**План**

1. Вступ

2. Передумови й історія виникнення UML

3. Короткий опис UML

4. Керована моделями інженерія. Огляд

5. Огляд англомовної літератури UML

6. Критика UML

7. Висновок

Література

**1. Вступ**

При створенні складних інженерних систем прийнято використовувати прийоми моделювання. Складність більшості створюваних сьогодні програмних систем не поступається складності багатьом інженерним спорудженням, тому моделювання програмних систем є досить актуальним завданням. Більш того, у таких концепціях, як MDA (Model Driven Architecture - архітектура на основі моделей) і MDD (Model Driven Development - розробка на базі моделей), моделям приділяється центральна роль у процесі створення програмного продукту. Основною ідеєю цих концепцій є представлення процесу створення програмного продукту у вигляді ланцюжка трансформацій його вихідної моделі в готову програмну систему.

Майже у всіх інструментальних засобах, що втілює ідеї MDD, як мова моделювання використовується мова UML (Unified Modeling Language - уніфікована мова моделювання), цілком або які-небудь його частини.

UML - це мова, призначена для візуалізації, спеціфікації, конструювання й документування програмних систем. Слово «уніфікований» у назві мови означає, що UML може використовуватися для моделювання широкого кола додатків від вбудованих систем і систем реального часу до розподілених web-додатків. Виразні засоби мови дозволяють описати систему зі всіх точок зору, що мають відношення до розробки й розгортання.

**2. Передумови й історія виникнення UML**

**Методики об’єктно-орієнтованого моделювання**

Розвиток об’єктно-орієнтованих мов моделювання в 1980-х 1990-х роках призвів до появи великого числа об’єктно-орієнтованих підходів до моделювання. Зокрема, у період з1989 по 1994 роки загальна кількість відомих мов моделювання зросло з 10 до більш ніж 50.[3]. Кожна з мов мала свої достоїнства й недоліки також як і свою нотацію. Той самий символ у різних нотаціях найчастіше міг мати абсолютно різне значення.

Це ситуація суворої конкуренції між методиками моделювання дістала назву «війни методів».

«Війна методів» обумовила необхідність створення єдиної мови, що поєднувала би сильні сторін відомих методів. І в жовтні 1994 року почалося створення мови UML, основу якої склали кілька об'єктно-орієнтованих методів, що зарекомендували себе щонайкраще на практиці, але кожний з яких був націлений не рішення окремого завдання аналізу й проектування.

- метод Граді Буча, умовна назва BOOCH ( Booch'91, BooCH Lite, Booch'93) вважався найбільш ефективним на етапах проектування й розробки програмних систем [ 1].

- метод Джеймса Рамбо, Object Modeling Technique (ОМТ, ОМТ-2) – оптимально походив для аналізу процесів обробки даних в інформаційних системах [5]; (Rumbaugh, et al., 1991);

- метод Айвара Джекобсона (Ivar Jacobson), Object-Oriented Software Engineering (OOSE) – містив засоби представлення варіантів використання, що мають істотне значення на етапі аналізу вимог у процесі проектування бізнес-додатків [6]. [Jacobson, et al., 1993].

Спочатку Г. Буч і Д. Рамбо з компанії Rational Software Corporation почали роботу з уніфікації своїх методів. Незважаючи на те, що самі по собі ці методи були досить популярні, спільна робота їх була спрямована на вивчення всіх відомих об’єктно-орієнтованих методів з метою виявлення їхніх достоїнств. Восени того ж року до них приєднався Айвар Якобсон, головний технолог компанії Objectory AB,, до того, щоб інтегрувати свій метод OOSE із двома вищезгаданими. Протягом усього року творці займалися збором відкликань від основних компаній, що працюють в області створення ПО. За цей час стало ясно, що більшість таких компаній, що працюють в області створення , порахувала UML мовою, що має стратегічне значення для їхнього бізнесу.

У результаті був створений консорціум UML, у який увійшли організації, що виявили бажання надати ресурси для роботи, спрямованої на створення повної специфікації UML. Версія 1.0 мови з'явилася в результаті спільних зусиль компаній Digital Equipment Corporation, Hewlett Packard, I-Logix, Intellicprp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational, Texas Instruments \_ Unisys. UML 1.0 виявився добре визначеною, виразною, потужною мовою, що може бути застосованою для рішення великої кількості різноманітних завдань.

На протязі декількох років спеціальна робоча група OMG (OMG Revision Task Force) підтримувала просування проекту UML. Були створені версії 1.3, 1.4, і 1.5. За 2000-2003 була розроблена версія UML 2.0.у листопаді 2007 випущена поточна версія UML2.1.2.

**Завдання UML.**

Мова UML призначена для рішення наступних завдань:

1.надати в розпорядження користувачів готову до використання виразну потужну мову візуального моделювання, що дозволяє розробляти осмислені моделі й обмінюватися ними;

2. передбачити внутрішні механізми розширюваності й спеціалізації базових концепцій мови;

3. забезпечити максимальну незалежність проекту створення програмного забезпечення від конкретних мов програмування й процесів розробки;

4. забезпечити формальну основу для однозначної інтерпретації мови;

5.стимулювати розширення ринку об’єктно-орієнтованих інструментальних засобів створення програмного забезпечення;

6. інтегрувати кращий практичний досвід використання мови й реалізації програмних засобів його підтримки.

У значній мірі мова UML не залежить від процесу розробки програмного забезпечення. Уніфікований процес розробки ПЗ (Rational Unified Process, RUP) [Kruchten, 2004] - це один з підходів до організації життєвого циклу ПЗ, який особливо добре сполучається з UML. Цей комерційний продукт задає строгий регламент розподілу завдань і відповідальності між виконавцями в процесі розробки ПЗ.

**3. Короткий опис UML**

Підсумую, **UML** ( Unified Modeling Language — уніфікована мова моделювання) — мова графічного опису для об'єктного моделювання в області розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаною UML моделлю. UML був створений для визначення, візуалізації, проектування й документування здебільшого програмних систем.

Використання UML не обмежується моделюванням програмного забезпечення. Його також використовують для моделювання бізнес-процесів, системного проектування й відображення організаційних структур.

UML дозволяє розроблювачам ПЗ досягти угоди в графічних позначеннях для представлення загальних понять (таких як клас, компонент, узагальнення (generalization), об'єднання (aggregation) і поведінка) і більше сконцентруватися на проектуванні й архітектурі.

**Загальна структура мови**

Семантика мови UML визначається для двох видів об'єктних моделей: структурних і поведінкових. Структурні (статичні) моделі описують структуру сутностей або компонентів системи, включаючи їхні класи, інтерфейси, атрибути й зв'язки. Моделі поводження (динамічні) описують поведінку або функціонування об'єктів системи, включаючи їхні методи, взаємодію (співробітництво) між ними, а також процес зміни станів окремих компонентів і системи в цілому [4]. (Буч,1994)

Формальний опис мови UML ґрунтується на наступній загальній ієрархічній структурі модельних подань, що складається із чотирьох рівнів абстракції:

- позначка-метамодель,

- метамодель,

- модель,

- об'єкти користувача [16].

Рівень метаметамоделі утворить базову основу для всіх метамодельних представлень і визначає мову для специфікації метамоделі. Позначка-Метамодель визначає модель мови UML на найвищому рівні абстракції (відповідно на найнижчому рівні конкретизації) і є найбільш компактним його описом. Метамодель - екземпляр або конкретизація позначка-метамоделі - визначає мову для специфікації моделей. Всі основні поняття мови UML - це поняття рівня метамоделі. Модель у контексті мови UML є екземпляром (конкретизацією) метамоделі в тім розумінні, що кожна (конкретна) модель системи повинна використовувати тільки поняття метамоделі, конкретизувавши їх стосовно відповідної ситуації. Змістовно говорячи, рівень моделі призначений для опису конкретної предметної області. Конкретизація понять моделі відбувається на рівні об'єктів, які є екземплярами моделі й містять конкретну інформацію про предметну область відповідно до поняттями моделі.

Основою представлення UML на метамодельнім рівні є опис трьох його логічних блоків (пакетів):

- основні елементи,

- елементи поводження

- загальні механізми [16].

**Концептуальна модель мови**

Концептуальна модель мови включає основні будівельні блоки, правила їхні сполучення й загальні механізми [13, 17, 18].

Словник мови UML містить сутності (абстракції, що є основними елементами моделі) і відносини (основні сполучні будівельні блоки), Сутності й відносини за певними правилами з'єднуються в конструкції - діаграми.

В UML визначено чотири типи сутностей [13]:

− **структурні сутності**,

що поділяються на основні

клас (Class), інтерфейс (Interface)

кооперація (Collaboration),

прецедент (Use case),

активний клас (Active class),

компонент (Component),

вузол (Node)

різновиди основних

актор (Actor),

сигнал (Signal),

утиліта (Utility, вид класів),

процес (Process),

нитка (Thread, вид активних класів))

інші

додатка (Application),

документ (Document),

файл (File), бібліотека (Library),

сторінка (Page),

таблиця (Table, вид компонентів));

**− сутності поводження** (Behavioral things)

взаємодія (Interaction)

автомат (State machine);

− **сутності, що** групують, - пакет (Packages);

− **анотаційні сутності** – примітка (Note).

Основними типами відносин в UML є відносини:

залежності (Dependency),

асоціації(Association) (різновидом асоціації є

відношення агрегації (Aggregation)),

узагальнення(Generalization)

реалізації (Realization).

Існують також їхньої варіації, наприклад, уточнення, трасування, включення й розширення (для відносин залежності).

Для побудови коректно оформленої моделі в UML визначені правила, що дозволяють коректно й однозначно визначати:

(1) імена сутностей, відносин і діаграм,

(2) область дії імен (контекст, у якому ім'я має деяке значення),

(3) видимість імен (для використання іншими елементами),

(4) цілісність (правильність і погодженість співвідношення елементів),

(5) виконання моделі [13].

Ефективність і спрощення застосування мови забезпечується використанням певних угод, так званих, загальних механізмів:

- специфікацій (Specifications),

- доповнень (Adornments),

- прийнятих розподілів (Common divisions

- механізмів розширення (Extensibility mechanisms) [7, 17, 18].

Кожний елемент нотації UML має унікальне графічне позначення й специфікацію - текстове подання синтаксису й змістовної семантики відповідного будівельного блоку.

Практично всі будівельні блоки характеризуються дихотомією “клас/ об'єкт” і“інтерфейс / реалізація”. Це основні підходи розподілу реальності при об’єктно-орієнтованом моделюванні систем.

UML допускає контрольовані розширення для адаптації мови до конкретних потреб. Наявність внутрішніх механізмів розширення принципово відрізняє UML від таких засобів моделювання як IDEF0, IDEF1X, IDEF3, DFD і ERM [37], що є замкнутими й не допускають расширення засобами самої мови.

До механізмів розширення UML ставляться:

- **стереотип** (Stereotype), що розширює словник мови (дозволяє створювати з існуючих блоків нові, специфічні для конкретного розв'язуваного завдання);

- тег-**значення** (Tagged value), що розширює властивості будівельного блоку (дає можливість включати нову інформацію в специфікацію елемента);

## - обмеження (Constraint), що розширює семантику будівельного блоку (дозволяє додавати нові або модифікувати існуючі правила за допомогою семантичних обмежень, заданих природною мовою або формальною мовою OCL). Деякі розширення придбали таку популярність, що були внесені в стандарт поточної версії UML [7, 21, 18].

## Діаграми

У нотації UML всі представлення про моделі складної системи фіксуються у вигляді спеціальних графічних конструкцій, що одержали назву діаграм [16]. Діаграма в UML - це графічне подання набору елементів, зображуване, як правило, у вигляді зв'язного графа з вершинами (сутностями) і ребрами (відносинами). Теоретично діаграми можуть містити будь-які комбінації сутностей і відносин. Однак на практиці застосовується невелика кількість типових комбінацій.

В UML використовуються наступні види діаграм (для виключення неоднозначності приведу також позначення англійською мовою):

|  |  |
| --- | --- |
| **Structure Diagrams:**   * Class diagram * Component diagram * Composite structure diagram   + Collaboration (UML2.0) * Deployment diagram * Object diagram * Package diagram   **Behavior Diagrams:**   * Activity diagram * State Machine diagram * Use case diagram * **Interaction Diagrams:**   + Communication diagram (UML2.0) / Collaboration (UML1.x)   + Interaction overview diagram (UML2.0)   + Sequence diagram   + Timing diagram (UML2.0) | **Структурні діаграми:**   * Класів * Компонентів * Композитної/складеної структури   + Кооперації (UML2.0) * Розгортання * Об'єктів * Пакетів   **Діаграми поводження:**   * Діяльності * Станів * Варіантів використання * **Діаграми взаємодії:**   + Комунікації (UML2.0) / Кооперації (UML1.x)   + Огляду взаємодії (UML2.0)   + Послідовності   + Синхронізації (UML2.0) |

### Діаграма класів (Class diagram) — статична структурна діаграма, що описує структуру системи, вона демонструє класи системи, їхні атрибути, методи й залежності між класами.

### Діаграма компонентів (Component diagram) — статична структурна діаграма, показує розбивку програмної системи на структурні компоненти й зв'язки (залежності) між компонентами. Як фізичні компоненти можуть виступати файли, бібліотеки, модулі, що виконуються файли, пакети й т.п.

### Діаграма композитної/складеної структури (Composite structure diagram) — статична структурна діаграма, демонструє внутрішню структуру класів і, по можливості, взаємодію елементів (частин) внутрішньої структури класу.

Підвидом діаграм композитної структури є **діаграми кооперації** (Collaboration diagram, уведені в UML 2.0), які показують ролі й взаємодія класів у рамках кооперації. Кооперації зручні при моделюванні шаблонов проектування.

Діаграми композитної структури можуть використовуватися разом з діаграмами класів.

### Діаграма розгортання (Deployment diagram) — служить для моделювання працюючих вузлів (апаратних засобів, node) і артефактів, розгорнутих на них. В UML 2.0 на вузлах розвертаються артефакти (англ. artifact), у той час як в UML 1.0 на вузлах розверталися компоненти. Між артефактом і логічним елементом (компонентом), що він реалізує, установлюється залежність маніфестації.

### Діаграма об'єктів (Object diagram) — демонструє повний або частковий знімок моделируємої системи в заданий момент часу. На діаграмі об'єктів відображаються екземпляри класів (об'єкти) системи із вказівкою поточних значень їхніх атрибутів і зв'язків між об'єктами.

### Діаграма пакетів (Package diagram) — структурна діаграма, основним змістом якої є пакети й відносини між ними. Твердого поділу між різними структурними діаграмами не проводиться, тому дана назва пропонується винятково для зручності й не має семантичного значення (пакети й діаграми пакетів можуть бути присутнім на інших структурних діаграмах). Діаграми пакетів служать, у першу чергу, для організації елементів у групи по якій-небудь ознаці з метою спрощення структури й організації роботи з моделлю системи.

### Діаграма діяльності (Activity diagram) — діаграма, на якій показане розкладання деякої діяльності на її складові частини. Під діяльністю (activity) розуміється специфікація поводження, що виконується, у вигляді координованого послідовного й паралельного виконання підлеглих елементів - вкладених видів діяльності й окремих дійhttp://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9\_%D1%8F%D0%B7%D1%8B%D0%BA ( action), з'єднаних між собою потоками, які йдуть від виходів одного вузла до входів іншого.

Діаграми діяльності використовуються при моделюванні бізнес-процесів, технологічних процесів, послідовних і паралельних обчислень.

Аналогом діаграм діяльності є схеми алгоритмів.

### Діаграма автомата (State Machine diagram) (діаграма кінцевого автомата, діаграма станів) — діаграма, на якій представлений кінцевий автомат із простими станами, переходами й композитними станами.

**Кінцевий автомат** (State machine) — специфікація послідовності станів, через які проходить об'єкт або взаємодія у відповідь на події свого життя, а також відповідні дії об'єкта на ці події. Кінцевий автомат прикріплений до вихідного елемента (класу, кооперації або методу) і служить для визначення поведінки його екземплярів.[40]

### Діаграма прецедентів (Use case diagram) (діаграма варіантів використання) — діаграма, на якій відбиті відносини, що існують між акторами і прецедентами.

### Основне завдання - являти собою єдиний засіб, що дає можливість замовникові, кінцевому користувачеві й розроблювачеві спільно обговорювати функціональність і поводження системи.

### Діаграми комунікації й послідовності транзитивни, виражають взаємодію, але показують його різними способами й з достатнім ступенем точності можуть бути перетворені одна в іншу.

**Діаграма комунікації** (Communication diagram) (в UML 1.x — діаграма кооперації, collaboration diagram) — діаграма, на якій зображуються взаємодії між частинами композитної структури або ролями кооперації. На відміну від діаграми послідовності, на діаграмі комунікації явно вказуються відносини між елементами (об'єктами), а час як окремий вимір не використовується (застосовуються порядкові номери викликів).

**Діаграма послідовності** (Sequence diagram) — діаграма, на якій зображене впорядковане в часі взаємодія об'єктів. Зокрема, на ній зображуються об'єкти, що беруть участь у взаємодії, і послідовність повідомлень, якими вони обмінюються.

### Діаграма огляду взаємодії (Interaction overview diagram) — різновид діаграми діяльності, що включає фрагменти діаграми послідовності й конструкції потоку керування.

### Діаграма синхронізації (Timing diagram) — альтернативне подання діаграми послідовності, що явно показує зміни стану на лінії життя із заданою шкалою часу. Може бути корисна в додатках реального часу.[15,40]

**4. Керована моделями інженерія. Огляд**

Останні п'ятдесят років дослідники й розроблювачі програмного забезпечення створюють абстракції, що допомагають їм програмувати в термінах цілей свого проекту, а не використовуваного комп'ютерного середовища, і захищаючі їх від складностей цього середовища. Із самого початку ці абстракції включали технології мов програмування й операційних систем. Наприклад, ранні мови програмування (мови асемблера й Fortran) захищали розроблювачів від складностей програмування в машинних кодах. [10.1]

Аналогічно, ранні операційні системи захищали їх від складностей програмування прямо на рівні апаратури. Хоча ці ранні мови й платформи підвищували рівень абстракції, вони явно були "орієнтованими на обчислення". Зокрема, вони забезпечували абстракції простору рішень (тобто області самих комп'ютерних технологій), а не абстракції, що дозволяють звістки розробку в термінах прикладної області. Згодом уживали численні спроби подальшого підвищення рівня абстракції. [14]

Одним з найбільш відомих напрямків, що утворилися в 1980-е рр., є інженерія програмного забезпечення з комп'ютерною підтримкою (Computer-Aided Software Engineering, CASE), що забезпечує методи й засоби розробки програмного забезпечення, що дозволяють розроблювачам виражати свої конструкції з використання графічних програмних засобів загального призначення, різного виду діаграм. Однієї із цілей CASE-Засобів було забезпечення більше ретельного аналізу графічних програм за рахунок їхньої меншої складності, чим у програм, представлених на традиційних мовах програмування (наприклад, у графічних програмах неможливі помилки, що приводять до ушкодження пам'яті).

Іншою проблемою підходу CASE була його нездатність масштабуватися до складних, виробничих систем у широкому діапазоні прикладних областей. Загалом кажучи, CASE-Засобу не підтримували паралельну розробку; на їхній основі можна було розробляти програми поодинці або групою, члени якої по черзі зверталися до файлів, використовуваним цими засобами. [10/1]

У результаті в 1990-е рр. підхід CASE робив порівняно невеликий вплив на розробку комерційного програмного забезпечення, фокусуючись на декількох прикладних областях, наприклад, на обробці викликів у телекомунікаційних системах, де добре підходили подання у вигляді кінцевих автоматів. Навіть там, де CASE-Засоби застосовувалися на практиці, звичайно використовувалася тільки їхня частина, що дозволяє розроблювачам малювати діаграми програмних архитектур і документувати свої рішення, на основі чого програмісти потім вручну робили й розвивали реалізації. Однак, оскільки був відсутній прямий зв'язок між діаграмами й реалізаціями, розроблювачі не прагнули до великої точності діаграм, оскільки вони рідко синхронізувалися із кодом на подальших стадіях проектів.

В останні двадцять років досягнення в області мов програмування й платформ привели до підвищення рівня абстракцій, доступних для розроблювачів, зм'якшивши один з недоліків підходу CASE. Наприклад, сьогодні розроблювачі звичайно використовують більше виразні об’єктно-орієнтовані мови (зокрема, C++, Java і C#), а не Fortran або C. Повторно використовувані бібліотеки класів і платформи підтримки додатків мінімізують потребу у винаході загальних і прикладних сервісів - транзакцій, відмовостійкості, оповіщення про події, безпеку, розподіленого керування ресурсами й т.д. Все це дозволяє розроблювачам краще захиститися від складності, пов'язаної зі створенням додатків на основі традиційних технологій.[14]

Незважаючи на ці досягнення, залишається кілька неприємних проблем. У центрі цих проблем лежить складність платформ, що росте швидше здатності мов загального призначення маскувати її. Наприклад, популярні платформи проміжного програмного забезпечення J2EE, .NET і CORBA містять тисячі класів і методів з багатьма складними залежностями й тонкими побічними ефектами, що вимагає значних зусиль при програмуванні й ретельному настроюванні.

Родинна проблема полягає в тому, що код більшості додатків і платформ як і раніше пишеться на мовах третього покоління, для чого потрібно багато часу й зусиль, особливо для виконання інтеграційних дій - розгортання, конфігурування й підтримка якості системи. Наприклад, на Java або C# важко написати код, коректно й оптимально розгортає великомасштабної розподіленої системи із сотнями тисяч взаємозалежних компонентів.

Ситуацію не рятує навіть використання описів розгортання мовою XML, використовуваних у компонентних і сервіс-орієнтованих платформах проміжного програмного забезпечення, оскільки в цьому випадку є семантичний розрив між метою розробки й вираженням цієї мети в тисячах рядків вручну зробленого XML, синтаксис якого не має відносини ні до семантики прикладної області, ні до мети розробки. Через наявність цих проблем індустрія програмного забезпечення наближається до гранично припустимої складності.

У той же час платформні технології нового покоління, наприклад, Web-Сервіси й архітектури продуктових ліній (product-line architecture) стають настільки складними, що роками опановують платформними API і паттернами використання й при цьому часто виявляються знайомими тільки із частиною можливостей регулярно використовуваної ними платформи. Більше того, при використанні мов третього покоління розроблювачі змушені приділяти настільки велику увагу тактичним деталям імперативного програмування, що вони часто не можуть концентруватися на стратегічних архітектурних проблемах, таких як коректність системи в цілому і її продуктивність. Такі фрагментовані представлення проекту в цілому затрудняють розроблювачам розуміння того, які частини їхніх додатків є чутливими до побічних ефектів, що виникають при зміні вимог замовників і/або середовища розробки. Часто це змушує розроблювачів робити неоптимальні рішення, у яких дублюються частини коду, порушуються ключові архітектурні принципи, ускладнюється розвиток системи й забезпечення необхідної якості.

Багатообіцяючим підходом, спрямованим на рішення цих проблем, є розробка технологій інженерії, керованої моделями, (Model-Driven Engineering, MDE). При використанні MDE розробка ведеться на предметно-предметно-орієнтованих мовах моделювання (Domain-Specific Modeling Language, DSML), у системах типів яких формалізується структура, поводження й вимоги додатка усередині відповідної предметної області. DSML описуються з використанням метамоделей, у яких визначаються зв'язки між поняттями предметної області й точно специфується основна семантика й обмеження, асоційовані із цими поняттями. [10.1]

**5. Огляд англомовної літератури UML**

Для того щоб добре орієнтуватися в UML, тобто успішно використовувати його на практиці, треба мати досить глибокі представлення про методи об’єктно-орієнтованого аналізу, проектування й програмування. Приведу список англомовної літератури по UML з коротким описом змісту кожної роботи. Цей список не претендує на повноту, але, по -моєму, дає деяке подання про літературу, що описує UML.

Перша група робіт - [4], [5], [6]

[4], Booch G. Object-oriented analysis and design with applications. Second edition. The Benjamin/Cummings Publishing Company, Inc. 1994. 589 p.//

[5] Rumbaugh J., Blacha M. Premerlani W., Eddy F. Lorensen W. Object-Oriented Modeling and Design. Prentice-Hall, Inc., 1991

[6] Jacobson I. Object-Oriented Software Engineering. A Use Case Driven Approach. Addison-Wesley Publishing Company, 1993.

містить опис об’єктно-орієнтованих методологій, які були покладені в основу UML.

[5] є описом методології OMT. Вона вийшла в 1991 році й на справжній момент існує багато CASE-Засобів, у тім або іншому ступені підтримуючих її. До UML ця методологія була однієї з найпоширеніших.

[4] є, очевидно, кращою книгою в цій області. Вийшло два її видання - перше в 1991 році, друге - в 1994. Обоє видання перекладені на російську мову.

[6] містить опис OOSE - об’єктно-орієнтованого підходу до створення програмних систем, заснованого на моделі випадків використання (use case driven approach) і є систематизацією більш ніж 20-літнього досвіду її автора, Айвара Джекобсона, в області створення більших систем. Ця книга вийшла у світло в 1992 році. Модель випадків використання й багато чого, з нею зв'язане, увійшли в UML.

Друга група літератури є канонічним описом стандарту UML

[2] Booch G.,Rumbaugh J. UML 1.1. Semantics. 1997.

і [7] Booch G., Rumbaugh J. UML 2.0. Notation Guide є основними документами по UML. Там описується метамодель UML і дуже мало уваги приділяється семантиці конструкцій. В [8] (A Rational Approach to Software Development Using Rational Rose 4.0 )описується приклад розробки програмної системи (реєстрація студентів на відвідування навчальних курсів) з використанням CASE-Засобу Rational Rose, що реалізує підмножину UML (аналіз і проектування). Всі ці документи вільно доступні і їх можна взяти на web-вузлі OMG.

Оскільки офіційна документація по UML скрутна для розуміння, виходить багато книг, що описують його з різними акцентами. Я відзначу книги, написані головними авторами UML - Г. Бучем, И Джекобсоном, Д. Рэмбо - [1], [11], [12]:

* [1] - G. Booch, Jim Rumbaugh, Ivar Jacobson The Unified Modeling Language User Guide: детальна інформація про використання UML. Покриває близько 80% мови. Застосування UML описується на великій кількості прикладів;
* [11] Ivar Jacobson, G. Booch, Jim Rumbaugh The Unified Software Development Process - опис процесу об’єктно-орієнтованої розробки ПО;
* [12] - J.Rumbaugh, I.Jacobson, G. Booch Unified Modeling Language Reference Manual: - довідник по UML, що охоплює вся мову.

Крім того, відзначу ще книгу [13] (B.P. Douglass Real-Time UML. Developing Efficient Objects for Embedded Systems), написану співробітником фірми i-Logix Брюсом Дугласом, у якій утримується гарний опис UML у контексті розробки систем реального часу.

Ще одним джерелом інформації з UML є матеріали, що випускаються компанією Rational Software Corp. Це насамперед величезна база даних RUP, що містить більше 1000 статей навколо UML. Крім того, із грудня 1998 року виходить у світло журнал "Rose Architect", що містить багато цікавих статей фахівців фірми Rational Software Corp. про UML, RUP, Rational Rose, про застосування Rational Rose у різних областях розробки ПО.

Нагадаю, початковою метою розвитку UML було забезпечення більш точного опису мови моделювання - підтримка формальної основи для розуміння мови моделювання. Однак дотепер формальна семантика не є частиною стандарту. Огляд декількох підходів, що стосується визначення такої семантики, наведений в [26] (Husman H. Loose Semantics for UML), де розглянуті теоретико-множинний, трансляційний, метамодельний підходи й запропонований так звана “вільна семантика”.

Слід зазначити, що в самій специфікації UML існує багато огріхів і протиріч: так, у роботі [27] (Genova G., Llorens J., Quintana V. Digging into Use Case) розглядаються відносини включення й розширення для варіантів використання, в [29] (Gogolla M., Henderson-Sellera B. Analysis of UML Stereotypes within the UML Metamodel) аналізуються стандартні стереотипи метамоделі UML. У роботі [30] (Naumenko A., Wegmann A. A Metamodel for the Unified Modeling) пропонується використовувати RMODP ((Reference Model Open Distributed Processing) [31] (RM-ODP Open Distributed Processing - Reference) для рішення проблем мови. Відзначу, що відповідно до специфікації UML модель RM-ODP виступає структурою, що безпосередньо впливає на архітектуру метамодели мови (розділ “Preface: Relationships to Other Models”). Крім того RM-ODP використовується в MOF (Meta-Object Facility) для керування типами. В [30] (Naumenko A., Wegmann A. A Metamodel for the Unified Modeling) ідентифікуються три проблеми метамоделі UML і пропонується їхнє рішення на базі RM-ODP:

- структурний хаос семантики мови - “висока техніка, лаконічність і складність для розуміння новачками”; рішення: використання структури RM-ODP і теорії типів Б. Расела;

- відсутність декларативної семантики, суперечливість семантики мови при описі відносин між моделями, побудованими з використанням мови, і безпосередньо суб'єктів моделювання; рішення: реалізація базової концепції моделювання (Basic Modeling Concept);

- недостатнє теоретичне обґрунтування використовуваної метамоделі мови UML; рішення: пропонована в статті концепція моделювання на основі RM-ODP, теорії типів Б. Расела до інтерпретації формальних вирахувань уважається повністю обґрунтованою.

У багатьох роботах, присвячених формалізації моделі й метамодели мови UML, розглядається не сама мова, а деяка його підмова, формальна й строго структурований. Так, в [32] / (Paige R., Ostroff J. Metamodelling and Conformance Checking with PVS) розглядаються BON (Business Object Notation, об’єктно-орієнтована мова моделювання, по суті співпадаюча з підмовою діаграм класів UML [33] (Walden K., Nerson J.-M. Seamless Object-Oriented Software Development) і PVS (Prototype Verification System, мова специфікацій, розроблена для автоматичного аналізу метамоделей мов моделювання [34]( Owre S., Shankar N., Rushby J., Stringer-Calvert D. The PVS Language Reverence Version 2).

Результатом цієї роботи є повна формальна специфікація метамодели об’єктно-орієнтованого мови моделювання у формі, придатної для автоматичного аналізу. Однак BON у порівнянні з UML більше формалізований і “підігнаний” під умови розв'язуваного завдання.

Аналогічний підхід використаний в [35] (Overgaard G. Formal Specification of OO Modeling), де як платформа для формалізації обраний формалізм Boom, що складається з метамоделі й мови формальних специфікацій Odal - простого строго типізованої мови, семантика якого задана в термінах так званого π -вирахування. В [36]( Clark T., Evans A., Kent S. The Metamodelling Language Calculus: Foundation Semantics for UML) на основі [37] (Cardeli L, Abadi M. A theory of Objects) розглядається формалізація мови MML (Metamodelling Language), що є підмножиною UML; ця формалізація запропонована авторами як база для всього UML 2.0.

Нарешті, у роботі [38] (Lellahi K. Conceptual Data Modeling: An Algebraic Viewpoint) демонструється застосовність алгебраїчного підходу для формального опису ER-Діаграм (Entity-Relationship diagrams), що є аналогами діаграм класів UML.

**Статті в комп'ютерних журналах.**

Зупинюся докладніше на огляді лютневого, 2005 р. номера журналу **Computer (IEEE Computer Society, V. 38, No 2, February 2005).**

Темою лютневого номера журналу є **"Керована моделями розробка програмного забезпечення"** ("Model-Driven Software Development"). Представлено повноцінну тематичну добірку статей із запрошеним редактором, Дугласом Шмідтом (Douglas C. Schmidt, Vanderbilt University). [10]

Об'ємна вступна замітка редактора називається **"Керована моделями інженерія"** ("Model-Driven Engineering"). У статті дається докладний аналіз залежності ручної праці розроблювача програмного забезпечення від рівня абстракції мови програмирвания. Обговорюються фактори виникнення програмного забезпечення з комп'ютерною підтримкою (Computer-Aided Software Engineering, CASE),його переваги й недоліки. Піднімається питання складності популярних платформ проміжного програмного забезпечення J2EE, .NET і CORBA. Ситуацію не рятує навіть використання описів розгортання мовою XML. Багатообіцяючим підходом, спрямованим на рішення цих проблем, є розробка технологій інженерії, керованої моделями, (Model-Driven Engineering, MDE).[10.1]

Перша основна стаття тематичної добірки називається **"Розробка додатків з використанням керованих моделями середовищ розробки"** ("Developing Applications Using Model-Driven Design Environments"). Автори статті - Krishnakumar Balasubramanian, Aniruddha Gokhale, Gabor Karsai, Janos Sztipanovits, Sandeep Neema, Vanderbilt University. [10.2]

Історично методології розробки програмного забезпечення фокусуються більшою мірою на вдосконалюванні засобів розробки систем, а не на створенні інструментів, що допомагають конструювати й інтегрувати системи. Компонентне програмне забезпечення проміжного шару (Enterprise Java-Beans (EJB), Microsoft .NET і CORBA Component Model (CCM)) сприяють підвищенню рівня повторного використання програмного забезпечення на основі абстракції компонента.

Однак при прийнятті розроблювачами на озброєння цих комерційних, готових до використання технологій виникає розривши між такими доступними й зробленими засобами розробки, як компілятори й отладчиками, і засобами, використовуваними розроблювачами для компонування, аналізу й тестування закінченої системи або системи систем. У результаті розроблювачі продовжують виконувати системну інтеграцію з використанням підручних методів без підтримки засобів автоматизації.

Розробка, керована моделями (Model-Driven Development, MDD) - це парадигма, що розвивається, вирішальні численні проблеми композиції й інтеграції великомасштабних систем і опирається при цьому на наявні досягнення в області технологій розробки програмного забезпечення (зокрема, на компонентне проміжне програмне забезпечення). MDD дозволяє перевести розробку програмного забезпечення на більше високий рівень абстракції в порівнянні з тим, що можливий при використанні мов третього покоління. Для подання елементів системи і їхніх зв'язків у підході MDD використовуються моделі. Моделі служать вхідними й вихідними даними на всіх стадіях розробки, впритул генерації закінченої системи.

Автори визнають, що популярним варіантом MDD є модельно-керована архітектура (Model-Driven Architecture, MDA), запропонована й розвиваєма консорціумом Object Management Group (OMG).У підході MDA системи представляються з використанням мови моделювання загального призначення Unified Modeling Language (UML)і її конкретних профілів. Ці моделі перетворяться в артефакти, виконувані на різноманітних платформах, зокрема, на EJB, .NET і CCM. [10.2]

Наступна стаття написана Adam Childs, Jesse Greenwald, Georg Jung, Matthew Hoosier, John Hatcliff, Kansas State University. Назва статті - **"CALM і Cadena: метамоделювання для заснованої на компонентах розробки продуктового ряду"** ("CALM and Cadena: Metamodeling for Component-Based Product-Line Development").[10.3]

Великомасштабні роботи зі створення програмного забезпечення все частіше ґрунтуються на **продуктових лініях.** У таких процесах розробки розроблювачі створюють програмне забезпечення для подібних сімейств продуктів на основі повторно використовуваної архітектури й загальних прикладних компонентів.

У підході продуктових ліній особливе значення надається систематичному повторному використанню, і проходження цьому підходу може скоротити час розробки й впровадження у виробництво, а також загальну вартість більш ніж в 10 разів. Підхід продуктових ліній підтримується використанням компонентного проміжного програмного забезпечення за рахунок забезпечення правильно певних інтерфейсів, які запобігають зайвій прив'язці клієнтського коду до низкорівневих реалізацій, і спрощення додавання й вилучення модулів, що сприяє повторному використанню й розвитку системи.

Розробка на основі підходу продуктових ліній з використанням компонентних каркасів успішно зарекомендувала себе в численних прикладних областях: від великомасштабних розподілених систем реального часу й убудованих систем, систем керування електромережами, систем керування виробничими процесами до операційних систем користувальницького рівня й систем інтеграції додатків персональних комп'ютерів.

Це явний поділ інфраструктури й модулів додатка, а також можливість простого компонування цих модулів, наводить на природну думку про наявність у розробці трьох ролей. **Архітектор продуктової лінії** формує архітектуру системи, вибирає інфраструктурні платформи й організує процес розробки, **розроблювач компонентів** створює модулі бізнес-логіки, і **інтегратор компонентів** збирає компоненти в систему.

Платформа підтримки розробки Cadena разом з її основним засобом моделювання CALM (Cadena Architecture Language with Metamodeling) дозволяє перебороти цей недолік на рахунок забезпечення адаптивного середовища моделювання з потужною, гнучкою й розширюваною інструментальною підтримкою. CALM - це мова опису архитектур, що підтримує строго типізоване моделювання платформ, компонентів цих платформ і складань компонентів конкретних сценаріїв. Мова також підтримує засновану на спадкуванні ієрархічну організацію платформ із використанням механізмів аспектів для включення в загальні архітектурні описи атрибутів конкретних платформ. Cadena забезпечує різноманітну підтримку створення, редагування, запиту, перегляду й перетворення CALM-Моделей. CALM-моделі зв'язуються з каркасами компонентного проміжного програмного забезпечення, а також із засобами генерації коду через підключаються модулі, ЩоМ, Cadena.

Стаття **"Автоматизація еволюції змін у модельно-модельно-керованій інженерії"** ("Automating Change Evolution in Model-Driven Engineering") написана Джефом Гріємо, Джейн Лін і Джинг Жанг. [10.4]

З розширенням областей застосування моделей програмного забезпечення й систем з'явилася термінова потреба в керуванні складною еволюцією змін усередині подання моделей. У розроблювачів повинна бути можливість швидкої й простої перевірки різних проектних альтернатив серед незліченних і різнотипних конфігураційних можливостей.

В ідеалі інструментальний засіб повинний був б робити імітаційне моделювання кожної нової проектної конфігурації для визначення того, яким образом деякий аспект конфігурації (наприклад, комунікаційний протокол) впливає на спостережувану властивість (наприклад, на пропускну здатність). Для забезпечення такого рівня підтримки еволюції моделей інструментальний засіб повинний забезпечувати дві категорії змін, які в цей час виконуються розроблювачами вручну й звичайно з поганими результатами.

Першу категорію становлять зміни, що перетинають ієрархію подання моделі. Прикладом є ефект зміни пропускної здатності на якість обслуговування компонентів авіаційного електронного встаткування, які повинні відображати в реальному часі відео-потік. Щоб оцінити наслідки такої зміни, розроблювач вручну обійти ієрархію моделі, рекурсивно клікая по кожної подмоделі. Цей процес стомлюючий і чреватий помилки, оскільки моделі систем часто містять ієрархії глибиною в кілька рівнів.

Друга категорія змін включає збільшення масштабу частин моделі, що доставляє особливі занепокоєння при розробці великомасштабних убудованих систем реального часу, що містять тисячі крупномодульних компонентів. Для цього типу зміни потрібне створення декількох модельних елементів і з'єднань між ними. При роботі з інструментом моделювання для масштабування базової моделі з декількох елементів до моделі з тисячами елементів потрібно разюче велика кількість дій з мишею й клавіатурою. При виконанні цього процесу легко робляться помилки, наприклад, можна забути встановити з'єднання між двома задубльованими елементами. Зрозуміло, що ручне масштабування впливає не тільки на ефективність моделювання, але й на коректність представлення.

Обидві ці категорії еволюції змін істотно виграли б від автоматизації. Із цією метою автори розробили узагальнений процесор трансформацій для маніпулювання моделями, названий ними C-Saw (Constraint-Specification Aspect Weaver). C-Saw – це - модуль, що підключається, для GME (див. вище огляд статті "Розробка додатків з використанням керованих моделями середовищ розробки").

Для роботи зі змінами, що перетинають ієрархію, в C-Saw використовується кілька принципів аспектно-орієнтованого підходу. Комбінація трансформації моделі з компонуванням аспектів забезпечує потужну технологію для швидкого перетворення успадкованих систем на основі високорівневих властивостей, описуваних моделлю. Далі, шляхом застосування аспектно-орієнтованих методів і перетворення програм невеликі зміни на модельному рівні можуть ініціювати дуже великі трансформації на рівні вихідного коду.

Останню статтю тематичної добірки - **"Модельно-модельно-орієнтована розробка з використанням UML 2.0: обіцянки й прорахунки"** ("Model-Driven Development Using UML 2.0: Promises and Pitfalls") - написали Роберт Франс, Судипто Гош, Трунг Динх-Тронг і Арнор Солберг (Robert B. France, Sudipto Ghosh, Trung Dinh-Trong, Colorado State University, Arnor Solberg, SINTEF). [10.5]

Досвід показує, що ефективні механізми керування складністю автоматизують звичайні завдання розробки й забезпечують надійну підтримку поділу відповідальностей. Наприклад, сучасні високорівневі мови програмування й інтегровані середовища розробки забезпечують абстракції, що захищають розроблювачів від складних низькорівневих деталей і підтримують автоматичне перетворення абстрактних представлень вихідного коду в дійсні форми, які виконує машина.

Досягнення в областях розробки програмного забезпечення й технологій обробки інформації привели до спроб створення більш складних програмних систем. Ці спроби демонструють неадекватність абстракцій, забезпечуваних сучасними мовами високого рівня. Виникає потреба в мовах, моделях і технологіях, що підвищують рівень абстракції, на якому замислюються, створюються й розвиваються.

OMG (Object Management Group) відповідає на цю вимогу підготовкою версії 2.01 мови UML і ініціативою MDA (Model Driven Architecture). Проблеми, на рішення яких на початку були націлені розроблювачі UML 2.0, включали очевидне розпухання ранніх версій UML і відсутність правильно певної семантики.

У ході розробки стандарту в число вимог була додана підтримка модельно-модельно-керованої розробки (MDD), а точніше, підходу MDA до MDD. Широка поінформованість про проблеми ранніх версій UML разом зі зростаючим інтересом до MDD породили надії на те, що розроблювачам UML 2.0 удасться зробити версію мови з істотно скороченим набором модельних понять для точного й зручного описів найрізноманітніших додатків.

Очікувалося також, що в цій версії буде бути точна семантика, що могла б полегшити автоматизацію, необхідну для просування MDD за межі традиційних можливостей існуючих CASE-Засобів. Деякі люди очікували, що розроблювачам UML 2.0 удасться зробити срібну кулю мов моделювання або, принаймні, тісно наблизитися до цього.

Ті, хто не знаком із внутрішніми роботами в області, проведеній співтовариством стандартизації мов, знаходять разючими розмір і складність стандарту UML 2.0. Дійсно, кінцеві результати здаються далекими від посилок, які мотивували початок цієї роботи з великого перегляду мови. З погляду очорнителя, численні модельні поняття, погано певна семантика й легковагі механізми розширень затрудняють вивчення мови і його застосування в середовищі MDD. Ці реальні проблеми вимагають рішення, але нам не слід дивуватися тому, що ця мова моделювання першого покоління далека від досконалості.

Як відзначають деякі розроблювачі UML, розробка мов моделювання усе ще переживає період становлення. Для прискорення процесу виявлення знань, пов'язаних з MDD, нам потрібна конструктивна критика UML. У цьому змісті UML 2.0 може зіграти важливу роль як явна форма експерименту з мовою моделювання, що може вивчатися, аналізуватися й критикуватися зацікавленими сторонами.

Захисники UML 2.0 відзначають, що в цьому стандарті відбитий кращий виробничий досвід застосування моделювання. Вони говорять про наступні основні вдосконалення:

* Поліпшено підтримку UML як сімейства мов за рахунок використання профілів і крапок семантичних варіацій (semantic variation point), які позначають частини UML, які навмисне залишені без семантики, щоб можна було навантажити їхньою семантикою, обумовленої користувачами.
* Поліпшено виразність моделювання, включаючи моделювання бізнес-процесів, підтримку класифікаторів повторного використання моделювання й підтримку архитектур розподілених неоднорідних систем.
* Зроблено інтеграцію семантики дій (Action Semantics), що може використовуватися розроблювачами для визначення семантики моделей під час виконання й забезпечує семантичну точність, необхідну для аналізу моделей і їхньої трансляції в реалізації.

На думку авторів, що занадто висока оцінка UML і відповідних технологій MDD може бути настільки ж пагубної, як і несправедлива критика. Це може привести до росту очікувань користувачів до недосяжного в цей час рівня. У своїй статті автори намагаються розвіяти хмари реклами, що оточують UML 2.0, і представити обґрунтовану оцінку забезпечуваної цієї версії мови підтримку MDD [10.4].

**6. Критика й переваги UML**

Незважаючи на те, що UML досить широко розповсюджений і використовуваний стандарт, його часто критикують через наступні недоліки [40, 41]:

* **Надмірність мови**. UML часто критикується, як невиправдано велика і складна. Вона включає багато надлишкових або практично невикористовуваних діаграм і конструкцій. Частіше це можна почути у відношенні UML 2.0, чим UML 1.0, тому що більше нові ревізії включають більше « розроблених-комітетом» компромісів.
* **Неточна семантика**. Тому що UML визначена комбінацією себе (абстрактний синтаксис), OCL (мовою опису обмежень — формальної перевірки правильності) і Англійської (докладна семантика), то вона позбавлена скутості властивим мовам, точно певним техніками формального опису. У деяких випадках абстрактний синтаксис UML, OCL і Англійської суперечать один одному, в інших випадках вони неповні. Неточність опису самої UML однаково відбивається на користувачах і постачальниках інструментів, приводячи до несумісності інструментів через унікальне трактування специфікацій.
* **Проблеми при вивченні й впровадженні**. Вищеописані проблеми роблять проблематичним вивчення й впровадження UML, особливо коли керівництво насильно змушує використовувати UML інженерів при відсутності в них попередніх навичок (стаття ACM "Death by UML Fever" на англ. містить цікаве оповідання про кількість таких випадків.) [41]
* **Тільки код відображає код**. Ще одна думка — що важливо робочі системи, а не гарні моделі. Як лаконічно виразився Джек Ривс, «The code is the design» (англ. «Код і є проект»). Відповідно до цієї думки, існує потреба в кращому способі написання ПЗ; UML цінується при підходах, які компілюють моделі для генерування вихідного або здійсненного коду. Однак цього все-таки може бути недостатньо, тому що UML не має властивостей повноти по Тьюрінгу і будь-який згенерований код буде обмежений тим, що може розглянути або припустити інтерпретуючий UML інструмент.
* **Кумулятивне навантаження/Неузгодженість навантаження** (Cumulative Impedance/Impedance mismatch). Неузгодженість навантаження — термін з теорії системного аналізу для позначення нездатності входу системи сприйняти вихід іншої. Як у будь-якій системі позначень UML може представити одні системи більш коротко й ефективно, чим інші. Таким чином, розроблювач відмінюється до рішень, які більш комфортно підходять до переплетенню сильних сторін UML і мов програмування. Проблема стає більш очевидної, якщо мова розробки не дотримується принципів ортодоксальної об’єктно-орієнтованої доктрини (не намагається відповідати традиційним принципам ВОП).
* **Намагається бути всім для всіх**. UML — це мова моделювання загального призначення, що намагається досягти сумісності з усіма можливими мовами розробки. У контексті конкретного проекту, для досягнення командою проектувальників певної мети, повинні бути обрані застосовні можливості UML. Крім того, шляхи обмеження області застосування UML у конкретній області проходять через формалізм, що не повністю сформульований, і який сам є об'єктом критики. [40]

## Переваги UML

## Підсумую:

* UML об’єктно-орієнтована, у результаті чого методи опису результатів аналізу й проектування семантично близькі до методів програмування на сучасних ВО-Мовах;
* UML дозволяє описати систему практично із всіх можливих точок зору й різні аспекти поведінки системи;
* Діаграми UML порівняно прості для читання після досить швидкого ознайомлення з його синтаксисом;
* UML розширює й дозволяє вводити власні текстові й графічні стереотипи, що сприяє його застосуванню не тільки в сфері програмної інженерії;
* UML одержала широке поширення й динамічно розвивається.

**7. Висновок**

UML є потужним, гнучким засобом моделювання, опис стандарту якого є відкритим для наступного вдосконалювання. Неоднозначність як деяких конструкцій самої мови, так і підходів до його формальної семантики, наявність у специфікації неформальних описів вимагає подальшого розвитку формальної основи для повної й несуперечливої інтерпретації мови.

**Перелік літератури**

[1] G. Booch, Jim Rumbaugh, Ivar Jacobson The Unified Modeling Language User Guide: Addison-Wesley Publishing Co., 1999, 512 p.

[2] Booch G., Rumbaugh J. UML 1.1 Semantics. (http://www.rational.com/uml/)1997.

[3]RogerR.Flynn,EditorinChief,Computersciences.Volume2.SoftwareandHardware.MacmillianreferenceUSA.ThomsonGaleCompany,Inc.2002572p.

[4]BoochG.Object-orientedanalysisanddesignwithapplications. Secondedition.TheBenjamin/CummingsPublishingCompany,Inc.1994.589p.//

[5]RumbaughJ.,BlachaM.PremerlaniW.,EddyF.LorensenW.Object-OrientedModelingandDesign.Prentice-Hall,Inc.,1991

[6]JacobsonI.Object-OrientedSoftwareEngineering. AUseCaseDrivenApproach.Addison-WesleyPublishingCompany,1993.

[7]BoochG.,RumbaughJ.UMLNotationGuide(www.rational.com/uml/)1997.

[8]ARationalApproach toSoftwareDevelopment Using Rational Rose4.0http://www.rational.com/support/techpapers/roseapproach/. 1997

[9] OMG Unified Modeling Language Specification (draft). Version 1.3R9. (http://www.rational.com/uml/)1999.

[10]Computer(IEEEComputerSociety,V.38,No2,February2005)"Model-DrivenSoftwareDevelopment"

[10.1]Model-DrivenEngineering("Керована моделями інженерія"), Douglas C. Schmidt

[10.2] Developing Applications Using Model-Driven Design Environments. (Розробка додатків з використанням керованих моделями середовищ розробки"). K.Balasubramanian, A.Gokhale, G.Karsai, J.Sztipanovits, S. Neema.

[10.3] CALM and Cadena: Metamodeling for Component-Based Product-Line Development. ("CALM і Cadena: метамоделювання для заснованої на компонентах розробки продуктового ряду")A.Childs, J.Greenwald, G.Jung, M.Hoosier, J.Hatcliff

[10.4]Automating Change Evolution in Model-Driven Engineering ("Автоматизація еволюції змін у модельно-модельно-керованій інженерії"") Джеф Греq, Джейн Лін і Джинг Жанг.

[10.5] Model-Driven Development Using UML 2.0: Promises and Pitfalls. S.Ghosh, Trung Dinh-Trong, A. Solberg. ("Модельно-модельно-орієнтована розробка з використанням UML 2.0: обіцянки й прорахунки"

[11] Ivar Jacobson, G. Booch, Jim Rumbaugh The Unified Software Development Process: Addison-Wesley Publishing Co., 1999, 512 p.

[12] Jim Rumbaugh, Ivar Jacobson, G. Booch Unified Modeling Language Reference Manual: Addison-Wesley Publishing Co., 1999, 576 p.

[13] B.P. Douglass Real-Time UML. Developing Efficient Objects for Embedded Systems: Addison-Wesley Publishing Co., 1998, 365 p.

[14] G. Booch The Visual Modeling of Software Architecture for the Enterprise. Rose Architect. October 1998, Vol. 1, No 1. p 18-25.

[15] Barker R. CASE Method. Entity-Relationship Modeling. Copyright Oracle Corporation UK Limited, Addison-Wesley Publishing Co., 1990

[16] Object Management Group, 2003. OMG Unified Modeling Language Specification / www. omg. org.

[17] .http://www.rational.com/uml.

[18] Chonoles M. J., Schardt J.A. UML 2 for Dummies. - Hungry Minds, 2003. - 412 р.

[19] Nock C. Data Access Patterns: Database Interactions in Object-Oriented Applications. - Addison Wesley, 2003. - 512 р.

[20] Fontoura M., Pree W., Rumpe B. UML Profile for Framework Architectures. First edition. Addison Wesley Longman, Inc.,, 2001, 240 pages

[21] Fowler M.,Scott K., UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language Second Edition, Addison Wesley Longman, Inc., 1999, 224 pages

[22] Gramma E.,Helm R.,Johnson R., Vlissides J. Design Patterns. Elements of reusable object-oriented software. , Addison Wesley Longman, Inc., 1999, 368 pages.

[23] Rebecca M. Riordan, Designing Effective Database Systems, Addison Wesley Professional., Inc., 2005, 384 pages

[24] Thomas A. Pender . UML Weekend Crash Course. Wiley Publising, Inc., 2002, 362 pages

[25] Rebecca M. Riordan, Seeing Data: Designing User Interfaces for Database Systems Using .NET, Addison Wesley Professional., Inc., 2004, 544 pages

[26] Husman H. Loose Semantics for UML/OCL // Society for Design and Process Science, 2002. - P. 32-39.

[27].Genova G., Llorens J., Quintana V. Digging into Use Case Relationships // Lect. Notes Comput. Sci. - 2002. - V. 2460. -P. 115-127.

[28] Kendal S. Fast Track UML 2.0. - Apress, 2004., 416 р.

[29] Gogolla M., Henderson-Sellera B. Analysis of UML Stereotypes within the UML Metamodel // Lect. Notes Comput. Sci. -2002. - V. 2460. - P. 84-99.

41./[30]Naumenko A., Wegmann A. A Metamodel for the Unified Modeling Language // Lect. Notes Comput. Sci. - 2002. - V. 2460.- P. 2-17.

42/[31].RM-ODP Open Distributed Processing - Reference Model - ISO, ITU.: ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation X.901, X.902, X.903, X.904, 1995-1998.

[32] Paige R., Ostroff J. Metamodelling and Conformance Checking with PVS // Lect. Notes Comput. Sci. - 2001. - V. 2029. -P. 2-16.

[33] Walden K., Nerson J.-M. Seamless Object-Oriented Software Development. - Prentice-Hall, 1995.

[34]. Owre S., Shankar N., Rushby J., Stringer-Calvert D. The PVS Language Reverence Version 2.3 - September, 1999. -(Tech. Rep. / SRI International Technical Report).

[35] Overgaard G. Formal Specification of OO Modeling // Lect. Notes Comput. Sci. - 2000. - V. 1783. - P. 193-207.

[36]Clark T., Evans A., Kent S. The Metamodelling Language Calculus: Foundation Semantics for UML // Lect. Notes Comput.Sci. - 2001. - V. 2029. - P. 17-31.

[37] Cardeli L, Abadi M. A theory of Objects. - Springer-Verlag, 1996.

[38] Lellahi K. Conceptual Data Modeling: An Algebraic Viewpoint // Lect. Notes Comput. Sci. - 2001. - V. 2244. - P. 336-348.

### [39] Wikipedia, the free encyclopedia,www. wikipedia.gb

[40] стаття ACM "Death by UML Fever"

[41] UML Forum. "UML FAQ". http://www.uml-forum.com/FAQ.htm.