OOAD, UML, DESIGN PATTERN, RUP 과 개발 가이드

Overview

2025.12

강태욱

Table of Contents

1. 소프트웨어 복잡성과 방법론의 진화

현대 소프트웨어 공학의 역사는 복잡성(Complexity)과의 끝없는 투쟁으로 정의될 수 있다. 시스템의 규모가 커지고 비즈니스 요구사항이 정교해짐에 따라, 단순히 코드를 작성하는 행위를 넘어 시스템을 구조화하고, 계획하며, 품질을 보증하는 체계적인 접근 방식이 필수적으로 요구되었다. 본 보고서는 이러한 배경 속에서 탄생한 객체지향 분석 및 설계(OOAD), 통합 모델링 언어(UML), Rational Unified Process(RUP), 그리고 익스트림 프로그래밍(XP)의 이론적 토대와 실무적 적용 방법을 포괄적으로 조사하였다. 특히, Grady Booch, James Rumbaugh, Ivar Jacobson 등 소위 'Three Amigos'가 주도한 방법론의 통합 과정과, 이와는 별개의 영역에서 디자인 패턴을 정립한 Erich Gamma(Gang of Four)의 역할을 명확히 구분하여 기술한다. 또한, 이러한 이론적 배경이 실제 산업 표준인 ISO 19166(BIM-GIS 매핑)과 디지털 트윈(Digital Twin) 개발에 어떻게 적용되는지 상세한 '따라하기 가이드' 형식으로 제시함으로써 이론과 실무의 간극을 메우고자 한다.

2. 객체지향 분석 및 설계 (OOAD)와 방법론 전쟁

2.1 객체지향 패러다임의 태동과 역사적 맥락

객체지향 분석 및 설계(OOAD)는 소프트웨어 시스템을 상호작용하는 객체(Object)들의 집합으로 모델링하는 접근 방식이다. 1980 년대 후반부터 1990 년대 초반은 소프트웨어 개발 방법론의 춘추전국시대라 불릴 만큼 다양한 방법론이 난립하던 시기였다. 1989 년에서 1994 년 사이, 모델링 언어와 방법론의 수는 10 개 미만에서 50 개 이상으로 급증하며 소위 "방법론 전쟁(Method Wars)"이 발발하였다.[1] 이 시기 개발자들은 각기 다른 표기법과 프로세스를 주장하는 방법론 사이에서 혼란을 겪었으며, 이는 산업 전반에 걸쳐 표준화된 언어에 대한 갈망을 증폭시켰다.

이 혼란 속에서 두각을 나타낸 세 가지 주요 흐름은 다음과 같다:

Grady Booch의 Booch Method: Rational Software의 수석 과학자였던 Grady Booch는 설계 및 구축 단계에서의 강력한 표현력을 바탕으로 한 Booch Method를 제안하였다.[1] 그의 표기법은 구름 모양의 클래스 아이콘으로 유명했으나, 이는 훗날 UML 표준에 포함되지 않았다.[3]

James Rumbaugh의 OMT (Object Modeling Technique): General Electric(GE)에서 근무하던 James Rumbaugh는 데이터 중심의 분석에 강점이 있는 OMT를 개발하였다. 이는 엔티티-관계(Entity-Relationship) 모델에 뿌리를 두고 있어 분석 단계에서 특히 유용했다.[1]

Ivar Jacobson의 OOSE (Object-Oriented Software Engineering): Ericsson의 Ivar Jacobson은 시스템의 정적 구조보다는 사용자 관점의 상호작용, 즉 '유스케이스(Use Case)'를 중심으로 한 OOSE를 주창하였다.[1]

이 세 사람은 훗날 UML의 창시자인 "Three Amigos"로 불리게 되지만, 이들과 동시대에 활동하며 소프트웨어 설계의 또 다른 축을 담당한 인물이 Erich Gamma이다. Erich Gamma는 Richard Helm, Ralph Johnson, John Vlissides와 함께 "Design Patterns: Elements of Reusable Object-Oriented Software"를 저술하며 'Gang of Four(GoF)'로 알려졌다.[2] 중요한 점은, Booch, Rumbaugh, Jacobson이 모델링 **언어와 프로세스**를 통합하려 노력했다면, Gamma는 객체지향 설계에서 반복적으로 나타나는 **문제 해결 패턴**을 정립했다는 것이다. 따라서 이들이 함께 찍은 사진이 드문 이유는 그들의 주 활동 영역과 소속 조직(Rational Software vs. 학계 및 타 연구소)이 달랐기 때문임을 이해하는 것이 중요하다.

2.2 OOAD의 핵심 원칙과 SOLID

객체지향 설계가 단순히 클래스를 정의하는 것을 넘어 유지보수 가능하고 확장 가능한 시스템을 만들기 위해서는 견고한 설계 원칙이 필요하다. Robert C. Martin이 제안한 SOLID 원칙은 OOAD의 정수(essence)를 담고 있으며, 나쁜 설계(Bad Design)에서 나타나는 경직성, 취약성, 부동성을 예방한다.[4]

2.2.1 단일 책임 원칙 (Single Responsibility Principle, SRP)

"클래스는 변경해야 할 이유가 단 하나여야 한다"는 원칙이다.[4] 이는 하나의 클래스가 너무 많은 기능을 수행하지 않도록 책임을 분리하는 것을 의미한다.

위반 사례: Employee 클래스가 급여 계산(비즈니스 로직)과 보고서 출력(프레젠테이션 로직)을 모두 담당하는 경우. 급여 산정 방식이 바뀌거나 보고서 형식이 바뀔 때마다 이 클래스를 수정해야 하므로 SRP를 위반한다.

적용: PayrollCalculator 클래스와 EmployeeReporter 클래스로 분리하여 각각의 변경 이유를 격리시킨다.[6]

2.2.2 개방-폐쇄 원칙 (Open/Closed Principle, OCP)

"소프트웨어 엔티티(클래스, 모듈, 함수 등)는 확장에 대해서는 열려 있어야 하고, 수정에 대해서는 닫혀 있어야 한다".[5]

매커니즘: 기존 코드를 건드리지 않고 새로운 기능을 추가할 수 있어야 한다. 이는 주로 추상화(Interface)와 다형성(Polymorphism)을 통해 달성된다.

심층 분석: 새로운 결제 방식(예: 암호화폐)이 추가될 때, 기존의 PaymentProcessor 클래스의 if-else 문을 수정하는 것이 아니라, Payment 인터페이스를 구현한 새로운 클래스를 생성하여 시스템에 플러그인하는 방식이 OCP의 전형적인 예이다.

2.2.3 리스코프 치환 원칙 (Liskov Substitution Principle, LSP)

"서브 타입은 언제나 자신의 기반 타입으로 교체할 수 있어야 한다".[4] 이는 상속의 오용을 방지하는 가장 중요한 원칙이다.

직사각형과 정사각형의 역설: 수학적으로 정사각형은 직사각형이다. 그러나 OOAD에서 Rectangle 클래스를 상속받아 Square 클래스를 만들고, setWidth 메서드에서 가로와 세로를 동시에 변경하도록 오버라이딩하면, Rectangle로 동작하기를 기대하는 클라이언트 코드(가로만 변경되기를 기대함)를 파괴하게 된다. 이는 LSP 위반의 대표적인 사례로, 상속보다는 구성(Composition)을 고려해야 함을 시사한다.[5]

2.2.4 인터페이스 분리 원칙 (Interface Segregation Principle, ISP)

"클라이언트는 자신이 사용하지 않는 메서드에 의존하도록 강요받아서는 안 된다".[4]

Fat Interface 문제: 하나의 거대한 인터페이스(God Interface)에 모든 기능을 넣으면, 이를 구현하는 클래스는 필요 없는 메서드까지 빈 구현체(Empty Implementation)로 남겨야 한다. 이는 불필요한 의존성을 낳는다.

해결책: 인터페이스를 역할별로 잘게 쪼개어(Role Interface), 클라이언트가 필요한 인터페이스만 구현하거나 사용하도록 한다.[6]

2.2.5 의존성 역전 원칙 (Dependency Inversion Principle, DIP)

"고차원 모듈은 저차원 모듈에 의존해서는 안 된다. 둘 다 추상화에 의존해야 한다".[4]

제어의 역전: 전통적인 절차적 프로그래밍에서는 상위 모듈이 하위 모듈을 직접 호출했다. DIP는 하위 모듈이 상위 모듈이 정의한 인터페이스를 구현하게 함으로써 의존성의 방향을 역전시킨다. 이는 시스템의 결합도를 낮추고 유닛 테스트를 용이하게 한다.

3. UML (Unified Modeling Language)

3.1 통합의 역사와 표준화

"방법론 전쟁"의 피로감 속에서, 1994 년 Rational Software에 합류한 James Rumbaugh와 Grady Booch는 각자의 방법론(OMT와 Booch Method)을 통합하기 시작했다. 1995 년 OOPSLA 컨퍼런스에서 이들의 연합이 공식화되었고, 이어 Ivar Jacobson이 합류하면서 'Three Amigos'가 완성되었다.[1] 이들은 1996 년 UML 0.9 를 발표하였고, 이는 1997 년 11 월 OMG(Object Management Group)에 의해 표준으로 채택되었다.[1]

UML의 목표는 새로운 표기법을 창조하는 것이 아니라, 기존의 검증된 방법론들(Booch의 설계, OMT의 분석, OOSE의 유스케이스, Harel의 Statechart 등)을 통합하고 단순화하여 범용적인 모델링 언어를 제공하는 것이었다.[10] 이후 UML 2.0(2005 년)과 현재의 UML 2.5.x로 발전하면서, 모델 주도 개발(Model Driven Development)을 지원하기 위한 정밀한 시멘틱(Semantics)과 메타모델(Metamodel)을 갖추게 되었다.[11]

3.2 UML 다이어그램의 분류 체계

UML 2.5 는 다이어그램을 크게 **구조(Structure)** 다이어그램과 **행위(Behavior)** 다이어그램으로 분류한다.[12]

3.2.1 구조 다이어그램 (Structure Diagrams)

시스템의 정적인 구조, 즉 시간의 흐름과 무관하게 존재하는 요소들을 표현한다.

다이어그램 유형	설명 및 주요 요소	활용 목적
클래스 다이어그램 (Class Diagram)	시스템의 클래스, 속성, 오퍼레이션 및 관계(연관, 일반화, 의존 등)를 표현.[14]	시스템의 정적 구조 설계, DB 스키마 도출, 코드 생성의 기초. 가장 많이 사용됨.
컴포넌트 다이어그램 (Component Diagram)	소프트웨어 컴포넌트(모듈, 패키지, 파일 등) 간의 와이어링과 의존성을 표현.[12]	대규모 시스템의 아키텍처 조망, 인터페이스 정의, 조립 구조 설계.
배치 다이어그램 (Deployment Diagram)	하드웨어 노드(서버, 디바이스)와 그 위에 실행되는 소프트웨어 아티팩트의 배치를	물리적 인프라 설계, 네트워크 토폴로지 매핑, 분산 시스템 구성.

	표현.[12]	
객체 다이어그램 (Object Diagram)	특정 시점에서의 클래스 인스턴스와 그들 간의 관계를 스냅샷 형태로 표현.[13]	복잡한 데이터 구조의 예시 설명, 런타임 상태 디버깅.
패키지 다이어그램 (Package Diagram)	모델 요소를 그룹화한 패키지와 그들 간의 의존 관계를 표현.	대규모 시스템의 네임스페이스 관리, 레이어 아키텍처 표현.

클래스 다이어그램의 상세:

클래스는 이름, 속성, 오퍼레이션의 3 단 구획으로 표현된다. 관계의 표현은 매우 중요한데, 단순 연관(실선), 집합(Aggregation, 빈 마름모), 합성(Composition, 채워진 마름모), 상속(Generalization, 빈 삼각형 화살표)을 정확히 구분해야 한다.14 특히 합성은 '부분'이 '전체'의 생명주기에 종속됨을 의미하는 강력한 결합이다.

3.2.2 행위 다이어그램 (Behavior Diagrams)

시스템 내 객체들의 동적인 상호작용과 상태 변화를 표현한다.

다이어그램 유형	**설명 및 주요 요소**	**활용 목적**
유스케이스 다이어그램 (Use Case Diagram)	시스템(Subject), 액터(Actor), 유스케이스(Use Case) 및 관계를 표현.[10]	요구사항 정의, 시스템 범위(Scope) 설정, 이해관계자와의 커뮤니케이션.
시퀀스 다이어그램 (Sequence Diagram)	시간 순서에 따른 객체 간의 메시지 교환을 표현. 라이프라인(Lifeline)과 메시지 화살표가 핵심.[12]	특정 유스케이스의 상세 로직 설계, API 호출 흐름 정의.
상태 머신 다이어그램 (State Machine Diagram)	단일 객체의 상태(State)와 이벤트(Event)에 따른	복잡한 라이프사이클을 가진 객체(예: 주문, 결제 세션)의

	전이(Transition)를 표현.[12]	로직 검증.
활동 다이어그램 (Activity Diagram)	업무 프로세스나 알고리즘의 실행 흐름을 표현. 분기(Decision), 병합(Merge), 포크(Fork), 조인(Join) 포함.[12]	비즈니스 워크플로우 분석, 병렬 처리 로직 설계.
통신 다이어그램 (Communication Diagram)	객체 간의 연결 관계와 메시지 흐름을 동시에 표현 (시퀀스와 유사하나 구조 강조).[12]	객체 간의 정적 연결과 동적 메시지를 한눈에 파악.

4. RUP (Rational Unified Process)와 4 단계/9 원칙

4.1 RUP의 철학: 반복과 위험 관리

Rational Unified Process(RUP)는 UML을 만든 Rational Software가 개발한 프로세스 프레임워크로, "누가, 언제, 무엇을, 어떻게" 해야 하는지를 정의한다. RUP는 반복적(Iterative)이고, 아키텍처 중심(Architecture-centric)이며, 유스케이스 주도(Use-case driven)의 개발 프로세스이다.[16]

RUP의 구조는 시간 축(Phases)과 내용 축(Disciplines)의 2 차원 매트릭스로 표현된다.[18] 이는 프로젝트가 진행됨에 따라 각 훈련(Discipline)의 비중이 어떻게 변화하는지를 시각적으로 보여주는 'Hump Chart'로 유명하다.

4.2 RUP의 4 단계 생명주기 (Phases)

각 단계는 엄격한 마일스톤(Milestone)을 가지며, 이를 통과해야만 다음 단계로 넘어갈 수 있다.[19]

4.2.1 도입 단계 (Inception Phase)

목표: 프로젝트의 범위를 설정하고 비즈니스 타당성을 검토한다. 이해관계자 간의 합의를 도출하는 것이 핵심이다.

주요 활동: 비전 수립, 핵심 유스케이스 식별(전체의 약 10-20%), 초기 리스크 평가, 비용 및 일정 추정.

산출물: 비전 문서(Vision), 비즈니스 케이스(Business Case), 리스크 목록(Risk List), 초기 유스케이스 모델.[20]

마일스톤: Lifecycle Objectives (LCO) - 프로젝트를 진행할 가치가 있는가?

4.2.2 정련 단계 (Elaboration Phase)

목표: 아키텍처를 확립하고 고위험 요소를 제거한다. RUP에서 가장 중요한 단계로, "실행 가능한 아키텍처 베이스라인(Executable Architecture Baseline)"을 구축해야 한다.[21]

주요 활동: 상세 요구사항 분석(유스케이스의 80% 상세화), 아키텍처 설계 및 프로토타이핑, 주요 기술적 리스크 해결.

산출물: 소프트웨어 아키텍처 문서(SAD), 실행 가능한 아키텍처 프로토타입, 수정된 리스크 목록, 상세 반복 계획.[21]

마일스톤: Lifecycle Architecture (LCA) - 아키텍처가 안정적이며 리스크가 통제 가능한가?

4.2.3 구축 단계 (Construction Phase)

목표: 남은 기능을 구현하고 시스템을 완성한다. 아키텍처가 안정되었으므로 효율적인 생산에 집중한다.[16]

주요 활동: 컴포넌트 개발, 코딩, 단위 테스트, 통합 테스트, 문서화.

산출물: 소프트웨어 제품, 사용자 매뉴얼, 테스트 스위트.[21]

마일스톤: Initial Operational Capability (IOC) - 제품이 베타 테스트를 할 준비가 되었는가?

4.2.4 전이 단계 (Transition Phase)

목표: 소프트웨어를 사용자 환경에 배포하고 피드백을 수렴한다.[16]

주요 활동: 베타 테스트, 데이터 마이그레이션, 사용자 교육, 유지보수 전환.

산출물: 릴리즈 노트, 최종 제품, 설치 가이드.[22]

마일스톤: Product Release - 제품이 최종 사용자 만족도를 충족하는가?

4.3 9 가지 핵심 훈련 (Disciplines)

각 반복(Iteration) 내에서 수행되는 논리적인 활동 그룹이다.[19]

비즈니스 모델링 (Business Modeling): 조직의 업무 프로세스를 이해하고 모델링.

요구사항 (Requirements): 이해관계자의 요구를 유스케이스로 변환.

분석 및 설계 (Analysis & Design): 요구사항을 시스템 아키텍처로 변환.

구현 (Implementation): 코드 작성 및 단위 테스트.

테스트 (Test): 통합 및 시스템 테스트 수행.[24]

배포 (Deployment): 패키징 및 설치.

구성 및 변경 관리 (Configuration & Change Management): 버전 관리 및 변경 요청 추적.[22]

프로젝트 관리 (Project Management): 반복 계획 수립 및 리스크 관리.[25]

환경 (Environment): 도구 및 프로세스 지원 인프라 구축.

4.4 반복 계획 (Iteration Plan)의 실제

RUP의 반복 계획은 전체 프로젝트 계획(Coarse-grained)과는 달리 매우 상세(Fine-grained)하다.[20]

구성: 작업(Task), 담당자(Role), 산출물(Artifact), 기간(Duration)이 명시된다.

예시: "반복 계획 수립" -> "유스케이스 상세화" -> "설계 모델 업데이트" -> "코드 구현" -> "테스트 및 평가" -> "반복 평가(Iteration Assessment)".[25]

5. XP (Extreme Programming) 실천법

5.1 XP의 핵심 가치와 철학

XP는 켄트 벡(Kent Beck)이 창시한 애자일 방법론으로, "변화에 대한 비용 곡선을 평탄하게 만드는 것"을 목표로 한다.[26] XP는 개발 과정에서의 불확실성을 인정하고, 이를 5 가지 핵심 가치를 통해 관리한다.

의사소통 (Communication): 문서보다는 대화를 중시하며, 고객과 개발자, 개발자와 개발자 간의 장벽을 허문다.[26]

단순성 (Simplicity): "지금 당장 필요한 것만 만든다(YAGNI - You Ain't Gonna Need It)." 미래를 위한 과도한 설계를 지양한다.

피드백 (Feedback): 짧은 주기의 릴리즈와 테스트를 통해 시스템의 상태를 지속적으로 확인한다.

용기 (Courage): 코드를 버리거나, 리팩토링하거나, 고객에게 정직한 일정을 말할 수 있는 용기를 의미한다.

존중 (Respect): 팀원 간의 전문성을 인정하고 기여를 존중한다.[28]

5.2 12 가지 실천 항목 (Practices)과 생명 주기

XP의 실천 항목들은 서로 독립적이지 않고 상호 보완적이다. 이를 "생명의 원(Circle of Life)"으로 표현하기도 한다.[29]

5.2.1 코딩 영역 (Inner Circle) - 개발자의 일상

짝 프로그래밍 (Pair Programming): 두 명이 하나의 컴퓨터에서 작업한다. 한 명은 코드를 작성(Driver)하고, 다른 한 명은 전략을 생각하며 검토(Navigator)한다. 이는 실시간 코드 리뷰 효과를 낸다.[31]

테스트 주도 개발 (TDD): 구현하기 전에 실패하는 테스트를 먼저 작성한다. 이는 코드의 결함을 줄이고 설계를 단순하게 유지한다.[31]

리팩토링 (Refactoring): 기능 변경 없이 코드의 내부 구조를 개선한다. TDD와 짝 프로그래밍이 이를 안전하게 만든다.[29]

5.2.2 팀 영역 (Middle Circle) - 협업과 통합

지속적 통합 (Continuous Integration, CI): 코드를 하루에도 수차례 통합하고 빌드한다. "통합 지옥"을 방지한다.[26]

공동 코드 소유 (Collective Code Ownership): 누구든지 시스템의 어떤 코드라도 수정할 수 있다. 병목 현상을 없앤다.[29]

코딩 표준 (Coding Standard): 모든 코드가 한 사람이 짠 것처럼 보이도록 규칙을 정한다.[29]

메타포 (Metaphor): 시스템의 구조를 설명하는 간단하고 공유된 이야기(비유)를 사용한다.[29]

5.2.3 프로세스 영역 (Outer Circle) - 비즈니스와의 연결

계획 게임 (Planning Game): 비즈니스(고객)는 가치를 결정하고, 개발자는 비용(노력)을 추정하여 협상한다.[31]

소규모 릴리즈 (Small Releases): 가능한 한 빨리, 자주 배포하여 피드백을 받는다.[31]

상주 고객 (Whole Team/On-site Customer): 고객이 팀과 함께 앉아 질문에 즉시 답한다.[29]

지속 가능한 속도 (Sustainable Pace): 과도한 야근은 생산성을 떨어뜨린다. 주 40 시간 근무를 원칙으로 한다.[31]

5.3 실천법 간의 상호 의존성 (Reinforcement)

XP의 강력함은 실천법들이 서로를 지탱하는 데서 온다.[26]

리팩토링은 **TDD**가 없으면 위험해서 할 수 없다.

공동 코드 소유는 **짝 프로그래밍**과 **코딩 표준** 없이는 혼란을 초래한다.

단순한 설계는 리팩토링을 통해 지속적으로 유지된다.
따라서 XP는 부분적인 적용보다 전체 실천법을 함께 적용할 때(Extreme) 그 진가가 발휘된다.

6. 부록: ISO 19166 및 디지털 트윈 구현

본 장에서는 앞서 다룬 OOAD, UML, RUP, XP의 개념을 실제 복잡한 엔지니어링 문제인 ISO 19166 (BIM to GIS Conceptual Mapping)과 **디지털 트윈** 프로젝트에 적용하는 과정을 '따라하기 가이드' 형식으로 서술한다.

6.1 시나리오 정의: 스마트 시티를 위한 BIM-GIS 통합

배경: 스마트 시티 구축을 위해 건물 내부의 상세 정보(BIM, IFC 표준)를 지리공간 정보(GIS, CityGML 표준)로 변환하여 도시 전체의 재난 대피 시뮬레이션을 수행하고자 한다.

문제: BIM의 IfcDoor는 물리적 문이지만, GIS의 Door는 논리적 개구부일 수 있다. 서로 다른 의미론적(Semantic) 차이를 극복해야 한다.

해결책: ISO 19166 표준의 매핑 프레임워크를 RUP 프로세스에 따라 구현한다.33

6.2 비즈니스 모델링 및 요구사항

비전 수립: "도시 차원의 재난 대응을 위한 건물 데이터의 GIS 통합 시스템."

관점 정의 (Perspective Definition): ISO 19166 의 첫 번째 메커니즘인 '관점 정의(B2G PD)'를 수행한다.

User Story (XP): "재난 관리자는 대피 경로를 계산하기 위해 건물의 출입구 위치와 폭 정보를 GIS 상에서 확인하고 싶다."

UML 유스케이스 다이어그램: Actor(재난 관리자) --(Uses)--> UseCase(BIM 데이터 가져오기) --(Includes)--> UseCase(좌표 변환).

6.3 매핑 아키텍처 및 요소 매핑 (Element Mapping)

이 단계에서는 **UML 클래스 다이어그램**을 사용하여 소스(IFC)와 타겟(CityGML) 간의 구조적 매핑을 정의한다. 이를 ISO 19166 의 '요소 매핑(B2G EM)'이라 한다.[34]

매핑 테이블 (Mapping Table) 예시:

소스 클래스 (IFC)	관계 (Cardinality)	타겟 클래스 (CityGML)	매핑 로직 및 조건 (Transformation Logic)
IfcBuilding	1:1	AbstractBuilding	속성 Name, Description 직접 매핑. 좌표계 변환 적용.

IfcWall	1:1	WallSurface	기하학적 형상(Geometry)의 면(Face) 추출. 두께 정보 제거.
IfcDoor	1:1	Door	OverallWidth, OverallHeight 속성 매핑. 위치 좌표 변환.
IfcSpace	1:N	Room (LOD4)	InternalOrExternal 속성이 INTERNAL인 경우만 매핑.
IfcSlab	N:1	RoofSurface	PredefinedType='ROOF'인 슬래브만 필터링하여 지붕으로 변환.[35]

OOAD 원칙 적용:

LSP (리스코프 치환): 매핑된 GIS 객체가 원래 BIM 객체의 '대피 가능 여부'라는 행위적 계약을 준수하는가? 만약 GIS Door 객체가 '잠김 상태' 정보를 잃어버린다면 시뮬레이션에서 대체 불가능하므로, 속성을 추가로 확장(Extension)해야 한다.

추상화 (Abstraction): BIM의 나사, 손잡이 등 불필요한 디테일은 제거하고 '개구부'라는 본질만 GIS로 넘긴다.

6.4 XP 실천법을 통한 매핑 로직 구현

매핑 규칙이 복잡하므로 XP의 TDD(테스트 주도 개발)를 적용하여 변환 로직을 구현한다.

Step 1: 실패하는 테스트 작성 (Red)

```
def test_ifc_door_to_citygml_door_transformation():
    # Arrange: 폭 2.0m, 높이 2.5m의 IFC 문 생성
    bim_door = IfcDoor(guid='12345', width=2.0, height=2.5)

    # Act: 매퍼 실행
    gis_door = Mapper.transform(bim_door, target_schema='CityGML')

    # Assert: GIS 문의 속성이 일치하는지 확인
    assert gis_door.width == 2.0
    assert gis_door.type == 'Door'
```

```
    # 아직 Mapper 클래스가 없으므로 이 테스트는 실패한다.
```

Step 2: 통과하는 최소한의 코드 작성 (Green)

```
class Mapper:
    @staticmethod
    def transform(element, target_schema):
        if isinstance(element, IfcDoor):
            return CityGMLDoor(width=element.width, height=element.height)
        return None
```

Step 3: 리팩토링 (Refactor)

단순 if-else 구조를 전략 패턴(Strategy Pattern)으로 변경하여 IfcWall, IfcWindow 등 다양한 요소에 대한 매핑 전략을 유연하게 교체할 수 있도록 개선한다. 이는 OCP(개방-폐쇄 원칙)를 준수하는 것이다.

6.5 LOD 매핑 및 디지털 트윈 고도화

ISO 19166 의 세 번째 메커니즘인 'LOD 매핑(B2G LM)'은 기하학적 단순화를 다룬다.[34]

LOD 1 (Block Model): IfcBuilding의 외곽 박스(Bounding Box)만 추출.

LOD 4 (Interior): IfcSpace와 IfcFlowTerminal(소화기 등)까지 상세 매핑.[36]

디지털 트윈 사례 적용:

제조 라인의 디지털 트윈 구축 사례 37 를 보면, 가상의 '로봇 에이전트'와 물리적 '센서' 간의 통신이 핵심이다.

UML 컴포넌트 다이어그램: ROS Master --(Topic)--> Virtual Robot --(Service)--> Physical Controller.

RUP 전이 단계: 시뮬레이션 환경(가상)에서 검증된 제어 로직을 실제 공장(물리)에 배포(Deploy)하고, 실제 센서 데이터와 가상 모델의 오차를 보정(Calibration)한다.

7. 결론

OOAD의 기본 원칙에서 시작하여 UML이라는 표준 언어, RUP라는 관리 프로세스, 그리고 XP라는 애자일 실천법이 어떻게 유기적으로 연결되는지를 확인하였다.

OOAD와 SOLID는 소프트웨어의 내부 품질(Maintainability)을 보장하는 근본적인 설계 원칙이다.

UML은 복잡한 시스템을 시각화하고 소통(Communication)하기 위한 필수적인 언어이다.

RUP는 대규모 프로젝트의 리스크를 관리(Management)하고 아키텍처를 안정화하는 프레임워크를 제공한다.

XP는 변화하는 요구사항 속에서도 실행 품질(Execution Quality)을 유지하게 하는 개발팀의 습관이다.

ISO 19166 과 디지털 트윈 사례에서 보듯, 복잡한 시스템은 단 하나의 방법론만으로는 해결할 수 없다. RUP의 거시적인 계획 하에, UML을 사용하여 데이터 구조를 엄밀히 정의하고, XP의 TDD와 CI를 통해 견고한 코드를 작성하는 통합된 접근(Integrated Approach)을 적절히 적용할 필요가 있다.

8. 참고 자료

1. UML: The Unified Modeling Language, 1 월 1, 2026 에 액세스, https://cs.smu.ca/~porter/csc/465/notes/uml.html
2. Grady Booch - Wikipedia, 1 월 1, 2026 에 액세스, https://en.wikipedia.org/wiki/Grady_Booch
3. Rebecca's Web – Object design and analysis methods before and after UML - Wirfs-Brock, 1 월 1, 2026 에 액세스, https://wirfs-brock.com/rebecca/blog/2025/04/03/
4. SOLID Design Principles: Hands-On Examples - Splunk, 1 월 1, 2026 에 액세스, https://www.splunk.com/en_us/blog/learn/solid-design-principle.html
5. SOLID Principles with Real Life Examples - GeeksforGeeks, 1 월 1, 2026 에 액세스, https://www.geeksforgeeks.org/system-design/solid-principle-in-programming-understand-with-real-life-examples/
6. SOLID Design Principles Explained: Building Better Software Architecture - DigitalOcean, 1 월 1, 2026 에 액세스, https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
7. SOLID Principles — explained with examples | by Raj Suvariya - Medium, 1 월 1, 2026 에 액세스, https://medium.com/mindorks/solid-principles-explained-with-examples-79d1ce114ace
8. 5. Interface Segregation Principle Explained | SOLID Principles in JavaScript & Node.js, 1 월 1, 2026 에 액세스, https://www.youtube.com/watch?v=-haC7a7Kpe0
9. SOLID Design Principles With Examples : r/ProgrammerTIL - Reddit, 1 월 1, 2026 에 액세스, https://www.reddit.com/r/ProgrammerTIL/comments/11tihj8/solid_design_principles_with_examples/
10. History of UML: Methods and Notations - SourceMaking, 1 월 1, 2026 에 액세스, https://sourcemaking.com/uml/basic-principles-and-background/history-of-uml-methods-and-

notations
11. About the Unified Modeling Language Specification Version 2.5.1, 1 월 1, 2026 에 액세스, https://www.omg.org/spec/UML/2.5.1/About-UML
12. UML 2.5 Diagrams Overview, 1 월 1, 2026 에 액세스, https://www.uml-diagrams.org/uml-25-diagrams.html
13. Blog - UML overview - where and why each UML diagram is used - draw.io, 1 월 1, 2026 에 액세스, https://www.drawio.com/blog/uml-overview
14. Class diagram - Wikipedia, 1 월 1, 2026 에 액세스, https://en.wikipedia.org/wiki/Class_diagram
15. The UML 2 class diagram - IBM Developer, 1 월 1, 2026 에 액세스, https://developer.ibm.com/articles/the-class-diagram/
16. What is RUP(Rational Unified Process) and its Phases? - GeeksforGeeks, 1 월 1, 2026 에 액세스, https://www.geeksforgeeks.org/software-engineering/rup-and-its-phases/
17. Using Rational Unified Process in an SME – A Case Study - ResearchGate, 1 월 1, 2026 에 액세스, https://www.researchgate.net/publication/221045898_Using_Rational_Unified_Process_in_an_SME_-_A_Case_Study
18. Demystifying the Rational Unified Process Diagram: A Visual Guide to Software Development - ONES.com, 1 월 1, 2026 에 액세스, https://ones.com/blog/demystifying-rational-unified-process-diagram/
19. A Manager's Introduction to The Rational Unified Process (RUP) - Ambysoft Inc., 1 월 1, 2026 에 액세스, https://www.ambysoft.com/downloads/managersIntroToRUP.pdf
20. Planning a Project with the Rational Unified Process - NYU, 1 월 1, 2026 에 액세스, https://www.nyu.edu/classes/jcf/CSCI-GA.2440-001_sp15/handouts/PlanningProjWithRUP.pdf
21. RUP - IBM Rational Unified Process/Phases - Wikibooks, 1 월 1, 2026 에 액세스, https://en.wikibooks.org/wiki/RUP_-_IBM_Rational_Unified_Process/Phases
22. Rational Unified Process: Examples, 1 월 1, 2026 에 액세스, https://files.defcon.no/RUP/examples/ovu_arex.htm
23. Rational unified process - Wikipedia, 1 월 1, 2026 에 액세스, https://en.wikipedia.org/wiki/Rational_unified_process
24. Iteration Plan Sample Schedule: Elaboration Phase, 1 월 1, 2026 에 액세스, https://files.defcon.no/RUP/process/itrwkfls/sip_iie.htm
25. Sample Iteration Plan: Inception Phase, 1 월 1, 2026 에 액세스, https://www.tesestec.com.br/pasteurjr/rup/examples/devcase_sp/sip_iii.htm
26. What is Extreme Programming (XP)? - Agile Alliance, 1 월 1, 2026 에 액세스, https://agilealliance.org/glossary/xp/
27. Extreme programming - Wikipedia, 1 월 1, 2026 에 액세스, https://en.wikipedia.org/wiki/Extreme_programming
28. XP Values, Principles & Practices: Complete Overview - Teaching Agile, 1 월 1, 2026 에 액세스, https://teachingagile.com/extreme-programming-xp/overview

29. The 4 Circles of Extreme Programming | JD Meier, 1 월 1, 2026 에 액세스, https://jdmeier.com/4-circles-of-extreme-programming/
30. Concept: XP Practices - Gemserk, 1 월 1, 2026 에 액세스, https://www.gemserk.com/sum/xp/guidances/concepts/xp_practices_36E149F4.html
31. Extreme programming practices - Wikipedia, 1 월 1, 2026 에 액세스, https://en.wikipedia.org/wiki/Extreme_programming_practices
32. The original 12 practices and their dependencies. Rather than add... - ResearchGate, 1 월 1, 2026 에 액세스, https://www.researchgate.net/figure/The-original-12-practices-and-their-dependencies-Rather-than-add-additional_fig1_221320577
33. ISO 19166 BIM-GIS conceptual mapping | PDF - Slideshare, 1 월 1, 2026 에 액세스, https://www.slideshare.net/slideshow/iso-19166-bimgis-conceptual-mapping/238678769
34. PD ISO/TS 19166:2021 - BSI Knowledge, 1 월 1, 2026 에 액세스, https://knowledge.bsigroup.com/products/geographic-information-bim-to-gis-conceptual-mapping-b2gm
35. CityGML in the Integration of BIM and the GIS: Challenges and Opportunities - Semantic Scholar, 1 월 1, 2026 에 액세스, https://pdfs.semanticscholar.org/d32f/d6864eb7e9255780ef2af0d7d74bf2128696.pdf
36. BIM to GIS (Intermediate) | IFC LOD 300 to LOD 4 CityGML - FME Support Center, 1 월 1, 2026 에 액세스, https://support.safe.com/hc/en-us/articles/25407718003341-BIM-to-GIS-Intermediate-IFC-LOD-300-to-LOD-4-CityGML
37. A Digital Twin-based paradigm for programming and control of cooperating robots in reconfigurable production systems - Taylor & Francis, 1 월 1, 2026 에 액세스, https://www.tandfonline.com/doi/full/10.1080/0951192X.2024.2428683