МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

О.В. Ізмайлова

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Рекомендовано вченою радою Київського національного університету будівництва і архітектури як навчальний посібник для студентів галузі знань 12 «Інформаційні технології»

Київ 2022

УДК 004.4 (075.8)
І37

Рецензент *О.В. Сєлюков*, д-р техн. наук, професор
Київський національний університет будівництва і архітектури

Затверджено на засіданні вченої ради Київського національного університету будівництва і архітектури, протокол № 4 від 24 січня 2022 року.

Ізмайлова О.В.
І37 Проектування інформаційних систем: навч. посіб. / О.В. Ізмайлова,–Київ: КНУБА, 2022. – 88с.

ISBN 978-966-627-194-8

Викладаються основні положення об’єктно-орієнтованої методології аналізу та проектування інформаційних систем, особливості застосування уніфікованої мови моделювання UML, визначені процеси моделювання в UML, види статичних та динамічних діаграм. Надані загальні вказівки до лабораторних робіт циклу «Об’єктно-орієнтований аналіз та проектування систем. Візуальне моделювання систем в StarUML», де визначені мета робіт, тематика, порядок виконання та зміст звіту по кожній роботі.

Призначені для студентів спеціальностей 123 «Комп’ютерна інженерія» та 125 «Кібербезпека» галузі знань 12 «Інформаційні технології».

УДК 004.4 (075.8)

ISBN 978-966-627-194-8

ЗМІСТ

ПЕРЕДМОВА

Складність сучасних інформаційних систем, багатоетапність та ітераційний підхід до їх розробки, висока вартість створення і супроводження, участь в розробці цих систем десятків та сотень спеціалістів різного профілю (спеціалістів предметної області, аналітиків, програмістів, математиків, спеціалістів по захисту даних та комп'ютерної інженерії, менеджерів) визначають вирішальну залежність ефективності інформаційної системи від якості реалізації такої стадії їх життєвого циклу як аналіз та проектування. При цьому базова задача цієї стадії – створення, аналіз і удосконалення проектних рішень на основі системи візуальних моделей. Ці моделі повинні бути створені до початку розробки та програмування, надати комплексне представлення шляхів створення і удосконалення рішень побудови системи і бути доступними для аналізу і роботи всім спеціалістам – користувачам і розробникам.

В даний час у світі поширені три методології (нотації) візуального моделювання: IDEF (Icam DEFinition), ARIS (Architecture of Integrated Information Systems) та UML (Unified Modelling Language). Перші дві використовуються найчастіше при моделюванні бізнес-процесів. Ці нотації буди предметом розгляду, освоєння та застосування при виконанні циклів практичних, лабораторних робіт та курсового проектування в рамках курсу «Системний аналіз» в п'ятому семестрі [12]. Розгляду третьої методології , що стала , по суті, сучасним світовим стандартом розробки інформаційних систем, присвячений цей посібник.

Посібник складається з двох частин – теоретичної та практичної.

В теоретичні частині (розділ1) в конспектній формі викладаються основні положення об'єктно-орієнтованої методології аналізу та проектування інформаційних систем, особливості застосування уніфікованої мови моделювання UML, визначені процеси моделювання в UML, види статичних та динамічних діаграм. Приділена увага розгляду Rational Unified Process (RUP) – однієї з кращих сучасних методологій розробки інформаційних систем, що ґрунтується на індустріальних методах розробки. Одним з основних засад, на які спирається RUP, є процес створення і застосування візуальних моделей на різних етапах життєвого циклу системи за допомогою уніфікованої мови моделювання (UML).

В теоретичні частині посібника увага приділена одному з UML орієнтованих інструментальних засобів аналізу та проектування системи -

StarUML, який буде застосований в навчальному процесі при виконанні лабораторних робіт та при курсовому проектуванні.

В практичні частині (розділи 2-6) надані загальні вказівки до лабораторних робіт циклу «Об'єктно-орієнтований аналіз та проектування систем. Візуальне моделювання систем в StarUML», де визначені мета робіт, тематика, порядок виконання та зміст звіту по кожній роботі. В чотирьох розділах (3-6) посібника викладені методичні матеріали по виконанню лабораторних робіт по побудові наступних моделей – діаграм прецедентів (варіантів застосування) **(use case diagrams)**, діаграм класів **(class diagram)**, діаграм послідовностей (**sequence diagram**) та діаграм діяльності (**activity diagram**). Методичні матеріали визначають мету лабораторної роботи, надають основний теоретичний матеріал для їх виконання, інструкції по застосуванню інструменту StarUML при побудові моделі.

1. ТЕОРЕТИЧНІ ПОЛОЖЕННЯ ВИКОНАННЯ ЦИКЛУ ЛАБОРАТОРНИХ РОБІТ

1.1. Об’єктно-орієнтована методологія аналізу та проектування інформаційних систем

Проектування інформаційної системи базується на необхідності формування у авторів **системного уявлення про предметну область розробки**, що може формуватися на основі сукупності різноаспектних, зв’язаних у цілісне представлення моделей функціонування інформаційної системи, що проектується.

Всі сучасні технології аналізу і проектування ІС ґрунтуються на **методології моделювання предметної області, які поділяються на структурну та об’єктно-орієнтовану.** Принципова відмінність між структурною та об'єктно-орієнтованою методологією проектування інформаційних систем полягає в підході до декомпозиції системи. Об'єктно-орієнтований підхід базується на об'єктній декомпозиції.

Методологія об'єктно-орієнтованого аналізу і проектування (ООАП) була запропонована Йорденом для проектування великих систем. Автор вважав, що дана методологія дозволяє більш адекватно зобразити предметну область в системі і забезпечити більш надійний і перебудовуваний проект. ООАП складається з п'яти головних кроків:

1. Визначення предметної області.
2. Визначення об'єктів області.
3. Визначення структури об'єктів за рахунок створення відношень "складається з" і "являється".
4. Визначення атрибутів об'єктів.
5. Визначення сервісу об'єктів (методів поведінки) і взаємодій за рахунок посилання повідомлень між об'єктами.

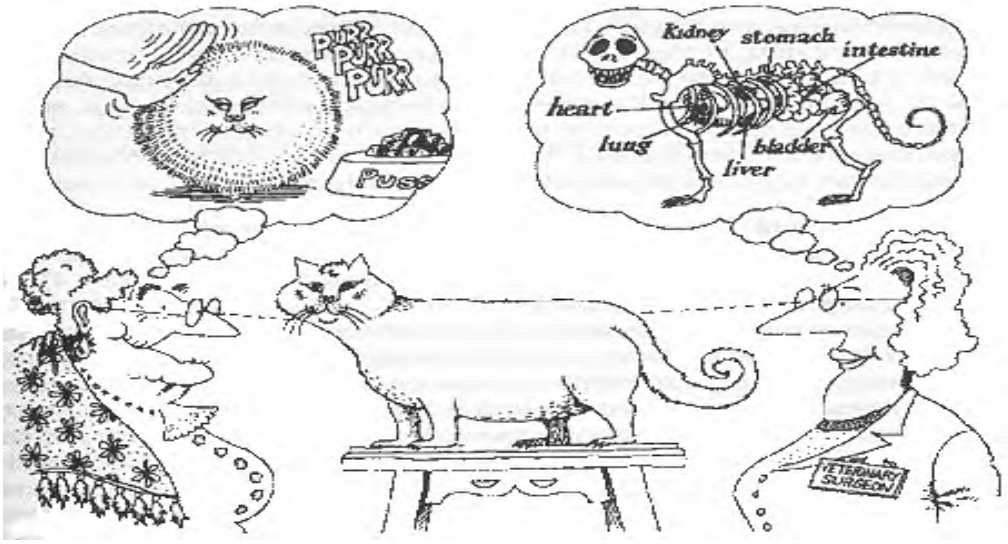
Статична структура інформаційної системи описується в термінах об'єктів і встановлених зв'язків між ними. Динамічна структура відображає поведінку системи і описується в умовах обміну повідомленнями між об'єктами. Кожен об'єкт системи має свою власну поведінку, що моделює поведінку об'єкта реального світу.

Концептуальною основою об'єктно-орієнтованого підходу є об'єктна модель. Основними положеннями її побудови є:

• абстрагування (abstraction);

• інкапсуляція (encapsulation);
• модульність (modularity);
• ієрархія (hierarchy).

Абстрагування - це виділення істотних характеристик деякого об'єкта, які відрізняють його від усіх інших видів об'єктів і, таким чином, чітко визначають його концептуальні кордони щодо подальшого розгляду та аналізу. Абстрагування концентрує увагу на зовнішніх особливостях об'єкта і дозволяє відокремити найістотніші особливості його поведінки від деталей їхньої реалізації. Вибір правильного набору абстракцій для заданої предметної області являє собою головне завдання об'єктно-орієнтованого проектування (рис.1.1).



Абстракція фокусується на суттєвих з точки зору спостерігача характеристиках об'єкта.

Рис. 1.1. Приклад вибору альтернативних зовнішніх характеристик реального об’єкту

З рис. 1.1. видно, що об’єктом реального світу є кіт. Але в голові дідуся та бабусі різні моделі цього об’єкту в їх системі представлення. Дідусь – ветеринар. Ціль побудови моделі – спостереження за здоров’ям кота. Тому при абстрагуванні він буде визначати наступні об’єкти системи: КІТ з такими характеристиками, як ім’я, порода, окрас, стан здоров’я; ОРГАНИ КОТА з такими характеристиками як назва органу, загальний стан; ЗАХВОРЮВАННЯ ОРГАНУ тощо. Бабуся - господиня кота. ЇЇ мета – забезпечення комфортного життя для кота. Тому при абстрагуванні вона застосує об’єкт КІТ, але додасть свої об’єкти системи такі як: ХАРЧУВАННЯ з такими характеристиками як назва їжі, частота застосування, кількість на один прийом; ПРОГУЛЯНКИ тощо.

Інкапсуляція - це процес **відокремлення** один від одного окремих елементів об'єкта, що визначають його пристрій і поведінку. Інкапсуляція служить для того, щоб ізолювати інтерфейс об'єкта, що відображає його зовнішнє поводження, від внутрішньої реалізації об'єкта. Об'єктний підхід передбачає, що власні ресурси, якими можуть маніпулювати тільки методи цього об’єкту, сховані від зовнішнього середовища. Абстрагування і інкапсуляція є взаємодоповнюючими операціями: абстрагування фокусує увагу на зовнішніх особливостях об'єкта, а інкапсуляція (чи, інакше, обмеження доступу) не дозволяє об'єктам-користувачам розрізняти внутрішній устрій об'єкта.

Ієрархія - це ранжирування або упорядкування системи абстракцій, розташування їх по рівнях. Прикладами ієрархії класів є просте і множинне спадкування (один клас використовує структурну або функціональну частину відповідно одного або декількох інших класів).

Основними поняттями об'єктно-орієнтованого підходу є об'єкт і клас(рис.1.2. ,1.3)

Експерт	***Ім’я класу***
ПІБ	***Поля даних (властивості)***
Звання	
Вчений ступінь	
Місце роботи	
Встановити рейтинг	***Методи (функції операції)***
Визначити допуск	

Рис. 1.2 Приклад опису класа

Об'єкт (рис.2.2) визначається як реальність (tangible entity) - предмет чи явище, що мають чітко визначену поведінку. Структура і поведінка схожих об'єктів визначають загальний для них клас. Терміни "екземпляр класу" і "об'єкт" є еквівалентними. Стан об'єкта характеризується переліком всіх можливих (статичних) властивостей даного об'єкта і поточними значеннями (динамічними) кожної з цих властивостей.

Петренко	***Ім’я об’єкту***
Петренко Юрій Іванович	***Поля даних (властивості)***
Професор	
Доктор технічних наук	
КНУБА	
Встановити рейтинг	***Методи (функції операції)***
Визначити допуск	

Рис. 1.3 Приклад опису об’єкта

Приклади об’єктів: Студент Петров, Київський національний університет будівництва та архітектури, Вища математика, Теорія прийняття рішень.

Клас - це безліч об'єктів, пов'язаних спільністю структури і поведінки. Будь-який об'єкт є екземпляром класу. Визначення класів і об'єктів - одна з найскладніших завдань об'єктно-орієнтованого проектування. Клас є моделлю ще не існуючої сутності (об'єкта). Клас має в складі поля даних та методи.

Два об’єкта - Вища математика та Теорія прийняття рішень належать одному і тому же класу - ДИСЦИПЛІНА. Назва дисципліни, семестр, обсяг годин, форма контролю – це його атрибути, що належать класу. Значення атрибутів належить об’єкту. Наприклад: Вища математика. 7 семестр, 100 годин, іспит. Всі об'єкти одного і того ж класу характеризуються однаковими наборами атрибутів.

Об'єднання об'єктів в класи дозволяє ввести в задачу абстракцію і розглянути її в загальнішій постановці. Клас має ім'я , яке відноситься до всіх об'єктів цього класу. Крім того, в класі вводяться імена атрибутів, які визначені для об'єктів. У цьому сенсі опис класу аналогічний опису типу структури (записи); при цьому кожен об'єкт має той же сенс, що і екземпляр структури (змінна або константа відповідного типа).

Поля даних (атрибути). Параметри (властивості) об'єкта що задають його стан. Фізично поля являють собою значення (змінні, константи), оголошені як такі, що «належать класу».

Методи. Процедури і функції, що пов'язані з класом. Вони визначають дії, які можна виконувати над об'єктом такого типу і які сам об'єкт може виконувати.

Об'єктно-орієнтоване проектування полягає в описі структури та поведінки проектованої системи, тобто, фактично, у відповіді на два основних питання:

• З яких частин складається система.

• У чому полягає відповідальність кожної з частин.

Виділення частин проводиться таким чином, щоб кожна мала мінімальний за обсягом і точно певний набір виконуваних функцій (обов'язків) і при цьому взаємодіяла з іншими частинами якомога менше.

1.2.Уніфікована мова моделювання UML

1.2.1. Загальна характеристика

Незважаючи на перевагу об’єктно-орієнтованих технологій аналізу та проектування інформаційних систем перед структурними, їх поширення до останнього часу було незначним, оскільки жоден з методів не давав єдиної і цілісної об’єктної моделі системи. Кожен метод висвітлював одну або декілька сторін реальної системи, залишаючи поза увагою багато інших, не менш важливих сторін. Крім того, відсутність єдиного стандарту не сприяла широкому поширенню об’єктно-орієнтованих методів при розробці програмного забезпечення. Мовою, яка об’єднала сильні сторони відомих методів і забезпечила найкращу підтримку моделювання, стала уніфікована мова моделювання UML (Unified Modeling Language). Вона з’явилась внаслідок розвитку методів об’єктно-орієнтованого аналізу і проєктування (OOA&D – Object-Oriented Analyses&Design). Мова моделювання пройшла процес стандартизації в рамках консорціуму OMG (Object Management Group) і на сьогодні прийнята як фактичний стандарт OMG. UML – це сімейство графічних нотацій, яке допомагає в описі та протктуванні програмних систем, особливо систем, побудованих з використанням об’єктно-орієнтованої парадигми. Графічні мови моделювання вже тривалий час широко використовуються в індустрії програмування і проектування інформаційних систем. Більшість існуючих методів об'єктно-орієнтованого аналізу і проектування (ООАП) включають як **мову моделювання**, так і **опис процесу моделювання**.

Мова моделювання - це нотація (сукупність умовних знаків у системі правил для опису), яка використовується методом для опису проектів.

Нотація являє собою сукупність графічних об'єктів, які використовуються в моделях. Вона є синтаксисом мови моделювання. Наприклад, нотація діаграми класів визначає, яким чином подаються такі елементи і поняття, як клас, асоціація і множинність.

Процес - це опис кроків, які необхідно виконати при розробці проекту.

Мова UML є мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів і інших систем. Мова UML одночасно є простим і потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних і графічних моделей складних систем самого різного цільового призначення. Ця мова

увібрала в себе найкращі якості методів програмної інженерії, які з успіхом використовувалися впродовж останніх років при моделюванні великих і складних систем.

Уніфікована мова моделювання UML (Unified Modeling Language) - це наступник того покоління методів ООАП, які з'явилися в кінці 80-х і початку 90-х рр. Створення UML фактично почалося в кінці 1994 р., коли Граді Буч і Джеймс Рамбо почали роботу по об'єднанню методів Booch і OMT (Object Modeling Technique) під егідою компанії Rational Software. До кінця 1995 р. вони створили першу специфікацію об'єднаного методу, названого ними Unified Method, версія 0.8. Тоді ж, у 1995 р., до них приєднався творець методу OOSE (Object-oriented Software Engineering) Івар Якобсон.

Таким чином, UML є прямим об'єднанням і уніфікацією методів Буча, Рамбо і Якобсона, однак доповнює їх новими можливостями.

Головними в розробці UML були наступні цілі:

• надати користувачам готову до використання виразну мову візуального моделювання, що дозволяє розробляти осмислені моделі та обмінюватися ними;

• передбачити механізми розширюваності і спеціалізації для розширення базових концепцій;

• забезпечити незалежність від конкретних мов програмування і процесів розробки;

• забезпечити формальну основу для розуміння цієї мови моделювання.

UML необхідний:

- керівникам проектів, які керують розподілом завдань і контролем за проектом;
- проектувальникам інформаційних систем, які розробляють технічні завдання для програмістів;
- бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії;
- програмістам, які реалізують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Володіння об'єктно-орієнтованою мовою програмування (наприклад, Java) та доступ до широкої бібліотеки ресурсів ще недостатня умова об'єктної

системи. Дуже важливу роль тут грають аналіз та проектування системи з точки зору об'єктної методології.

Вибір правильного набору абстракцій для заданої предметної області являє собою головне завдання об'єктно-орієнтованого аналізу та проектування.

1.2.2.Процес моделювання в UML

Моделювати складну систему слід з кількох різних точок зору, кожен раз беручи до уваги один аспект, що моделюється, і абстрагуючись від інших.

Ця теза є одним з основоположних принципів UML і може бути найважливішим принципом, що зумовив практичний успіх UML.

Моделювання в UML змістовно поділяється на моделювання використання, моделювання структури та моделювання поведінки системи, зв'язок між ними (рис.1.4)представлений у вигляді діаграми діяльності.

Моделювання використання. Опис використання повинний відповідати на питання, що робить система корисного. Визначальною ознакою для віднесення складових моделі до опису використання є встановлення для системи зовнішніх кордонів та зовнішніх дійових осіб, котрі взаємодіють із системою, внутрішніх варіантів використання, що описують різні сценарії такої взаємодії. Таким чином, єдиним виразним засобом представлення використання виявляються діаграми використання. В UML це діаграми прецедентів та опис прецедентів.

Моделювання структури. Подання структури покликане відповідати (з різним ступенем деталізації) на питання - з чого складається система.

Визначальною ознакою для віднесення складових моделі до подання структури є явне виділення складових частин системи і опису взаємозв'язків між ними. Принциповим є чисто статичний характер опису, тобто відсутність поняття часу в будь-якій формі, зокрема, у формі послідовності в формі послідовності подій і / або дій.

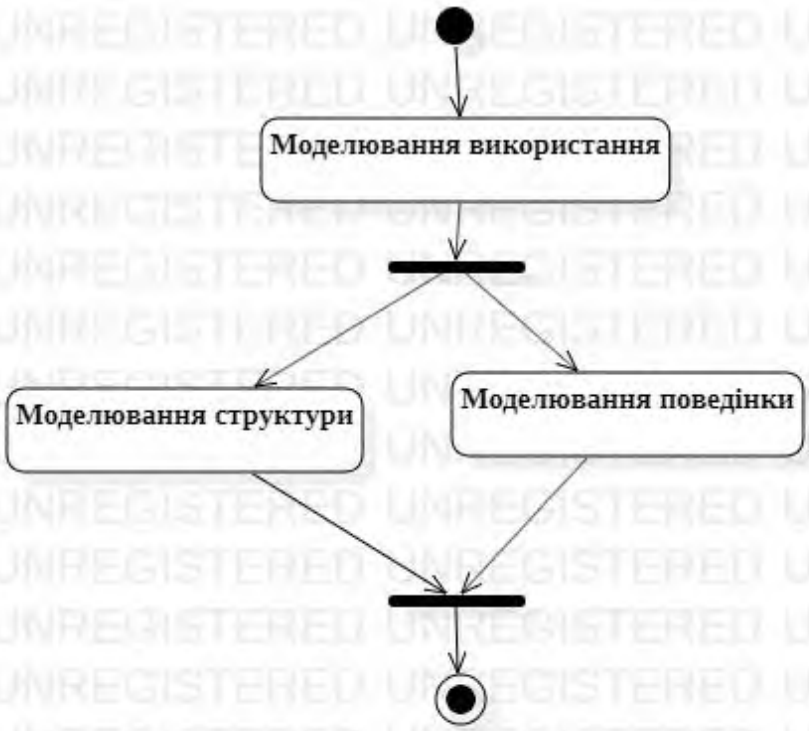


Рис 1.4. Діаграма діяльності процесу моделюванні в UML

Подання структури описується головним чином діаграмами класів, а також додатково діаграмами компонентів, розміщення, внутрішньої структури і , в окремих випадках, діаграмами об'єктів.

Моделювання поведінки. Моделювання поведінки покликане відповідати на питання: як працює система. Визначальною ознакою для віднесення складових моделі до подання поведінки є явне використання поняття часу, зокрема, у формі опису послідовності подій / дій. Подання поведінки описується діаграмою діяльності, а також оглядової діаграмою взаємодії, діаграмами комунікації і послідовності.

Моделі UML можуть створюватися і можуть використовуватися з різними цілями. Іноді буває навіть так, що автор створював модель з однієї певною метою, а використовується вона іншими людьми абсолютно несподіваним для автора чином. Таким чином, призначення моделі не є чимось постійним і важко змінним. Трансформації призначення моделей цілком можливі, але, тим не менш, практика моделювання підказує , що моделі дають більший ефект, якщо при моделюванні брати до уваги призначення моделі .

Розглянемо три типу призначення моделі:

- Концептуальні моделі
- Моделі проектування
- Моделі реалізації

Концептуальна модель (conceptual model). Формулювання змістовного і внутрішнього представлення, що **поєднує концепцію користувача й розробника моделі.** Це абстрактна модель, що визначає структуру модельованої системи, властивості її елементів і причинно-наслідкові зв'язки, властиві системі , що є суттєві для досягнення мети моделювання.

Основні елементи концептуальної моделі:

- умови функціонування об’єкта, що визначені характером взаємодії між об'єктом і його оточенням а також між елементами об'єкта;
- мета дослідження об'єкта та напрямок покращення його функціонування;
- можливості керування об'єктом, визначення складу керованих змінних об'єкта.

Концептуальною основою об'єктно-орієнтованого підходу є діаграма класів. Існують різні точки зору на побудову діаграм класів в залежності від цілей їх застосування.

Концептуальна точка зору - діаграма класів описує модель предметної області, в ній присутні тільки класи прикладних об'єктів.

Точка зору проектування (специфікації) - діаграма класів застосовується при проектуванні інформаційних систем. Деталізує концептуальні рішення для можливостей побудови системи і здійснення процесу програмування

Точка зору реалізації - діаграма класів містить класи, використовувані безпосередньо в програмному коді (при використанні об'єктно - орієнтованих мов програмування) .

В рамках курсу буде розглядатися моделі з концептуальної та проектної точки зору.

1.2.3. Діаграми UML

Діаграма (diagram) - графічне представлення сукупності елементів моделі у формі зв'язного графа, вершинам і ребрам (дугам) якого приписується визначена семантика.

При моделюванні систем, незалежно від предметної області, UML передбачає можливість побудови діаграм різних типів, що відображають найбільш важливі аспекти побудови системи. Перелік цих діаграм і їх назв є канонічними в тому сенсі, що являють собою невід'ємну частину графічної нотації мови UML. Більше того, процес об'єктно-орієнтованого проектування

нерозривно пов'язаний із процесом побудови цих діаграм. Сукупність побудованих у такий спосіб діаграм є самодостатньою в тому сенсі, що в них міститься вся інформація, яка необхідна для реалізації проекту складної системи (рис.1.5).

Рис. 1.5. Структура діаграм UML

Структурні діаграми. В UML існують структурні діаграми для візуалізації, специфікації, конструювання та документування статичних аспектів системи, які складають її основу.

Статичні аспекти інформаційних систем відображають наявність та розташування класів, інтерфейсів, кооперацій, компонентів, вузлів та інших сутностей.

До структурних діаграм віднесені:

діаграми класів (class diagrams) призначені для моделювання структури об'єктно-орієнтованих додатків - класів, їх атрибутів і заголовків методів, наслідування, а також зв'язків класів один з одним;описують статичну структуру класів. Дозволяють (на концептуальному рівні) формувати "словник предметної області" та (на рівні специфікацій і рівні реалізацій) визначати структуру класів у програмній реалізації системи;

діаграми компонентів (component diagrams) використовуються при моделюванні компонентної структури розподілених додатків; всередині кожна компонента може бути реалізована за допомогою безлічі класів;

діаграми розгортання (deployment diagrams) призначені для моделювання апаратної частини системи, з якою програмне забезпечення безпосередньо зв'язано (розміщено або взаємодіє);

діаграми поведінки. П’ять основних діаграм поведінки в UML використовується для візуалізації, специфікування, конструювання та документування динамічних об’єктів системи. Можна вважати, що динамічні об’єкти системи представляють собою її змінювані частини. Динамічні аспекти

системи охоплюють такі її елементи як потік повідомлень у часі та фізичне переміщення компонентів у мережі.

Діаграми поведінки в UML умовно розділяються на п’ять типів у відповідності з основними способами моделювання динаміки системи:

- **Діаграми прецедентів (варіантів використання)** описують організацію поведінки системи. Такі діаграми описують *функціональність*, яка буде надаватись користувачам системи, котра проектується. Представляються шляхом використання *прецедентів* та *акторів*, а також *відношень між ними*. Набір усіх прецедентів діаграми фактично визначає *функціональні вимоги*, за допомогою яких може бути сформульоване *технічне завдання*.
- **Діаграми послідовностей** акцентують увагу на часовій упорядкованості повідомлень;
- **Діаграми кооперації** сфокусовані на структурній організації об’єктів, що відсилають та отримують повідомлення;
- **Діаграми станів** описують зміни стану системи у відповідь на події;
- **Діаграми діяльності** демонструють передачу управління від однієї діяльності до іншої.

У більшості випадків створювані розробником діаграми можна буде віднести до одного з типів, та тільки зрідка знадобляться нові, специфічні для окремого проекту. У кожної діаграми повинне бути унікальне ім’я, з допомогою якого можна посилатись на неї та відрізняти від іншої. При роботі із складною системою доведеться об’єднувати діаграми в пакети.

Діаграми ***UML*** в своїй основі є графами спеціального виду, що складаються з вершин у формі геометричних фігур, які пов'язані між собою ребрами або дугами. Оскільки інформація, яку містить в собі граф, має в основному топологічний характер, ні геометричні розміри, ні розташування елементів діаграм не мають принципового значення.

Для діаграм мови UML існують три типи візуальних позначень, які є важливими з точки зору укладеної в них інформації:

- **Зв'язки**, які представляються різними лініями на площині. Зв'язки в мові UML узагальнюють поняття дуг і ребер з теорії графів, але мають менш формальний характер.
- **Текст**, який міститься всередині кордонів окремих геометричних фігур на площині. При цьому форма цих фігур (прямокутник, еліпс) відповідає деяким елементам мови UML (клас, варіант використання) і має фіксовану семантику.

• **Графічні символи** – візуальні змістовні елементи діаграм.

При графічному зображенні діаграм слід дотримуватися наступних основних рекомендацій:

• Кожна діаграма повинна служити закінченим представленням відповідного фрагмента модельованої предметної області. Йдеться про те, що в процесі розробки діаграми необхідно врахувати всі сутності, важливі з точки зору контексту даної моделі і діаграми. Відсутність тих чи інших елементів на діаграмі є ознакою неповноти моделі, а також вимагати її подальшого доопрацювання.

• Всі сутності на діаграмі моделі повинні бути одного концептуального рівня. Тут мається на увазі узгодженість не тільки імен однакових елементів, але і можливість вкладення окремих діаграм один в одного для досягнення повноти уявлень. У разі досить складних моделей систем бажано дотримуватися стратегії послідовного уточнення або деталізації окремих діаграм.

• Діаграми не повинні містити суперечливої інформації. Суперечливість моделі може служити причиною серйозних проблем при її реалізації і подальшому використанні на практиці.

При графічному зображенні діаграм слід дотримуватися наступних основних рекомендацій:

• Діаграми не слід перевантажувати текстовою інформацією. Прийнято вважати, що візуалізація моделі є найбільш ефективною, якщо вона містить мінімум пояснювального тексту. Як правило, наявність великих фрагментів розгорнутого тексту служить ознакою недостатньої опрацьованості моделі або її неоднорідності, коли в рамках однієї моделі представляється різна за характером інформація. Оскільки загальна декомпозиція моделі на окремі типи діаграм здатна задовольнити детальні уявлення розробників про систему, важливо вміти правильно відображати ті чи інші сутності та аспекти моделювання в відповідні елементи канонічних діаграм.

• Кожна діаграма повинна бути самодостатньою для правильної інтерпретації всіх її елементів і розуміння семантики всіх використовуваних графічних символів. Будь-які пояснювальні тексти, які не є власними елементами діаграми (наприклад, коментарями), не повинні прийматися до уваги розробниками.

1.3 UML як базова складова RUP (Rational Unified Process) методології створення систем

1.3.1. Загальна характеристика RUP методології

Rational Unified Process (RUP) – одна з кращих методологій розробки складних інформаційних систем. Вона створена в компанії Rational Software, ґрунтуючись на індустріальних методах розробки. Одним з основних засад, на які спирається RUP, є процес створення моделей за допомогою уніфікованої мови моделювання (UML).

RUP - це керівництво по тому, як ефективно використовувати UML на різних етапах життєвого циклу інформаційних систем. RUP увібрав в себе все найкраще, що є на сьогоднішній день в області організації розробки ІС, включаючи розв'язання:

- концептуальних питань (бізнес-моделювання, аналіз вимог);
- аналіз і проектування,
- розробку;
- тестування;
- управління конфігурацією і управління змінами.

RUP - **це технологічний процес** по створенню ІС, що дозволяє поліпшити продуктивність колективної розробки шляхом надання ***для всіх етапів життєвого циклу методик виконання основних видів діяльності,*** шаблонів документів, інструкцій по роботі з інструментальними засобами.

RUP розроблявся сумісно з UML - стандартом об'єктно-орієнтованого моделювання, тим же колективом авторів.

В чому особливість RUP ?

- RUP – в дослівному перекладі "раціональний уніфікований процес".
- **RUP - це ітеративний процес** (Controlled Interactive). Передбачає наскрізне застосування **апарату Use Cases (Use Case Driven) (прецеденти).**
- Особлива увага приділяється **розробці архітектури** (Architecture Centric).
- Включає управління вимогами та змінами (Requirements Configuration and Change Management). Базується на візуальному моделюванні (Visual Modeling Techniques).

1.3.2 . Особливості ітераційного процесу в RUP методології

Класичний **водоспадний життєвий цикл** включає етапи аналізу вимог, проектування, розробки, побудови та тестування ІС, що виконуються **послідовно.** При цьому підході необхідність внесення змін в роботи системи виявляється на завершальних етапах розробки, тому потрібно правити модель всієї системи і вносити зміни великого обсягу. Такий підхід хороший для маленьких проектів і в тих випадках, коли вимоги до ІС суворо визначені.

RUP базується на **спіральній (ітеративній**) моделі життєвого циклу системи.

При цьому пропонує чотири стадії (фази):

- входження в проект (дослідження),
- розвиток (уточнення плану),
- конструювання,
- розгортання.

Особливості ітераційного процесу:

1. **Кожна фаза** складається з послідовності ітерацій, число яких може бути будь-яким.
2. У кожній ітерації перераховані вище технологічні процеси послідовно **застосовуються до розробки невеликої частини ІС.**
3. При цьому раціональним є пред'явлення результату замовникові. Він має можливість оцінити виконану реалізацію, видати свої зауваження, які можуть привести до зміни та уточнення вимог до ІС.
4. **Наступна ітерація** передбачає **розширення вже розробленої частини** шляхом:
   - реалізації та інтеграції чергової порції вимог,
   - обліку зміни вимог відповідно до зауважень замовника.

Переваги ітераційного процесу:

- Зміни та уточнення вимог виявляються вже в ранній період розробки, коли обсяг програмного коду порівняно невеликий, тому трудомісткість внесення змін істотно нижче.
- Вже на ранній стадії процесу розробки є можливість залучення фахівців замовника для оцінки проміжних версій (прототипів) ІС. Як результат - значно більш висока ймовірність того, що кінцевий продукт буде робити саме те, чого чекає від нього замовник, тобто, гарантується висока якість ІС
- Знижуються архітектурні та інтеграційні ризики. При ітеративному підході створюється стійка **архітектура,** яка **опрацьовується на**

стадіях дослідження, а потім перевіряється і уточняються в декількох початкових ітераціях.

• **Кожна ітерація передбачає інтеграцію нових елементів** в систему, тобто, **кількість інтегрованих елементів в кожній ітерації невелика** і легше простежується, ніж при глобальної інтеграції всіх розроблених елементів ІС.

• Ітеративний підхід сприяє більш повному повторному використанню програмних елементів. Аналіз результатів кожної ітерації дозволяє архітекторам ІС виділити фрагменти, потенційно підлягають повторному використанню, а в наступній ітерації оформити їх як повторно використовувані.

1.3.3 Принципи та умови реалізації RUP методології

В основі RUP лежать наступні принципи :

- **Рання ідентифікація** і безперервне (до закінчення проекту) усунення основних ризиків побудови системи.
- **Концентрація на виконанні вимог замовників** до виконуваній програмі - аналіз і побудова моделі прецедентів (варіантів використання).
- **Очікування змін у вимогах**, проектних рішеннях і реалізації в процесі розробки.
- **Компонентна архітектура**, реалізована і протестована на ранніх стадіях проекту.
- Постійне забезпечення якості на всіх етапах розробки проекту (продукту).
- Робота над проектом в згуртованій команді, **ключова роль в якій належить архітекторам.**

Умови реалізацій RUP – методології:

1. **Базування на тому, що весь процес будується на варіантах використання (прецедентах).** На етапах аналізу вимог, проектування і реалізації use cases виступають в якості варіантів використання системи будучи тією «грубкою», від якої «танцюють» аналітики і розробники при виконанні проектування і реалізації ІС.

При аналізі вимог, виділивши прецеденти, ми тим самим визначаємо вимоги або рівень ієрархії вимог до ІС.

При деталізації прецедентів визначаються об'єкти, і способи їх взаємодії, які повинні бути реалізовані в умовах програмування.

Слід особливо наголосити роль варіантів використання при **плануванні ітерацій розробки прототипів системи**. В цих умовах на діаграмах прецедентів, для кожного з низ встановлюється пріоритет, який визначає в якій ітерації розвитку можливостей системи його слід реалізувати. Можна показати всі прецеденти, що реалізуються в черговий ітерації, на окремій діаграмі (або декількох діаграмах).

2. **RUP - процес, заснований на архітектурі системи.**

Розробка архітектури ІС в RUP переслідує такі цілі:

• Опис організації ІС.

• Визначення компонентів системи і їх інтерфейсів.

• Визначення підсистем і об'єднання компонентів у підсистеми.

• Створення основи для повторного використання компонентів.

• Створення базису для розробки.

• Контроль над проектом і підтримка цілісності системи.

3.При розробці архітектури акцентується увага тільки на таких елементах, які мають істотний вплив на структуру ІС.

До числа таких елементів відносяться:

• **Класи,** що моделюють основні об'єкти діяльності.

• **Механізми**, що визначають **поведінку цих класів**.

• **Рівні і підсистеми**.

• **Інтерфейси**.

• **Основні процеси і керуючі потоки**.

4.Особливістю RUP є те, що в результаті роботи над проектом створюються і удосконалюються моделі. Замість створення величезної кількості паперових документів, RUP спирається на розробку і розвиток семантично збагачених моделей, всебічно представляють розроблювану систему. **RUP - це керівництво по тому, як ефективно використовувати UML. Стандартна мова моделювання, який використовується усіма членами групи, робить зрозумілими для всіх опису вимог, проектування і архітектуру системи.**

5.**RUP підтримується інструментальними засобами, які автоматизують великі розділи процесу.** Вони використовуються для створення і вдосконалення різних проміжних продуктів (прототипів) на різних

етапах процесу створення ІС , наприклад, при візуальному моделюванні, програмуванні, тестуванні і т.д.

1.3.4.Процес розробки по RUP методології

Стадії (фази) розробки системи:

1. Початкова стадія (Inception) – входження в проект.
2. Уточнення (Elaboration) – проектування.
3. Побудова (Construction) – конструювання.
4. Впровадження (Transition).

RUP визначено дев'ять технологічних процесів, для кожного з яких запропонована методика виконання і визначається, що виконується на кожній стадії.

Технологічні процеси діляться на дві категорії :

1. Основні процеси.
2. Процеси підтримки

До основних відносяться:

- Бізнес-аналіз.
- Управління вимогами.
- Аналіз і проектування.
- Реалізація.
- Тестування.
- Розгортання.

Допоміжні процеси включають:

- Управління проектом.
- Управління конфігурацією.
- Управління середовищем.

Графічне представлення процесу розробки по RUP представлено на рис. 1.6.

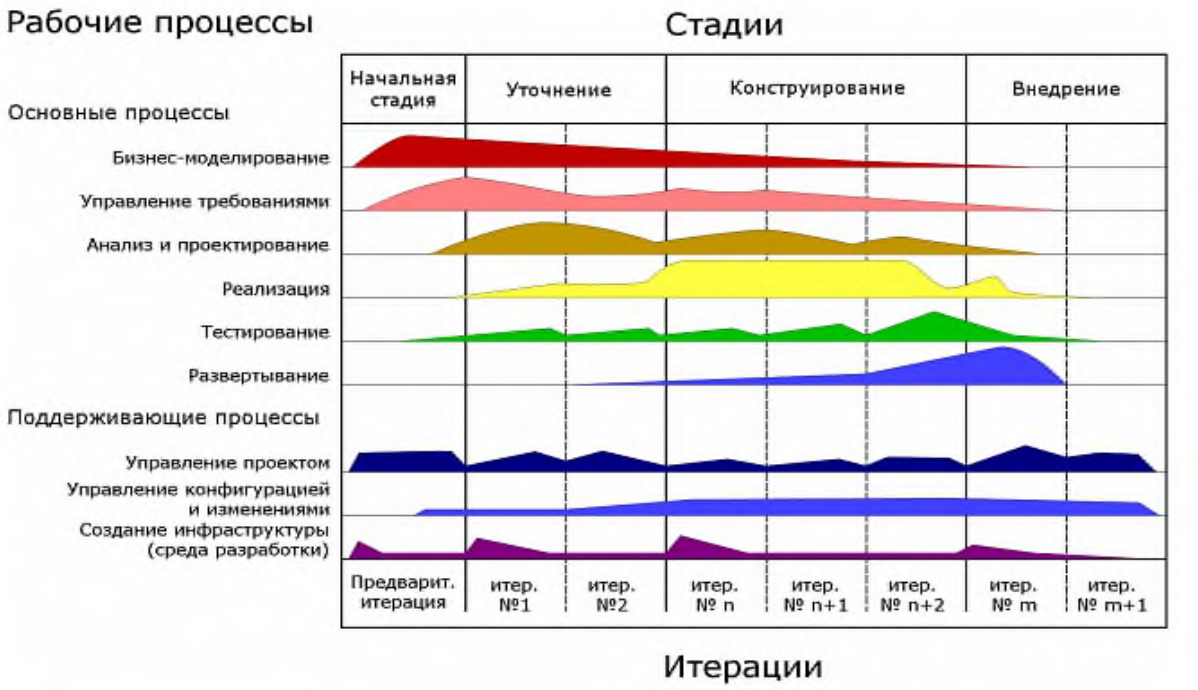


Рис 1.6. Процес розробки по **RUP методології**

1.3.5. Характеристика стадій (фаз) розробки

1.3.5.1. Початкова стадія – входження в проект

Первинною ціллю є адекватна оцінка системи як бази для обчислення початкових розцінок та бюджету. Базується на **бізнес-моделюванні, що спрямоване на те, щоб визначити** бачення організації, в якій буде розгортатись система і визначенні, як використати це бачення для виділення процесу, ролей та обов'язків. Розробники системи повинні розуміти структуру і динаміку цільової організації (клієнта), нинішні проблеми в організації, а також можливі удосконалення. При реалізації робіт виконується:

- Формуються бачення і межі проекту.
- Створюється економічне обґрунтування (business case).
- Визначаються основні вимоги, обмеження і ключова функціональність продукту.
- Створюється базова версія моделі прецедентів.

При завершенні початкової стадії оцінюється досягнення віхи цілей життєвого циклу (англ. Lifecycle Objective Milestone), яке передбачає угоду зацікавлених сторін про продовження проекту.

1.3.5.2.Стадія (фаза) уточнення (проектування)

На етапі проектування проводиться **аналіз предметної області та побудова виконуваної архітектури.**

Ця фаза має пройти віху життєвого циклу архітектури (LCA), задовольняючи такі критерії:

- Модель прецедентів, в якій ідентифікуються прецеденти та актори, та розробляється більшість описів прецедентів. Модель прецедентів повинна бути завершена на 80%.
- Опис архітектури системи.
- Виконувана архітектура, яка реалізує архітектурно значимі прецеденти.
- Бізнес — випадки та список ризиків переглядаються.
- План розвитку проекту в цілому.
- Прототипи, що явно зменшили кожен виявлений технічний ризик.

Успішне виконання фази проектування означає досягнення віхи архітектури життєвого циклу (англ. Lifecycle Architecture Milestone)

1.3.5.3.Стадія (фаза) конструювання

Під час цієї фази відбувається реалізація більшої частини функціональності продукту. ***Фаза завершується першим зовнішнім релізом системи і віхою початкової функціональної готовності (Initial Operational Capability).***

Основна мета полягає в створенні інформаційної системи. Це етап, коли відбувається основна частина кодування. У більш великих проектах, може бути кілька фаз конструювання, в спробі поділити прецеденти на керовані сегменти, які можуть утворити презентабельні прототипи.

Цей етап створює перший реліз(показ) програмного забезпечення.

1.3.5.4.Стадія (фаза) впровадження

Створюється фінальна версія продукту і передається від розробника до замовника. Це включає в себе програму бета-тестування, навчання користувачів, а також визначення якості продукту. У разі, якщо якість не відповідає очікуванням користувачів або критеріям, встановленим у фазі Початок, фаза Впровадження повторюється знову. Виконання всіх цілей

означає досягнення віхи готового продукту (Product Release) та завершення повного циклу розробки.

Основна мета полягає в переведенні системи **з розробки у продукт,** зробивши її доступною та зрозумілою для кінцевого споживача.

Якщо всі вимоги задоволені, досягається віха релізу продукту, і цикл розробки завершується.

1.4.StarUML як інструментальний засіб аналізу та проектування систем

CASE є методологією проектування ІС, а також набором інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розроблення і супроводу ІС і розробляти застосування відповідно до інформаційних потреб користувачів.

Більшість існуючих CASE-засобів ґрунтується на методології структурного або об’єктно-орієнтованого аналізу і проектування, що використовують специфікації у вигляді діаграм або текстів для описання зовнішніх вимог, зав’язків між моделями системи, динаміки поведінки системи й архітектури програмних засобів.

Перевагою застосування розробниками систем інструментальних засобів є:

- Підтримка колективної роботи - можливість паралельної розробки окремих компонент проекту різними групами розробників з можливістю інтеграції результатів в один загальний проект.
- Використання типових рішень - використання раніше накопиченого досвіду при прийнятті рішень, наприклад, використання готових фрагментів моделі.
- Автоматичне створення компонент - наприклад, автоматична кодогенерація (створення комп'ютерних програм, баз даних на основі введених моделей і діаграм), формування різного роду звітів, документації по заданим шаблоном і т.д.

Існує досить багато CASE-інструментів аналізу та проектування систем (рис. 1.7).В рамках об’єктно-орієнтованої методології аналізу та проектування систем UML є загальновизнаним стандартом, застосування якого підтримується багатьма об’єктно-орієнтованими CASE-продуктами. Знання UML є

необхідним не лише для системних аналітиків і проектувальників, але й для звичайних програмістів і тестувальників програмного забезпечення.

Постійно збільшується ринок UML орієнтованих інструментальних засобів, призначених для автоматизації процесів життєвого циклу створення інформаційних систем. У цьому посібнику увага приділена одному з UML орієнтованих інструментальних засобів аналізу та проектування системи, StarUML [13], який буде застосований в навчальному процесі при виконанні лабораторних робіт та курсовому проектуванні. Вибір цього інструментального засобу обґрунтований наступними факторами:

- Ця програмна платформа має вільну ліцензію та доступна для встановлення з офіційного сайту StarUML. *StarUML* - програмний інструмент моделювання, який підтримує UML (Уніфікована мова моделювання).
- StarUML орієнтований на UML версії 1.4 і підтримує одинадцять різних типів діаграм, прийнятих в нотації UML 2.0.
- StarUML активно підтримує підхід MDA (Модельна керована архітектура), реалізуючи концепцію профілів UML.

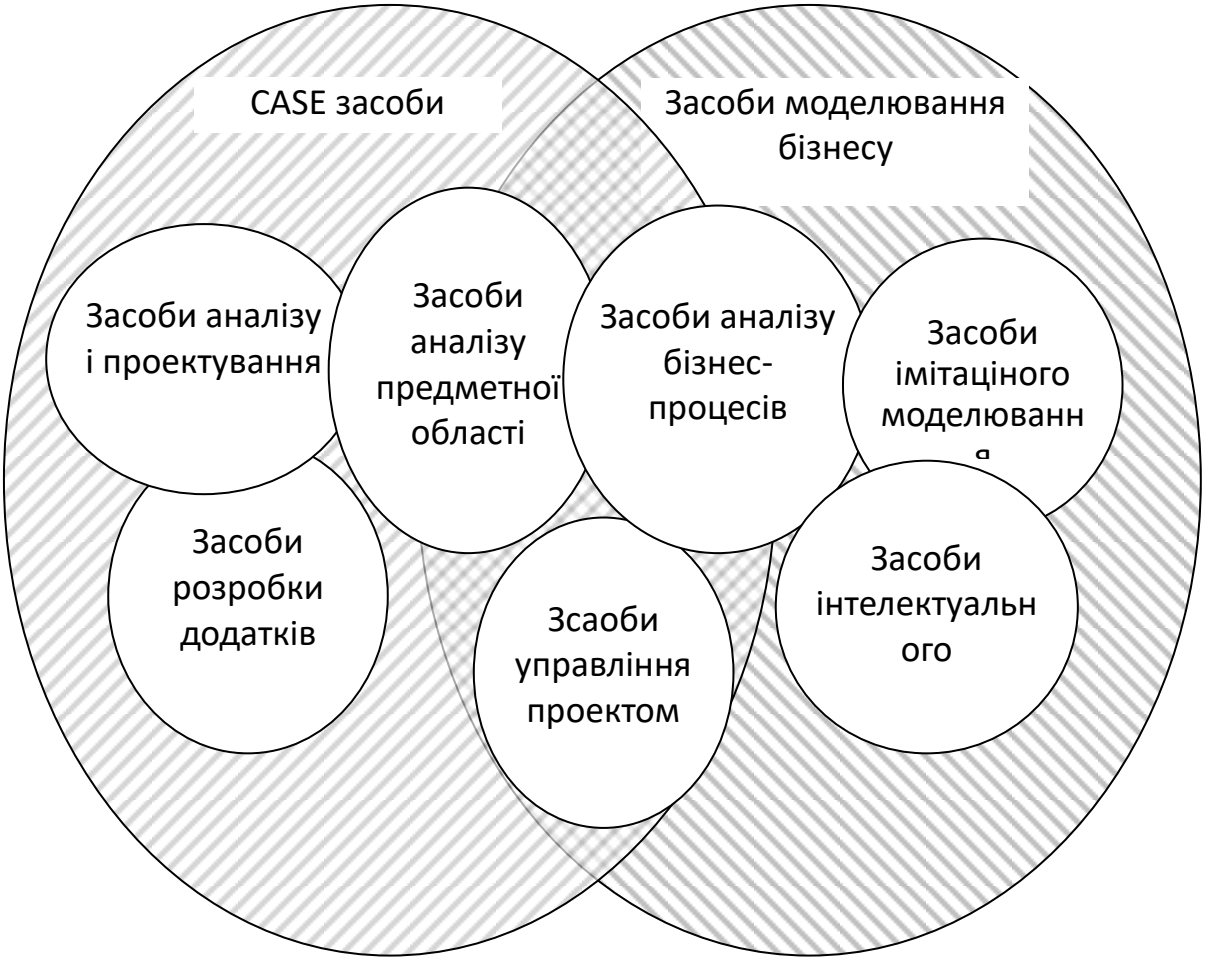


Рис. 1.7. Класифікація CASE-інструментів аналізу та проектування систем

- Середовище розробки StarUML чудово налаштовується відповідно до вимог користувача і має високий ступінь розширюваності, особливо в області своїх функціональних можливостей.
- Використання StarUML, одного з провідних програмних інструментів моделювання, гарантує досягнення максимальної продуктивності і якості програмних проектів.
- Програмний продукт StarUML (укр. Стар ЮЕМЕЛ) від розробника MKLabs призначений для створення та застосування графічних моделей у нотації UML.
- Система заснована на UML 2.0 та надає одинадцять різних видів діаграм, активно підтримуючи таким чином підхід побудови архітектур на базі моделей (англ. Model Driven Architecture, MDA).
- Система може ефективно застосовуватися системними аналітиками, проектувальниками та архітекторами систем, інженерами-програмістами.
- StarUML строго дотримується специфікації UML, розробленої OMG для моделювання програм. Слід розуміти, що використання нерегулярного синтаксису UML в інформаційних проектах через 10 або більше років може привести до сумних наслідків. StarUML ™ максимально відповідає стандарту UML 1.4 і слід нотації UML 2.0 на основі стійкої мета-моделі.
- На відміну від багатьох існуючих програм, які використовують власні неефективні формати файлу моделі, StarUML оперує файлами в стандартному форматі XML. Коди, написані в легких для читання структурах і форматах, можуть бути легко змінені з допомогою синтаксичного аналізатора XML. З огляду на факт, що XML є світовим стандартом, це, звичайно, велика перевага, яке гарантує, що програмні моделі залишаться корисними більше ніж через десятиліття.
- StarUML – це проект з відкритим кодом для розробки швидких, гнучких, діаграм які розширюються, мають велику функціональність і головне розповсюджуються безкоштовно на платформі UML / MDA для 32-розрядних та 64-розрядних системах, таких як: Windows, Linux, Mac OS X.
- Проект StartUML включає мету створення універсальної безкоштовної платформи для моделювання, яка послужить аналогом для таких комерційних проектів, як Rational Rose, Together та інших.
- StarUML є одним з найпопулярніших інструментів UML в світі. Він був завантажений більш ніж на 5.000.000 і використовується в більш ніж 150 країнах світу.

2.ЗАГАЛЬНІ ВКАЗІВКИ ДО ЛАБОРАТОРНИХ РОБІТ ЦИКЛУ «ОБ'ЄКТНО-ОРІЄНТОВАНИЙ АНАЛІЗ ТА ПРОЕКТУВАННЯ СИСТЕМ. ВІЗУАЛЬНЕ МОДЕЛЮВАННЯ СИСТЕМ В STARUML»

2.1. Мета лабораторних робіт циклу

Визначений цикл лабораторних має на меті формування у студентів навичок в трьох напрямках:

- застосування об'єктно-орієнтованої методології аналізу та проектування захищених інформаційних систем та мереж;
- застосування мови UML для моделювання та проектування інформаційних систем;
- застосування відповідного програмного інструментарію обєктно-орієнтованного моделювання StarUML.

2.2.Тематика лабораторних робіт

Практичні та лабораторні роботи студент виконує по єдиній для всього циклу обраній особисто та затвердженій викладачем темі. Тема циклу робіт повинна відповідати назві окремої інформаційної системи або одного з її функціональних модулів (підсистеми, комплексу задач, задачі). Вважається доцільним, щоб обрана тема робіт відповідала напряму теоретичних та практичних інтересів студента. Тому вибір студента вільний, він полягає в його пропозиції щодо бажаної теми робіт або в виборі теми з переліку, що пропонує викладач. В якості прикладу сформульовані наступні теми – варіанти вибору:

1. Система оцінки ризиків порушення конфіденційності інформації.
2. Система проектування комп'ютерної мережі підприємства.
3. Система захисту даних на основі рольової моделі.
4. Система моніторингу бездротової локальної мережі.
5. Система захисту від несанкціонованого доступу до інформаційно-телекомунікаційних систем.
6. Оцінка ризику інформаційної безпеки.
7. Система захисту розумного будинку
8. Система тестування знань студентів.
9. Система захисту комерційної інформації.
10. Система оцінки рейтингу вищого навчального закладу.
11. Система поставки ресурсів.

12. Система оцінки ринкової вартості нерухомості.
13. Система календарного планування виконання робіт.
14. Система оцінки рейтингу викладача.
15. Система оцінки програмного продукту.
16. Система відбору персоналу.
17. Система захисту програмного забезпечення
18. Система управління доступом користувача до інформаційних систем.
19. Система прийняття рішень по підвищенню конкурентоспроможності продукції підприємства.
20. Система прийняття рішень оцінки основних напрямків розвитку будівельного підприємства.

2.3. Порядок виконання лабораторних робіт

Для виконання всіх лабораторних робіт пропонується єдиний порядок, який передбачає наступні кроки:

1. Ознайомитися з постановкою завдання і вихідними даними.
2. Провести аналіз предметної області теми своєї роботи.
3. Розробити запропоновану в роботі модель застосуванням інструментального засобу StarUML.
4. Зберегти файл моделі і надати опис виконаної роботи.
5. Скласти звіт про виконану роботу.

2.4. Зміст звіту про виконання лабораторних робіт

Звіт оформляється по кожній лабораторній роботі і складається з наступних розділів:

- Тема лабораторної роботи.
- Мета роботи.
- Індивідуальне завдання.
- Розроблена модель і опис виконаної роботи.

3.ЛАБОРАТОРНА РОБОТА №1.
ДІАГРАМИ ПРЕЦЕДЕНТІВ АБО ДІАГРАМИ ВИКОРИСТАННЯ (USE CASE DIAGRAMS)

3.1 Мета роботи

Освоїти правила аналізу предметної області та формування діаграм варіантів використання системи (діаграм прецедентів) , що описують функціональність, котра буде запропонована користувачам та розробникам системи, що проектується.

3.2. Теоретичні положення

3.2.1 Суть діаграми прецедентів

Діаграма прецедентів служить визначеній меті: документування дійових осіб-акторів (все, що знаходиться поза системою), варіантів використання (всього, що знаходиться в межах системи) і відносин (взаємозв'язків) цих компонентів.

Дійові особи (актори) - це всі, хто взаємодіє з розробляється системою. В UML дійова особа позначається фігуркою людини.

Існують три основні типи дійових осіб: користувачі системи; інші системи, які взаємодіють з даною, і час.

Прецеденти - *опис "на високому рівні" функцій, що надаються системою, тобто* ***вони ілюструють як можна використовувати систему***

На початок роботи над проектом виникає питання: "Як виявити прецеденти?" Прецеденти не залежать від реалізації та надають високорівневий опис системи. Обговоримо кожну частину цього визначення.

По-перше, прецеденти не залежить від реалізації. Прецеденти загострюють увагу на те, **що** повинна робити система, а не на те, **як** вона буде це робити.

По-друге, прецеденти дають високорівневу картину системи. Набір прецедентів повинен ясно і просто показати користувачеві всю систему Прецедентів не повинно бути багато, щоб користувачеві не доводилося знайомитися з великим обсягом документації тільки для того, щоб дізнатися, що робить ця система. У той же час набір прецедентів зобов'язаний надати

повний опис системи. У типових системах виявляється від 20 до 70 прецедентів (якщо ж їх буде 3000, то втрачається перевага простоти опису).

Нарешті, прецеденти повинні акцентувати те, що отримає користувач від системи. Кожен варіант використання повинен являти собою закінчену транзакцію між користувачем і системою, причому результат транзакції повинен мати важливе значення для користувача. Прецеденти іменуються за термінологією користувачів, а не технічними назвами, які не завжди зрозумілі виконавцям.

Для включення в діаграму вибрані прецеденти повинні задовольняти наступним критеріям:

• прецедент повинен описувати, ЩО треба робити , а не ЯК ;

• прецедент повинен описувати дії з точки зору ВИКОНАВЦЯ;

• прецедент повинен повертати виконавцю деяке ПОВІДОМЛЕННЯ;

• послідовність дій всередині прецеденту повинна являти собою один неподільний ланцюжок.

Суть діаграми прецедентів складається в наступному: проектована система представляється у вигляді безлічі сутностей або акторів, що взаємодіють з системою за допомогою так званих варіантів використання. При цьому актором (actor) або дійовою особою називається будь-яка сутність, що взаємодіє з системою ззовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка може служити джерелом впливу на систему, що моделюється так, як визначить сам розробник. У свою чергу, варіант використання (use case) служить для опису сервісів, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який чинять системою при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів з системою сервісів, які система надає актору. Ці і діаграми описують **функціональність,** яка буде надаватись користувачам системи, котра проектується. Представляються шляхом використання **прецедентів** та **акторів,** а також **відношень між ними**. Набір усіх прецедентів діаграми фактично визначає **функціональні вимоги**, за допомогою яких може бути сформульоване **технічне завдання**.

Визначимо поняття діаграми прецедентів:

Прецедент – опис процесів предметної області, що представляє собою варіант використання системи.

Діаграма прецедентів - структура всіх прецедентів та визначення відношень між ними та з зовнішніми об’єктами.

3.2.2.Змістовна сутність прецеденту

Системи створюється для роботи з користувачем (людиною, іншою системою,програмою тощо), вони не ізольовані. Користувач в UML визначений як АКТОР - людина або програма, яка використовує систему в своїх цілях, причому кожен актор очікує, що вона буде вести себе певним, цілком передбачуваним чином. ПРЕЦЕДЕНТ (Use case) специфікує поведінку системи або її частини і являє собою опис безлічі послідовностей дій (включаючи варіанти), що виконуються системою для того, щоб актор міг отримати певний результат.

За допомогою прецедентів можна описати поведінку розроблюваної системи, не визначаючи її реалізацію. Вони дозволяють :

- досягти сукупної роботи та взаєморозуміння між розробниками, експертами та кінцевими користувачами продукту,
- перевірити архітектуру системи в процесі її розробки.

3.2.3.Шаблони опису прецедентів (сценаріїв використання)

Немає ніякого стандартного шаблону для документації детальних сценаріїв використання. Існує багато конкуруючих схем, але найкраще використовувати ті шаблони, які краще підходять для проекта. Є, однак, основні моменти, на які варто звернути увагу:

Ім'я прецеденту (сценарію). Варто писати в форматі дієслово-іменник (наприклад, Розрахувати сітьову модель або Розрахунок сітьової моделі). Воно повинно описувати досяжну мету (наприклад, Розрахунок сітьової моделі краще, ніж Розраховується сітьова модель), і має пояснювати сенс сценарію використання.

Ціль. Ціль коротко описує те, чого користувач має намір досягти з цим сценарієм використання

Актори (дійова особа). Актор це хтось або щось поза системою і впливає на систему або знаходиться під її впливом. Актор може бути людиною, пристроєм, іншою системою або підсистемою, або часом. Людина в реальному світі може бути представлений декількома акторами, якщо у них є кілька різних ролей і цілей по відношенню до системи. Вони взаємодіють з системою і роблять над нею деякі дії

Попередні умови (перед-умови). Варто визначити всі умови, при яких виконання сценарію має сенс. Таким чином, якщо система не знаходиться в стані, описаному в попередніх умовах, сценарій не діє.

Опис. Надає загальну характеристику призначення прецеденту для розв'язання поставленої задачі. Опис прецеденту – це текстовий документ, що описує сценарії застосування системи. Прецеденти описуються в різних форматах в залежності від етапу створення системи. Наведемо в якості прикладу форму опису прецедентів в табличній формі у вигляді двох колонок. В таблицях 3.1 та 3.2 наведені приклади зжатого опису прецедента (прецедент високого рівня). При описі прецедента високого рівня в короткій формі визначається назва, виконавець, тип і надається у вигляді одного абзацу опис дій.

Таблиця 3.1

ПРЕЦЕДЕНТ ВИСОКОГО РІВНЯ «Розрахунок сітьової моделі»

Прецедент	Розрахунок сітьової моделі
Виконавець	Відділ підготовки виробництва
Тип	Основний
Опис	Вводиться опис сітьової моделі, проводиться розрахунок всіх часових показників.

Таблиця 3.2

ПРЕЦЕДЕНТ ВИСОКОГО РІВНЯ – «Визначення потреби в ресурсі»

Прецедент	Формування варіанту календарного плану
Виконавці	Відділ ОПР
Тип	Головний
Опис	Відділ ОПР формує варіанти створення календарного плану будівництва, надає запит про потрібність в ресурсі для виконання роботи визначеного обсягу та визначає тривалість робіт

В таблиці 3.3. наведений приклад представлення розгорнутого прецеденту, що передбачає детальний стиль опису, де визначаються поступові кроки реалізації , альтернативні варіанти розвитку сценарію. РОЗГОРНУТІ ПРЕЦЕДЕНТИ мають форму діалогу між виконавцем та системою.

Таблиця 3.3

РОЗГОРНУТИЙ ІДЕАЛЬНИЙ ПРЕЦЕДЕНТ «Розрахунок сітьової моделі»

1. Прецедент	Розрахунок сітьової моделі
Виконавець (актор)	Відділ підготовки виробництва
Ціль	Визначення часових параметрів по об’єкту в цілому та по кожній роботі
Опис	Вводиться опис сітьової моделі, проводиться розрахунок всіх часових параметрів: ...
Тип	Основний, ідеальний
Посилання	Функції:
Перед-умови	Визначені топологія та час виконання всіх робіт
Типовий хід подій	
Дії виконавця	Відгук системи
1.Виклик визначеної сітьової моделі(СМ)	Надається опис СМ
2.Замовлення розрахунку ранніх термінів виконання кожної роботи	Розрахунок $T_{кр,}$
3.Замовлення розрахунку пізніх термінів виконання кожної роботи	Розрахунок$\{ Tij^{n.з.} \}$
…	...

3.2.4 Моделювання прецедентів (варіантів використання)

Моделювання прецедентів традиційно поділяють на бізнес-моделювання (діаграми бізнес-прецедентів) і системне моделювання (діаграми прецедентів). У бізнес-моделюванні робиться акцент на саму організацію, тоді як в системному моделюванні основна увага націлене на систему, що розроблюється. Поняття, які використовуються в діаграмах прецедентів при бізнес моделюванні і системному моделюванні, наведені в таблиці 3.4.

Таблиця 3.4

Поняття моделювання діаграм прецедентів

Поняття	Бізнес-моделювання	Системне моделювання
Варіант використання (прецедент, Use case)	Описує особливості бізнесу	Описує дії системи в контексті бізнесу
Дійова особа (суб'єкт, роль, Actor)	Зовні по відношенню до організації	Зовнішнє по відношенню до системи (але може належати організації)
Співробітник (Business worker)	Належить організації	Не використовується

Діаграма прецедентів служить визначеній меті:документування дійових осіб (все, що знаходиться поза системою), варіантів використання (всього, що знаходиться в межах системи) і відносин (взаємозв'язків) цих компонентів.

Аналіз прецедентів дозволить клієнтові побачити те, що він отримає від системи, та погодити межі дії системи на самому початку розробки проекту.

3.2.5 Відношення

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

- асоціації (англ. *association relationship*);
- включення (англ. *include relationship*);
- розширення (англ. *extend relationship*);
- узагальнення (англ. *generalization relationship*)

Асоціації (асоціативні відношення) (*association elationship*) служать для показу взаємозв'язків між прецедентом і актором. Це одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

В діаграмах прецедентів асоціація служить для позначення специфічної ролі актора при його взаємодії з окремим варіантом використання. Іншими словами, асоціація специфікує семантичні особливості взаємодії акторів і варіантів використання в графічній моделі системи. На діаграмі варіантів використання, так само як і на інших діаграмах, ставлення асоціації позначається суцільною лінією між актором і прецедентом(рис.3.1.).

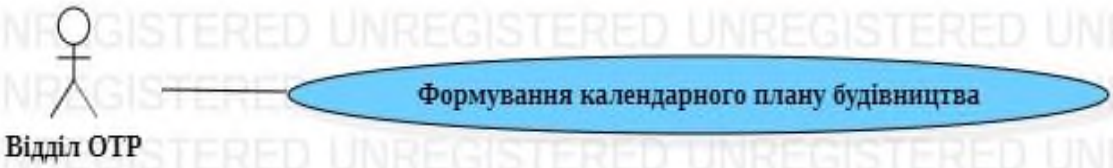


Рис. 3.1. Приклад асоціативних відношень

Ця лінія може мати деякі додаткові позначення, наприклад, ім'я та кратність.

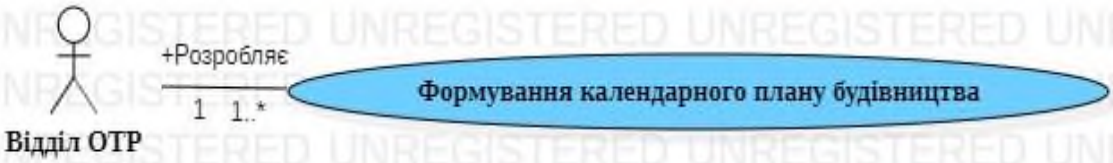


Рис. 4.3 Асоціація з додатковими позначеннями

Прецеденти позначаються дуже простим чином - у вигляді еліпса, всередині якого зазначена його назва.

Прецеденти і актор з'єднуються за допомогою ліній. Інколи на одному з кінців лінії зображують стрілку, причому спрямована вона на того, у кого запитують сервіс, іншими словами, чиїми послугами користуються (рис.3.2).

Рис.3.2. Спрямована асоціація

Між прецедентами розглянемо головні відносини трьох типів: включають, розширюють, узагальнюють.

За винятком включаючих і розширючих відносин, прецедент зобов'язаний ініціюватися дійовою особою.

Включаючі відносини. Включаючі відносини дозволяють одному прецеденту надавати функції іншим прецедентів. Такі відносини необхідні в двох випадках. По-перше, якщо кілька прецедентів мають багато в чому ідентичну функціональність, її можна розділити на окремі прецеденти, які включаються (входять до складу) загального прецеденту. По-друге, що включають відносини, що допоможуть в ситуації, коли один з прецедентів володіє незвично широкою функціональністю. У цьому випадку в моделі з'являться два менших прецедентів. Включаючі відносини відзначаються пунктирною стрілкою зі словом «include» (рис. 3.2).

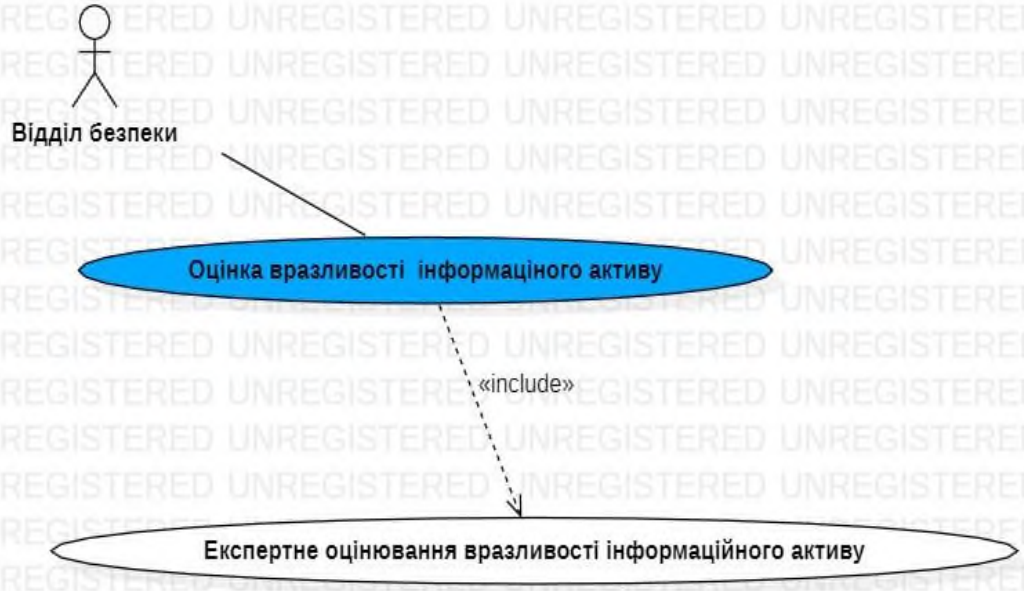


Рис. 3.2. Включаючі відносини

Включаючі відносини припускають, що один прецедент **завжди** користується функціональністю іншого прецеденту. Так при реалізації прецеденту «Оцінка вразливості інформаційного активу» базовий прецедент завжди звертається до реалізації і користується результатами роботи прецедента «Експертне оцінювання вразливості інформаційного активу».

На діаграмі прецедентів відношення включення зображують у вигляді залежності зі стереотипом include (пунктирна стрілка спрямована від базового прецеденту до того, що включається. На рис. 3.3 наведений приклад побудови діаграми прецедентів системи тестування знань студентів, що включає включаючі відносини.

Розширюючі відносини. На відміну від включаючих розширюючі відносини дозволяють одному варіанту використання доповнити при необхідності свою функціональність за рахунок іншого. Ці відносини схожі на ті, що включають, оскільки обидва типи відносин виділяють деякі функції в

окремому варіанті використання. В UML розширюючі відносини відзначаються пунктирною стрілкою зі словом «extend» (рис. 3.4) з напрямом від прецедента, що включається до того, що його включає.

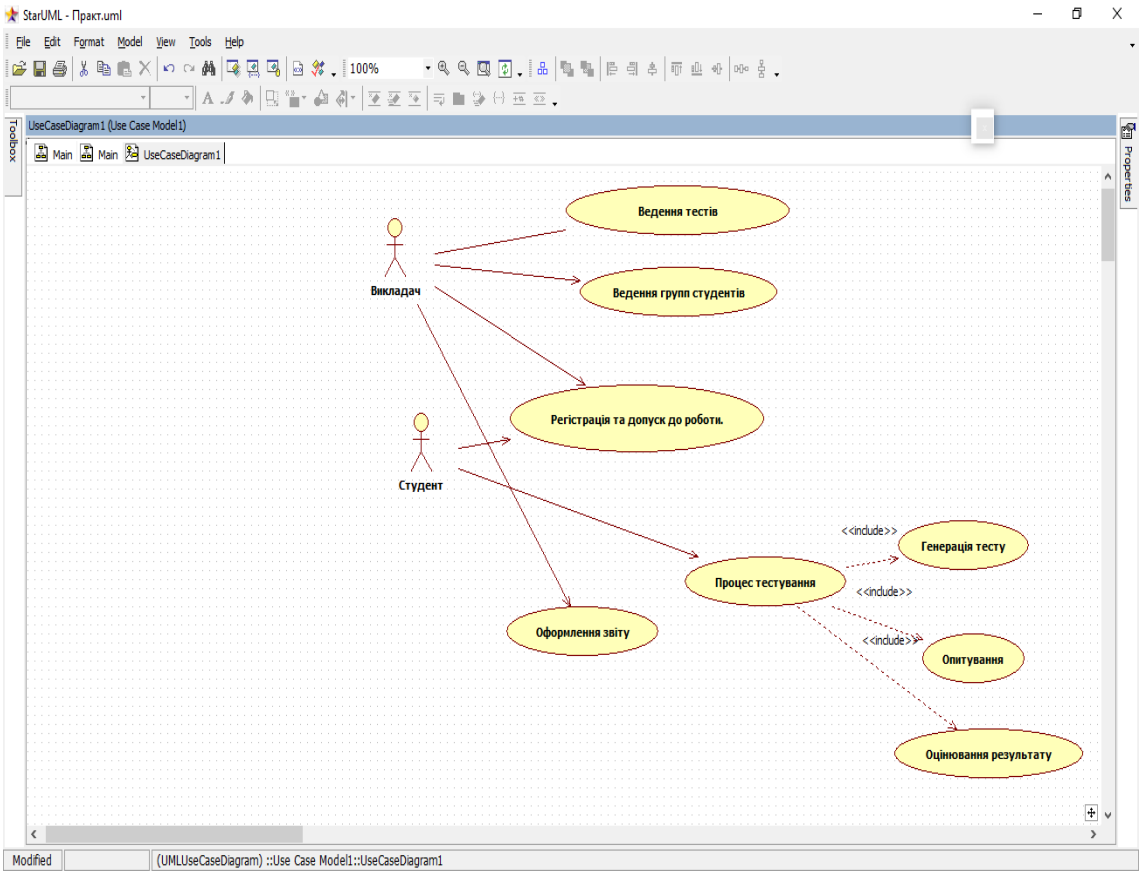


Рис. 3.3. Діаграма прецедентів системи тестування знань студентів.

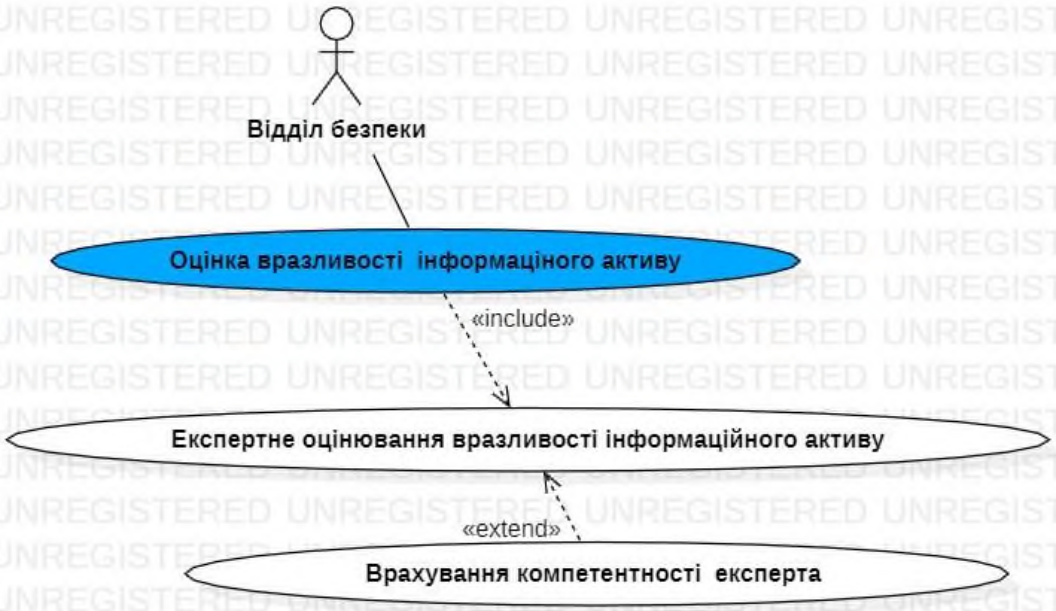


Рис. 3.4 Включаючі та розширюючі відносини

У цьому прикладі прецедент "Врахування компетентності експерта»" розширює прецедент "Експертне оцінювання вразливості активу". Під час виконання прецеденту "Експертне оцінювання вразливості активу " прецедент " Врахування компетентності експерта " виконується тоді і тільки тоді, коли особа, що приймає рішення (ОПР), вважає необхідним при узагальнені експертних оцінок враховувати вагу оцінки окремого експерта. Якщо ОПР так не вважає, прецедент «Врахування компетентності експерта» не підключається до процесу оцінювання».

Відношення узагальнення – визначає зв'язок, коли два й більше актори мають загальні властивості, тобто взаємодіють з тією самою множиною варіантів використання однаковим чином. Така спільність властивостей і поведінки представляється у вигляді відношення узагальнення з іншим, можливо, абстрактним актором, що моделює відповідну сукупність ролей.

Графічно відношення узагальнення позначається суцільною лінією зі стрілкою у формі не зафарбованого трикутника, що вказує на батьківський варіант використання (рис. 3.5). Ця лінія зі стрілкою має спеціальну назву - стрілка-узагальнення.

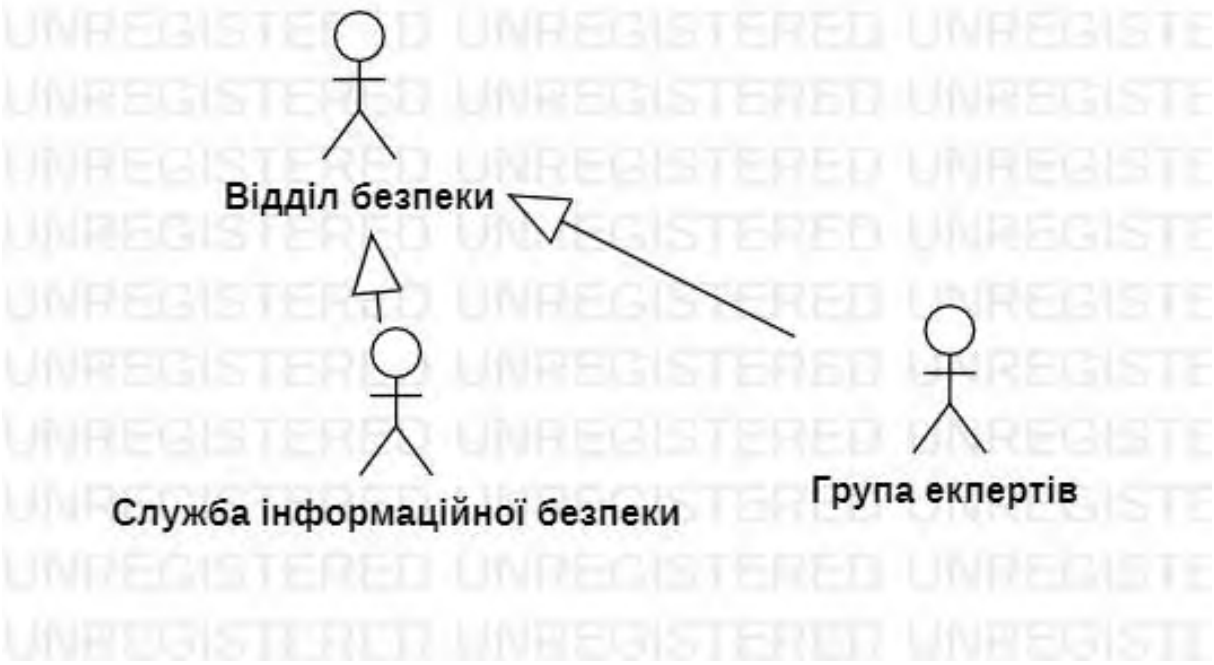


Рис.3.5 Відношення узагальнення для акторів

У наступному прикладі(рис. 3.6) відношення узагальнення вказує на те, що прецедент (варіанти використання) "Оцінка ризику інформаційної безпеки"має такі варіанти реалізації (такі нащадки): "Оцінка ризику на основі статистичних даних", "Оцінка ризику на основі експертного оцінювання" та "Оцінка ризику на основі аналізу об’єктів-аналогів" - спеціальні випадки

варіанта використання "Оцінка ризику інформаційної безпеки". Варто підкреслити, що нащадок успадковує всі властивості поведінки свого батька, а також може мати додаткові особливості.

Рис.3.6 Відношення узагальнення для прецедентів

3.3. Побудова діаграми прецедентів в StarUML

1. **Створення нового проекту.** Для виконання комплексу практичних і лабораторних робіт необхідно в системі створити окремий проект. Для створення нового проекту в вікні обирається вкладка “File -> New Project”
2. **Для створення діаграми прецедентів** треба у вікні вибрати вкладку ‘Model-> Use Case Diagram (рис 3.7).

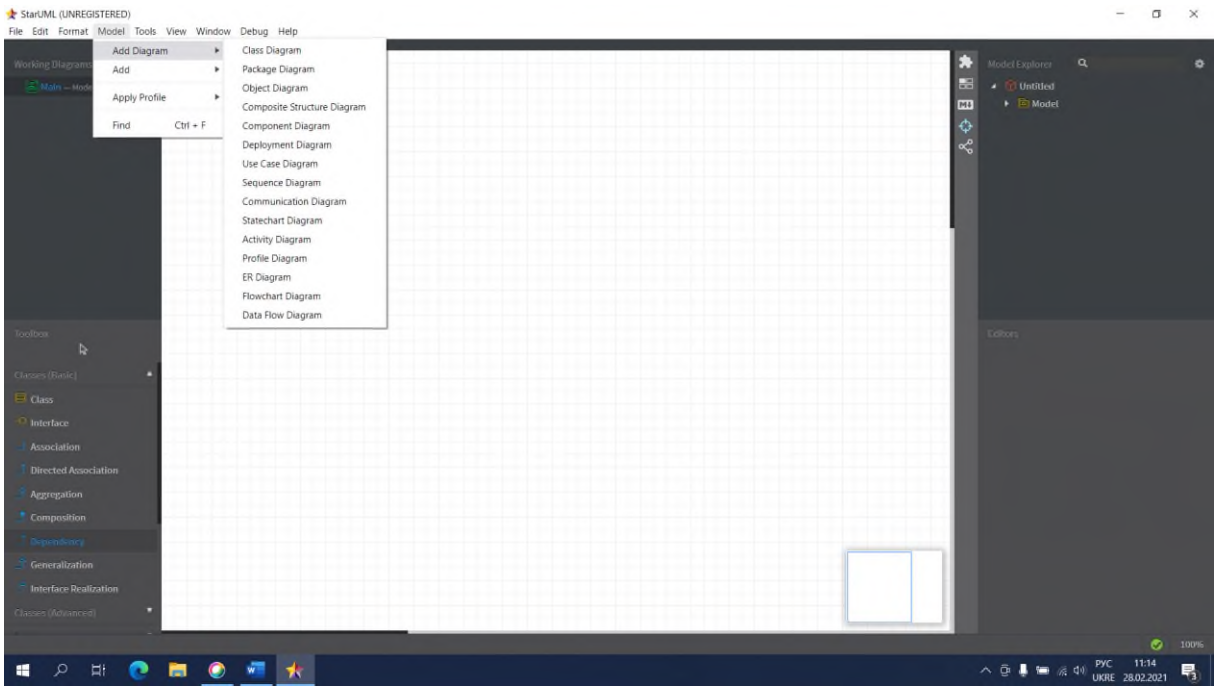


Рис. 3.7 Створення Use Case Diagram

3. **Створення складової «Актор».** Для включення в діаграму актора треба натиснути кнопку [Toolbox]-> [Use Case Diagram]-> [Actor]. На діаграмі

визначте позицію, де ви хочете розмістити фігуру актора (рис.3.8). Побудову діаграму прецедентів ми будемо ілюструвати на конкретному прикладі, що стосується побудови системи тестування.

Так, на діаграмі визначаємо двох акторів – «Студент» , «Викладач». Назви можна встановлювати на англійській або український мові. Зупиняємось на українській

Актора можна відобразити на діаграмі в різних режимах. Для вибору режиму треба обрати пункт меню [Format]-> [Stereotype Display]-> [Decoraition] при виборі декоративного режиму або -> [None] (рис. 3.8.,3.9)

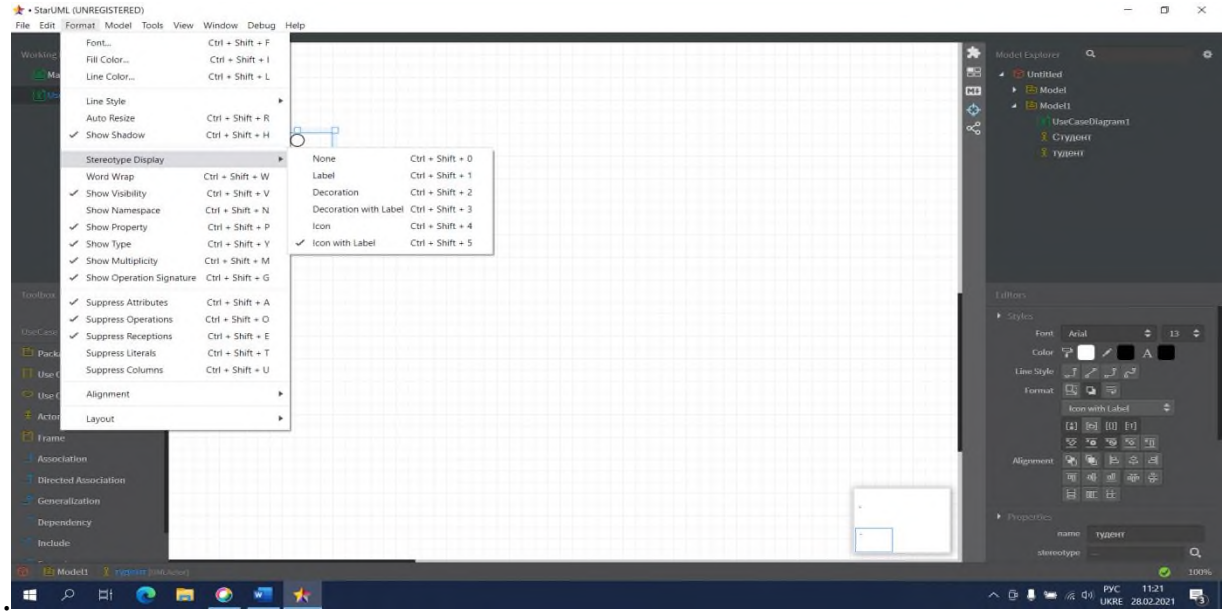


Рис. 3.8 Вибір режиму

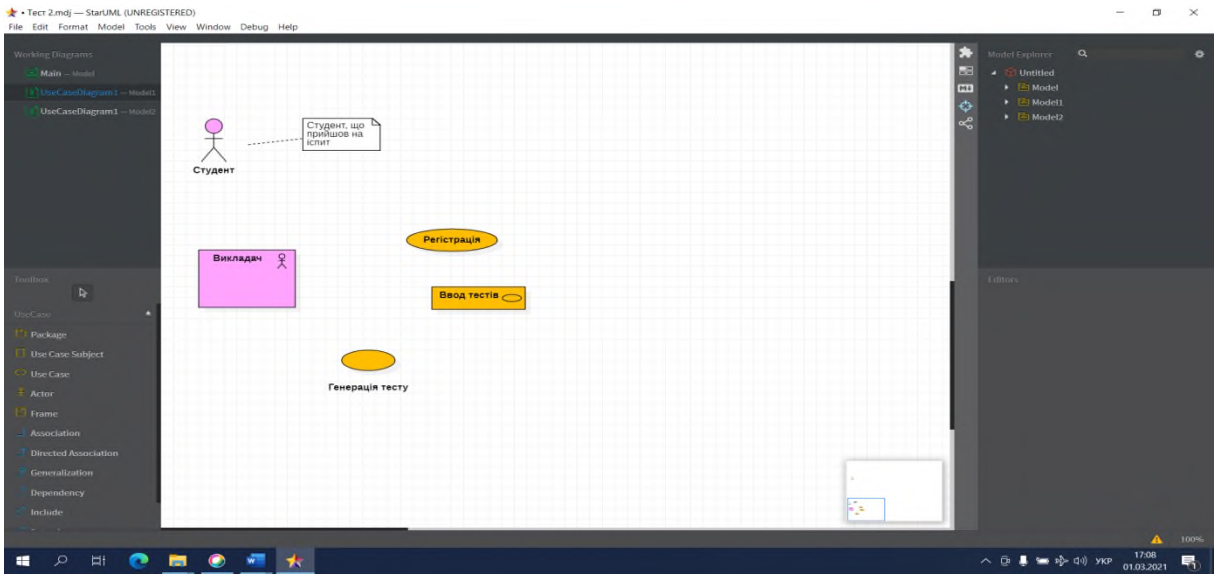


Рис.3.9. Приклади представлення прецедентів та акторів в різних режимах

В подальшому ми будемо застосовувати текстовий режим представлення акторів.

4.Створення прецедентів. Натисніть кнопку [Toolbox]-> [Use Case] ,визначте місце на діаграмі, де треба розмістити прецедент, надайте назву прецеденту (рис.3.10.)

Прецедент може бути зображений в різних форматах. Щоб обрати формат представлення прецеденту, виберіть пункт меню [Format]-. [Stereotype Display]. Так в нашому прикладі ми створюємо наступні прецеденти (рис. 3.11):

- Формування інформаційної платформи.
- Регістрація та допуск до роботи.
- Проведення тестування.
- Оцінка результатів тестування.
- Оформлення результатів тестування.

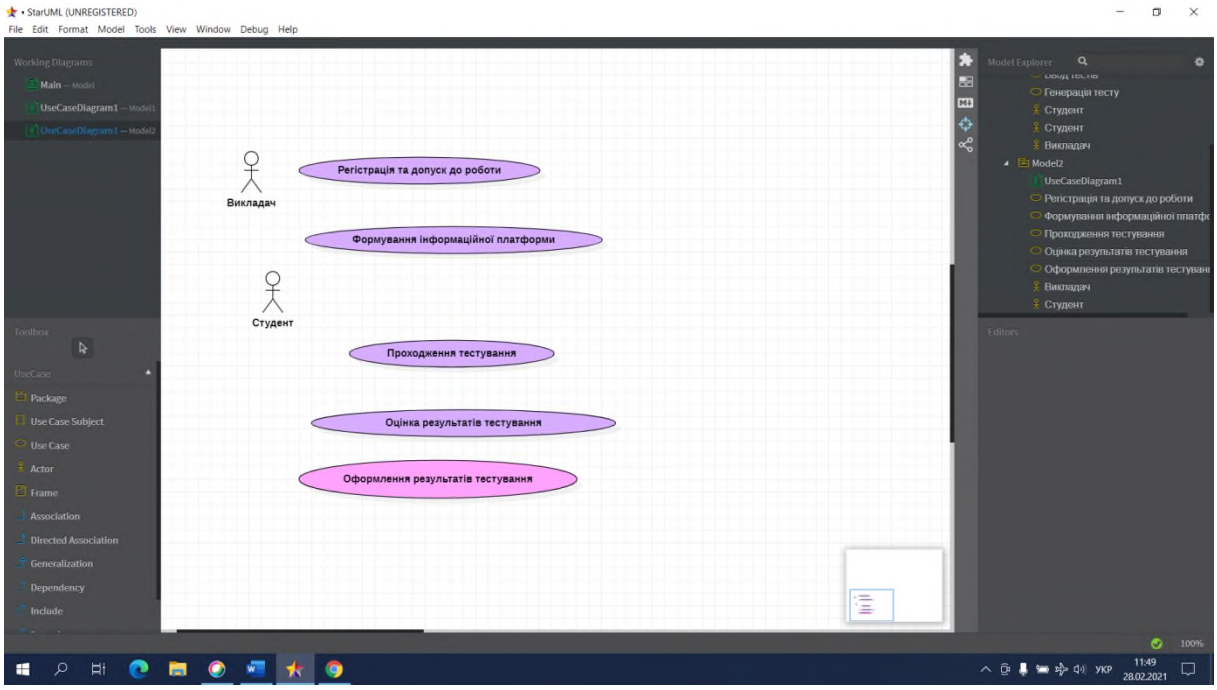


Рис.3.11. Приклад створення прецедентів

5.Створення асоціацій. Асоціація – це відношення між актором і прецедентом. Щоб відтворити асоціацію (не спрямовану) треба натиснути кнопки [Toolbox]-> [Use Case] -> [Association] і провести лінію від Актору до визначеного Прецеденту. На рис. 3.12 встановлені неспрямовані асоціації для акторів і процесів (рис. 3.12). Є альтернатива - створення спрямованої асоціації. Для цього треба натиснути кнопки [Toolbox]-> [Use Case] -> [Direct Association].

6.Створення відношень включення.Для створення відносин включення треба натиснути кнопки [Toolbox]->[Use Case] ->[Include].

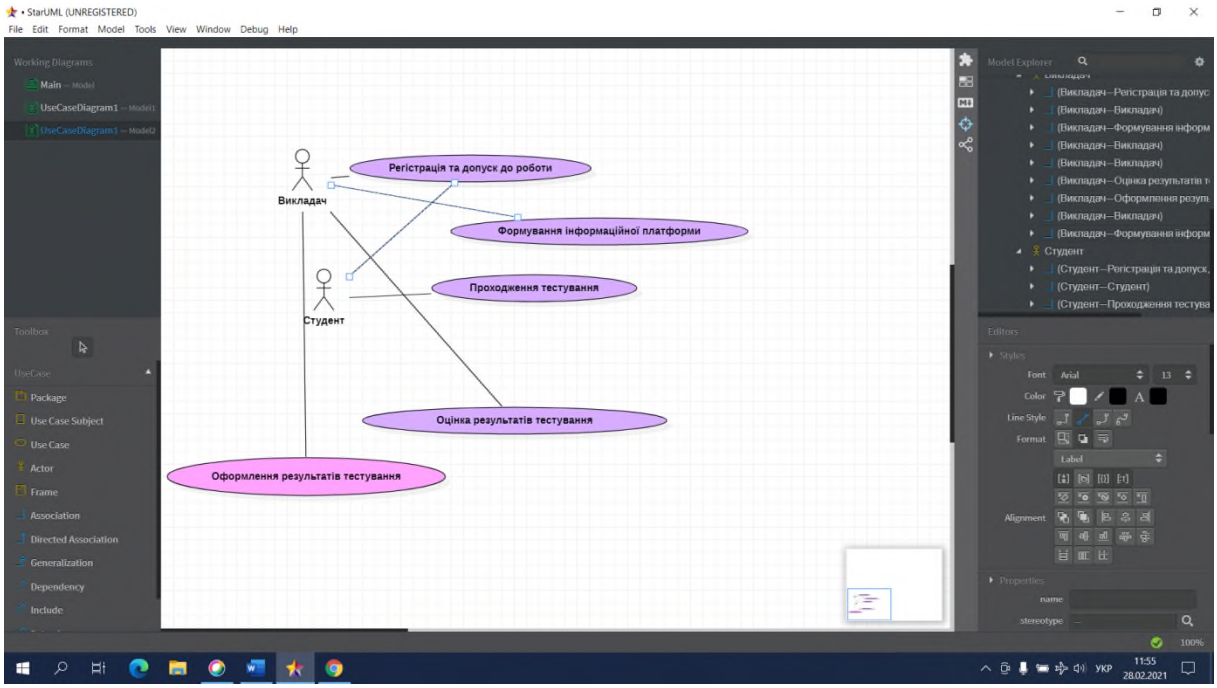


Рис. 3.12. Приклад неспрямованої асоціації

Для нашого прикладу ми визначили доцільним передбачити розділення узагальненого прецеденту «Регістрація та допуск до роботи» на два складові прецеденти: «Регістрація користувача» та «Управління доступом», узагальгнені прецеденти «Формування інформаційної платформи» та «Оцінка результатів тестування» включають прецедент «Допуск до роботи» (рис. 3.13.) Лінія проводиться від прецеденту, що включає, до тих, які включаються.

7. Створення відносин розширювання. Для створення відносин розширювання треба натиснути кнопки [Toolbox]-> [Use Case] -> [Extend]. Прикладом для нашої предметної області розширення можливостей прецеденту «Проведення тестування» є прив’язка для нього двох альтернативних прецедентів «Генерація тестів » та «Формування тестів з питань» (рис. 3.14).

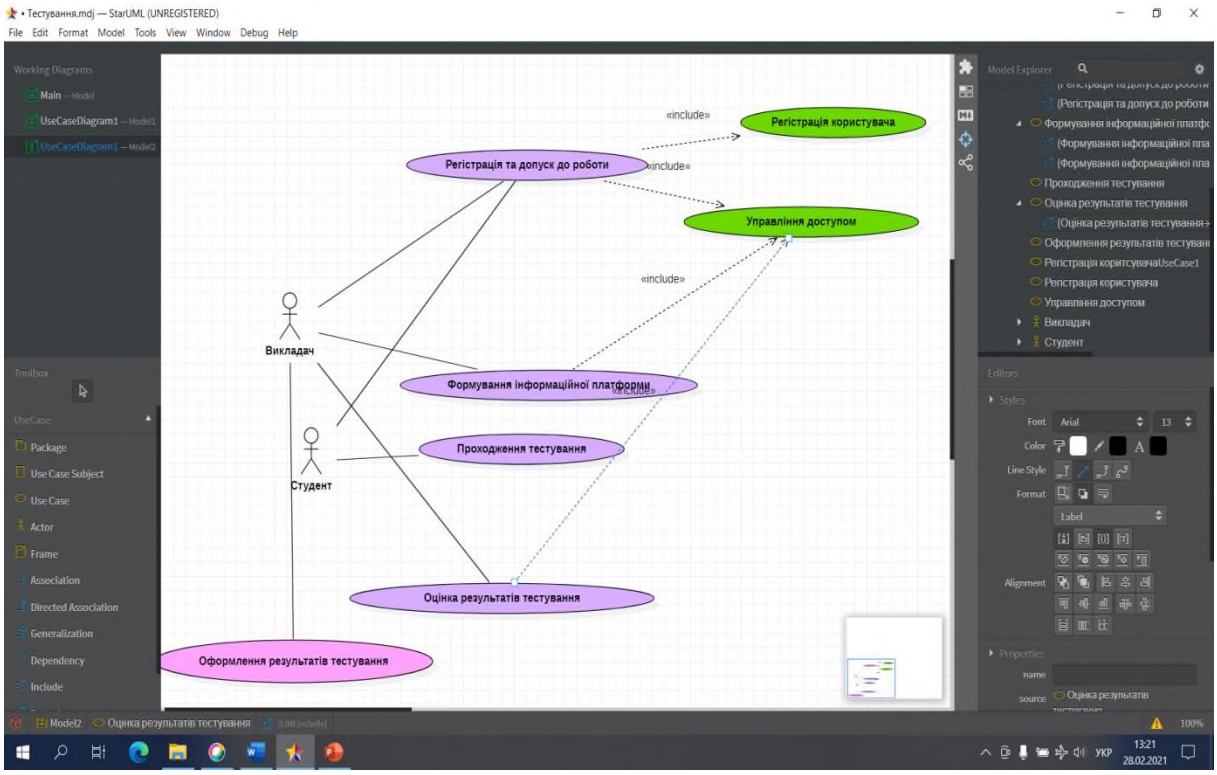


Рис. 3.13 Відносини включення

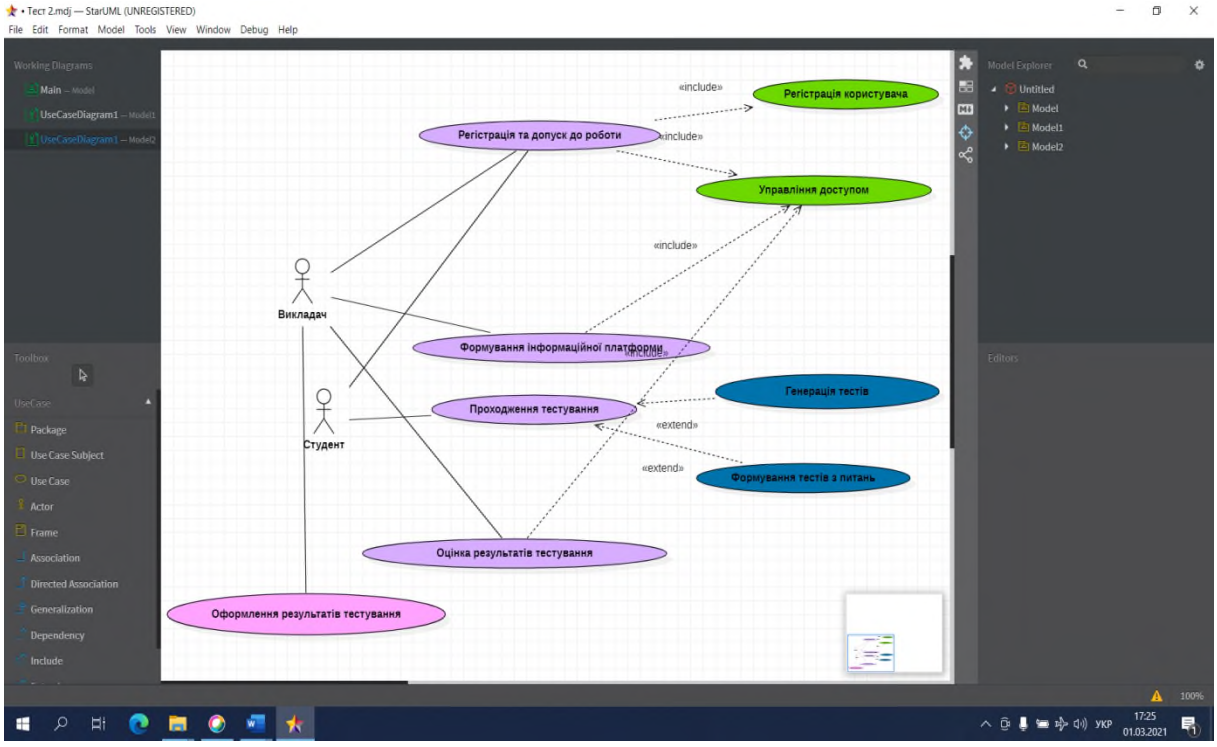


Рис. 3.14. Встановлення відносин розширювання

8.Створення відносин узагальнення. Для створення відносин узагальнення треба натиснути кнопки [Toolbox]-> [Use Case] ->[Generalization]. Так, наприклад, якщо в системі закладено прецедент «Формування тестів», як

такий, що включається до прецеденту «Проведення тестування» і він має два варіанта реалізації :перший «Генерація тестів » (передбачає ведення готових тестів), другий – «Формування тесту з питань», на основі яких буде застосований підхід формування тестів на основі закладених питань системою, то це потребує змін в діаграмі прецедентів з застосуванням відносин узагальнення (рис. 3.15).

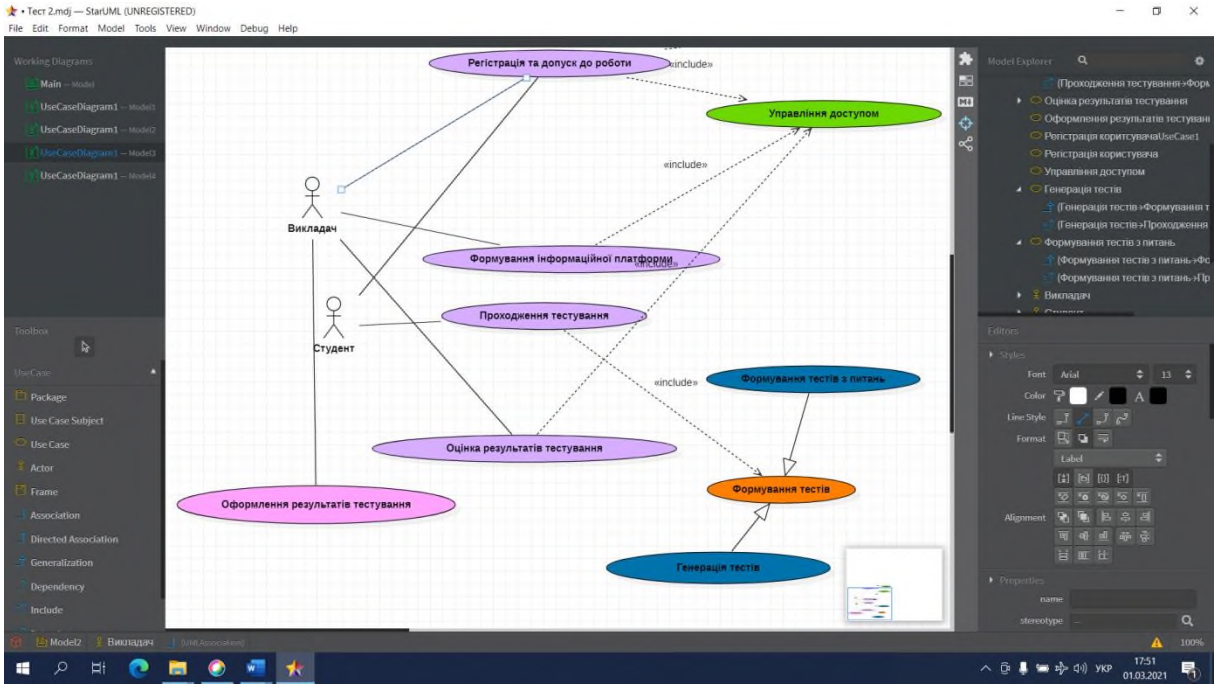


Рис. 3.15. Встановлення відносин узагальнення

3.4.Контрольні питання до роботи

1.Призначення діаграми прецедентів.
2.Мета розробки діаграми прецедентів.
3. Елементи діаграми прецедентів. Актори.
4. Елементи діаграми прецедентів . Асоціація.
5.Елементи діаграми прецедентів . Відносини включення.
6.Елементи діаграми прецедентів . Відносини розширювання.
7.Елементи діаграми прецедентів . Відносини узагальнення .

4.ЛАБОРАТОРНА РОБОТА №2.
ДІАГРАМА КЛАСІВ(CLASS DIAGRAM)

4.1 Мета роботи

Мета роботи - освоєння студентами правил розробки статичної структури моделі системи в термінології класів об'єктно-орієнтованого аналізу, проектування та програмування. Ставиться задача побудови діаграми класів як частини логічної моделі системи, що представляє її статичну картину. Для кожної системи в роботі перед студентом може бути поставлене завдання побудови не однієї, а кількох діаграм класів – окремо для кожного прецедента або для окремого сценарію функціонування системи.

4.2. Теоретичні положення

4.2.1.Діаграма класів (class diagram)

Центральне місце в об’єктно-орієнтованому аналізі і проектуванні систем займає розробка моделі системи в виді діаграми класів. Елементи діаграм класів пов’язані різними структурними зв’язками. Опрацьовуючи при проектуванні статичне представлення системи, зазвичай використовують діаграми класів з однією із трьох цілей:

1. Для моделювання словника системи. Воно допомагає вирішити питання стосовно того, які абстракції стануть предметом уваги системи, а які виходять за її межі. Використовуємо діаграми класів, щоб специфікувати ці абстракції та їх призначення.
2. Для моделювання простих кооперацій. Кооперація – це угрупування класів та інших елементів, які працюють для формування деякої спільної поведінки при реалізації прецеденту або одного з сценаріїв роботи системи.
3. Для моделювання логічної схеми бази даних.

4.2.2.Клас

Клас у мові UML використовується для позначення множини об'єктів, які мають однакову структуру, поведінку і відносини з об'єктами з інших класів. Графічно клас зображується у виді прямокутника, який додатково може бути поділений горизонтальними лініями на секції. У цих секціях можуть вказуватися ім'я класу, атрибути (змінні) і операції (методи) (рис.4.1).

Рис.4.1 Приклади графічного представлення класу

Обов'язковим елементом позначення класу є його ім'я. На початкових етапах розробки діаграми (концептуальний рівень) окремі класи можуть позначатися простим прямокутником з зазначенням тільки імені відповідного класу. У процесі розробки (рівень проектування) окремих компонентів діаграми описи класів доповнюються атрибутами і операціями. Передбачається, що кінцевий варіант (рівень реалізації) діаграми містить найбільш повний опис класів, який складається з трьох секцій. Іноді в позначеннях класів використовується додаткова четверта секція, в якій приводиться семантична інформація довідкового характеру або явно вказуються виняткові ситуації. Навіть якщо секція атрибутів і операцій пусті, в позначенні класу вони виділяються горизонтальною лінією, щоб відразу відрізнити клас від інших елементів мови UML.

Ім'я класу повинне бути унікальним в межах пакету, який описується деякою сукупністю діаграм класів (розробка діаграм класів для різних прецедентів). Воно вказується в першій верхній секції прямокутника. Ім'я класу записується по центру секції імені жирним шрифтом і повинно починатися з великої букви. Рекомендується як імена класів використовувати іменники, які записуються без прогалин. Саме імена класів утворюють словник

предметної області при об’єктно-орієнтованому аналізі та проектуванні (ООАП)

У другій зверху секції прямокутника класу записуються його атрибути або властивості. Кожному атрибуту класу відповідає окрема стрічка

Ім'я атрибуту представляє собою стрічку тексту, яка використовується як ідентифікатор відповідного атрибуту і тому повинна бути унікальною в межах даного класу. Ім'я атрибуту є єдиним обов'язковим елементом синтаксичного позначення атрибуту.

У третій зверху секції прямокутника записуються операції або методи класу. Операція (operation) представляє собою деякий сервіс, який надає кожен екземпляр класу на певну вимогу. Сукупність операцій характеризує функціональний аспект поведінки класу.

4.2.3.Відношення між класами

Крім внутрішньої будови або структури класів на відповідній діаграмі вказуються різні відношення між класами. При цьому сукупність типів таких відношень фіксована в мові UML. Базовими відношеннями або зв'язками є:

- Відношення залежності.
- Відношення асоціації.
- Відношення агрегацій.

Відношення залежності в загальному випадку вказує деяке семантичне відношення між двома елементами моделі або двома множинами таких елементів, яке не є відношенням асоціації, узагальнення або реалізації.

Відношення залежності використовується в такій ситуації, коли деяка зміна одного елемента моделі може потребувати зміну іншого залежного від неї елемента моделі.

Відношення залежності графічно зображується пунктирною лінією між відповідними елементами зі стрілкою на одному з її кінців. На діаграмі класів дане відношення зв'язує окремі класи між собою, при цьому стрілка направлена від класу-клієнта залежності до незалежного класу або класу-джерела (рис.4.2).

Відношення асоціації відповідає наявності деякого відношення між класами. Дане відношення позначається суцільною лінією з додатковими спеціальними символами, які характеризують окремі властивості конкретної асоціації. Як додаткові спеціальні символи можуть використовуватися ім'я асоціації, а також імена і кратність класів-ролей асоціації. Ім'я асоціації є

необов'язковим елементом її позначення. Якщо воно задане, то записується з великої букви поруч з лінією відповідної асоціації.

Асоціація - взаємна залежність між об’єктами різних класів, кожен з яких є рівноправним членом залежності. Для асоціації може позначатися кількість екземплярів об’єктів кожного класу, які беруть участь у зв’язку (0 - якщо жодного, 1 - якщо один, * - якщо багато). Можуть вказуватися мінімальна й максимальна кількість, наприклад, 0,1...* означає, що на відповідному кінці асоціації може не бути жодного екземпляра, бути один або багато.

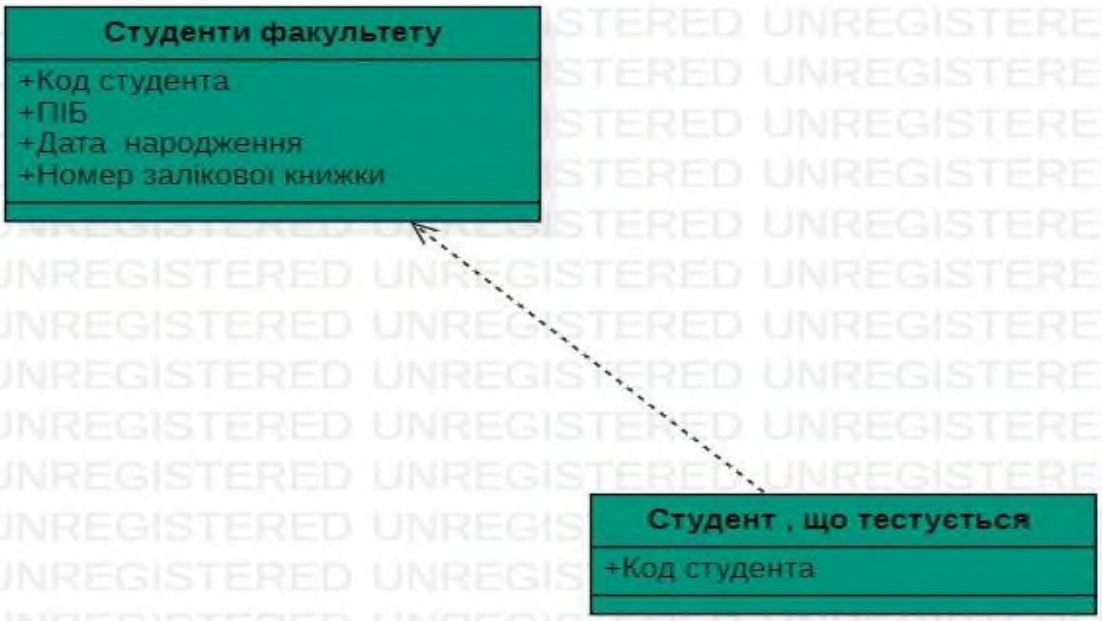


Рис. 4.2 Приклад застосування відношення залежності

Рис. 4.3. Приклади типів відношень

Спеціальною формою або частковим випадком відношення асоціації є **відношення агрегації**, яке, в свою чергу, також має спеціальну форму - **відношення композиції**.

Відношення агрегації може бути між декількома класами у тому випадку, якщо один із класів представляє собою деяку сутність, що включає в себе як складові частини інші сутності.

Графічно відношення агрегації зображується суцільною лінією, один з кінців якої представляє собою не зафарбований всередині ромб. Цей ромб вказує на той з класів, який являє собою "ціле". Решта класи є його "частинами" (рис. 4.4).

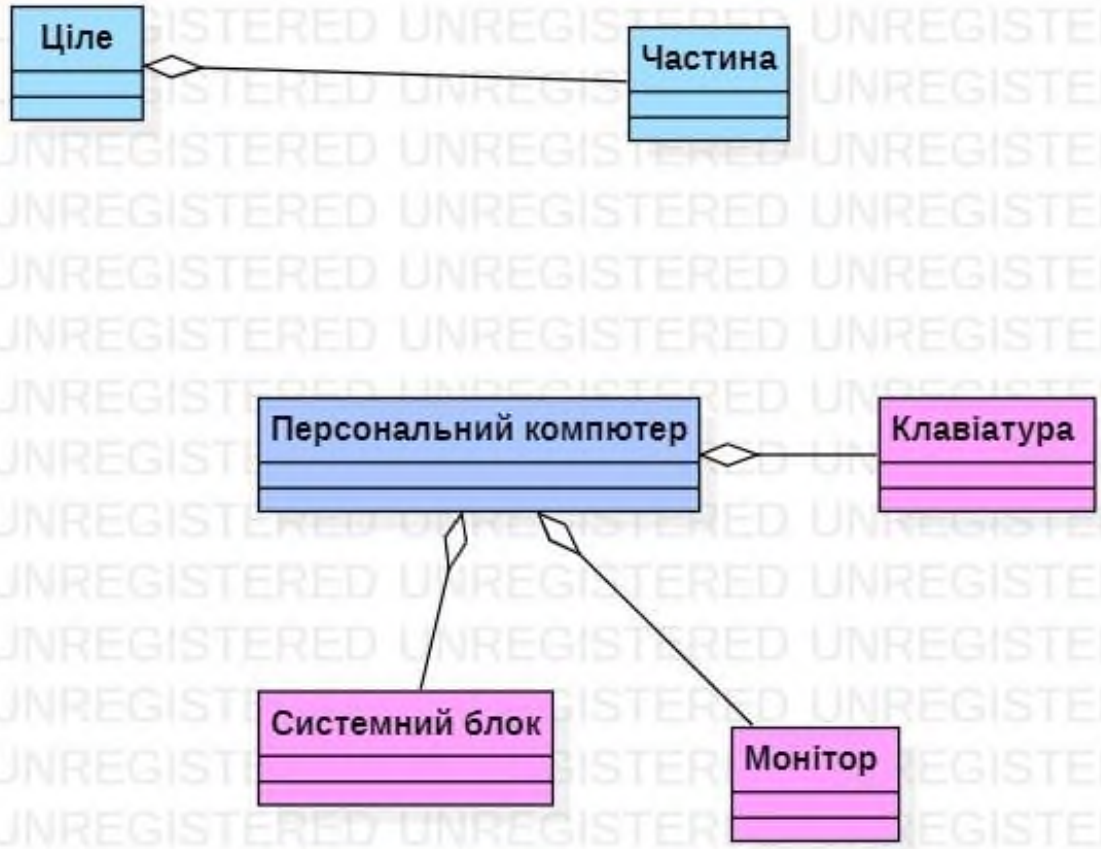


Рис. 4.4. Відношення агрегації

Відношення композиції, як вже згадувалося раніше, є окремим випадком відносини агрегації. Це відношення служить для виділення спеціальної форми відносини "частина-ціле", при якій складові частини в деякому сенсі знаходяться всередині цілого. Специфіка взаємозв'язку між ними полягає в тому, що частини не можуть виступати у відриві від цілого, тобто зі знищенням цілого знищуються і всі його складові частини(рис. 4.5).

Рис. 4.5. Відношення композиції

4.3.Побудова діаграми класів в StarUML

1. **Відкриття свого проекту.** Для виконання зазначеного циклу лабораторних робіт необхідно в системі відкрити або створити окремий проект. Для створення нового проекту необхідно в вікні вибрати вкладку “File -> New Project”
2. **Для створення діаграми класів** треба у вікні вибрати вкладку ‘Model-> Class Diagram(рис.4.6).

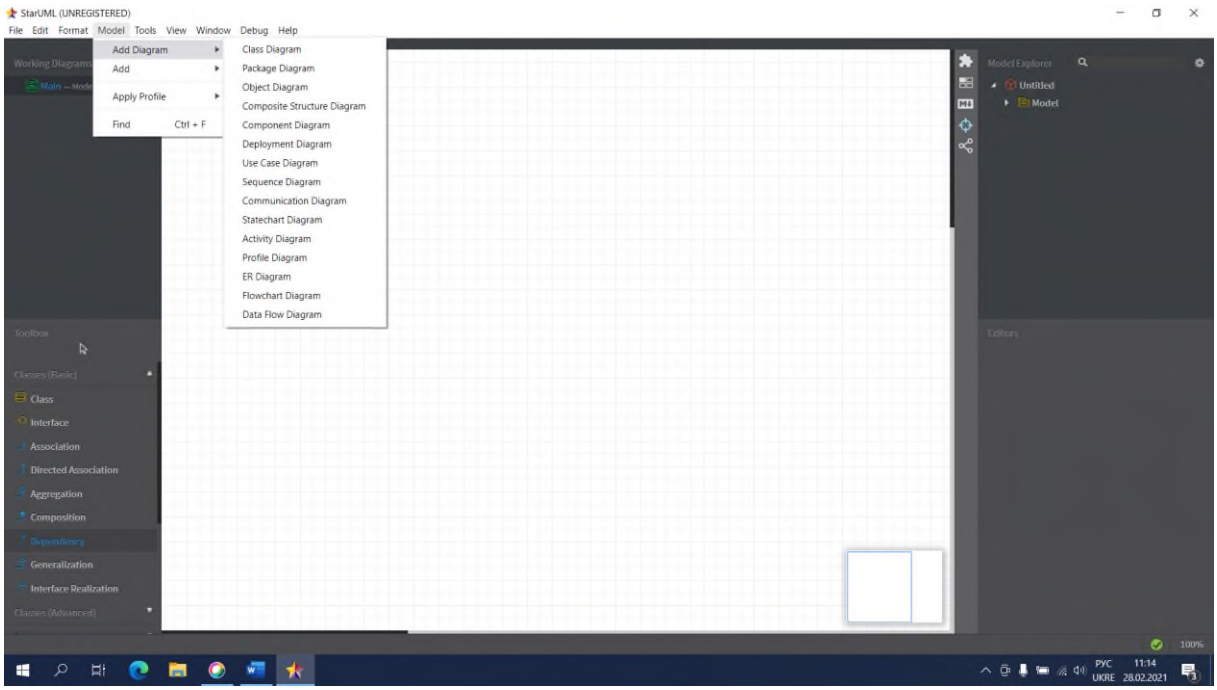


Рис4.6. Створення Class Diagram

Щоб створити клас (рис4.7) треба:

1) Клацніть кнопку [Toolbox] -> [Class] ->
2) Клацніть діаграму в позиції, куди потрібно помістити клас.

3) На діаграмі буде створений новий клас і буде відкритий його гарячий діалог.
4) У гарячому діалозі, введіть ім'я класу, і натисніть клавішу [Enter].

Рис.4.7. Створення класу

3. Додання атрибуту.

Рекомендовані методи додавання атрибута до класу:

- з використанням гарячого діалогу;
- з використанням навігатора моделі.

У разі використання гарячого діалогу:

1) Зробіть подвійне клацання на класі.
2) Натисніть кнопку [Add Attribute] в гарячому діалозі, і атрибут буде додано (рис.4.8.).

Рис 4.8. . Використання гарячого діалогу

У разі використання навігатора моделі:

1. Виберіть клас в навігаторі моделі.
2. Клацніть правою кнопкою миші обраний клас, виберіть пункт [Add] -> [Attribute] в контекстному меню (рис.4.9).

4.Додання операції.

Додати операцію в клас можна наступними шляхами:

- з використанням гарячого діалогу;
- з використанням навігатора моделі.

У разі використання гарячого діалогу:

1. Двічі клацніть клас, щоб викликати його гарячий діалог.
2. Натисніть кнопку [Add Operation] в гарячому діалозі (рис.4.10).

У разі використання модельного елемента:

1. Виберіть клас, клацніть його правою кнопкою миші, і виберіть пункт [Add] -> [Operation] в контекстному меню (рис.4.11).

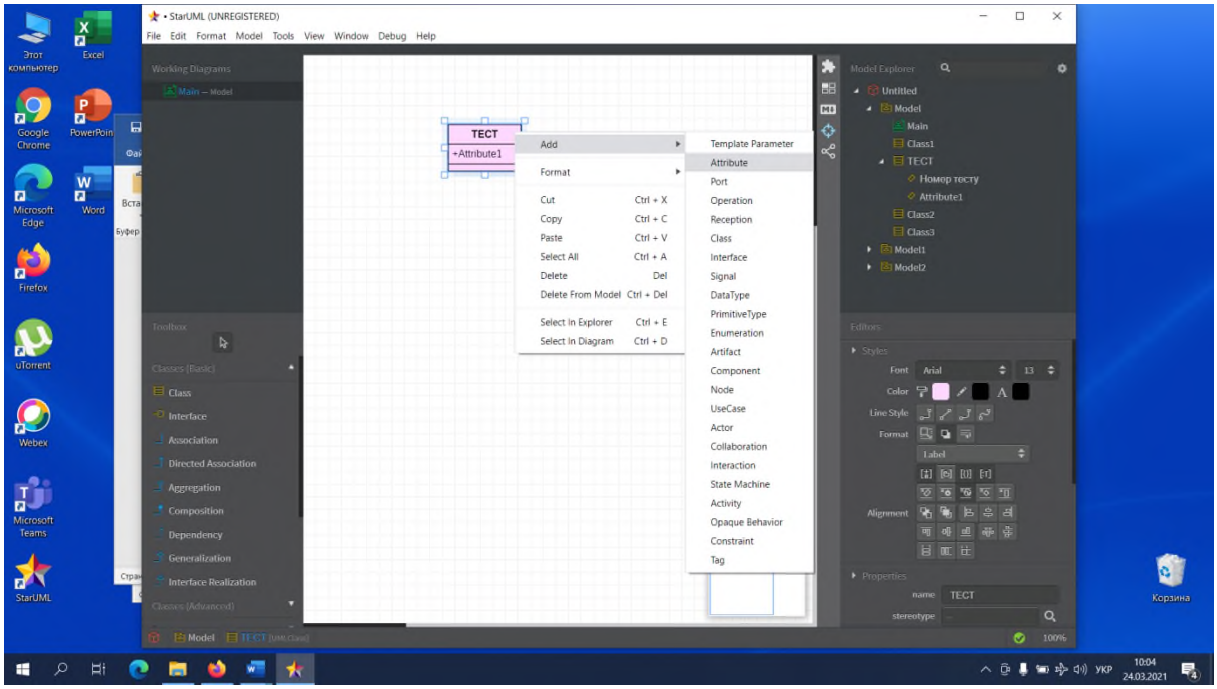


Рис.4.9.Використання навігатору моделі

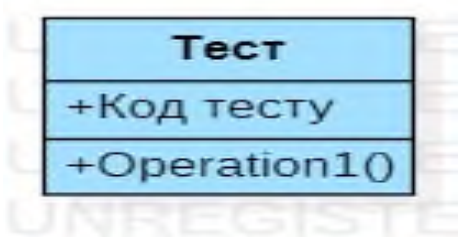


Рис.4.10 Додання операції з використанням гарячого діалогу

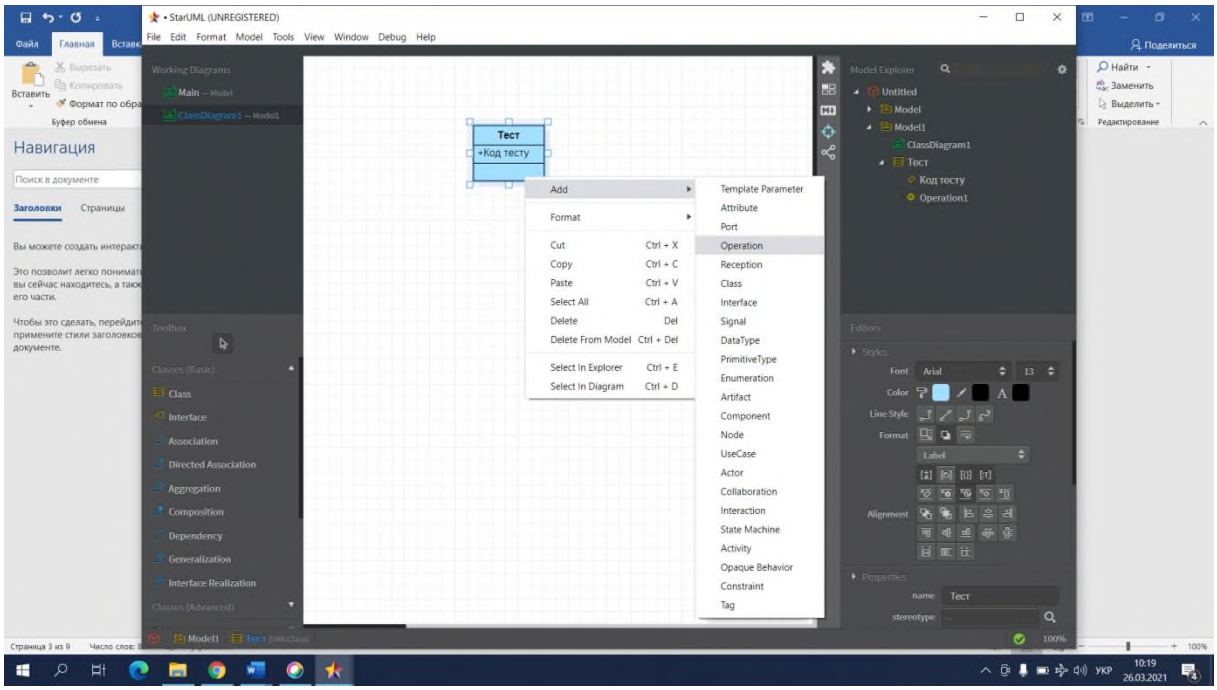


Рис.4.11. Додання операції з використанням навігатора моделі

6. Встановлення асоціацій

Семантика:

Асоціація – відношення між двома класами (включаючи можливість асоціації класу з самим собою).

Щоб створити асоціацію:

1. Натисніть кнопку [Toolbox] -> [Class] -> [Association].
2. На діаграмі проведіть лінію від одного пов'язують елемента до іншого.
3. Нова асоціація між двома класами буде створена, як показано нижче (рис.4.12).

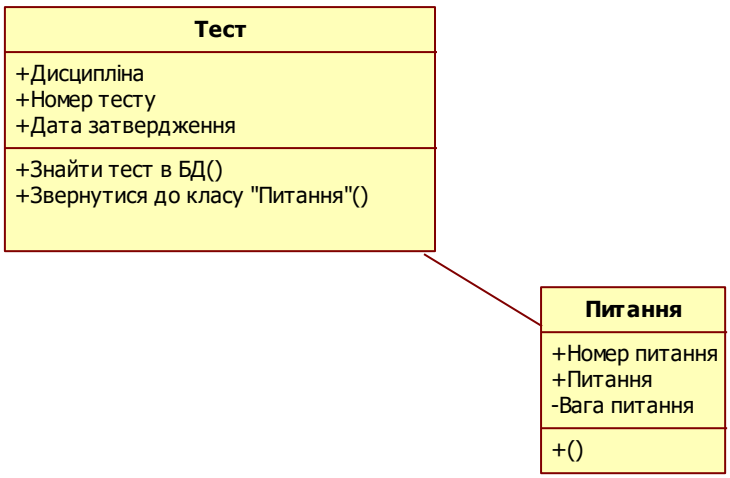


Рис.4.12. Встановлення асоціації

Спрямована асоціація. Процедура створення спрямованої асоціації аналогічна процедурі створення простої асоціації.

1. Натисніть кнопку [Toolbox] -> [Class] -> [DirectedAssociation].
2. Проведіть лінію між двома елементами в потрібному напрямку.
3. Результат буде наступним (рис.4.13).

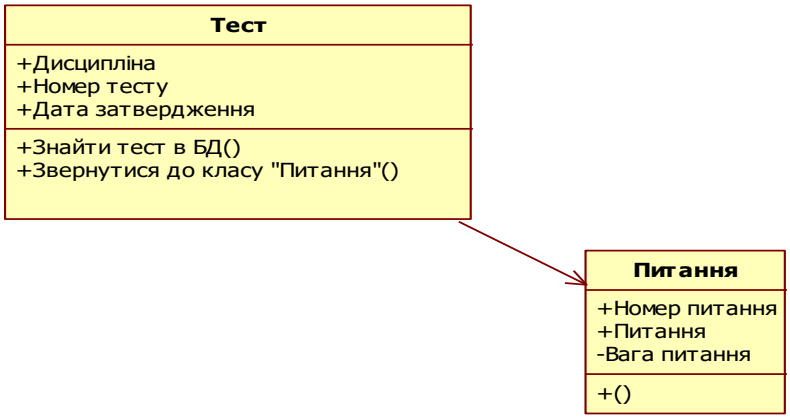


Рис.4.13. Встановлення спрямованої асоціації

Встановлення кількості екземплярі об’єктів кожного класу. Для асоціації може позначатися кількість екземплярів об’єктів кожного класу, які

беруть участь у зв’язку (0 - якщо жодного, 1 - якщо один, * - якщо багато). Можуть вказуватися мінімальна й максимальна кількість, наприклад, 0,1...* означає, що на відповідному кінці асоціації може не бути жодного екземпляра, бути один або багато (рис.4.14, див. рис 4.3).

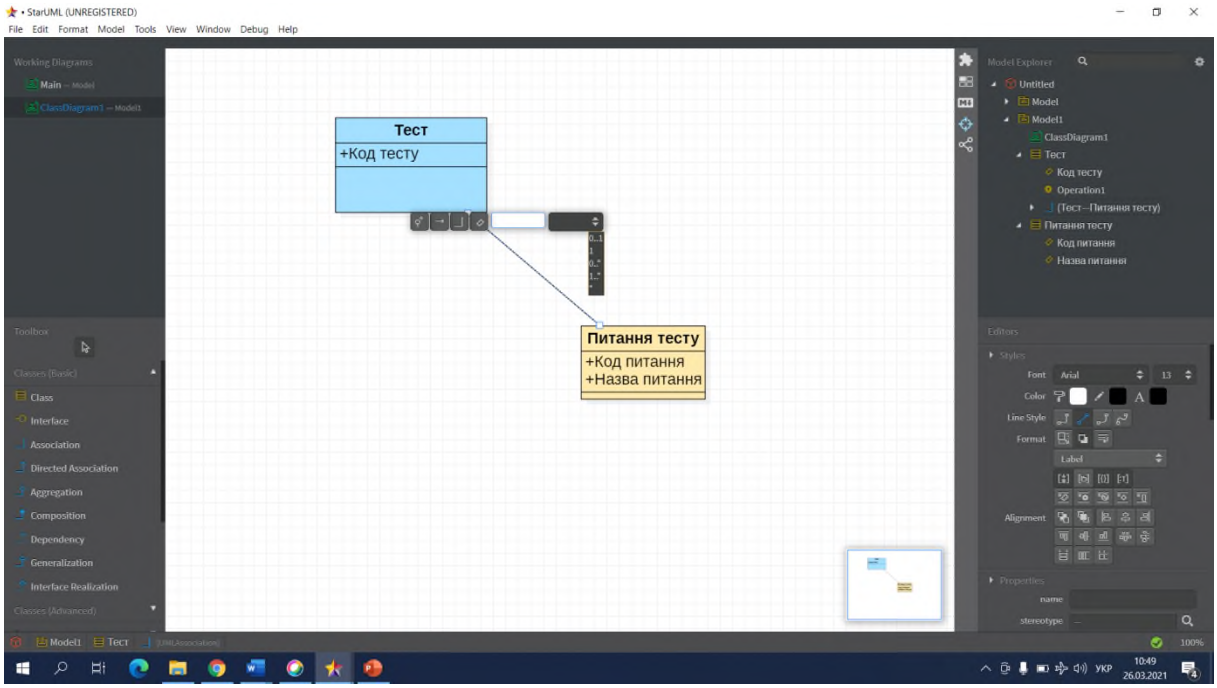


Рис. 4.14. Встановлення кількості екземплярів об’єктів

7. Агрегація. Агрегація – специфічний тип асоціації. Спеціальною формою або частковим випадком відношення асоціації є **відношення агрегації**.

Відношення агрегації може бути між декількома класами у тому випадку, якщо один із класів представляє собою деяку сутність, що включає в себе як складові частини інші сутності. На агрегат показує порожній ромб в точці, де асоціація з'єднується з класифікатором (кінець асоціації). Агрегація позначає відношення "ціле - частина". Клас, поблизу якого розташований порожнистий ромб, - ціле.

Процедура створення агрегату.

Щоб створювати агрегацію:

1. Клацніть кнопку [Toolbox] -> [Class] -> [Aggregation].
2. Проведіть лінію від класу-частини до класу-агрегату.
3. Результат буде наступним (рис.4.15).

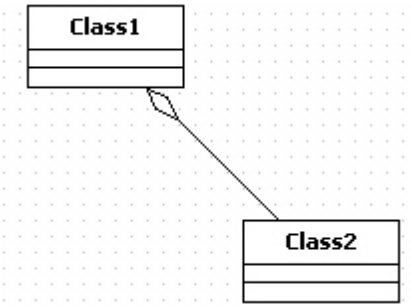


Рис. 4.15 . Агрегація

Процедура створення агрегованого класу за допомогою гарячого синтаксису створення елементів.

Щоб створити клас, агрегований в обраний клас, використовуйте гарячий синтаксис створення елементів (див. рис. 4.4)

1. Двічі клацнувши визветься гарячий діалог поточного класу. У гарячому діалозі, введіть символи "<> -", а потім імена класів, агрегованих в поточний клас.

2. Натисніть клавішу [Enter], і класи, агреговані в обраний клас, будуть створені і розміщені автоматично.

8.Композиція.Композиція – специфічний тип асоціації. Композиція позначається заповненим ромбиком в точці, де асоціація з'єднується з класифікатором (кінець асоціації). Композиція позначає відношення цілого і її невіддільною складовою частини, існування якої неможливе без цілого. Класифікатор, поблизу якого розташований заповнений ромб, - ціле.

Процедура створення композиції.

Щоб створити композицію:

1. Натисніть кнопку [Toolbox] -> [Class] -> [Composition].

2. Проведіть лінію від класу-частини до класу-композиту.

3. Між двома класами, буде створено нове ставлення композиції, як показано нижче (рис. 4.16).

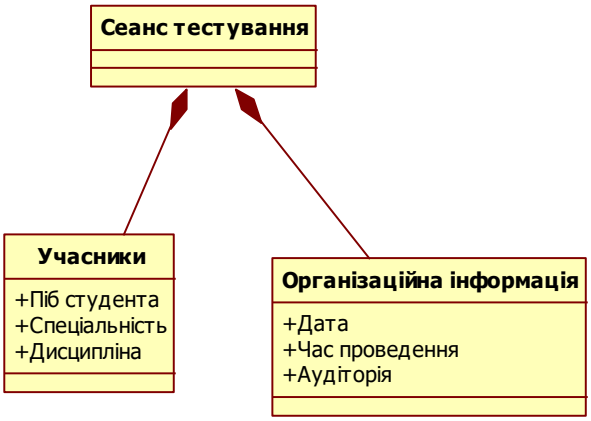


Рис.4.16. Приклад композиції класів

9.Залежність. Залежність визначає семантичні відносини між двома модельними елементами (або двома множинами модельних елементів). Вона пов'язує модельні елементи безпосередньо і не вимагає безлічі інстанцій для їх позначення. Вона вказує на ситуацію, коли зміна цільового елемента може спричинити необхідність зміни вихідного елемента залежності.

Процедура створення залежності.

Щоб створити залежність:

1. Клацніть кнопку [Toolbox] -> [Class] -> [Dependency].
2. Проведіть лінію між елементами в [основному вікні] в напрямку залежності.
3. Нова залежність між двома класами буде створена (див. рис.4.2).

10.Клас-ассоціатор. Клас-асоціатор - асоціація, яка є також класом. Вона не тільки з'єднує класифікатори, але також і визначає набір атрибутів, які безпосередньо належать відношенню, а не будь-якого з класифікаторів.

Процедура створення клас-асоціатора:

1. Клацніть кнопку [Toolbox] -> [Class] -> [AssociationClass].
2. Проведіть лінію між асоціацією та класом асоціації в [основному вікні] (рис.4.17).

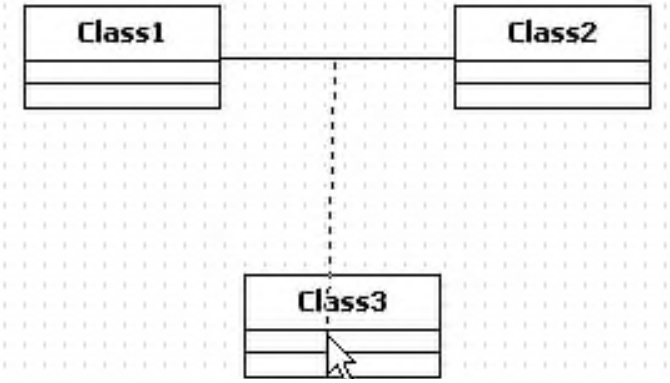


Рис. 4.17. Лінія між асоціацією та класом асоціації

Результат буде наступним (рис.4.18).

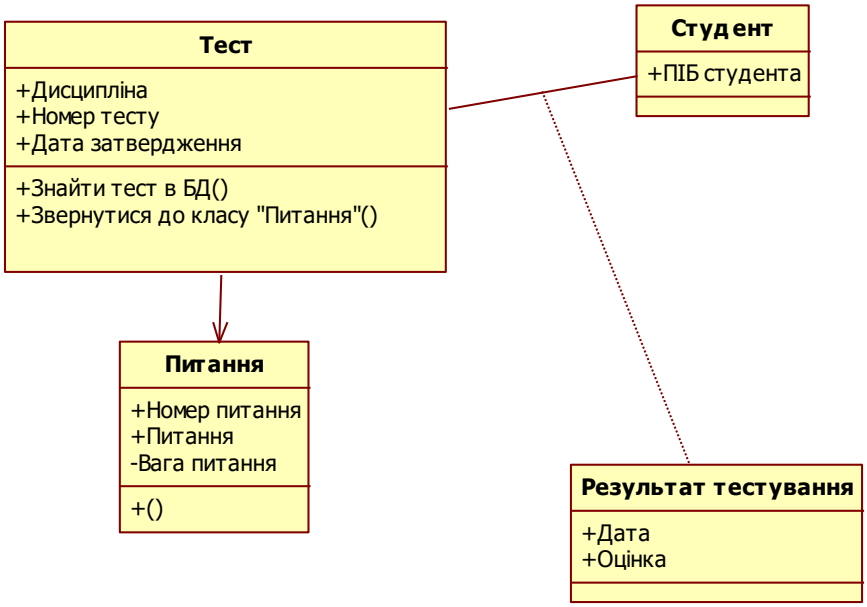


Рис.4.18. Зв'язок між асоціацією та класом асоціації

4.4. Контрольні питання до роботи

1. Цілі розробки діаграми класів.
2. Елементи діаграми класів. Класи.
3. Елементи діаграми класів. Відношення.

5.ЛАБОРАТОРНА РОБОТА №3.
ДІАГРАМА ПОСЛІДОВНОСТЕЙ (SEQUENCE DIAGRAM)

5.1 Мета роботи

Освоїти особливості взаємодії об’єктів проектованої системи при реалізації функцій окремого прецеденту, вивчити правила оформлення діаграм послідовності.

5.2 Теоретична частина

5.2.1. Діаграма послідовностей

Серією **діаграм прецедентів** визначається **зовнішня функціональність** системи, що проектується (виділяються всі прецеденти та актори, а також відношення між ними). З отриманих описів шляхом виявлення об'єктів, що задіяні в реалізації прецедентів, будуються **діаграми класів.** Уся подальша робота над проектом має здійснюватися на основі прецедентів: для кожного прецеденту формується опис його динаміки у вигляді серії ***діаграм взаємодії*** та ***діаграм діяльності***. Однією з важливіших видів діаграм взаємодії є діаграма послідовності.

Діаграми послідовностей (Sequence diagram) використовуються **для уточнення діаграм прецедентів, більш детального опису логіки сценаріїв використання.** Це відмінний засіб документування проекту з точки зору сценаріїв використання. **Діаграмою послідовностей** називається діаграма взаємодій, що акцентує увагу на часовій впорядкованості повідомлень, якими обмінюються об’єкти. Об’єкти це екземпляри класу - сутності, що на відміну від класу реально існують в системі, що проектується. Графічно така діаграма являє собою таблицю, об'єкти в якій розташовуються уздовж осі X, а повідомлення в порядку зростання часу - уздовж осі Y. Діаграма послідовності — відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень. Іншими словами, діаграма послідовностей відображає часові особливості передачі і прийому повідомлень об'єктами. Приклад побудови діаграми послідовності наведений на рис. 5.1.

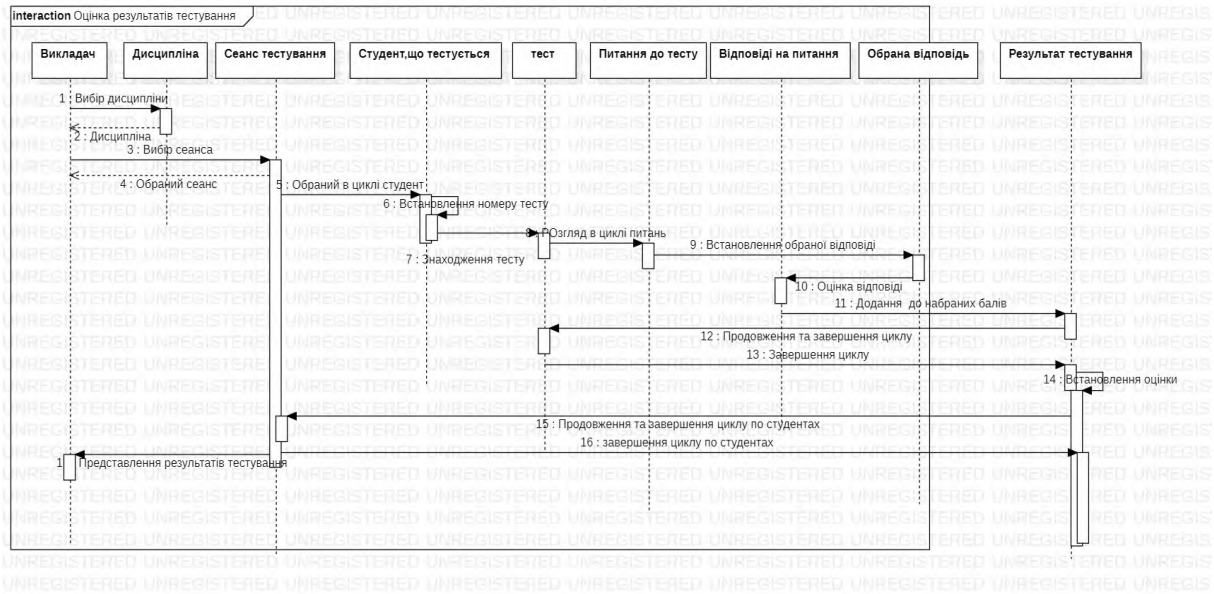


Рис. 5.1 Приклад діаграми послідовності прецеденту «Тестування»

Діаграми послідовностей зазвичай містять:

- **об'єкти,** які взаємодіють в рамках сценарію,
- **повідомлення**, якими вони обмінюються,
- **результати,** що повертаються як реакція на повідомлення.

В діаграмі **об'єкти** позначаються прямокутниками з підкресленими іменами (щоб відрізнити їх від класів), **повідомлення** (виклики методів) - лініями зі стрілками, р**езультати**, що повертаються - пунктирними лініями зі стрілками. Прямокутники на вертикальних лініях під кожним з об'єктів показують "час життя" (фокус) об'єктів. (*Втім, досить часто їх не зображують на діаграмі, все це залежить від індивідуального стилю проектування).*

На діаграмах послідовностей увага акцентується **насамперед на часовій впорядкованості повідомлень.** Для створення такої діаграми треба насамперед розташувати об'єкти, що беруть участь у взаємодії, у верхній її частині уздовж осі X. Зазвичай об’єкт, що ініціює взаємодію, розміщують ліворуч, а решта - правіше (тим далі, чим більш підлеглим є об'єкт). Потім уздовж осі Y розміщуються повідомлення, які об'єкти посилають і приймають, причому більш пізні виявляються нижче. Це дає читачеві наочну картину, що дозволяє **зрозуміти розвиток потоку управління в часі.**

На цій діаграмі показуються **взаємодії об'єктів для реалізації певного бізнес завдання** в якомусь певному процесі. Показується, як об'єкти один одного викликають і які параметри передають, повідомлення і що вони повертають.

Також можна подивитися за часом скільки кожен об'єкт живе і коли він знищується. І на відміну від Class diagram Sequence diagram показує всі об'єкти в дії. Тобто є якийсь певний процес і в ньому описується які об'єкти беруть участь і як вони взаємодіють (які повідомлення передають і в якій послідовності). **Це власне і є Sequence diagram.**

5.2.2. Лінія життя об’єкта

Лінія життя об'єкта на діаграмі відображається як вертикальна пунктирна лінія, що відображає існування об'єкта в часі. Велика частина об'єктів, представлених на діаграмі взаємодій, існує протягом всього сеансу роботи, тому їх зображують у верхній частині діаграми, а їх лінії життя промальовані зверху до низу. Об'єкти можуть створюватися і під час взаємодій. Лінії життя таких об'єктів починаються з отримання повідомлення зі стереотипом **create**. Об'єкти можуть також знищуватися під час взаємодій; в такому разі їх лінії життя закінчуються отриманням повідомлення зі стереотипом **destroy**, а в якості візуального образу використовується велика буква X, що позначає кінець життя об'єкта.

5.2.3. Фокус управління

Друга особливість цих діаграм - **фокус управління**. Він зображається у вигляді витягнутого прямокутника, що показує проміжок часу, протягом якого об'єкт виконує будь-яку дію, безпосередньо або за допомогою підпорядкованої процедури. Верхня грань прямокутника вирівнюється по тимчасовій осі з моментом початку дії, нижня - з моментом його завершення (і може бути позначена повідомленням про повернення).

5.2.4. Повідомлення

Як було зазначено вище, мета взаємодії в контексті мови UML полягає втому, щоб специфікувати комунікацію між безліччю взаємодіючих об'єктів. Кожна взаємодія описується сукупністю повідомлень, якими обмінюються між собою об'єкти, що беруть участь у цій взаємодії. У цьому сенсі повідомлення (message) являє собою закінчений фрагмент інформації, який відправляється одним об'єктом іншому. При цьому прийом повідомлення ініціює виконання певних дій, спрямованих на вирішення окремого завдання тим об'єктом, якому

це повідомлення надіслано. Таким чином, повідомлення не тільки передають деяку інформацію, але і вимагають або припускають від приймаючого об'єкта виконання очікуваних дій. Повідомлення можуть ініціювати виконання операцій об'єктом відповідного класу, а параметри цих операцій передаються разом з повідомленням.

На діаграмі послідовності всі повідомлення впорядковані за часом свого виникнення в системі. У такому контексті кожне повідомлення має напрямок від об'єкта, який ініціює та відправляє повідомлення, до об'єкту, який його отримує. Зазвичай повідомлення зображуються горизонтальними стрілками, що з'єднують лінії життя або фокуси управління двох об'єктів на діаграмі послідовності.

У мові UML кожне повідомлення асоціюється з деяким дією, яка має бути виконаною прийнятим його об'єктом. При цьому дія може мати деякі аргументи або параметри, в залежності від конкретних значень яких може бути отриманий різний результат. Відповідні параметри матиме і повідомлення, що викликало цю дію. Більш того, значення параметрів окремих повідомлень можуть містити умовні вирази, утворюючи розгалуження або альтернативні шляхи основного потоку управління.

5.3 Побудова діаграми послідовності у StarUML

1.Створення діаграми послідовності. Для створення діаграми послідовності необхідно у вікні програми вибрати вкладку “Model => Add Diagram =>Sequence Diagram” (рис 5.2).

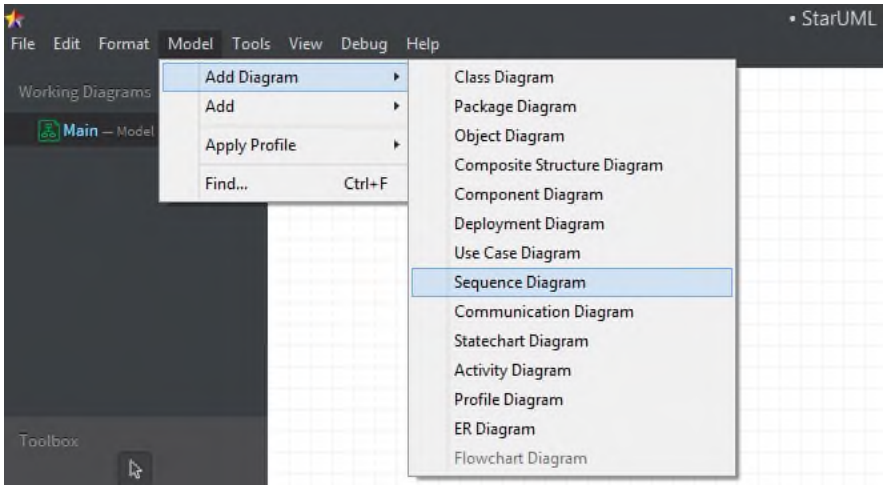


Рис. 5.2 Створення діаграми послідовності

Після цього на екрані зправа з’явиться панель управління діаграмою (рис.5.3):

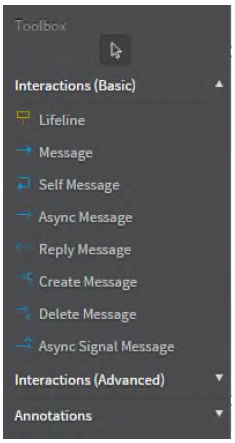


Рис.5.3. Панель управління Sequence diagram (Interactions (Basic))

Також в діаграммі послідовності можуть використовувати такі елементи(Interactions(Advanced)) (рис.5.4):

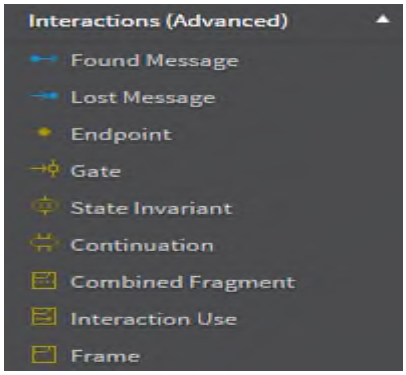


Рис.5.4. Меню Interactions (Advanced)

2.Створення елементів меню Sequance diagram.

Лінія життя об’єкта*(Lifeline*). На першому етапі роботи ми вибираємо об’єкти (екземпляри класу), що будуть застосовані при реалізації прецеденту. На діаграмі послідовності зображуються виключно ті об'єкти, які безпосередньо беруть участь у взаємодії Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів в часі. При цьому діаграма послідовності має якби два виміри-об'єкти, які розташовуються уздовж осі X, а повідомлення в порядку зростання часу - уздовж осі Y.

Лінія життя кожного об’єкта визначається зліва направо у вигляді вертикальних ліній. Графічно кожен об'єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя. Виберіть Lifeline в панелі

інструментів і розмістить свої об’єкти. Усередині прямокутника записуються ім'я об'єкта.

Повідомлення (*Message).*

1). Виберіть **Message(або Self-Message)** в панелі інструментів.

2) Перетягніть з обраного **Lifeline** і визначте «падіння» на іншому **Lifeline.**

Щоб відредагувати **Message**, ви можете виконати наступні дії:

За допомогою **Quick Edit** для повідомлення подвійним клацанням миші або натисніть Enter на обраному **Message** (рис.5.5).

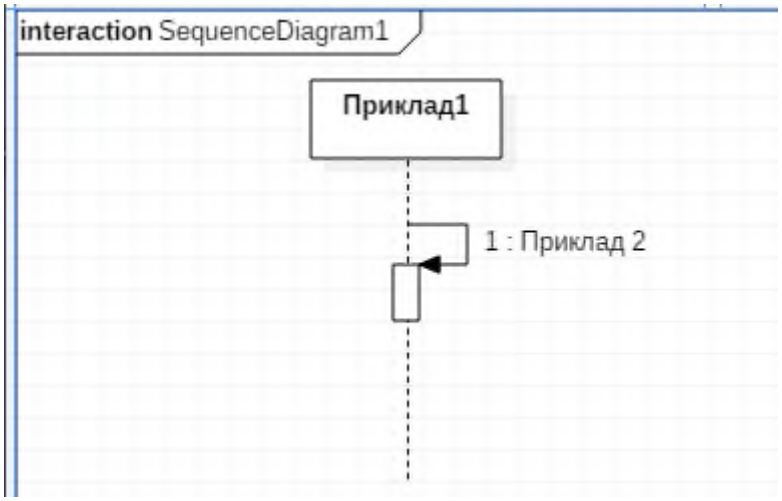


Рис.5.5. Приклад застосування Self Message

Кінцева точка (Endpoint). Для задання кінцевої точки зробіть наступне:

1) Виберіть **Endpoint** в панелі інструментів.

2) Натисніть на позицію на діаграмі.

3. Приклад побудовидіаграми послідовності

Розглянемо побудову діаграми послідовності на конкретному прикладі. **Постановка задачі**: виконати проектування додатку для автоматизації обліку **нових надходжень та реєстрації видачі в пункті прокату компакт-дисків. Розглянемо такий сценарій**: отримання клієнтом компакт-диска в пункті прокату.

Сценарій: Коли клієнт робить замовлення, службовець пункту прокату переглядає його картку, щоб визначити, чи не числяться за клієнтом будь-які порушення. Після цього перевіряється наявність запитуваного примірника. Після підтвердження формується запис про видачу, в якій вказується номер

видаваємого примірника, ПІБ клієнта, поточна дата і дата повернення. На основі сценарію можна визначити наступні об'єкти:

- Клієнт
- Службовець
- Компакт-диск
- Видача напрокат
- Картка видачі

1.Для створення нового проекту в системі StarUML необхідно у вікні програми вибрати вкладку “ File => NewProject ” (рис. 5.6).

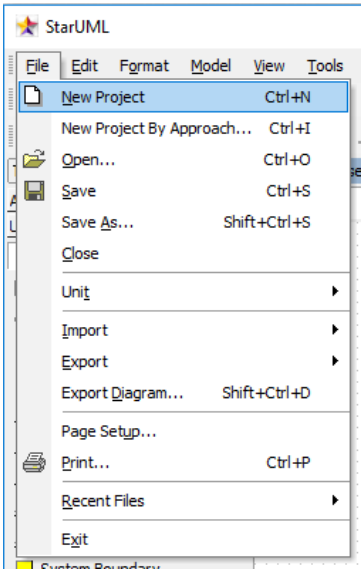


Рис. 5.6. Створення нового проекту StarUML

2.Для створення діаграми послідовності необхідно у вікні програми вибрати вкладку “Model => Add Diagram =>Sequence Diagram” (рис. 5.7).

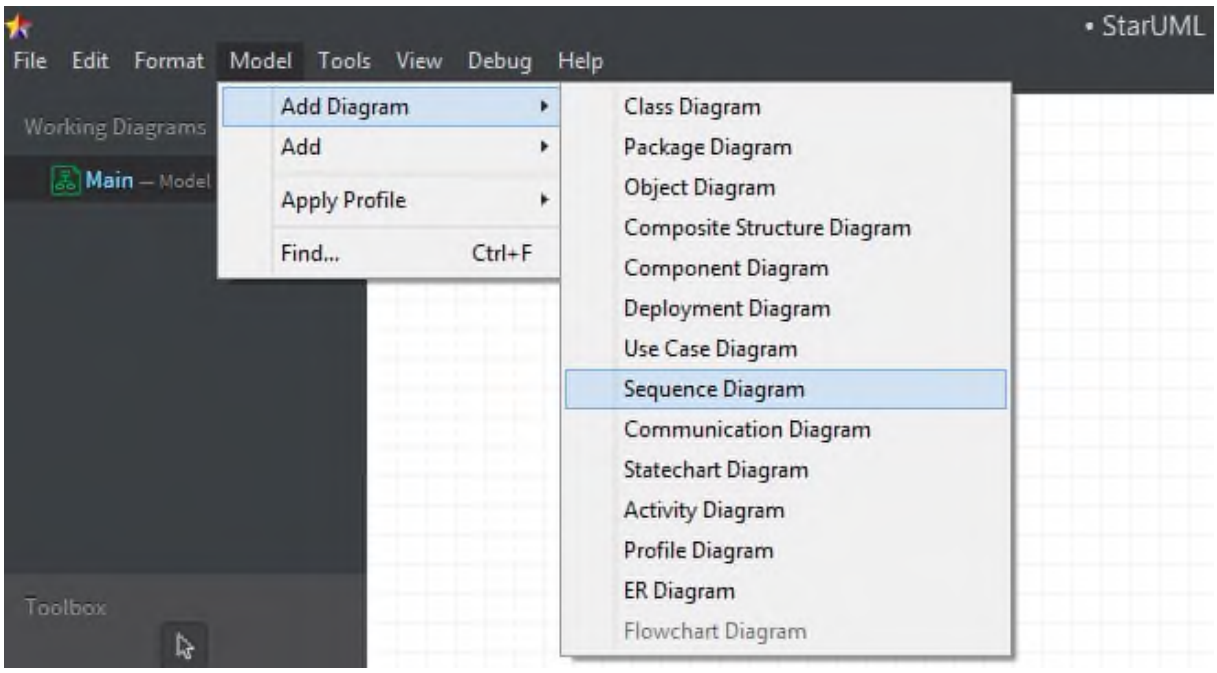


Рис. 5.7. Створення діаграми послідовності

Задаємо назву діаграми. Для цього знайдемо у меню «Properties» вкладку «name» (рис.5.8).

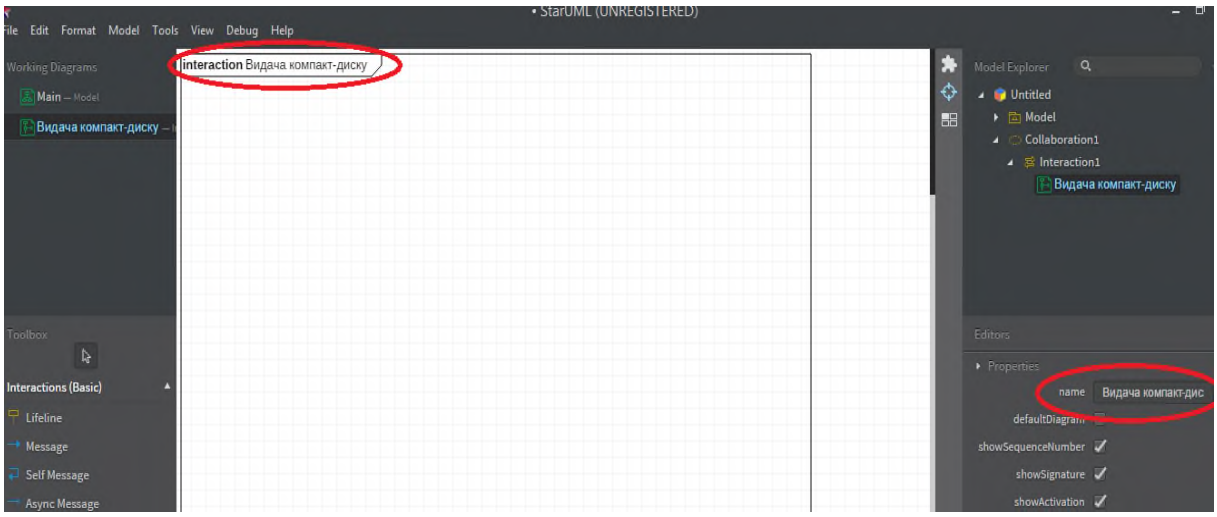


Рис. 5.8 Зміна назви діаграми

3.На панелі інструментів вибираємо кнопку з назвою **«Lifeline»** щоб додати об’єкт та надаємо йому ім’я (рис. 5.9).

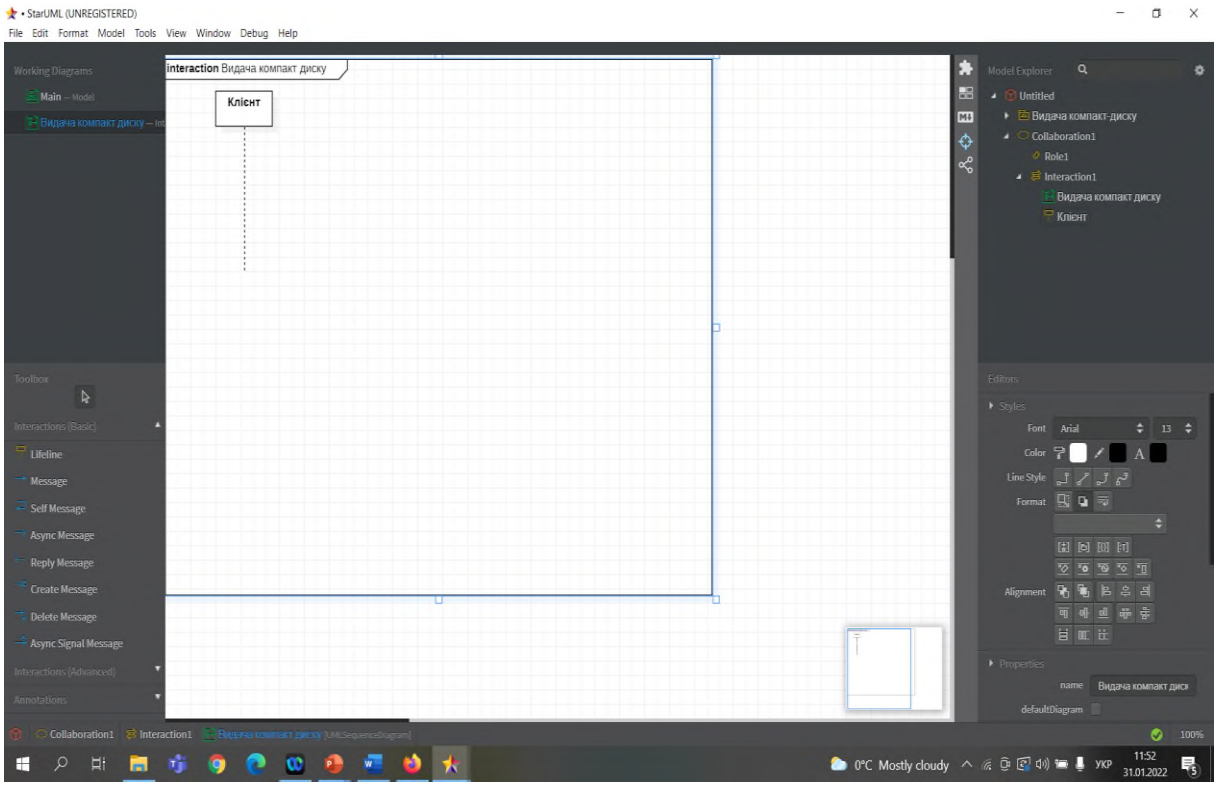


Рис.5.9. Створення об’єкту в StarUml

Аналогічно додаємо інші об’єкти (рис.5.10):

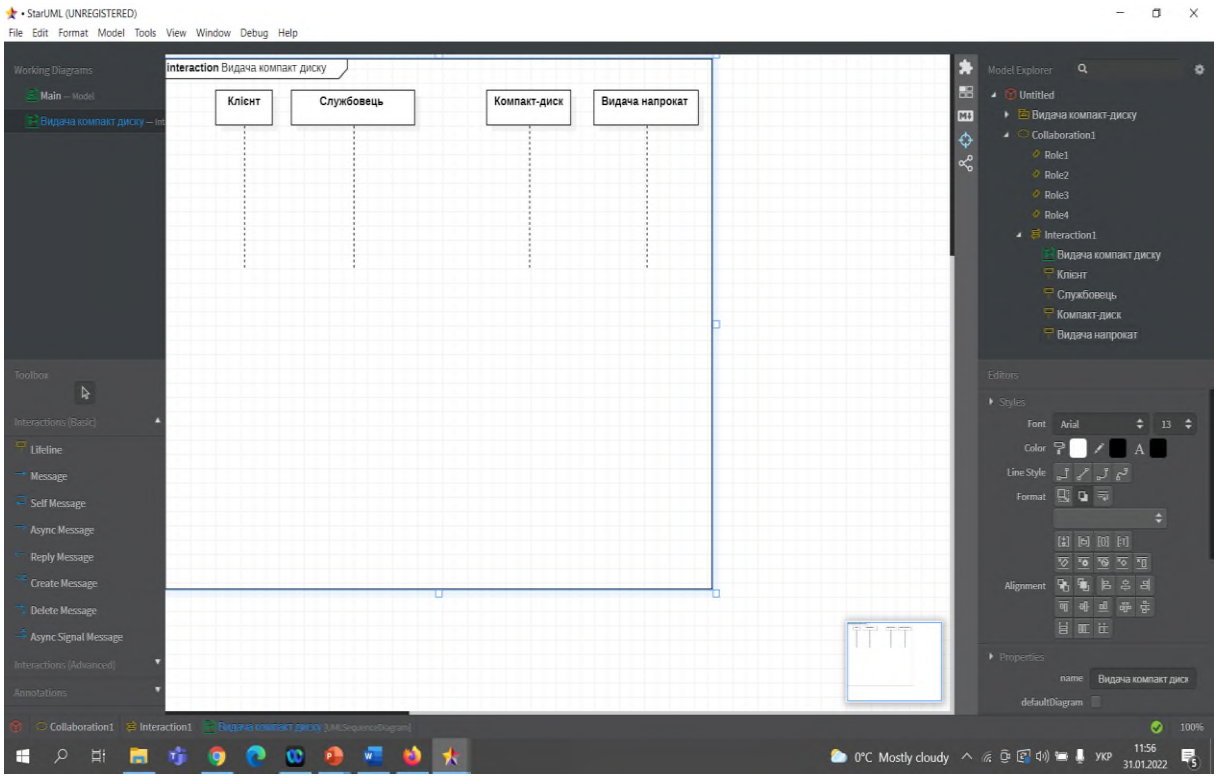


Рис.5.10 Об’єкти на панелі програми

4.Для зображення передачі повідомлення використовуємо суцільну стрілку (рис.5.11). З'єднуємо лінії життя відповідних об'єктів та підписуємо зміст повідомлення.

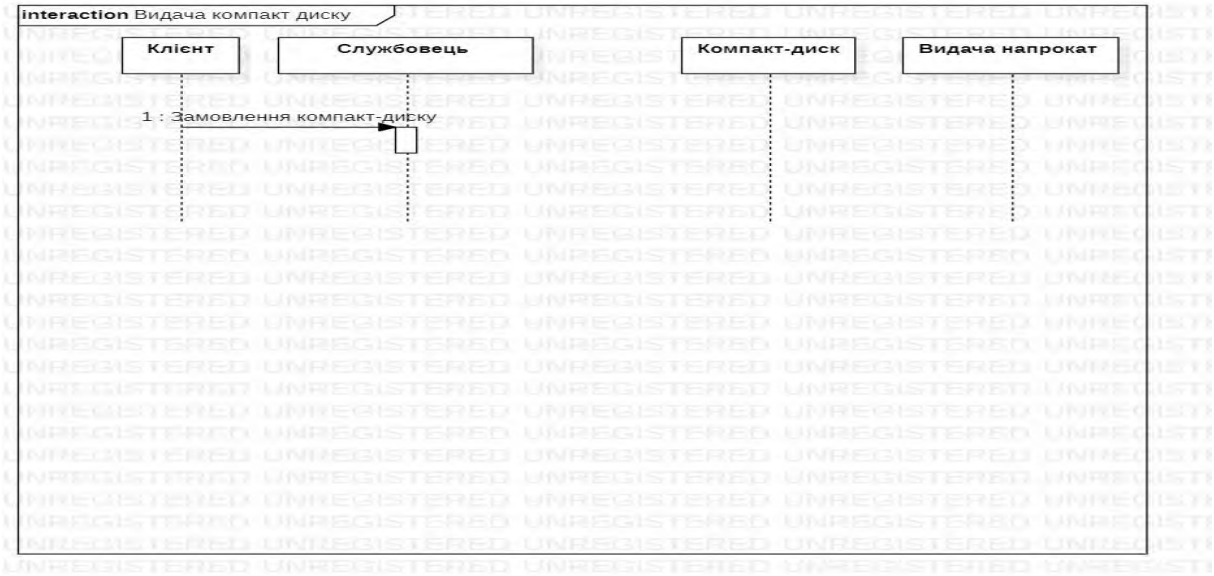


Рис.5.11. З'єднання ліній життя відповідних об'єктів

Далі службовець запитує «Чи є диск в наявності?» (рис. 5.12):

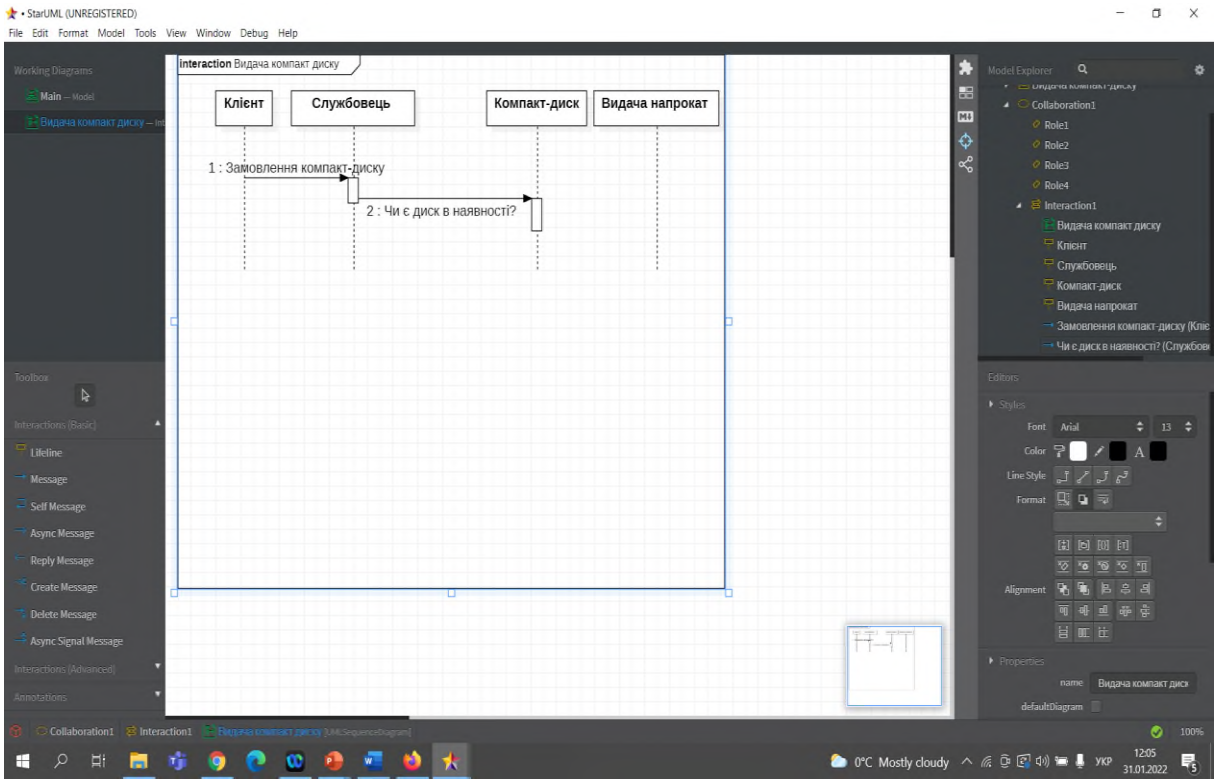


Рис.5.12. З'єднання ліній життя відповідних об'єктів

Службовцю надходить зворотна відповідь. Її зображують пунктирною стрілкою (рис.5.13).

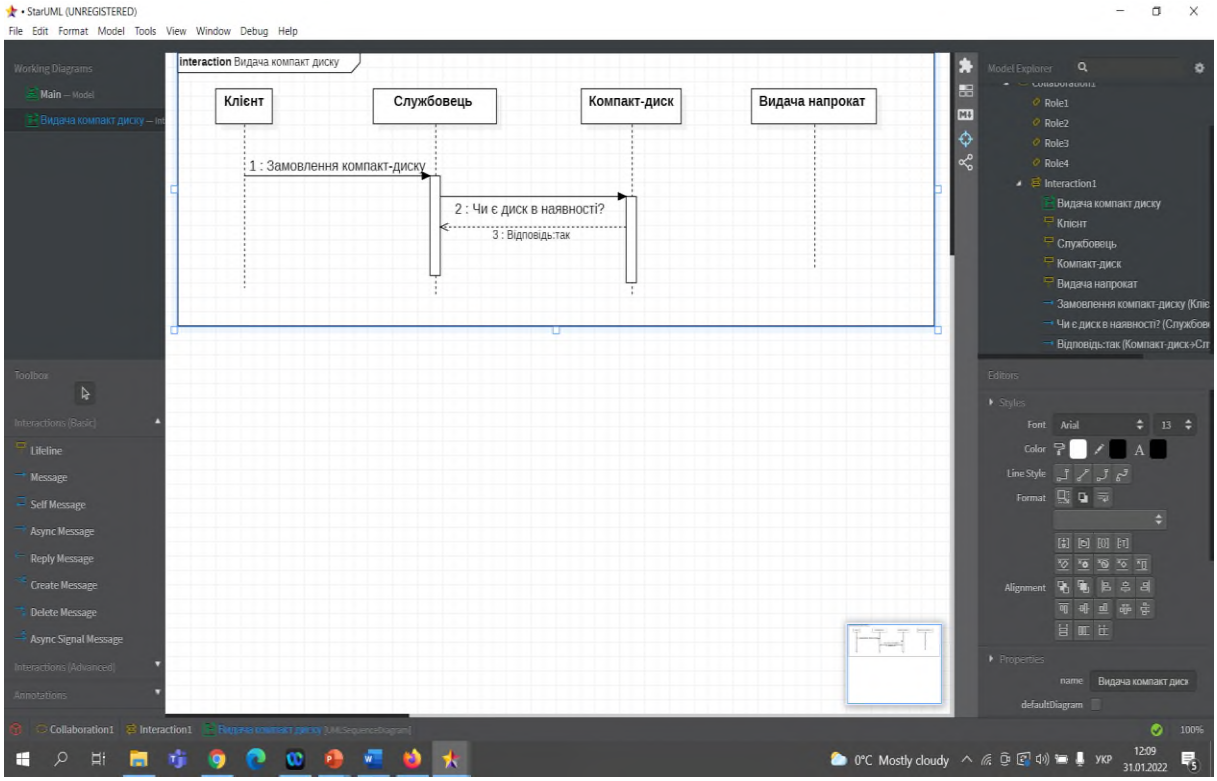


Рис.5.13. Зворотна відповідь

Якщо диск є в наявності, службовець зменшує кількість компакт дисків (рис. 5.14):

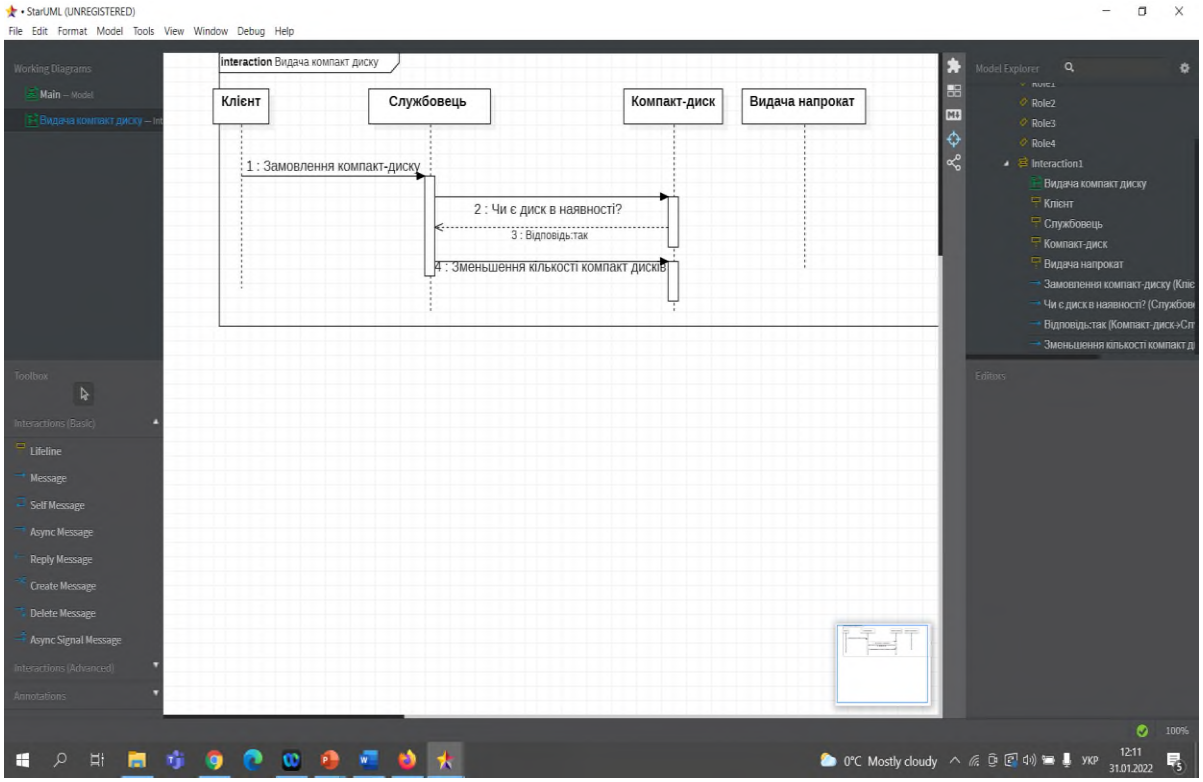


Рис.5.14. З'єднання ліній життя відповідних об'єктів

Службовець обчислює термін повернення компакт-диска (рис. 5.15):

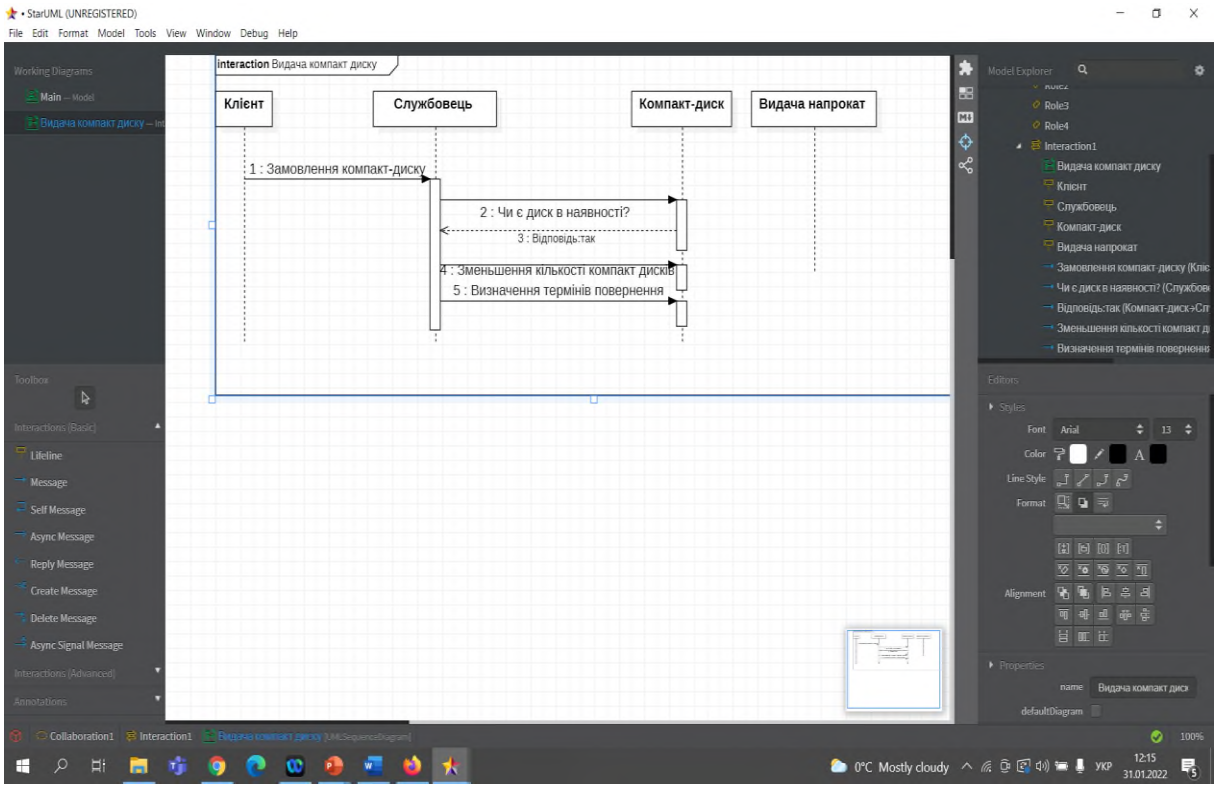


Рис.5.15. З'єднання ліній життя відповідних об'єктів

Йому надсилають зворотну відповідь про термін повернення рис.5.16):

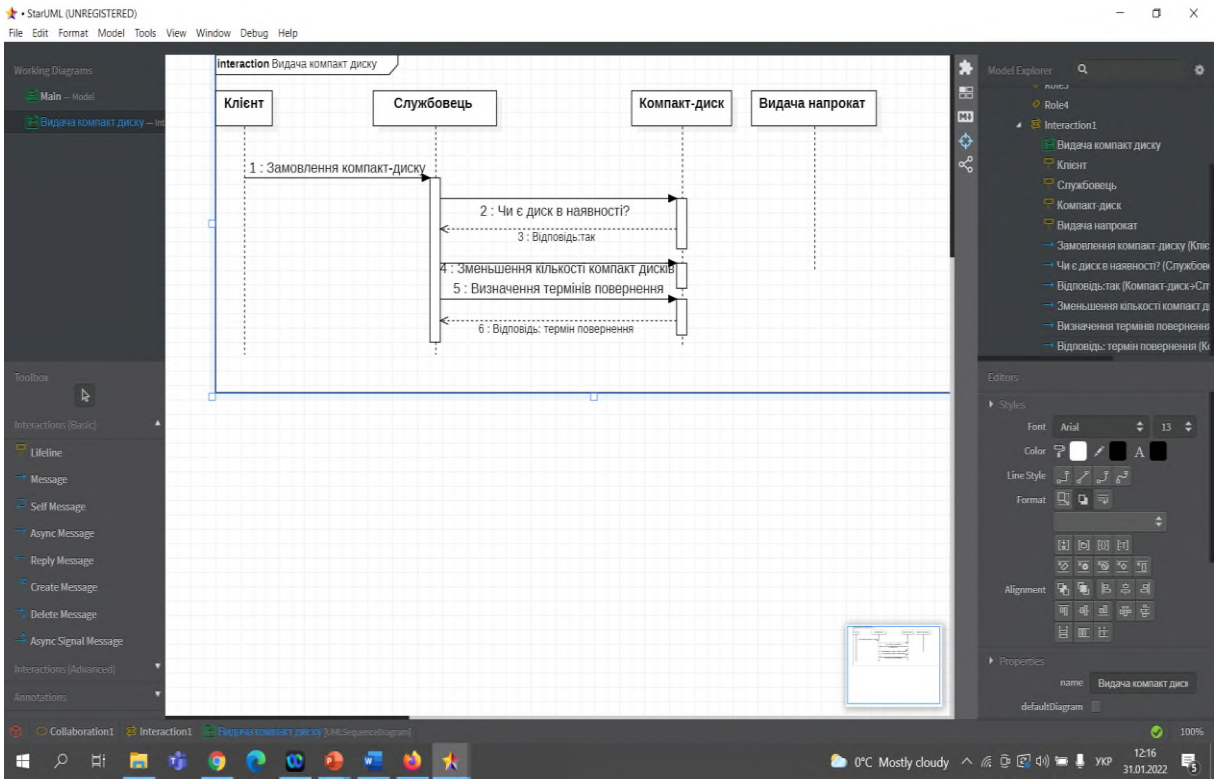


Рис.5.16. Реалізація зворотньої відповіді

Після цього службовець здійснює створення об'єкта видачі напрокат (рис.5.16):

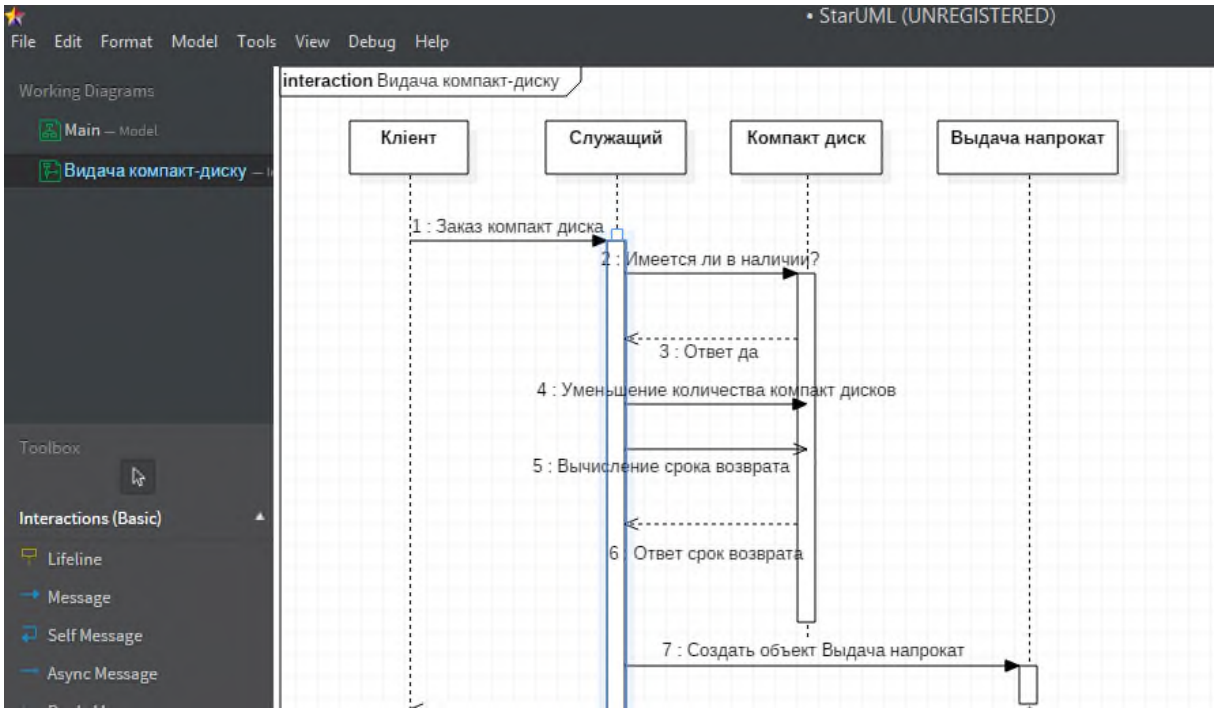


Рис.5.16. З'єднання ліній життя відповідних об'єктів

Таким чином ми отримали діаграму послідовності для обраного сценарію

5.4. Контрольні питання до роботи

1.Поняття класу і об’єкту системи.

2. Призначення діаграми послідовності.

3. Компоненти діаграми послідовності: об’єкти, повідомлення, лінія життя, фокус управління.

6. ЛАБОРАТОРНА РОБОТА №4.
ДІАГРАМА ДІЯЛЬНОСТІ (*ACTIVITY DIAGRAM*)

6.1. Мета роботи

Освоїти призначення та основи побудови діаграми діяльності –одного з п'яти видів діаграм, застосовуваних у UML для моделювання динамічних аспектів поведінки системи, вивчити правила оформлення діаграм діяльності.

6.2 Теоретичні положення

При моделюванні поведінки системи в процесі її розробки виникає необхідність не тільки уявити процес зміни її станів, а й деталізувати особливості алгоритмічної і логічної реалізації виконуваних системою операцій. Традиційно для цієї мети використовувалися блок-схеми або структурні схеми алгоритмів. Кожна така схема акцентує увагу на послідовності виконання певних дій або елементарних операцій, які в сукупності призводять до отримання бажаного результату. Алгоритмічні і логічні операції, що вимагають виконання в певній послідовності, оточують нас постійно. Важливо підкреслити ту обставину, що зі збільшенням складності системи суворе дотримання послідовності виконуваних операцій набуває все більш важливе значення. Якщо спробувати заварити каву холодною водою, то ми можемо тільки зіпсувати одну порцію напою. Порушення послідовності операцій при ремонті двигуна може привести до його поломки або виходу з ладу. Ще більш катастрофічні наслідки можуть статися в разі відхилення від встановленої послідовності дій при зльоті або посадці авіалайнера, запуск ракети, регламентних роботах на АЕС.

Для моделювання процесу виконання операцій в мові UML використовуються так звані діаграми діяльності (**activity diagram**). Кожен стан на діаграмі діяльності відповідає виконанню деякої операції, а перехід в наступний стан спрацьовує тільки при завершенні операції в попередньому стані. Графічно діаграма діяльності представляється у формі графа діяльності, вершинами якого є стани дії, а дугами - переходи від одного стану дії до іншого.

Діаграми діяльності - це один з п'яти видів діаграм, застосовуваних у UML для моделювання динамічних аспектів поведінки системи (інші види: діаграми послідовностей і кооперації, станів, прецедентів, відповідно). Діаграма

діяльності - це, по суті, блок-схема, яка показує, як потік управління переходить від однієї діяльності до іншої.

Діаграми діяльності можна використовувати для моделювання динамічних аспектів поведінки системи. Як правило, вони застосовуються, щоб промоделювати послідовні (а іноді і паралельні) кроки обчислювального процесу. За допомогою діаграм діяльності можна також моделювати життя об'єкта, коли він переходить з одного стану в інший в різних точках потоку управління. Діаграми діяльності можуть використовуватися самостійно для візуалізації, специфікації, конструювання та документування динаміки сукупності об'єктів, але вони придатні також і для моделювання потоку управління при виконанні деякої операції. На рис. 6.1. наведений приклад діаграми послідовності для системи управління інформаційними ризиками, де в якості станів визначні дії по реалізації прецедентів системи, на рис. 6.2 – приклад діаграми для дії, що пов’язані з реалізацією окремої операції класу.

Якщо в діаграмах взаємодій акцент робиться на переходах потоку управління від об'єкта до об'єкта, то діаграми діяльності описують переходи від однієї діяльності до іншої.

Графічно діаграма діяльності представляється у вигляді графа, що має вершини і ребра. Діаграми будуються з обмеженої кількості фігур, з’єднаних стрілочками (див. рис 6.1, 6.2). Найважливіші типи фігур:

- *скруглені прямокутники* - позначають *дії*;
- *ромби* - позначають *рішення*;
- *риски* позначають початок (*розподіл*) чи кінець (*об’єднання*) паралельних активностей;
- *чорний кружок* - позначає старт (*початковий стан*) процесу;
- *чорний кружок в колі* - позначає кінець (*кінцевий стан*).
- *стрілки* - ведуть від старту до кінця і позначають порядок в якому відбуваються активності.

Опишемо наступні базові логічні операції, що застосовуються при побудови діаграми діяльності: дії, переходи, рішення (розгалуження), розділення і злиття, область дії (доріжки).

Дії. Дія (або стан дії) являє собою виконання простої дії або операції. Стан дії - простий стан, що припускає виконання дії та вихід з цього стану. Тобто воно відповідає виконанню безпосередньо дії і переходу, який буде активізований, як тільки виконання дії буде завершено. Дія може бути записано на природній мові, деякому псевдокоді або мовою програмування. Ніяких додаткових або неявних обмежень при запису дій не накладається.

Рекомендується в якості імені дії використовувати дієслово з пояснювальними словами. Якщо ж дія може бути представлено в деякому формальному вигляді, то доцільно записати його на тій мові програмування, на якому передбачається реалізовувати конкретний проект.

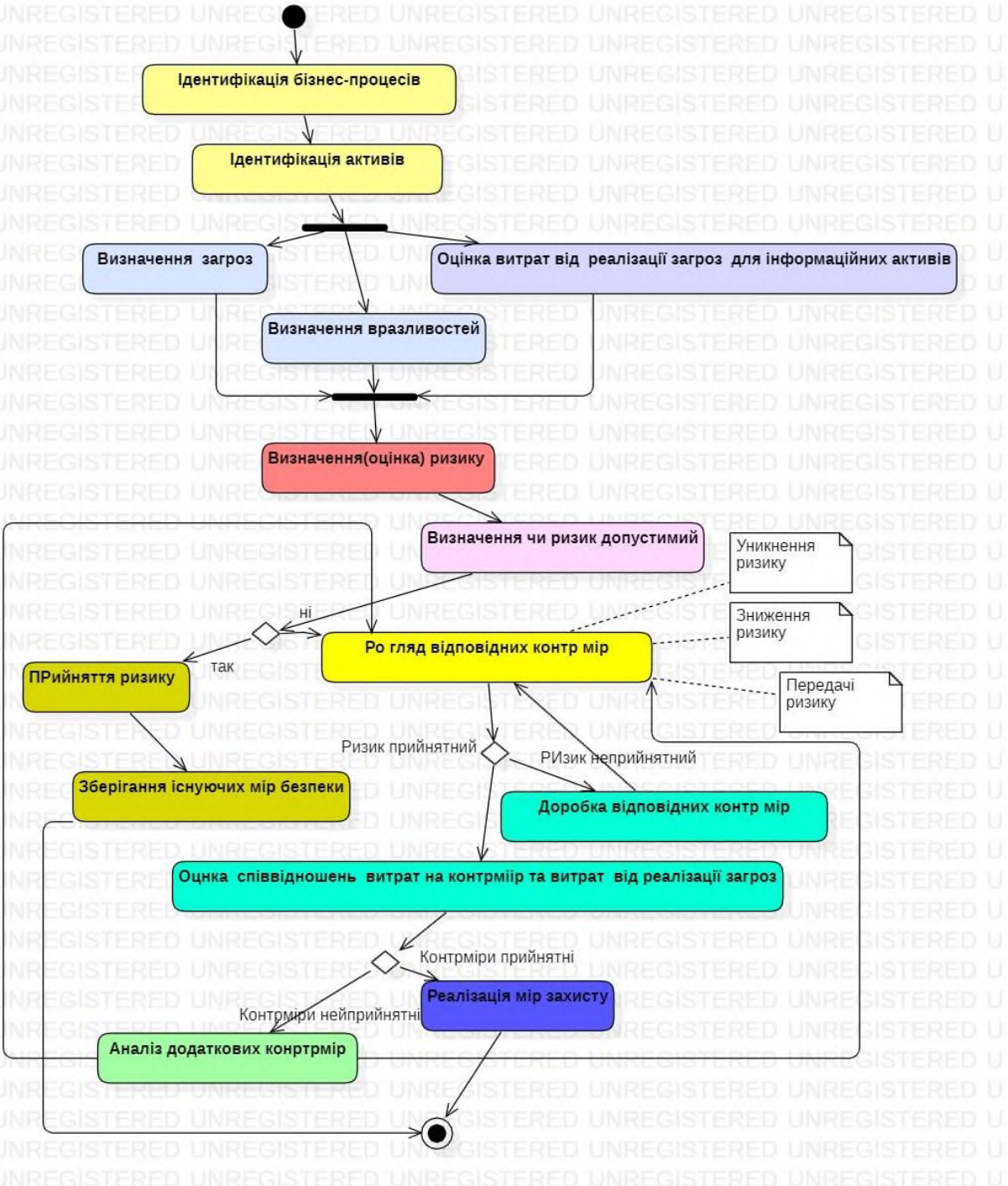


Рис. 6.1. Діаграма діяльності системи управління інформаційними ризиками

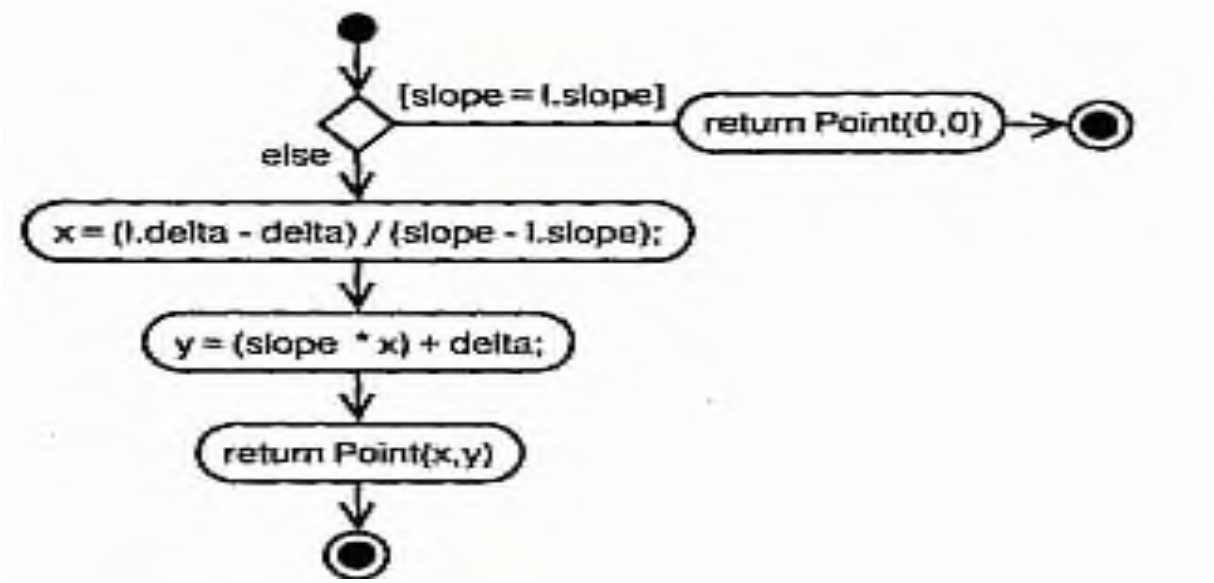


Рис. 6.2. Діаграма діяльності для операції

Дії зображуються прямокутниками з закругленими краями. Усередині такого символу можна записувати довільний вираз (рис.6.3., див. рис 6.1) або вираз, що пов'язаний з відповідною мовою програмування (див. рис. 6.2)

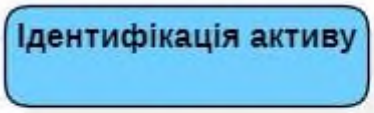


Рис. 6.3. Довільний вираз опису дії

Переходи. Коли дія або діяльність в деякому стані завершується, потік управління відразу переходить у наступний стан дії або діяльності. Для опису цього потоку використовуються переходи (**Transitions**), що показують шлях з одного стану дії або діяльності в інше. В UML перехід представляється простий лінією зі стрілкою, як показано на рис. 6.4.

Рис. 6.4. Перехід

Рішення (розгалуження). Діаграма станів (і похідна від неї діаграма діяльності) використовує елемент рішення, коли обчислюються деякі граничні умови, необхідні, щоб вказати можливі переходи, які залежать від результатів обчислення цих логічних умов. Графічно розгалуження на діаграмі діяльності позначається невеликим ромбом, усередині якого немає ніякого тексту. У цей ромб може входити тільки одна стрілка від того стану дії, після виконання якого потік управління повинен бути продовжений за однією з взаємно виключають гілок. Прийнято вхідну стрілку приєднувати до верхньої або лівої вершині символу розгалуження. Вихідних стрілок може бути дві або більше, але для кожної з них явно вказується відповідна сторожова умова в формі булевого вираження.

Як видно з рис.6.5, точка розгалуження представляється ромбом. В точку розгалуження може входити рівно один перехід, а виходити - два або більше. Для кожного вихідного переходу задається вираз, який обчислюється тільки один раз при вході в точку розгалуження. Ні для яких двох вихідних переходів ці сторожові умови не повинні одночасно приймати значення "істина", інакше потік управління виявиться неоднозначним. Але ці умови повинні покривати всі можливі варіанти, інакше потік зупиниться.

Рис. 6.5. Розгалуження

Розділення і злиття. Прості і розгалужені послідовні переходи в діаграмах діяльності використовуються найчастіше. Однак можна зустріти і паралельні потоки, і це особливо характерно для моделювання бізнес-процесів. В UML для позначення розділення і злиття таких паралельних потоків виконання використовується синхронізаційна риса, яка малюється у вигляді жирної вертикальної або горизонтальної лінії (рис. 6.6).

Доріжки. При моделюванні течії бізнес-процесів іноді буває корисно розбити стан діяльності на діаграмах діяльності на групи, кожна з яких представляє «актора», що веде та відповідає за ту чи іншу роботу (відділ компанії, посадовець, група виконавців). В UML такі групи називаються доріжками (Swimlanes), оскільки візуально кожна група відділяється від сусідніх вертикальною рисою, як плавальні доріжки в басейні (рис. 6.7).

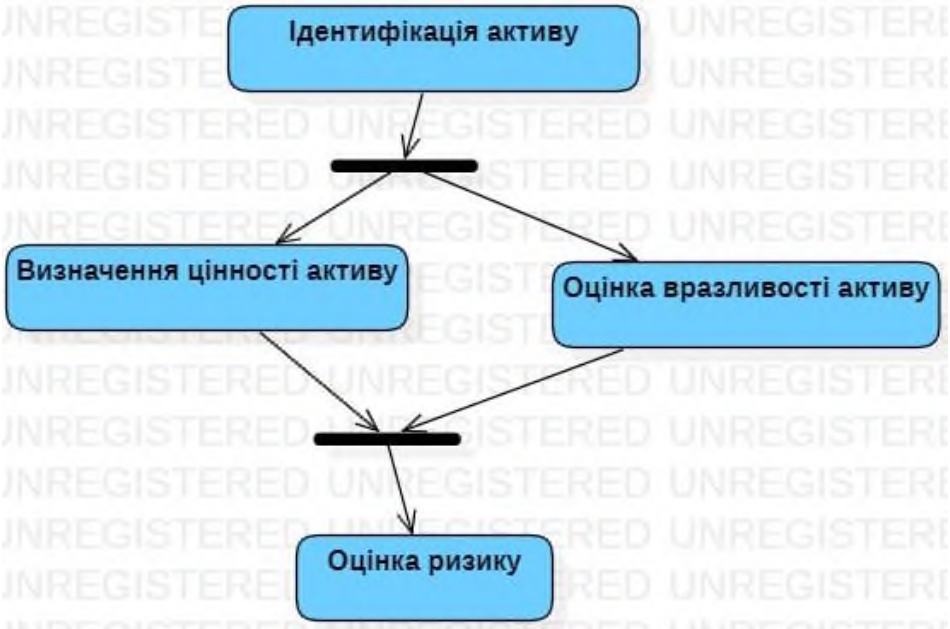


Рис. 6.6. Розілення і злиття інформаціних потоків

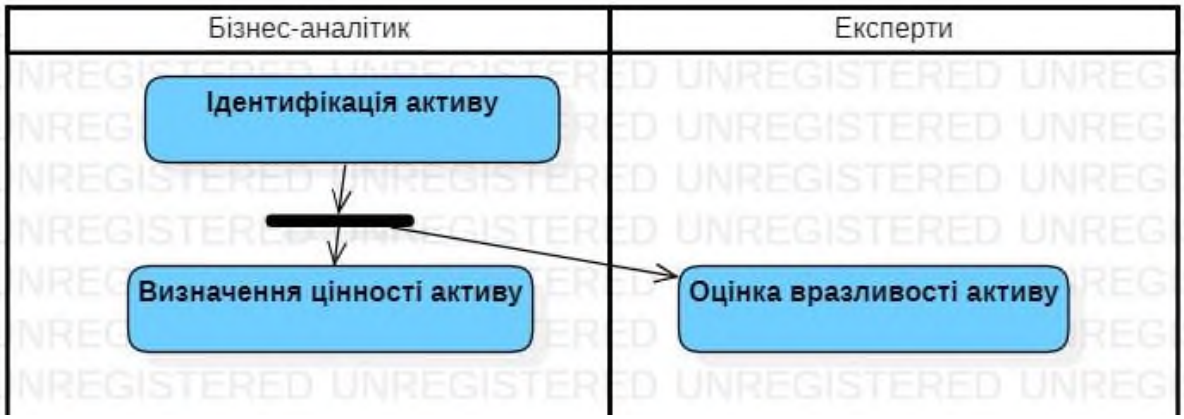


Рис. 6.7. Доріжки

6.3. Побудова діаграми діяльності в Star UML

1.Створення діаграми діяльності. Для створення діаграми діяльності необхідно в меню клацнути на Model – Add Diagram – Activity Diagram (рис. 6.8).

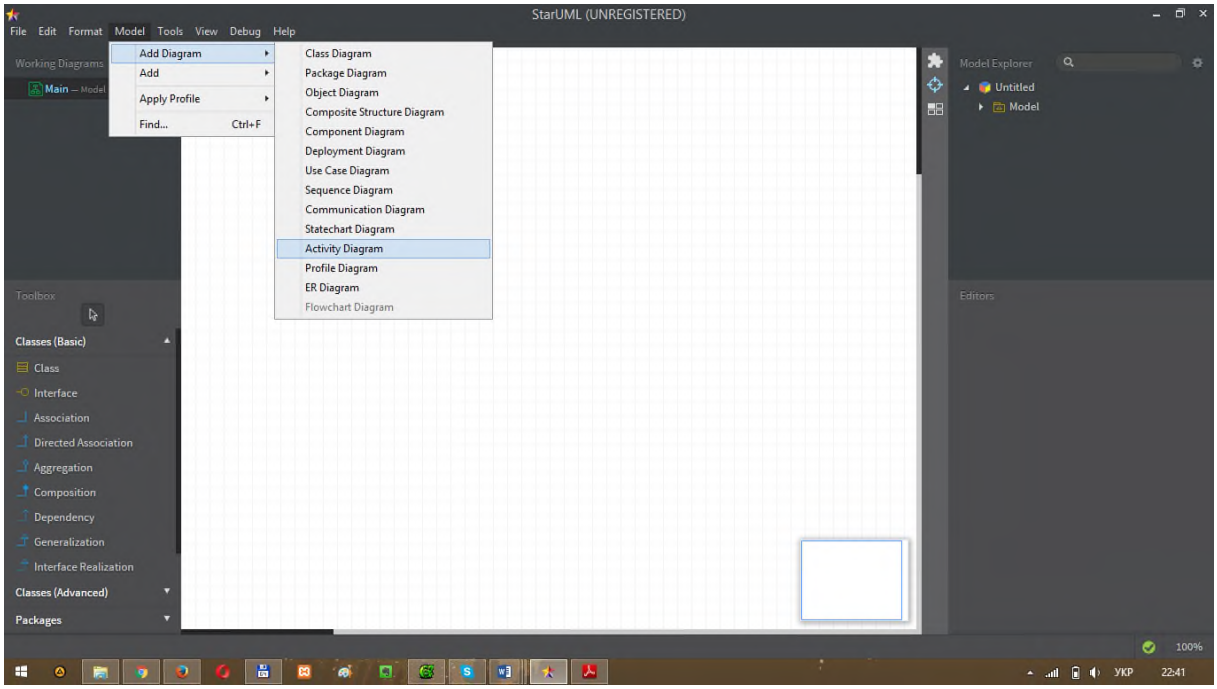


Рис.6.8. Створення діаграми діяльності

В панелі інструментів з’являються необхідні елементи, які дозволять нам створити діаграму діяльності (рис. 6.9).

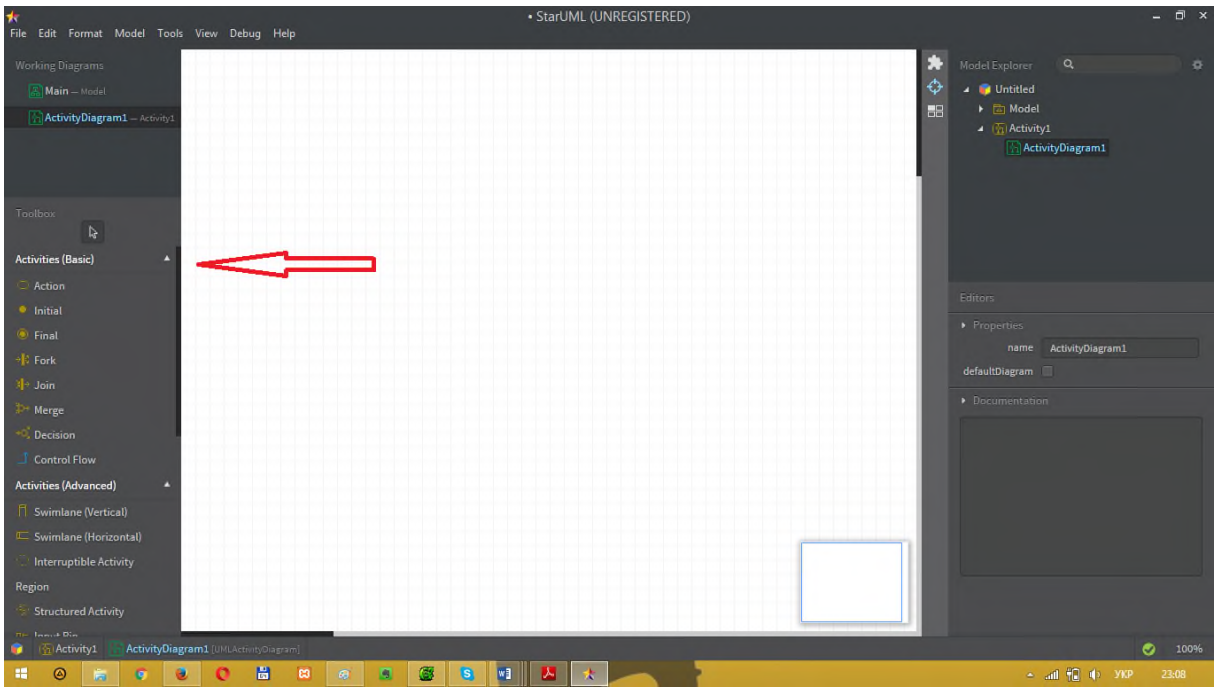


Рис. 6.9. Панель інструментів для діаграми діяльності

Основні з них це:

- **Action** (дія) - дозволяє створити дію, дія позначає якийсь крок (етап) процесу.
- **Initial** (початковий стан), **Final** (кінцевий стан) - два стани на діаграмі діяльності - початковий і кінцевий - визначають тривалість потоку. Початковий стан обов'язково має бути зазначено на діаграмі, він визначає початок потоку. Кінцевих станів може бути кілька або не одного. Він визначає точку завершення потоку. Початковий повинен бути тільки один. Початковий стан зображується жирною крапкою, а кінцевий - жирною крапкою в колі.
- **Fork** (розпаралелювання) – один потік управління розділяється на декілька, потоки виконуються паралельно.
- **Join** (синхронізація) - кілька потоків управління зливаються в один.
- **Decision** (умовний вибір) - при моделюванні керуючих потоків системи часто буває необхідно показати місця їх поділу на основі умовного вибору. Вибір на діаграмі показується ромбом.
- **Swimlane** (доріжки) - секції ділять діаграму діяльності на кілька ділянок. Це потрібно для того, щоб показати хто відповідає за виконання дії і в якому порядку.
- **Control Flow** (перехід) - перехід показує, як потік управління переходить від однієї дії до іншої.
- **Send Signal** (передача сигналу) - стан передачі сигналу.

- **Accept Signal** (прийом сигналу) - стан прийому сигналу

2.Завдання початково стану.Початковий стан визначає початок потоку дії по реалізації компонента системи. Для його активізації натискаємо **Toolbox – Initial**, і додаємо початковий стан на робочу область (рис. 6.10).

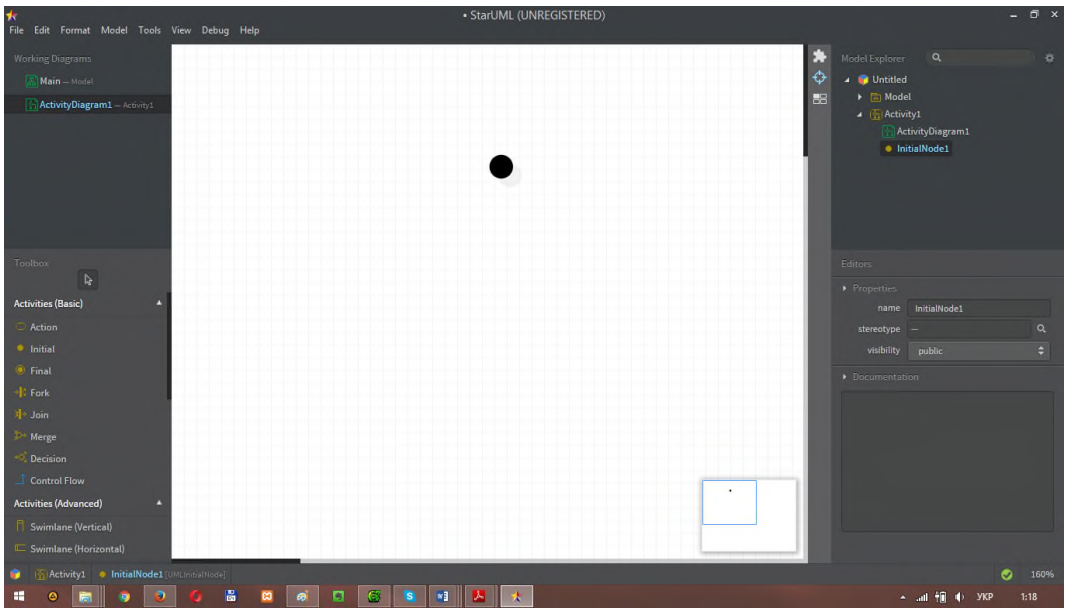


Рис.6.10.. Початковий стан

4. **Встановлення дії.** Для того щоб створити дію, необхідно в панелі інструментів в пункті **Activities (Basic)** вибрати **Action** і клікнути на робочу область. Далі необхідно назвати дію. І з’єднати початковий стан з дією, за допомогою **Control Flow** (рис. 6.11). В процксі моделювання створюються необхідні дії між якими встановлюються неохідні типи звязків.

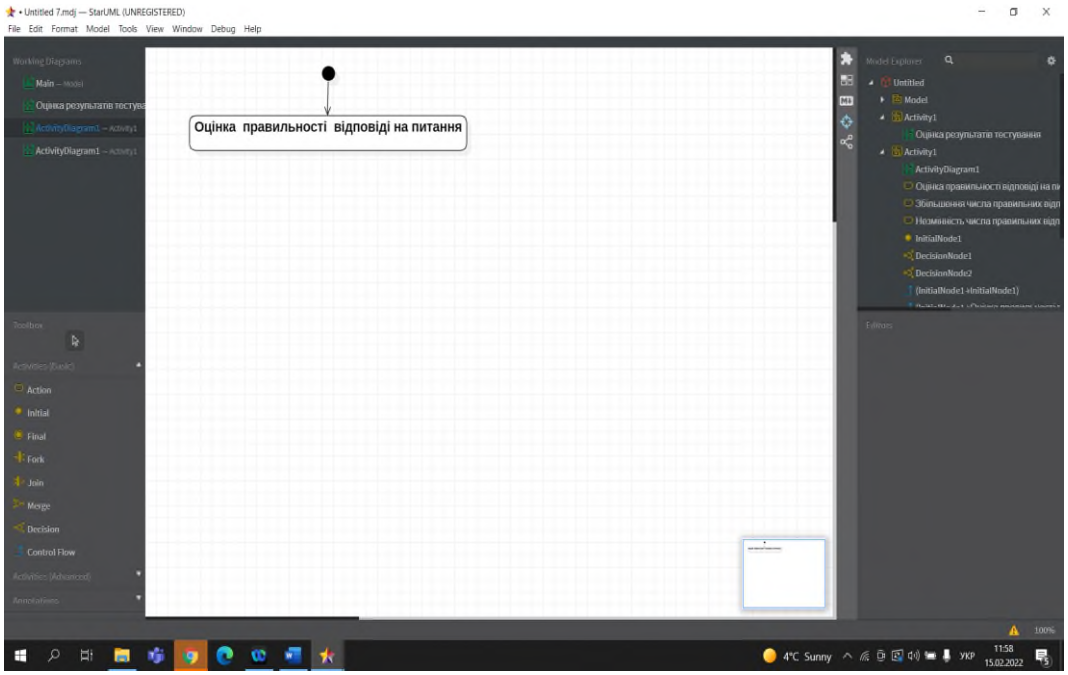


Рис. 6.11. Зєднання початкового стану з дією

5. **Розгалуження.** Для застосування розгалуження вибираємо **Toolbox - Decision** (умовний вибір) (рис.6.12), який буде визначати наші подальші дії в залежності від виконання встановленої умови вибору. Так, в нашому прикладі в процесі тестування при перевірки правильності відповіді є варіанти – відповідь правильна або неправильна.

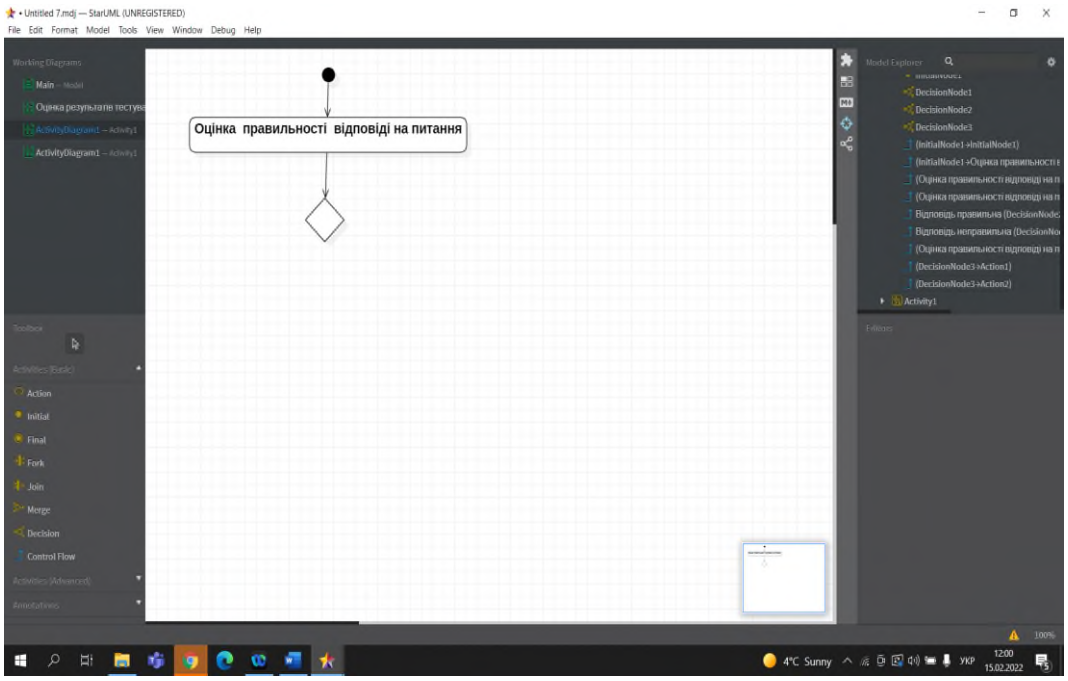


Рис. 6.12 Розгалуження

6. **Визначення альтернатив дій** в результаті розгалуження . Якщо в відповідь правильна, то ми поповнюємо у студента, що проходить тестування, число (лічильник) правильних відповідей поповнюється на одиницю , якщо ні - кількість правильних відповідь незмінна, лічильник не змінюється (рис. 6.13).

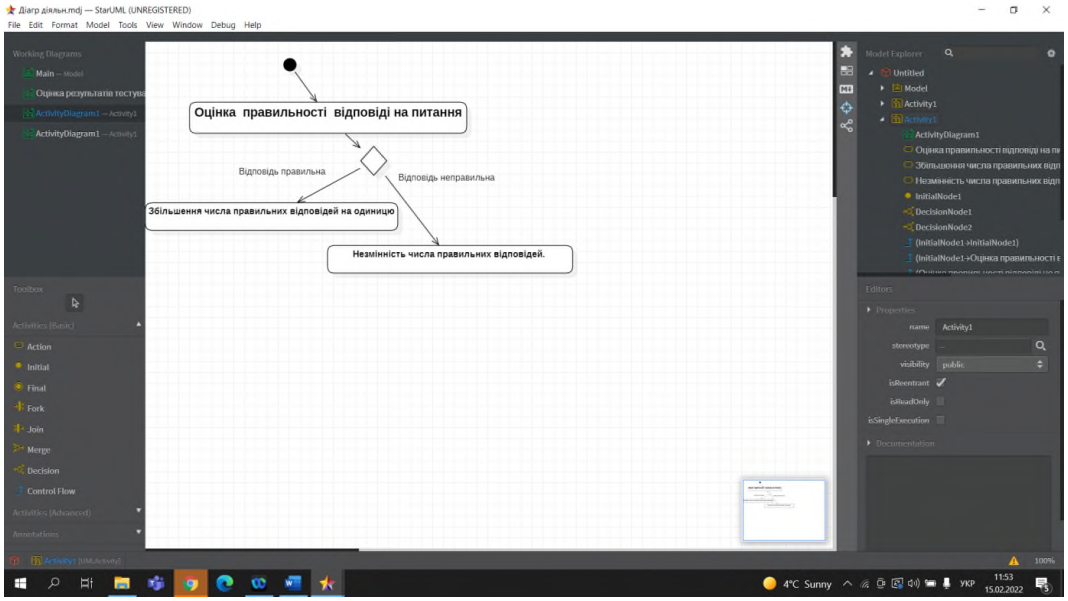


Рис.6.13 Альтернативи дій після розгалуження

7. **Встановлення розділення.** Для застосування розділення з метою моделювання паральних потоків застосовується інструмент **Fork** (розпаралелювання Ї панелі. При цьому один потік управління розділяється на декілька, потоки виконуються паралельно (рис. 6.14). Так, в нашому прикладі при виконанні дії «Регістрація викладача» можуть одночасно (паралельно) виконуватись дві дії «Введення тестів» та «Формування груп студентів».

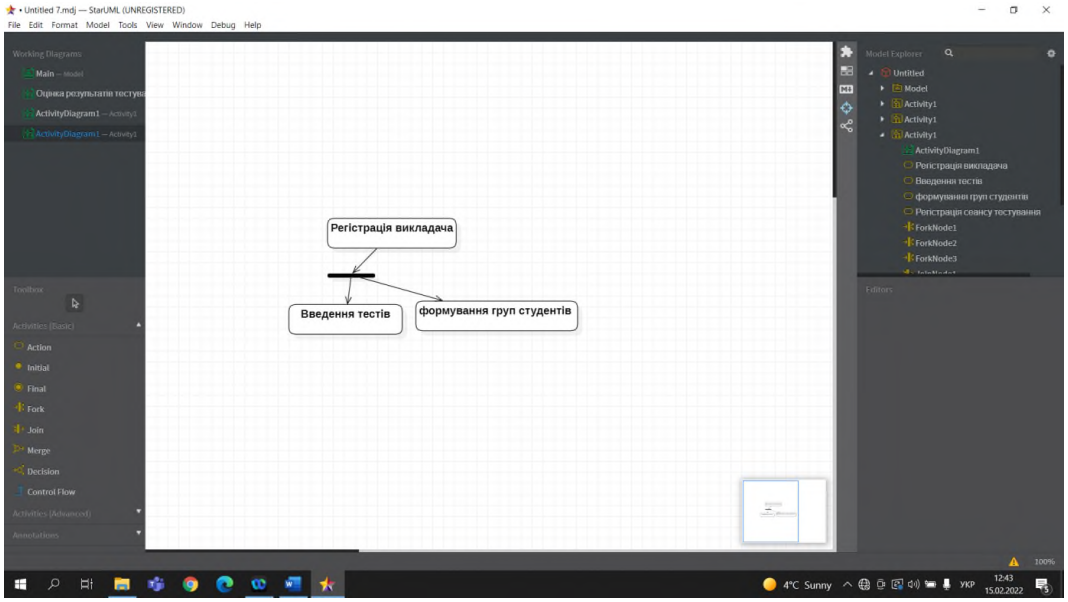


Рис. 6.14. Розділення дій

8.Злиття потоків.Застосовується інструмент Join (синхронізація) панелі - кілька потоків дій зливаються в один (рис. 6.15). Так в нашому прикладі дія «Регістрація сеансу тестування» може відбуватися тільки після завершення реалізації двох дії – «Введення тестів» та «Формування груп студентів».

9.Встановленні доріжок - Swimlane (доріжки) - секції ділять діаграму діяльності на кілька ділянок. Доріжки дозволять врахувати при побудові діаграми діяльності різні ролі виконавців. Доріжка - частина області діаграми діяльності, на якій відображаються тільки ті діяльності, за які відповідає конкретний об'єкт. Призначені вони для розбиття діаграми відповідно до розподілу відповідальності за дії. Ім'я доріжки може означати роль або об'єкт, якому вона відповідає. Для встановлення доріжок треба Для того щоб створити дію, необхідно в панелі інструментів в пункті Activities кликнути Swimlane і ввести її назву. (рис. 6.16).

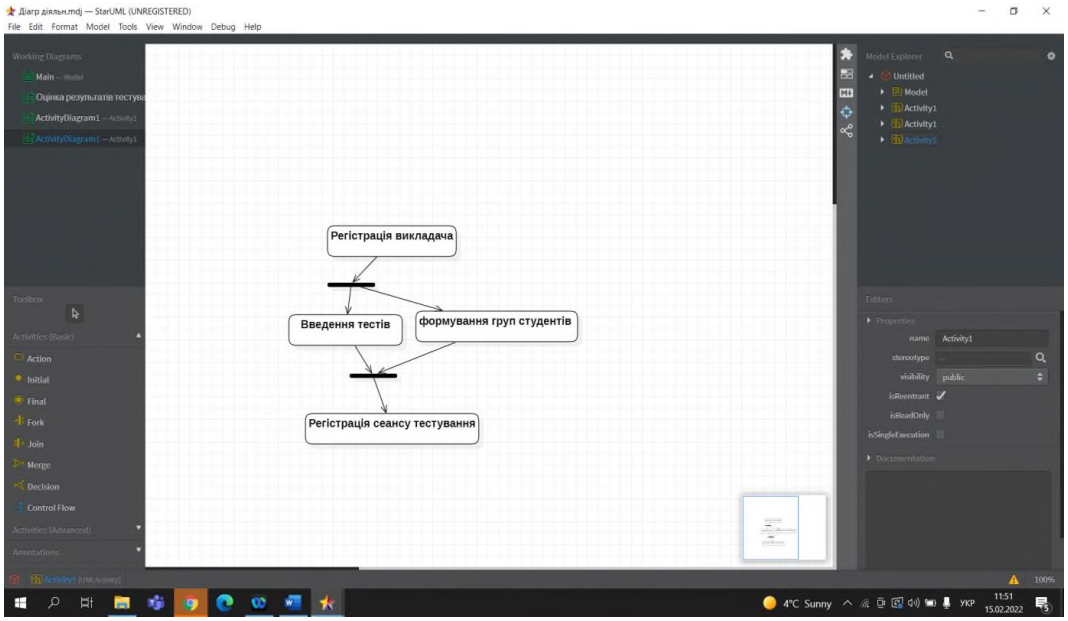


Рис.6.15 Розділення та злиття потоків

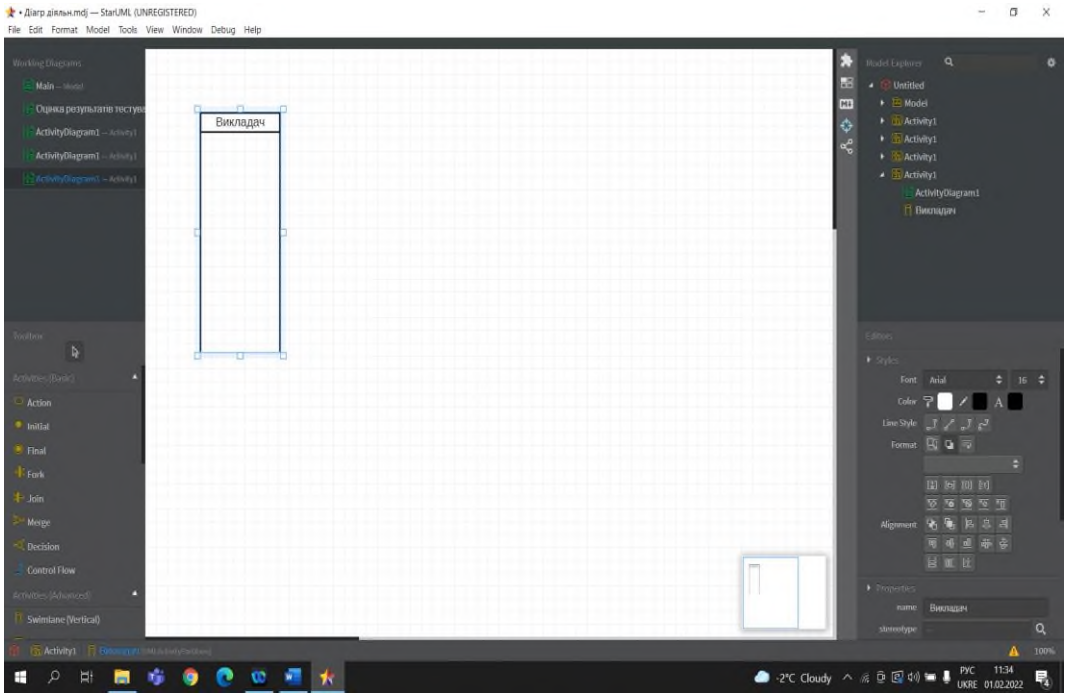


Рис. 6.16 Встановлення доріжки

На рис. 6.17 наведений приклад застосування двох доріжок між якими розподіляється хід виконання відповідних дій учасників реалізацій процесу тестування.

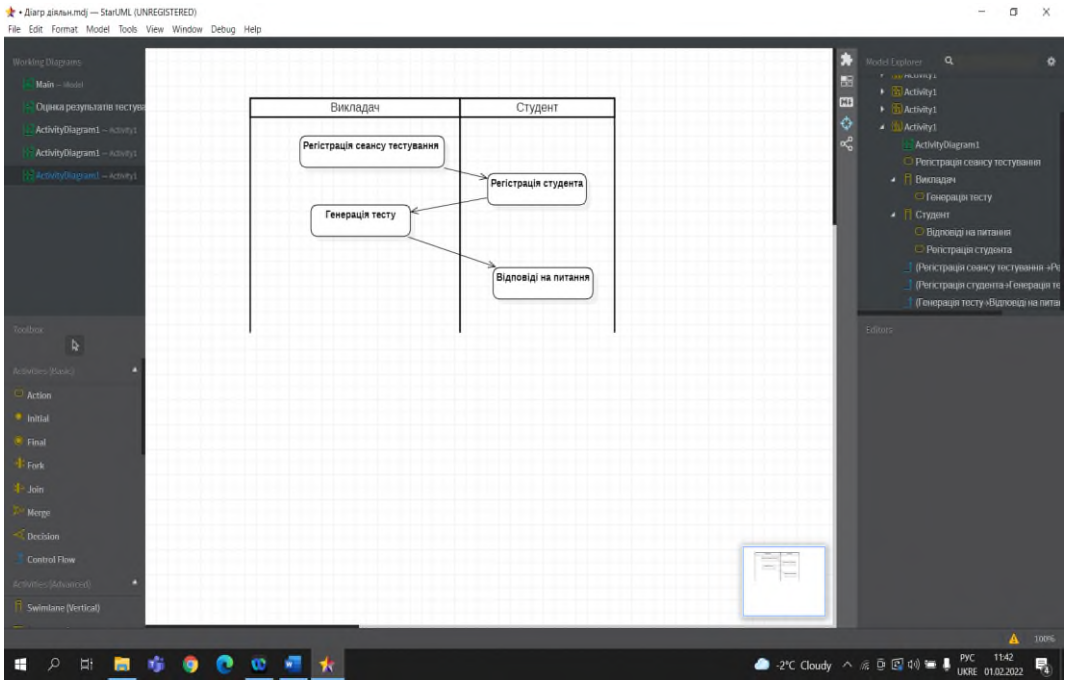


Рис. 6.17. Використання двох доріжок

10. Завдання завершального стану. Завершальний стан означає завершення дій по реалізації процесу. Для його введення на панелі інструментів використовується **Toolbox – Final**. На рис. 6.18. наведений фрагмент діаграми діяльності, де визначений завершальний стан роботи.

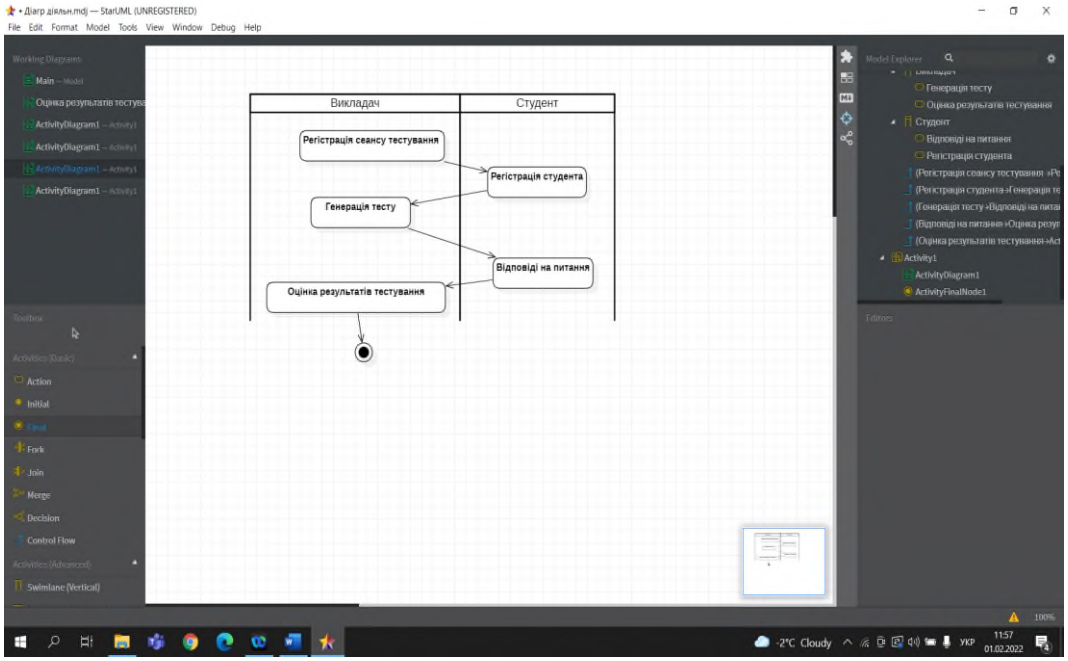


Рис. 6.18. Фрагмент діаграми діяльності

6.4. Контрольні питання до роботи

1. Поняття діаграми діяльності.

2.Призначення діаграми діяльності на різних етапах аналізу та проектування систем.

3.Компоненти діаграми діяльності: дії, переходи, рішення (розгалуження), розділення і злиття, область дії (доріжки).

Список джерел інформації

1. Богуш Б.М., Довидков О.А. Проектування захищених інформаційних систем і мереж. -К.: ДУІКТ, 2006. - 414 с.
2. Богуш В.М., Довидков О.А., Кривуца В.Г. Теоретичні основи захищених інформаційних технологій: навч. посібн. .– К. : ДУІКТ, 2010.– 454 с.
3. Буч Г., РамбоД., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК Пресс, 2007 – 489 с. (http://www.knigafund.ru/books/106240)
4. Буч Г., Якобсон А., Рамбо Дж., UML. Класика CS. 2-е вид. / Пер. з англ.; Під загальною редакцією проф. С. Орлова - СПб.: Пітер, 2006. - 736 с.
5. Грайворонський М.В., Новіков О.М. Безпека інформаційно-комунікаційних систем.-К.: Видавнича група BHV, 2009.-608с.: іл
6. Джозеф Шмуллер. Освоюй самостійно UML 2 за 24 години. Практичний посібник. - М.: "Вільямс", 2005. - 416 с.
7. Ларман К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку. – М.: Вильямс, 2013. – 736с.
8. Лешек А., Мацяшек, Аналіз та проектування інформаційних систем за допомогою UML 2.0. 3 вид.: - М.: Вільямс, 2004. - 816с.
9. Кендалл Скотт, UML. Основні концепції: - М.: Вільямс, 2002. - 144с.
10. Боггс, UML і StarUml: - М.: Лорі, 2007. - 286с.
11. Крег Ларман, Застосування UML 2.0 і шаблонів проектування. 3-є вид. - М.: "Вільямс", 2006. - 736 с.
12. Хлапонін Ю.І., Ізмайлова О.В. Системний аналіз:Посібник до виконання циклу практичних і лабораторних занять з розділу «Методології, моделі та методи структурного аналізу та проектування інформаційних систем». –К.: КНУБА, 2019. – 46с.
13. StarUml. Руководство пользователя. http://staruml.sourceforge.net › docs › user-guide

Навчальне видання

ІЗМАЙЛОВА Ольга Василівна

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Навчальний посібник

Редагування та коректура *О.В. Ізмайлова*
Комп’ютерне верстання *М.М. Власенко*

Підписано до друку 27.12.2022. Формат 60x84 1/16.
Ум. друк. арк. 5,12. Обл.-вид. арк. 2,645
Тираж 80 прим. Вид. № 16/І-17. Зам. № 15/1- 18

Видавець і виготовлювач
Київський національний університет будівництва і архітектури

Повітрофлотський проспект, 31, Київ, Україна, 03680

Свідоцтво про внесення до Державного реєстру суб’єкту
Видавничої справи ДК №808 від 13.02.2002