기술적 한계 : QOS[14] 제공 부족, 개발자 동시 개발 지원 미비, 미들웨어 지원 부족 등

○ 이 후 개발 프레임워크[15]의 발전으로 재사용이 가능한 클래스 라이브러리와 도메인 관련 미들웨어가 제공되었으나, 플랫폼의 복잡성 증가로 개발의 어려움은 여전히 존재하여 이 문제를 완화시켜 줄 수 있는 방안의 필요성 대두

12) Computer Aided Software Engineering, 즉 소프트웨어 생명 주기(software life cycle)의 전체 단계를 연결시켜 주고 자동화시켜 주는 통합된 도구들을 제공하여 소프트웨어의 생산성 향상을 도모하자는 것

13) Model-Drive Engineering, D.C. Schmidt, Vanderbilt Univ. 2006.02, IEEE Computer society

14) Quality of Service (분산 환경, Fault Tolerance, 보안 등 지원), 통신서비스 품질. 네트워크상에서 일정 정도 이하의 지연 시간이나 데이터 손실률 등의 보장을 일컫는 말로, 사전에 합의 또는 정의된 통신 서비스 수준을 뜻함

15) 소프트웨어 어플리케이션이나 솔루션의 개발을 수월하게 하기 위해 소프트웨어의 구체적 기능들에 해당하는 부분의 설계와 구현을 재사용 가능하도록 협업화된 형태로 제공하는 소프트웨어 환경을 말함

[표 1] 소프트웨어 개발방법론 비교

구분	구조적 방법론	객체지향방법론	CBD	MDD
특징	프로그램 로직 중심 프로그램 모듈화	개체에 데이터와 로직 통합 상속에 의한 재사용	컴포넌트에 인터페이스 추가	메타모델을 이용한 개발 자동화
주요 지원언어	COBOL, C, PASCAL	C++, JAVA, c#	모델링 언어, 개발언어는 무관	모델링 언어, 개발언어는 무관
출현시기	1970 년대	1990년대	2000년대	2000년대
핵심요소	프로세스	객체	컴포넌트	모델
목적	비즈니스 프로세스 자동화	실제 현실세계를 반영한 유연한 시스템	컴포넌트 개발 및 재사용	개발플랫폼에 자유로운 자동화된 소프트웨어 개발

□ 사용자와 개발자간 요구사항을 명확히 정의하고, 경영층의 지원과 사용자의 참여를 높일 수 있는 새로운 개발방법론의 필요성 대두

○ 2015년에는 전 세계적으로 50,000여개의 프로젝트가 수행되었으며, 예정된 시간 안에, 예정된 금액으로 성공적으로 완료된 프로젝트는 29% 수준[16)]

- 명확한 요구사항의 정의 및 사용자 요구사항 변화에 따른 변화관리가 프로젝트 성공에 중요한 요소임

[표 2] 소프트웨어 개발 프로젝트 성공도

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

자료: 2015 CHAOS Report by Standish Group, http://www.infoq.com/articles/standish-chaos-2015

* Successful은 예정된 시간 내 예정된 예산으로 프로젝트 완료한 경우
Challenged는 프로젝트는 완료되었으나 예정된 시간이나 예산이 넘었거나, 기술된 요구사항보다 부족한 기능을 구현한 경우
Failed는 프로젝트를 종료하지 못한 경우임

16) 2015 CHAOS Report by Standish Group, http://www.infoq.com/articles/standish-chaos-2015

- 요구사항의 불완전성을 해결하기 위해 소프트웨어 공학의 한 분야에 요구공학이 있고 “소프트웨어사업 요구사항분석, 적용가이드”[17] 와 같은 정부차원의 노력도 있음
- 여전히 요구사항 수집 및 분석은 소프트웨어 개발 프로젝트 수행에 해결하기 어려운 문제임

○ 프로젝트가 성공하지 못하는 또 다른 이유로는 경영층의 지원 부족 및 사용자의 참여 미흡 등을 들 수 있으며 기술, 자원의 한계, 프로젝트에 대한 과도한 기대, 개발 기간 부족 등도 요인으로 작용

- 프로젝트 성공을 위한 개선은 성공 요인에 대한 모든 요소에 대한 고려할 수 있는 개발방법론이 필요함
- 프로젝트 성공을 위해 이러한 문제들을 개선한 수 있는 개발방법론이 필요함

□ 소프트웨어 개발과 업무간의 연계성이 더욱 강화되면서, 업무 변경 시 소프트웨어에 즉시 반영할 수 있는 개발방법론이 요구됨[18]

○ 사용자와 IT 기술자간의 커뮤니케이션이 중요

- 소프트웨어가 비즈니스 가치를 창출하는 사회에서 소프트웨어 개발자와 사용자 간 소통이 중요함

○ 소프트웨어 시스템의 규모와 복잡성이 증가하였음에도 업무에 적합한 소프트웨어를 적정한 가격에 구축 요구

- 비즈니스가 복잡해지고 공유경제[19], 플랫폼형[20], 가치사슬[21] 변경에 의한 새로운 비즈니스 모델이 생성됨에 따라 새로운 소프트웨어 시스템이 필요하나, 기업은 사용연한이 짧은 시스템에 많은 투자를 하지 않음

17) 소프트웨어 요구사항분석, 적용 가이드, 지식경제부, 2012.12
18) Patterns: Model-Driven Development Using IBM Rational Software Architect, Peter Swithinbank et all, ibm.com/redbooks, 2005.12
19) ‘물건을 소유하는 개념이 아닌 서로 빌려 쓰는 경제활동’ 이라는 의미로 2008년 로렌스 레식 하버드대 교수가 처음 사용했으며, 에어비앤비(Airbnb), 우버(Uber) 등이 성공모델로 알려짐
20) 이 문서에서 여러 가지 의미의 플랫폼이 기술되며, 여기서는 다양한 종류의 시스템이나 서비스를 제공하기 위해 공통적이고 반복적으로 사용하는 기반 모듈, 어떤 서비스를 가능하게 하는 일종의 “토대”를 말함. 제품, 서비스, 자산, 기술, 노하우 등 모든 형태가 가능
21) 기업 활동에서 부가가치가 생성되는 과정

○ 특정 기술자, 특정 개발 환경에 종속되지 않은 개발환경 구축이 필요함

- 개발자의 변경이나 소프트웨어 구축 환경 변화에 소프트웨어가 민감하게 변경되지 않는 프로젝트 개발 환경이 프로젝트 성공에 중요한 요소가 됨

□ 기존 소프트웨어 전체를 변경하지 않고도 새로운 기술을 적용할 수 있는 소프트웨어 개발방법론이 요구됨

○ 신기술을 이용한 소프트웨어 개발이 가속화되면서 개발자들이 학습해야 할 기술의 양이 많아져 새로운 기술 적용 시 소프트웨어 변경을 최소화하는 방안 필요

- 과거 어셈블리어가 개발자를 기계어보다는 하드웨어 환경에서 자유롭게 한 것과 동일하게, 현재 쓰이고 있는 프로그램 언어보다는 시스템 아키텍처에서 자유로운 방안 모색 필요

○ 프로세서, 스토리지, 소프트웨어 등을 인터넷을 통하여 제공하는 클라우드 컴퓨팅은 여러 다른 하드웨어 환경, 플랫폼[22]에서 제공됨[23]

- 하위 플랫폼 변경에 영향을 적게 받는 상위 어플리케이션 소프트웨어 아키텍처 고려 필요

○ 업무에 모바일을 적용하는 경우가 많아져 기업들은 모바일 기술을 가진 개발자를 구해야 하는 부담이 되고 있음[24]

- 모바일 소프트웨어 플랫폼은 안드로이드, 아이폰, 윈도우즈 폰 등으로 다양하지만 똑같은 콘텐츠의 어플리케이션을 재사용하지 못해 다른 플랫폼 개발 시 새로운 코드를 작성해야만 함

- RMAD(Rapid mobile app development) 도구에 대한 여러 시도가 있으며, 드래그앤 드랍 도구, 코드 생성 도구, 컴포넌트 어셈블리 등으로 자동화 도구를 제공하고 있음

- 그 시도 중에 하나로 MDD 도구 제공

22) 각종 서비스의 기반이 되는 하드웨어나 소프트웨어 환경, 이 후 언급이 언급 경우는 같은 의미로 쓰임

23) Cloud SaaS and Model Driven Architecture, Rim sharma, Proc. of the International Conf. on Advanced Computing & Communication Tech. 2011

24) Market Guide for Rapid Mobile App Development Tools, Gartner, 2014.11

2. 개발방법론으로서 MDD의 유용성 검토

(1) MDD 이론적, 기술적 유용성

□ MDD에 의해 생성된 도메인 모델에는 업무 도메인에 대한 기술이 축적되어 있으며, MDD 개발과정에서 모델에 대한 최적화가 이루어짐

○ 기존 개발 방법론에 의해 구축된 소프트웨어에도 도메인 기술이 축적되어 있으나, 소프트웨어와 도메인 기술의 분리가 쉽지 않음

- 기존엔 설계와 개발이 분리되어 있어, 설계 산출물과 소프트웨어의 동기화가 어려움
- 설계 산출물은 개발된 소프트웨어와 일치되는 경우가 적어 실질적으로 개발 후 활용이 어렵고 도메인 기술 축적 자료로 사용되기 어려움
- 최종산출물인 소프트웨어의 경우는 도메인전문가가 해석하기 어려움

○ MDD에 의해 개발된 도메인 모델은 도메인 전문가가 이해하기 용이하고, 자동 생성된 문서가 제공되는 소프트웨어는 사용자가 이해하기 용이함

- 도메인 전문가가 실질적으로 알 필요가 없는 데이터베이스 모델, 하드웨어 모델과 같은 IT 관련 기술이 분리됨
- 업무관련 모델은 IT 기술자의 도움 없이 변경하고 최적화 가능
- 모델과 소프트웨어 간의 동기화가 보장되어, 모델로써 소프트웨어 기능에 대한 설명과 문서화가 가능하여 일반 사용자도 소프트웨어에 대한 이해가 용이함

□ MDD에 의해 생성된 모델은 MDD 도구에 의해 소스코드로 자동 변환되므로 모델과 소스코드가 일치됨

○ 기존 방법론에 의한 객체, 컴포넌트는 설계 문서와 소스코드간 일치여부 알 수 없음

○ 모델에서 일부 소프트웨어 코드가 자동 생성되어 프로젝트 생산성을 높임

- MDD에서는 분석, 설계, 개발의 단계에서 산출물이 자동으로 생산되어 각각의 단계의 산출물의 가시화되어 프로젝트 참여자들의 커뮤니케이션을 돕고 프로젝트 진행 상태를 관리할 수 있음

- 설계 산출물과 소스코드가 동기화되어 소프트웨어 생명 주기 모델 중 반복적 개발모델에 적용에 적합함

□ MDD에 의해 생성된 소프트웨어는 자동 문서화로 개발자 변경에 영향을 적게 받으며, 유지보수가 용이함

○ MDD의 업무 모델은 IT 기술과 분리되어 있고 하위 시스템 관련 모델은 업무 모델과 분리되어 있어 IT 기술자들이 이해하기 용이함

○ 모델이 자동문서화 되어 소프트웨어와 문서와의 동기화를 보장하고 개발과 동시에 개발문서가 업데이트됨

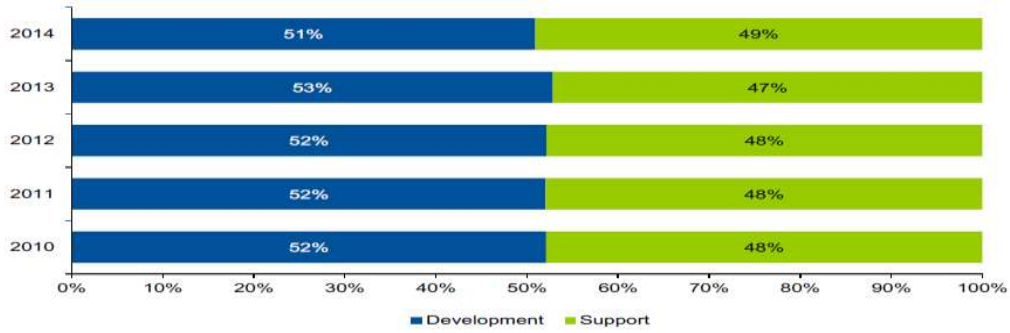
○ 소프트웨어 라이프사이클에서 유지보수가 큰 비중을 차지해, 유지보수 비용을 줄이는 것이 IT 비용 절감에 큰 영향 줌[25)]

- 소프트웨어 개발에 소요되는 비용은 개발에 이용되는 하드웨어, 상용 소프트웨어, 오피스 공간, 인적요소이며, 전체 IT 비용 중 소프트웨어 개발에 소요되는 비용은 34%임

- 프로젝트에 소요하는 비용 중 소프트웨어 유지보수 비용은 평균적으로 48%임

* Development : 개발 비용, Support : 소프트웨어 유지보수 비용

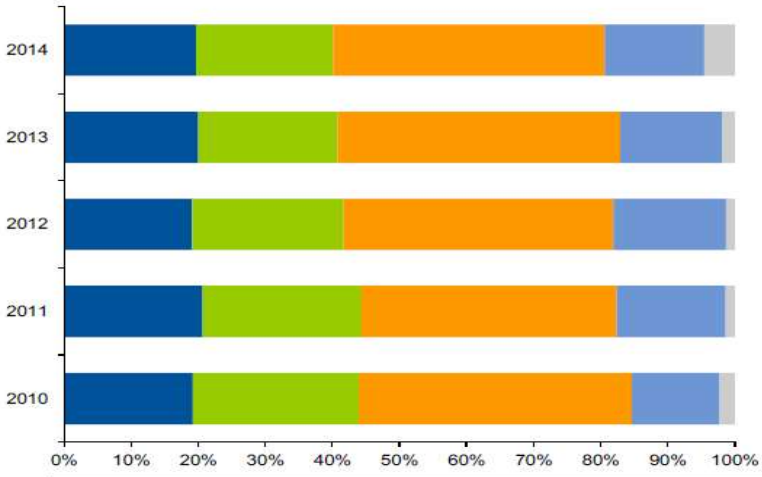
[그림 3]프로젝트 소요비용 : 개발 대 유지보수



자료 : Gartner IT Key Metrics Data (December 2014)

25) IT Key Metrics Data 2015:Key Applications Measures: Cost and staff Profile: Multi year, Gartner, 2014.12

[그림 4] 유지보수 활동



	2010	2011	2012	2013	2014
Bug Fixes	19%	21%	19%	20%	20%
Minor Enhancements	25%	24%	23%	21%	20%
Business & Operations Support	41%	38%	40%	42%	40%
Technical Upgrades	13%	16%	17%	15%	15%
Other	2%	1%	1%	2%	5%

자료 : Gartner IT Key Metrics Data (December 2014)

* 유지보수: 버그 수정, 2인/주에 수정이 가능한 기능 추가, 어플리케이션 기능에 대한 사용자 지원, 소프트웨어 하드웨어 업그레이드 (그림 4 참조)

* 직접적으로 소프트웨어 수정 및 변경에 관련된 비용은 Bug fixes, Minor Enhancements, Business & Operations Support로 약 80 ~ 85% 가량으로 전체 소프트웨어 라이프사이클에서 소프트웨어 유지보수 비용은 매년 40%[26)]

○ 개발자 변경 시 개발된 소프트웨어의 인수인계가 용이하고, 신규 개발자도 소프트웨어를 쉽게 이해 가능

- MDD 도구 개발자는 도메인 기술과 하위 소프트웨어 기술을 알고 개발할 소프트웨어에 맞는 MDD 도구를 수정하며, 소프트웨어 아키텍트가 코드 생성을 위한 규칙, 제한, 패턴을 정의하면, 모델을 사용하는 개발자는 직관적으로 모델을 사용할 수 있음

- 소프트웨어 유지보수 시 MDD 도구 개발자 및 소프트웨어 아키텍트는 참여하지 않고 기존 소프트웨어 아키텍처 위에서 비즈니스 로직 변경을 위한 개발자의 작업이 주로 일어남

26) T Key Metrics Data 2015:Key Applications Measures: Application Support : Multi year, Gartner, 2014.12

(2) 제약요인

□ MDD 도구의 유연성 및 비용 산정 문제

○ MDD 도구가 급변하는 경영환경에 맞추어 소프트웨어를 개발할 수 있도록 지원이 가능한지, 비용은 얼마나 절감되는지 여부를 쉽게 검증할 수 있는 방법이 충분치 않아 경영층이 MDD 적용을 결정하는데 어려움으로 작용

○ 특히 MDD 도입을 위해서는 MDD도구의 사용이 필수적이나 개발하려는 소프트웨어에 맞는 MDD 도구를 찾기 어렵고, 도구 개발에 들어가는 비용대비 효과 측면의 분석도 미비

- MDD의 소스 코드 자동생성에 의한 생산성 향상 효과 측정 시 MDD 도구 개발에 대한 오버헤드를 고려해야 함
- 모델 정의를 위한 설계비용 대비, 코드 자동생성에 의한 개발기간 단축을 고려해야 생산성 향상 정도를 측정 가능함
- 프로젝트 생산성 향상정도는 MDD 도구의 도입 및 학습 비용을 비교해서 감안해야 함

□ 소프트웨어 전문가들도 MDD 사례 등에 대한 정보 공유가 부족하고, MDD 기술 문헌이나 연구가 많지 않아, 필요성이나 구체적 작동원리에 대해 충분히 이해하지 못하고 있는 실정임

○ 일부 적용중인 사례에 대한 소개 및 연구가 필요하며 이를 통해 소프트웨어 개발 전문가들의 관심이 확대될 필요

□ MDD 도입 시 소프트웨어 설계자, 개발자, 사용자간 역할이 변화하여야 하나 단기간에 이들의 역할변화를 효율적으로 이끌어 내기는 쉽지 않음

○ MDD로 개발방법론을 변경하면 설계자, 개발자, 사용자 등 역할이 부분적으로 변경되며, 새로운 역할의 가진 담당자 필요

○ 새로운 역할에 대한 당위성 증명이 필요하고, 변경된 역할에 대한 공감대 형성과 교육이 필요함

3. MDD 사례분석

(1) 국내외 사례

○ MDD로 구축된 국내외 사례들은 아래 표와 같음. 다양한 분야에 도입되고 있으며, 전체 개발 프로세스에 적용되기보다, 일부 핵심 모듈 위주의 사용이 많음

[표 3] 국내 MDD를 이용한 소프트웨어 개발 현황

업 체	적용 분야	효 과
캐나다 솔콥 (현 HP)[27]	푸르덴셜생명 차세대계약관리 시스템 (2009), 시각화된 3가지 모델링 툴로 화면설계, 업무 로직 설계, 계산식 구현 등 구현	금융회사의 경우 개발과 변경에 필요한 업무단계가 줄어들어 신상품 개발 기간 단축
LG CNS	전북은행 차세대, 수협 정책보험 완료 PCA생명, JB캐피탈, 광주은행 등 진행	금융 프로젝트 Model자산화로 금융 프로젝트에 MDD 지속적 적용
SKC&C[28] (IBM RSA 기반)	대신증권 (2009)	자바, C 혼합, 대신증권 자체 유지보수 및 신규 시스템 개발
자동차 업계, 국방, 항공[29]	BMW, 현대모비스	ISO26262,AUTOSAR[30] MBD적용 권장 : 국내는 많이 적용되지 있지 않으나, 관심 커짐 DO-278C(항공) 에 MBD 도입
Lockheed Martin[36]	차세대 어플리케이션 개발 - F-16 전투기에 장착될 시스템 - 모델/지식자산화	MDD 초기 수용자 (90년대) 생산성 향상 정량적 분석 시도
Conquest, Inc. and Popkin, Inc[31] (미국 IT 서비스 업체)	미국 정부 공공기관에 SI시스템과 SW 솔루션을 제공	기존 시스템을 모델기반의 아키텍처로 구축함, 시스템에 대한 문서를 자동 생성 다이어그램을 적극 활용하여 커뮤니케이션 효율성을 높임
BankHOST[32]	인터넷 뱅킹 플랫폼의 게이트웨이 경보 시스템 개발	UML을 사용하여 명확한 커뮤니케이션이 가능, 코드 자동생성으로 에러를 90%정도 없앰 서버 사이드 코딩 없이 유저 사용 샘플 화면 생성 가능
M1 Global Solutions[33]	M1 글로벌은 기업운영을 더욱 효과적으로 할수 있도록 개발툴, 인프라스트럭처, 인력을 제공하는 회사	MDD를 고객 프로젝트에 적용하여 개발자 생산성 50% 향상, 프로젝트 기간 62%를 줄임

(2) Lockheed Martin[34] 사례[35]

□ 항공기 소프트웨어에 OMG의 MDA를 활용하여 여러 플랫폼에 사용가능하고, 생산성과 품질을 높이는 소프트웨어 생산

○ Application Software Interface의 변경에도 불구하고 어플리케이션 소프트웨어 변경이 없는 소프트웨어 개발

- 기존 : 객체지향 방법론을 이용한 CASE 도구를 사용
- 변경 : Kennedy Carter's iUML 도구 사용

[그림 5] 개발 소프트웨어 아키텍처

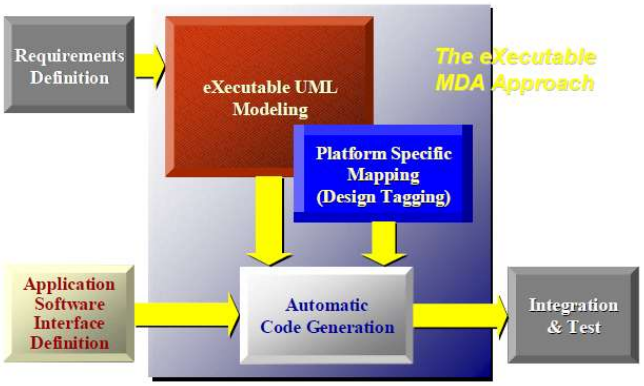
자료 : Lockheed Martin(MDA Success Story)에서 재구성

○ 사용자 요구사항(Requirements Definetions)을 모델링(eXecutable UML Modeling)하고, 하드웨어 관련 사항(Platform Specfic Mapping)을 추가하여 개발함

- 실행가능한 UML(xUML)의 시뮬레이션과 디버깅툴을 이용하여 초기에 테스팅을 수행하고, 시스템의 상세화를 통해서 어플리케이션이 수행하는 바를 정의하여 어플리케이션 개발

27) http://www.zdnet.co.kr/news/news_view.asp?artice_id=20091207142928&type=det&re=
28) http://www.ddaily.co.kr/news/article.html?no=58233
29) http://www.etnews.com/201309270318
30) AUTOSAR(AUTomotive Open System Architecture) : 자동차 제어기에 들어가는 소프트웨어 규격 표준화 단체
31) http://www.omg.org/mda/mda_files/popkinconquest.htm
32) http://www.omg.org/mda/mda_files/Metanology-BankHOSTstory.htm
33) http://www.omg.org/mda/mda_files/M1Global.htm
34) Lockheed Martin 사는 전투기 제작사이자 첨단 기술회사, 2013년 차세대 3군 통합 전투기 사업(Joint Strike Fighter)으로 선정된 F-35 Lightning Ⅱ의 주계약자로서 개발을 진행 중임
35) Lockheed Martin(MDA Success Story), www.omg.org/mda/mda_files/LockheedMartin.pdf

[그림 6] MDD 개발 프로세스

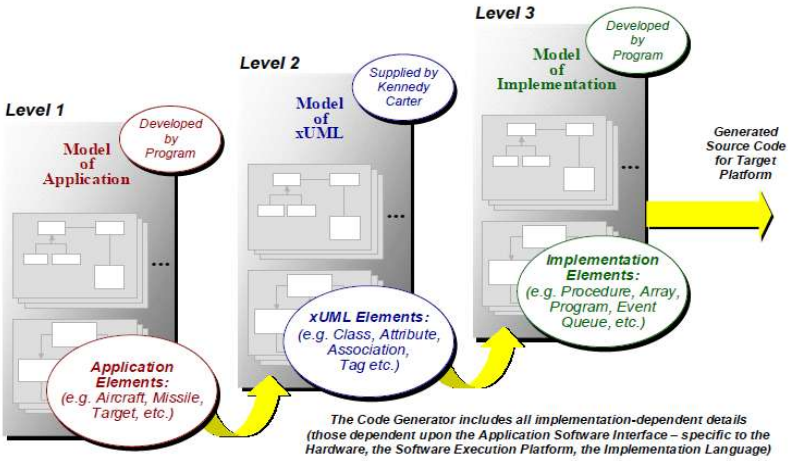


자료 : Lockheed Martin(MDA Success Story)

○ MDA의 PIM에서 PSM, 소스코드 변환 단계와 각 단계의 컴포넌트는 아래와 같이 설명됨

- Level1은 사용자 요구사항 수집단계, Level2은 플랫폼에 연관이 없는 메타모델 단계(PIM), Level3는 플랫폼에 종속된 모델 단계(PSM)임

[그림 7] PIM에서 PSM 변경 프로세스



자료 : Lockheed Martin(MDA Success Story)

□ 이 사례에서 어플리케이션 개발 시간이 20% 감소되고, 여러 플랫폼에서 사용가능한 소프트웨어가 개발되었음

○ UML을 사용하는 도메인 전문가는 하드웨어와 소프트웨어를 분리하여 작업

○ 하드웨어와 소프트웨어 플랫폼이 항공기 어플리케이션에 영향을 주지 않고 업그레이드 가능함

○ 코드 자동생산에 의해 코드 수작업이 감소되어 생산성 향상을 이룸

(3) 전북은행 프로젝트 사례

□ 은행 계정계 시스템을 MDD 방식으로 개발

○ 계정계 시스템 전체 업무(분석, 설계, 개발, 통합테스트, 이행)를 JAVA언어 기반의 MDD 방법론으로 구축

- 총 350여명의 전문 인력이 약 20개월간 프로젝트 진행
- 요구사항 3,811개, 화면 및 보고서 4,017개, 프로그램 11,261개, 테이블 2,063개, 인터페이스 4,201개의 산출물 생산

□ 금융시스템에 Full MDD[36] 방식을 적용하여 성공적으로 개발 구축한 첫사례

○ 기존의 은행 시스템은 주로 코볼이나 C언어로 구축되어 있는 경우가 많기 때문에, 새로운 언어로 구축하려면 큰 리스크가 따름

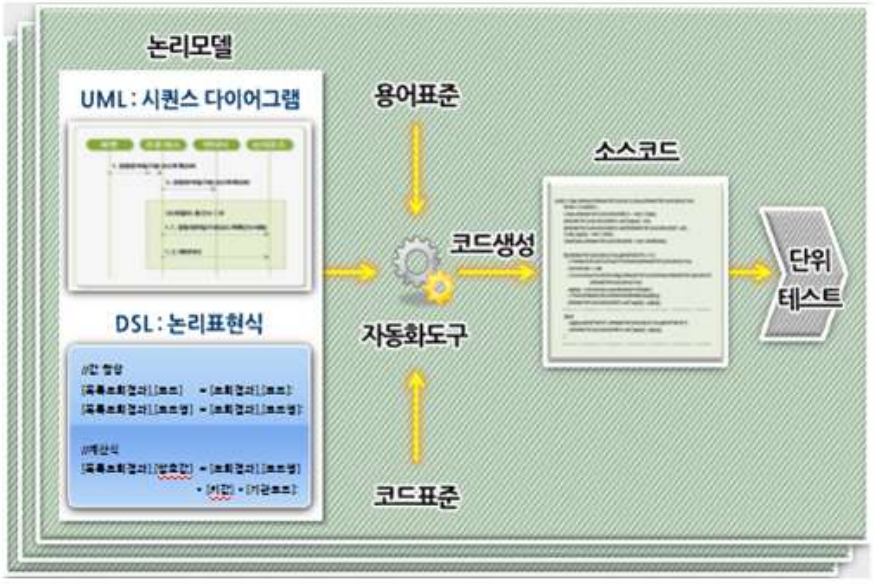
- 해당 프로젝트에서 JAVA언어를 경험한 인력은 약 15%에 불과

○ 이러한 리스크를 자체 개발도구를 활용한 Full MDD 방식을 적용해 극복

- 프로그램 코드와 산출물 문서는 수작업으로 작성하지 않고, 자체 개발한 MDD 도구를 통해 모델로부터 자동 생성
- 영문 텍스트 기반의 프로그램 코드 보다는 한글로 작성된 그래픽 모델 중심으로 프로젝트를 진행
- 자체개발한 표준화된 논리표현식을 사용하여 개발언어와 플랫폼에 대한 종속성을 줄임
- 모델을 중시하고 프로그램 코드를 자동생성 하기 때문에 개발 단계 보다는 설계단계에 많은 비중을 둠

36) 생성한 모델에서 100% 코드 자동 생성 되는 MDD 방식

[그림 8] MDD 개발 과정



자료 : LG CNS 홈페이지(http://mdd.lgcns.co.kr/mdd/html/main.html)

○ MDD를 도입하여 기존 프로젝트 대비 6.6% 생산성 증가[37)]

- 모델과 논리표현식을 활용한 개발로 초급 개발자도 중급 개발자와 비슷한 정도의 생산성 확보

  * 초급 등급 개발자 오퍼레이션 개발 시간 20% 감소

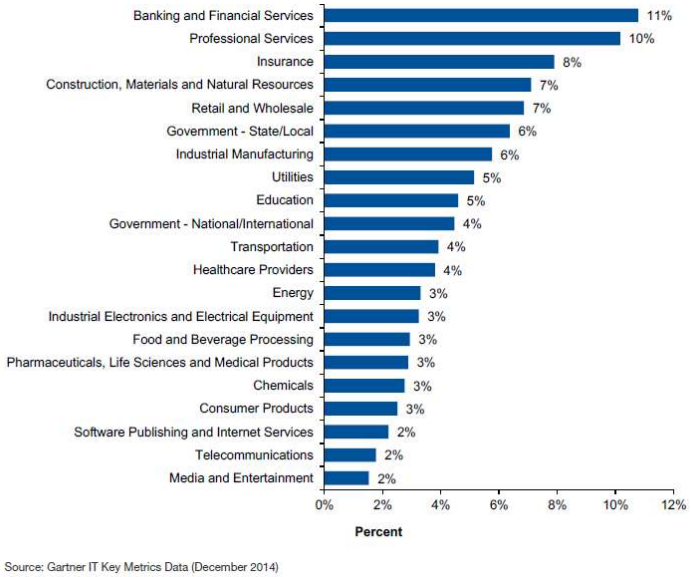
- 개발 경력자 능력에 의존도가 높은 ISD(Information System Development) 프로젝트의 한계점 극복

○ 은행 프로젝트를 분석부터 이행까지 MDD를 이용하여 수행하였으며, 은행 업무 프로세스에 대한 지식이 축적됨

- 어플리케이션 중 은행 및 파이넌스 서비스에 대한 비중은 11%로 은행 업무에 대한 지식 축적은 어플리케이션 매출에 간과할 수 없는 비중임[38)]

37) Software 개발방법론의 최전선 MDD의 개발 생산성 제고 사례, ENTRUE WORLD 2015, 이병태
38) IT Key Metrics Data 2015:Key Applications Measures: Project Measures : Current Year, Gartner, 15 Dec. 2014

[그림 9] 분야별 IT 서비스 프로젝트 점유율



자료 : Gartner IT Key Metrics Data (December 2014)

4. MDD 전망과 과제

(1) 향후 전망

□ 지금까지 살펴본 바와 같이, MDD는 특정분야에 있어 기능적인 유용성이 있는 것으로 보임

○ 기존에 설계와 개발이 분리되던 소프트웨어 개발방식에서 설계와 개발이 동기화되어 문서 작성, 유지보수 등에 도움을 줌

○ 이에 따라 업무와 소프트웨어 개발을 모두 알아야 하는 전문가의 수는 줄어듦

- 소프트웨어 설계는 도메인 전문가가 수행하여, 급변하는 경영환경에 따른 사용자 요구사항의 변화를 소프트웨어에 정확히 반영하도록 함[39]

□ 모바일, 클라우드 등의 영향으로 앞으로의 소프트웨어는 개발 주기가 점점 짧아지고, 개발 플랫폼도 급속도로 다양화해질 것으로 전망

○ 이에 따라 도메인 기술이 축적된 어플리케이션을 보다 간단하고 다양하게 적용할 수 있는 소프트웨어 개발방법론이 필요할 것으로 보여 MDD의 중요성이 커질 것으로 예상됨

□ 앞에서 살펴본 바와 같이, 국내는 다양한 분야에서 MDD 적용이 시도되고 있으며, 특히 은행 분야는 일부 성과를 보여줌

○ 따라서 은행 분야에 있어서는 도메인 지식이 모델에 많이 축적될 것으로 예상되며, 은행 및 재무 분야 - 특히 핀테크 등 - 관련된 산업 비중도 커서 효과가 기대됨

- 은행분야 외에도 비슷한 프로세스를 사용하는 분야를 발굴하여 지속적으로 모델을 발전시키는 것이 중요함

39) Transition to model-driven engineering: what is revolutionary, what remains the same?, Jorge Aranda, Proceeding MODELS'12 Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems

□ 앞으로 모델에 대한 다양한 사례와 시행착오가 축적이 되고, 개발참여자들과 경영층의 이해가 증가하면서 MDD의 활용도도 증가할 것으로 예상됨

○ 작은 규모부터 점차적으로 적용하여 도메인 지식을 축적하는 것이 모델의 활용성에 도움이 될 것으로 보임

□ 특히 국내의 경우 다음의 두 가지 측면에서 MDD의 활용도를 높여나가는 것이 필요

○ 첫째, 국내 소프트웨어 개발인력의 구성이 고급인력은 부족한 반면, 초, 중급 인력이 상대적으로 많아 소프트웨어 개발, 운영 측면에서 상당히 유용한 소프트웨어 개발방법론이 될 수 있음

- 인력수준별 부족인식률은 초급 3.3%, 중급 45%, 고급 51.7%임[40)]

[표4] 기술 등급 분류

기술 등급	기술자격자
고급	중급기술자 자격 취득 후 3년 이상 SW 기술 분야 업무를 수행한 자 또는 박사학위 및 기사 자격을 가진 자
중급	기사 자격증을 취득한 자로서 3년 이상 SW 기술 분야 업무를 수행한 자
초급	기사 자격증을 취득한 자

○ 둘째, 국내 소프트웨어 개발인력의 노동집약적 특성을 통한 MDD 활용

- 모델 기술 적용 등에 대한 다양한 축적이 MDD 적용의 중요한 경쟁력임

- 소프트웨어 산업이 발전하지 못한 우리나라에서 이를 한 단계 업그레이드할 수 있는 중요한 계기가 될 수 있음

* 세계 100대 IT서비스 기업 중 한국 기업은 3개(매출규모 순위: 삼성SDS 27위, LGCNS 51위, SK C&C 71위)이고, 한국의 IT서비스 시장 점유율은 1.4%(매출규모)[41)]

40) 산업기술인력의 경력경로에 관한 연구-소프트웨어 산업을 중심으로, 산업연구원, 최희선, 김주영, 조진환, p106, 2012.12

(2) 향후 과제와 제언

□ MDD 도구의 기본 기능인 모델변환 방법 및 기술적 실현 가능성에 대한 검토가 필요

○ MDD의 성공여부는 MDD의 장점을 얼마나 기술적으로 잘 살릴 수 있느냐에 달림

- 즉, 요구사항을 명확하게 하고, 급변하는 경영 환경과 기술 환경에서 MDD를 이용하여 어떻게 소프트웨어의 변경을 최소화하면서 사용자에 요구사항에 맞추어야 할지에 좌우될 것임

○ 이를 위해서는 기능이 충분하고 효율적인 MDD 도구가 필요하나 현재는 이론적 역할을 수행할 만한 충분한 기능을 가진 MDD 도구는 없으며[42], 성공한 개별 프로젝트에서 개발한 여러 종류의 MDD 도구가 있음

- MDD 도구는 개별 프로젝트에 맞추어 변경되어야 하며, 업무 및 모델에 밀접하게 관계될 것으로 예상됨. 얼마나 업무에 자유로운 도구를 만들 수 있냐는 프로젝트 생산성 향상과 밀접한 관계가 있음

□ MDD를 이용한 소프트웨어 개발을 위한 조직 연구가 필요

○ Jon Whittle에 따르면 MDD 적용의 성공 요인은 기술적 요인뿐만 아니라, 조직적 요인도 중요하다고 함[43]

○ MDD를 생산성 향상뿐만 아니라 비즈니스에서 발생하는 문제해결의 방편으로도 고려되며, MDD 적용 기업들의 경우 그 효과가 불확실하더라도 어느 정도 위험을 감수하는 경향이 있는 것으로 나타남[43]

- 사례로 프린터 개발에 있어서 소프트웨어 개발이 병목이 되어 프린터 출시에 문제가 있던 회사가 기존과 다른 소프트웨어 개발 방법을 모색하고 MDD를 적용 후 MDD의 효과에 의해 소프트웨어 개발에 문제없이 프린터를 출시한 경우가 있음

○ MDD 성공 여부는 조직적으로 얼마나 MDD 역할에 중요도를 두느냐에 달림

41) Gartner 2015.03, 2014년 기준 자료
42) Industrial Adoption of Model-driven Engineering : Are the Tools Really the Problem?, Jon Whittle et al., Model-Driven Engineering Languages and systems
43) The State of Practice in Model-Driven Engineering, Jon Whittle et al., IEEE Computer, 2013.4.23.

□ 특히 앞으로 MDD가 성공하기 위해서는 프로젝트 참여자들이 적극적으로 MDD를 수용하게 하는 것이 기술적 문제만큼이나 중요함

○ MDD 적용에 있어 참여자간 역할변화에 따른 차이점 극복이 필요

- 소프트웨어 아키텍트는 자신이 만든 형식, 제한, 패턴을 MDD 코드에 적용하기 때문에, 아키텍트 표준 준수에 용이한 방법론으로 MDD 적용에 긍정적

- 그 외 참여자들은 추상적으로 생각하는 방식을 학습하는 것에 어려움을 겪을 수 있음

- MDD 프로세스에 대한 이해와 모델링 방법 및 MDD 도구 사용방법 등의 교육을 강화할 필요가 있음

○ 고급개발자의 MDD 도구개발자로 자연스러운 역할 변경 유도

- 고급개발자들은 회사 내에서 자신의 중요성이 감소되고 창작성에 방해가 되기 때문에 MDD 적용에 부정적 경향이 있음

- MDD 도구를 개발할 도메인 기술과 소프트웨어 기술을 가진 전문가로 전환 유도

□ 국내에서 MDD가 보다 활성화되고 소프트웨어 산업 발전에 기여하기 위해서는 현실적이고 전략적인 접근이 필요함

○ MDD의 특성 및 국내 인력자원 구조 이용

- 소프트웨어 산업은 기초 인프라 부터 현장 비즈니스 여건이 맞아야 하는 장기적이고 질적인 도약이 필요한 어려운 과제

- 국내 인력자원 구조에 적합하고 국제적 분업이 가능한 MDD 특성을 이용하여 국내 소프트사업의 부족한 기술을 보완하는 전략적 접근

○ 따라서 우리 소프트웨어 산업의 현실을 감안한 보다 실현가능한 타겟을 정해 이를 이루어 가는 목표와 전략이 필요

- 지금 우리 소프트웨어 산업은 MDD 개발방법론을 적용 발전시킬

준비가 충분치 않은 초기 단계

- 현 단계에서는 기업들이 MDD에 대한 효용을 느끼고 MDD 적용 시도가 활성화될 수 있도록 관련 정보를 소개하고 공유할 수 있도록 하고 모델적용사례를 확대 하는 등의 지속적 시도가 필요
- 이를 바탕으로 기틀이 잡히면 기술의 비즈니스화가 본격화될 수 있도록 유도하고 본격적인 기업간 경쟁, 세계 시장진출을 확대토록 하는 정책 필요

  예) 정부 소프트웨어 조달 시장에서 MDD 적용 등의 수요 확대 정책 등

[참고문헌]

1. 국내문헌

이병태. (2015). Software 개발방법론의 최전선 MDD의 개발 생산성 제고 사례. ENTRUE WORLD

산업연구원. (2012). 산업기술인력의 경력경로에 관한 연구-소프트웨어 산업을 중심으로

지식경제부. (2012). 소프트웨어 요구사항분석, 적용 가이드

2. 국외문헌

B.hailpern, P,Tarr. (2006). Model-driven software development, Model-driven development : The good, the bad, and the ugly. IBM Systems Journal

CHAOS Report by Standish Group. (2015). http://www.infoq.com/articles/standish-chaos-2015

D.C. Schmidt. (2006). Model-Drive Engineering. IEEE Computer society

Gartner. (2014). Market Guide for Rapid Mobile App Development Tools

Gartner. (2014). IT Key Metrics Data 2015:Key Applications Measures: Cost and staff Profile: Multi year

Gartner. (2014). IT Key Metrics Data 2015:Key Applications Measures: Application Support : Multi year

Gartner. (2014). IT Key Metrics Data 2015:Key Applications Measures: Project Measures : Current Year, Gartner

Jon Whittle et al. (2013) Industrial Adoption of Model-driven Engineering : Are the Tools Really the Problem?. Model-Driven Engineering Languages and systems

Jon Whittle et al. (2013) The State of Practice in Model-Driven Engineering. IEEE Computer

Jorge Aranda. (2012). Transition to model-driven engineering: what is revolutionary, what remains the same?. Proceeding MODELS'12 Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems

Object Management Group Model Driven Architecture(MDA) MDA Guide rev. 2.0, (2014)

Peter Swithinbank et all. (2005) Patterns: Model-Driven Development Using IBM Rational Software Architect.

Rim sharma. (2011). Cloud SaaS and Model Driven Architecture. Proc. of the International Conf. on Advanced Computing & Communication Tech

주 의

1. 이 보고서는 소프트웨어정책연구소에서 수행한 연구보고서입니다.
2. 이 보고서의 내용을 발표할 때에는 반드시 소프트웨어정책연구소에서 수행한 연구결과임을 밝혀야 합니다.