МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСІТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ ТА ЗВ'ЯЗКУ

Кафедра інформаційних технологій

МОДЕЛЮВАННЯ ТА АНАЛІЗ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ

Методичні вказівки до практичних та лабораторних занять з дисципліни «Моделювання та аналіз програмного забеспечення» для студентів спеціальності 121–Інженерія програмного забезпечення

\

Одеса 2021

УДК 004.43 План НМВ 2021 р.

Рецензент– д.т.н., доцент кафедри Інденерної механіка Військової академії (м. Одеса)– Ісмаілова Н.П.

Моделювання та аналіз програмного забеспечення: методичні вказівки до практичних та лабораторних занять. / Укладач: Глазунова Л.В. – Одеса:ДУІТЗ, 2021 -

Методичні вказівки містять короткі теоретичні відомості з понять і принципів аналізу та моделювання програмного забеспечення. Згідно з навчальними планами дисциплін «Моделювання та аналіз програмного забеспечення» в методичних вказівках для закріплення теоретичного матеріалу також надані приклади застосування методів аналізу та моделювання різноманітних аспектів представлення розробляємої програмної системи. Для набуття практичних навичок моделювання за допомогою структурного та об’єктного підходів було розглянуто застосування CASE засобів BPwin, ARIS та StarUML. Велика увага приделяється створенню діаграми класів для відображення об’єктів системи, її структурування в рамках взаїмодії та поведінки об’єктів. Для закріплення практичних навичок студенти виконують аналіз та моделювання власної автоматизированої інформаційної системи. Методичні вказівки будуть корисні студентамb для закріплення лекційного матеріалу, а також при підготовці до практичних занять та виконанні лабораторних робіт.

Методичні вказівки призначено студентам для здобуття теоретичних і практичних знань зі спеціальності 121 – Інженерія програмного забезпечення.

СХВАЛЕНО
на засіданні кафедри
Інформаційних технологій
та рекомендовано до друку.
Протокол № 1
від 27 серпня 2021 р.

ЗАТВЕРДЖЕНО
методичною радою
ДУІТЗ.
Протокол № 1
від вересня 2021 р.

ТЕМА 1. АНАЛІЗ ВИМОГ ТА ПОБУДОВА ЦІЛІВОЇ МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ.

Мета: набути практичних навичок виявлення бізнес та користувацьких вимог для своєї предметної області.

1.1 Короткі теоретичні відомості з аналізу вимог до АІС

Перш, ніж розбирати докладно безпосередньо сам документ вимог коротко розглянемо місце і роль цього документа в усьому процесі розробки програмного забезпечення (ПЗ). Життєвий цикл розробки ПЗ в спрощеному вигляді виглядає так, як це представлено на рис. 1.1

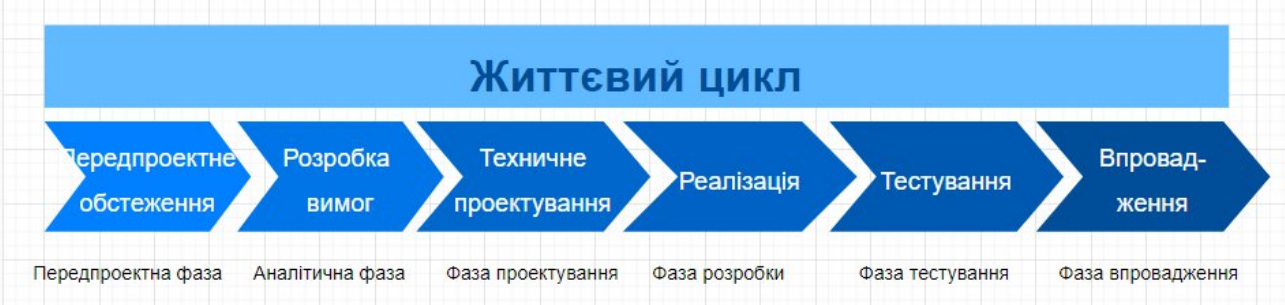


Рисунок 1.1 Спрощене уявлення життєвого циклу розробки ПЗ

Життєвий цикл (ЖЦ) включає в себе фази створення програмного забезпечення, кожна з яких може виконуватися ітераційно. Крім того, сам життєвий цикл не є строго лінійним, оскільки існує досить велика кількість зворотних зв'язків, що впливають на весь хід проекту.

Крім основних процесів ЖЦ, представлених на рис.1.1, існують так звані процеси управління, що підтримують і супроводжують весь цикл розробки ПЗ: управління вимогами, управління проектом в цілому, конфігураційне управління, управління ризиками і т.п. Однак ці процеси знаходяться за межами розглянутої нами теми.

З рис. 1.1 явно видно, що розробка вимог є відправною точкою всього процесу створення програмного забезпечення. Дійсно, маючи на вході концептуальне уявлення про продукт, на виході аналітичної фази проектна команда отримує документ вимог, що описує продукт настільки детально, як це необхідно для подальшого проектування ПЗ.

Таким чином, документ вимог представляється своєрідним "коренем" і породжує ціле дерево інших проектних документів: документ архітектури ПЗ, опис UI, модель бази даних, плани реалізації, плани тестування, руководство користувача та адміністратора.

Виходячи з цього можна стверджувати, що якісно розроблений документ вимог дозволить уникнути непотрібних переробок вже готової системи, знизити вартість і швидкість безпосередньо розробки та тестування, не допустити розповзання меж проекту і, нарешті, домогтися більшої задоволеності замовників.

Верхньо рівнева структура документу вимог визначається структурою вимог, виявлення яких є необхідною умовою успішного проектування АІС рис.1.2

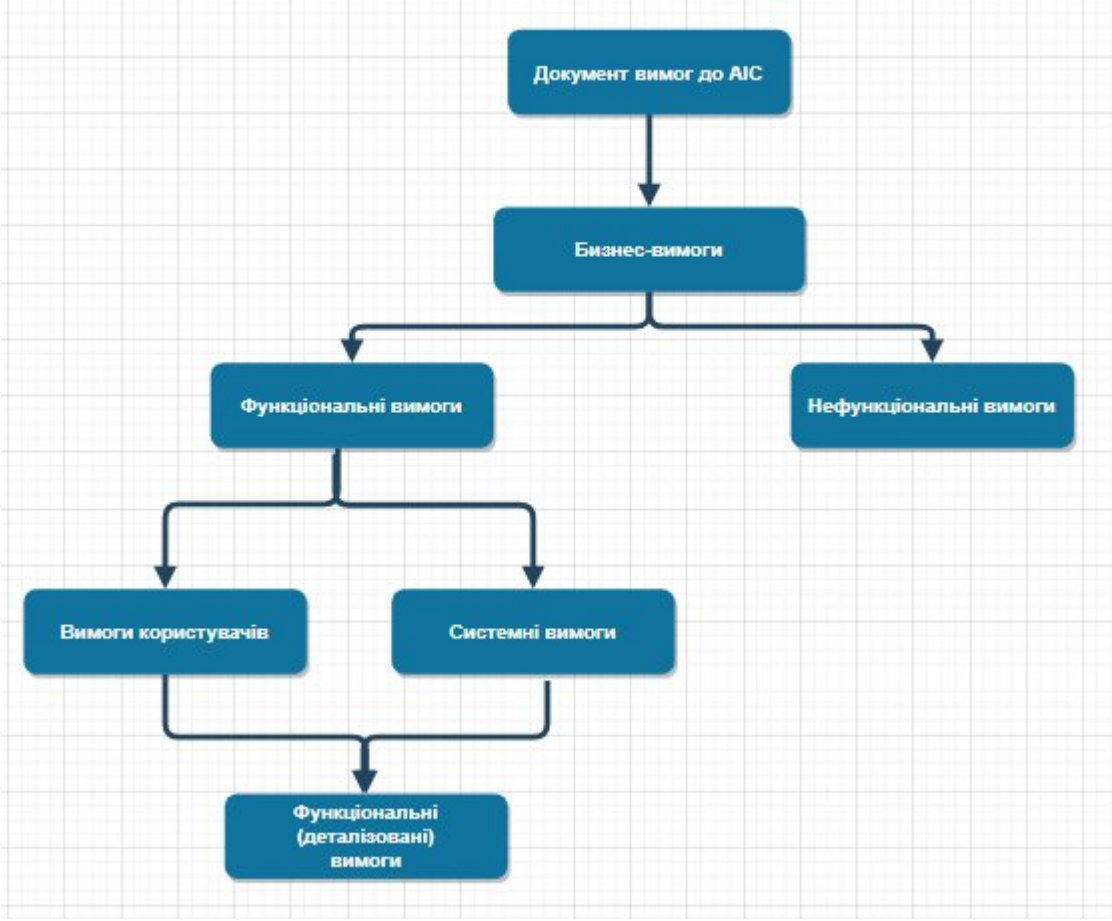


Рисунок 1.2 – Верхньо рівнева структура документу вимог до АІС.

Основою для початку визначення вимог є ***цілі проекту***. Однозначне трактування цілей дозволяє сформувати однозначне бачення Замовником і Розробником того ефекту, який буде отриманий бізнесом Замовника в результаті створення і впровадження проектованого ПЗ. Цілі проекту служать свого роду

точкою зору зацікавлених сторін на досягнуті в ході реалізації проекту результати.

Цілі створення продукту є основою для визначення ***бізнес-вимог*** до системи (Business requirements). Це найперший, самий верхній рівень документа вимог. ***Бізнес-вимоги*** описують, ЯКИМ ЧИНОМ система, яка розробляється, пов'язана з досягненням бізнес-цілей організації і ЩО вона повинна для цього робити. Можна сказати, що бізнес-вимоги є свого роду завданнями, які повинна вирішувати АІС для досягнення цілей свого створення.

Як видно зі схеми, представленої на рис. 2, бізнес-вимоги визначають функціональні і нефункціональні вимоги. На практиці може вийти так, що функціональні і нефункціональні вимоги можуть "заходити" за кордон бізнес-вимог, однак вони ніколи не повинні їм суперечити.

До ***функціональних вимог*** відноситься все те, що описує функціональність системи. Тобто, ЩО, ЯК і КОЛИ робить система. А також, ХТО приймає в цьому участь.

Нефункціональні вимоги - це характеристики системи, які не мають прямого відношення до процесів, які автоматизуються, але, тим не менш, без яких працювати з системою було б, м'яко кажучи, проблематично. Як приклад, сюди можна віднести вимоги по зручності у використанні, продуктивності, масштабованості, вимоги до документування, до організаційного та методичного забезпечення, до інтеграції із зовнішніми системами.

Функціональні вимоги включають в себе, так звані, вимоги користувачів і системні вимоги. ***Вимоги користувачів*** (Customer requirements) - це ті вимоги верхнього рівня, які пред'являють до системи майбутні бізнес-користувачі системи. Іншими словами, ті завдання, які система повинна вирішувати з точки зору бізнесу Замовника. Як приклад можна привести такі вимоги, як "Система повинна дозволяти створювати нові договори", "Система повинна дозволяти змінювати статус існуючого договору", "Система повинна дозволяти створювати нове додаткову угоду до існуючого договору" і т.д. Ще раз необхідно підкреслити, що ***вимоги користувачів описують верхнеуровневую функціональність системи, не вдаючись у подробиці її технічної реалізації***. ***Системні вимоги*** (System requirements) мають рівно такий же сенс, що і призначені для користувача вимоги, з тією різницею, що в більшій мірі продиктовані не потребами бізнесу, а особливостями IT -інфраструктури Замовника. Тобто, це вимоги, які визначають регламент взаємодії створюваного ПЗ з зовнішніми системами, регламентують програмно-апаратні платформи

функціонування продукту і т.п. Наприклад, "Система повинна мати можливість здійснювати автоматичний обмін даними з системою обліку персоналу".

У той час як вимоги користувачів і системні вимоги, як правило, відповідають на питання: ЩО повинна робити система, то ***деталізовані функціональні вимоги*** (Functional requirements) відповідають на питання: ХТО, ЯК і КОЛИ. Вимоги цього типу визначаються на основі аналізу і детального опису процесів, озвучених у вимогах верхнього рівня - призначених для користувача і системних. З усіма відповідними атрибутами - функціями, умовами, подіями, виконавцями, входами і виходами. Тобто, фактично, є детальний опис реалізованих системою потоків завдань і даних.

Користувацькі історії та концептуальні сценарії потрібні для розуміння основних мотивів користувачів і поглиблення в світ клієнта. Користувацькі історії та концептуальні сценарії оформлюються у вигляді ***сценарієв використання***.

Хоча сценарії використання є одним з найпопулярніших методів для опису функціональності систем, вони також використовуються для вивчення їх не функціональних характеристик. Найпростіший спосіб це зробити - зафіксувати їх в рамках самих сценаріїв використання. Наприклад, зв'язати вимоги до продуктивності та часу між конкретними кроками сценарію використання або перерахувати рівень очікуваного обслуговування для сценарію використання як частина його самого.

В ***осносної діючої особи*** є мета, система повинна допомогати діючій особі досягти цієї мети. Деякі сценарії призводять до досягнення заданої мети інші закінчуються, якщо мета виявляється недосяжною. Кожен сценарій містить послідовність кроків, що показують, як розкриваються дії і взаємодії. Варіант використання збирає всі ці сценарії разом, показуючи всі шляхи, які можуть привести або не привести до мети.

Варіант використання для діючієї особи містить набір можливих сценаріев для досягнення мети і повинні вдовольняти наступним критеріям:

— Всі взаємодії мають відношення до однієї і тієї ж мети однієї і тій ж основної діючої особи.

— Варіант використання починається з ініціірующей події і продовжуєся, доки мета не досягнута або її досягнення не перервано і система звільняється від своїх зобов'язань по відношенню до даного взаємодії.

Щоб задовольнити інтереси учасників, повинні бути описані три види дій варіанта використання як згода на поведінку:

— Взаємодія між двома діючими особами (щоб содіяти досягненню мети)
— Перевірка достовірності (щоб захистити учасника)
— Зміна внутрьошного стану (від імені учасника)

Модель Учасники та інтереси вносить лише невеликі зміни в загальну процедуру створення варіанту використання: перелік учасників та їх інтересів і застосування цього переліку в якості подвійної перевірки для гарантіі, що ніщо не пропущено в тексті варіанти використання. Ця зміна істотно впливає на якість варіанти використання.

Функціональна область дії відноситься до послуг, які надає ваша система і які врешті-решт будуть зафіксовані в варіантах використання. Оскільки ви тільки починаєте розробляти проект, цілком ймовірно, що ви не знаєте деталей. Ви визначаєте функціональну область дії одночасно зі створенням варіантів використання, ці два завдання переплетені. Список Усередині / Поза допомогає вирішити ці завдання, так як дозволяє провести граніцу між тим, що всередині області дії і тим, що поза цією областю. Два інших інструмента це список Дійова особа / Мета і короткий опис варіантів використання.

Кожен варіант використання повинен мати мітку його області дії проектування. Припустимо, що компанія MyTelCo проектує систему NewApp, яка включає підсистему Searcher. Нижче наведені імена областей дії проектування:

– ***Підприємство*** (MyTeICO). Ви обговорюєте поведінку цілої організаціі або підприємства для досягнення мети головної діючії особи. Запишить в поле Область дії варіанти використання назву організаціі (MyTelCo), а не просто "компанія". Обговорюючи відділ, вживайте його назву. Варіанти використання для бізнесу пишуться в області дії підприємства.
— ***Система*** (NewApp). Це обладнання або програмне забезпечення, яке вам доручили розробляти. Поза цією системою перебувають всі частини обладнання, програмного забезпечення та людських ресурсів, з якими система повинна взаємодіяти.
— ***Підсистема*** (Seacher). Ви розкрили головну систему і збираєтесь роздивитися, як працює її частина.

Користувацькі історії та концептуальні сценарії потрібні для розуміння основних мотивів користувачів і занурення в світ клієнта.

Конкретні сценарії і варіанти використання вже можуть використовуватися для проектування інформаційної архітектури та інтерфейсів, а також при проведенні тестів і досліджень юзабіліті.

Три рівня цілей:

- узагальнені
- призначені для користувача
- подфункции

Шаблон варіанта використання:

- Передумова
- Мета
- З точки зору
- Область дії
- Бізнес процес
- Альтернативний процес

Передумова варіанти використання оголошує, виконання якої умови гарантує система перед тим, як дозволити запуск цього варіанту використання. Так як умова виконується системою, і відомо, що воно істинне, то воно не буде перевірятися знову в період виконання варіанта використання. Загальновідомий приклад: користувач вже увійшов в систему і система ідентифікувала его.

Найбільший інтерес представляє ***мета користувача***. Це та мета, яку переслідує основна діюча особа, намагаючись домогтися від системи виконання певної роботи, або користувач, що працює з системою. Вона відповідає "елементарному бізнес-процесу" в технологіі бізнес процесів. Мета користувача характеризується питанням: "Чи буде основна діюча особа задоволена, виконавши це?" Найкоротший виклад функцій системи - це список цілей (задач) користувача, які вона підтримує. Це основа для розстановки пріоритетів, здачі системи, поділу робіт, оцінки та розробки. Такі цілі як "Здійснити покупку на онлайновому аукціоні" і "Увійти в сістему" не рахуються цілями користувача. Онлайнові аукціони вимагають кілька днів і тому не проходять односеансовий тест. Вхід в систему 32 рази поспіль не відповідає (як правило) посадовими обов'язками окремих людей чи цілі застосування системи. "Зареєструвати нового клієнта" і "Купити книгу" цілком можуть бути цілями користувачів. Завдання реєстрації 32 нових клієнтів не позбавлена сенсу для менеджера з продажу. Покупку книги можна зробити за один сеанс.

Система характеризується підтримкою цілей користувача. Ось одна з таких цілей: Ви службовець, що сидить на робочому місці. Дзвонить телефон, ви

берете трубку. Хтось надругом кінці говоріт: "...". Ви повертаєтеся до комп'ютера. У цей момент ви думаєте про те, що необхідно виконати роботу G. Ви деякий час працюєте з комп'ютером і клієнтом і, нарешті, завершуєте роботу G. відвертається від комп'ютера, прощаєтеся і покладаєте трубку.

Цілі узагальненого рівня включають кілька цілей користувача. При опису системи вони виконують три призначення:

- Показують контекст, в якому виконуються цілі користувача.
- Показують життєвий цикл послідовності пов'язаних цілей.
- Формують перелік варіантів використання більш нізкого рівня.

Узагальнені варіанти використання зазвичай виконуються протягом годин, днів, тижнів, місяців або років. Тут представлений основний сценарій "довгоіграющого" варіанту використання, завдання якого зв'язати користувацьки варіанти використання, розкидані по годам.

Цілі рівня підфункції - це цілі, досягнення яких потрібно для реалізації цілей користувачів. Включайте їх тільки в міру необхідності. Вони іноді потрібні для легкого прочитання або тому, що їх застосовують багато інших варіантів використання. Приклади варіантів використання рівня подфункции: Знайти продукт, Знайти замовника і Зберегти в файлі.

Спосіб оцінити рівні цілей - подивитися на довжину варіанта використання. Більшість добре написаних варіантів використання мають від двох до восьми кроків.

Набор варіантів використання - це історія переслідування основними дійовими особами своїх цілей, яка безперервно розгортається. Кожен варіант використання має ветвящуюся сюжетну лінію, яка показує, як система забезпечує досягнення мети або отвергает її. Ця сюжетна лінія представлена основним сценарієм і набором фрагментів сценарію в якості розширень до нього. Кожен сценарій або фрагмент стартує при виконанні умови запуску, яке вказує, коли відбувається запуск, і залишається справжнім, поки сценарій не завершиться або мета не буде отвергнута. Всі цілі різні за складністю, тому ми використовуємо однакову форму для опису процесу досягнення мети будь-якої складності на будь-якому рівні сценарію.

1.2 Аналіз предметної області: бізнес та користувацькі вимоги.

Завдання: побудувати цільові моделі бізнес процесу та вимог використання користувачів для співробітників фірми з ремонту побутової техніки.

Побудова цільової моделі бізнес процесу

1. Спочатку ми створюєм опис одного простого для розуміння і типичного сценарія, в якому досягається бізнес-мета і задовольняються інтереси всіх учасників, та виявляємо процеси , які треба віднести до розробляємої інформаційної системи. Це основний сценарій.

Концептуальна бізнес модель: клієнт приносить б.т. для ремонту на фірму «РемПобТех», яка заказує необхідні деталі для ремонту, виконує ремонт, отримає оплату та повертає б.т. клієнту. Вхідні дані: замовлення клієнта, запасні частини, оплата. Вихідні дані:виконане замовлення.

Передумова: клієнт приносить поломану побутову техніку (б.т.) на фірму по ремонту «РемПобТех»

Мета: якісно відремонтувати б.т. клієнта для отримання прибутку

З точки зору: бізнесмена (господаря бізнесу)

Область дії: фірма «РемПобТех»

Бізнес процес (основний сценарій): внутри +/вне -

1 Відділ роботи з клієнтами

1.1 Адміністратор оформлює заяву клієнта на ремонт б. техніки +

1.2 Адміністратор приймає б.т. і видає квітанцію клієнту +

1.3 Адміністратор передає б. т. у відділ Діагнастики та тестування +

2 Відділ Діагностики та тестування

1.3 Мастер встановлює причину поломки, -

1.4 Мастер оцінює вартість ремонту та термін ремонту, створює накладну на ремонт +

1.5 Мастер передає б. т. у відділ Ремонту після узгодження вартості ремонту з клієнтом. +

3 Відділ ремонту

3.1 Мастер замовляє на Складі необхідні запчастини +

3.2 Мастер виконує ремонт б.т., -

3.3 Мастер відремонтовану б.т. передається у відділ Діагнастики та тестування для тестування, +

4 Відділ Діагностики та тестування

4.1 Мастер виконує тестування б/т -

4.2 Мастер передає б/т Адміністратору +

5 Відділ роботи з клієнтами

5.1 Адміністратор приймає оплату за ремонту б.т., +

5.2 Адміністратор передає її клієнту по квітанції, продемонстрував її -

працездатність.

5.3 Адміністратор фіксує термін виконання +

Альтернативна поведінка:

2.3 клієнт не згоден з ціною, Адміністратор повертає б.т. клієнту +

3.1 на Складі немає деталей для ремонту

3.1.1 Адміністратор зсуває ремонт в термінах до отримання деталей +

3.1.2 Адміністратор повернути б.т. клієнту +

4.1 тестування не пройдено, Мастер відділу Діагностики та тестування повертає б.т. в відділ Ремонту +

5.4 термін виконання превищено, Адміністратор нараховує штраф на Майстера по ремонту +

1.3 Визначення цілей та варіатів використанн для користувачів АІС

Концептуальна модель для АІС «RepairTech» дозволяє виявити наступні основні діючі особи (користувачів):

Адміністратор – обслуговування клієнта

Мастер з діагностики – діагностування та тестування

Мастер з ремонту – ремонт б.т.

Цілі користувачів можливо зафіксувати у вигляді таблиці «Задачі користувачів»

Задача	Вхідна інформація	Вихідна інформація
1 Адміністратор		
1.1 Створення замовлення	Клієнт номер пристрою причина поломки дата прийому дата повернення майстер діагностики статус заяви	
1.2 Прийом б.т. і видача квітанцію клієнту	номер замовлення	квітанція
1.3 Підтверження вартості ремонту	номер замовлення	Статус=ремонт
1.4 Перегляд списку заяв на ремонт б/т по фільтру	проміжок дат/кліент/ тип прилада/статус	список заяв
1.5 Перегляд заяв, виконання яких було затримано		список заяв

1.6 Видача відремонтованої техніки клієнту	Номер замовлення Статус Дата виконання	
1.6 Визначення прибутку фірми	Проміжок часу	прибуток фірми
2 Мастер по діагностики та тестуванню		
2.1 Створення накладної на ремонт б/т	Заява Тип поломки Дата виконання вартість ремонту	
2.2 Створення звіта про тестування	накладна результат	
2.3 Призначення майстра з ремонту	Номер накладной статус	
2.4 Переглядає свого активного списоку накладних на ремонт	статус≠ «готова»	
3 Майстер по ремонту		
3.1 Створення списку деталей для ремонту	Накладна Назва деталі Кількість ціна	
3.2 Зміна статусу в накладній по ремонту	Накладна, статус	
3.3 Переглядає свого активного списоку накладних на ремонт	Майстер,накладна	Список накладних

Опис варіантів використання рівня системи для АІС «RepairTech» в рамках основного сценарію.

Специфікація варіанту використання для актора «Адміністратор»	«Оформлення замовлення»
Контекст використання	Прийом б/т від клієнта для ремонту
Дійові особи	актора «Адміністратор»
Передумова	вхід в систему
Тригер	клієнт приніс поломану б/т
Сценарій	1) Створити замовлення (клієнт, тип б/т, назва, номер,що не робить, дата) 2) Додати інформацію 3) Призначити майстра з діагностики за правилами фірми 4) Зберегти замовлення

	5) Роздрукувати квітакцію
Постумова	Створено замовлення від клієнта з необхідною інформацією

Специфікація варіанту використання для актора «Адміністратор»	«Підтверження вартості ремонту»
Контекст використання	Отримати підтверженя клієнта на суму ремонту
Дійові особи	актора «Адміністратор»
Передумова	вхід в систему створена накладна на ремонт
Тригер	згода клієнта на визначену вартість ремонту
Сценарій	1) Ввести номер замовлення 2) Змінити статус на «ремонт» 3) Зберегти
Постумова	В замовленні змінен статус на «ремонт»

Специфікація варіанту використання для актора «Адміністратор»	«Видача б/т клієнту»
Контекст використання	Видача б/т клієнту після ремонту
Дійові особи	актора «Адміністратор»
Передумова	вхід в систему квітанція на б/т
Тригер	Значення статусу в замовленні є «готова» Клієнт прийшов за б/т
Сценарій	1) По квітанції ввести номер замовлення 2) Приймає оплату та змінює статус на «видана» 3) Фіксує дату видачі 4) Зберегти
Постумова	У замовлення змінений статус на «видана»

Специфікація варіанту використання для актора «Майстер з діагностики»	«Оформлення накл_ремонт»
Контекст використання	Виявлення поломок б/т
Дійові особи	актора «Майстер з діагностики»
Передумова	вхід в систему
Тригер	Замовлення зі статусом «діагностика»
Сценарій	1) Перегляд нових замовлень

	2) Створити накладну з полями id замовлення, причина поломки, вартість ремонту, дата виконання 3) Призначити майстра з ремонту 4) Зберегти накладну
Постумова	Створено накладну на ремонт

Специфікація варіанту використання для актора «Майстер з діагностики»	«Передача б/т на ремонт»
Контекст використання	передати б/т на ремонт
Дійові особи	актора «Майстер з діагностики»
Передумова	вхід в систему
Тригер	Согласована вартість ремонту з клієнтом
Сценарій	1) Ввести id замовлення 2) Змінити статус на «ремонт» 3) Зберегти накл_прийому
Постумова	Змінено значення статусу на «ремонт»

Специфікація варіанту використання для актора «Майстер з діагностики»	«Тестування б/т»
Контекст використання	Протестувати роботоздатність б/т
Дійові особи	актора «Майстер з діагностики»
Передумова	вхід в систему
Тригер	Значення статусу заяви «тестування»
Сценарій	1) Ввести id заяви 2) Змінити статус на «готова» 3) Зберегти заяву
Постумова	Змінено значення статусу на «готова»

Специфікація варіанту використання для актора «Майстер з ремонту»	«Замовлення запчастин»
Контекст використання	замовити запчастини для ремонту б/т
Дійові особи	актора «Майстер з ремонту»
Передумова	вхід в систему
Тригер	Накладна з результатами діагностики
Сценарій	1) Створити новий список запчастин 2) Ввести значення

	3) Зберегти список
Постумова	Створено список запчастин

Специфікація варіанту використання для актора «Майстер з ремонту»	«Передача на тестування»
Контекст використання	Передати відремонтовану б/т на тестування
Дійові особи	актора «Майстер з ремонту»
Передумова	вхід в систему
Тригер	Завершення ремонту
Сценарій	1. Ввести id замовлення 2. Змінити статус на «тестування» 3. Зберегти
Постумова	Змінено статус на «тестування»

Питання для самоконтролю

1. Надати визначення каскадної, спіральної та гнучкої моделей життевого циклу.
2. Навети приклади застосування каскадної та гнучкої моделей ЖЦ.
3. Яка структура вимог для АІС?
4. Що таке користувацьки вимоги та варіан використання?
5. Що таке бізнес сценарій та сценарій використання?
6. Що таке фаза «Аналіз вимог»? Яка структура цієї фази?
7. Чим відрізняються функціональні від нефункціональних вимог?
8. Які документи (артефакти) створюються на даній фазі? Яким етапам розробки вимог вони відповідають?
9. Яку роль відіграє визначення мети проекту та цілей користувачів?
10. Визначити поняття «варіант використання» та «сценарій користувача». Як ці поняття пов’язані?
11. В чому полягає модель «Учасники та інтереси»?
12. Описати шаблон для створення сценарія користувача.

Завдання до лабораторної роботи №1

1. Обрати та обосновати модель ЖЦ для своєї АІС таблиця 1.1
2. Створити концептуальну модель для своєї АІС.
3. Створити діаграму «риб’я кість» можливостей вашої АІС.

4. Створити таблицю задач користувачів для своєї АІС.
5. Створити варіанти використання для користувачів АІС (3 найбільш складних).
6. Оформити протокол до лабораторної роботи.

Таблиця 1.1

№ вар.	Завдання
1	Розробити АІС для центру занятості.
2	Розробити АІС для інспектора відділу кадрів університету.
3	Розробити АІС для працівників автостоянки.
4	Розробити АІС для інвестиційної компанії.
5	Розробити АІС для працівників автобусного парку.
6	Розробити АІС для мережі готелей.
7	Розробити АІС для фірми по автопрокату.
8	Розробити АІС для працівників складу мережі комп’ютерних магазинів.
9	Розробити АІС для співробітників фітнес клубу.
10	Розробити АІС туристичного агенства.
11	Розробити АІС для копіювального центру.
12	Розробити АІС для співробітників інтернет-клубу.
13	Розробити АІС електроної картотеки для пацієнтів поліклініки.
14	Розробити АІС агенства нерухомості.
15	Розробити АІС мережі АЗС.
16	Розробити АІС шкільний «Щоденик».
17	Розробити АІС для центру дозвілля.
18	Розробити АІС обліку сімейного бюджету.
19	Розробити АІС для логістичної компанії (морських вантажоперевезень).

№ вар.	Завдання
20	Розробити АІС для міні пекарні.
21	Розробити АІС розклад занять факультету ВФН.
22	Розробити АІС «Гуртожиток».
23	Розробити АІС «Домашній кухар».
24	Розробити АІС букмекерської контори.
25	Розробити АІС для мережі магазинів побутової техніки.
26	Розробити АІС для клубу знайомств.
27	Розробити АІС пошуку попутчиків (типу blablacar).
28	Розробити АІС «Міський транспорт».
29	Розробити АІС «Мультімідійна бібліотека» (фото, відео, аудіо).
30	Розробити АІС «Ресторан».
31	Розробити АІС для співробітників телеканалу.
32	Розробити АІС для ОСББ.

ТЕМА 2. СТУКТУРНЕ МОДЕЛЮВАННЯ ПЗ. МЕТОДОЛОГІЇ ER. НОТАЦІЯ МАРТІНА.

Мета: набути практичних навичок моделювання інформаційних аспектів представлення системи в рамках структурного підходу.

2.1 Короткі теоретичні відомості за методологію ER.

Елементи ER-моделі

Сутність (Entity) - реальний або уявний об'єкт, що має істотне значення для розглянутої предметної області, інформація про який підлягає зберіганню. Кожна сутність повинна мати унікальний ідентифікатор. Кожен екземпляр сутності повинен однозначно ідентифікуватися і відрізнятися від всіх інших примірників даного типу сутності.

Кожна сутність повинна володіти деякими властивостями:

- повинна мати унікальне ім'я, і до одного і того ж імені повинна завжди застосовуватися одна й та ж інтерпретація. Одна і та ж інтерпретація не може застосовуватися до різних іменах, якщо тільки вони не є псевдонімами;
- володіє одним або декількома атрибутами, які або належать сутності, або успадковуються через зв'язок;
- володіє одним або декількома атрибутами, які однозначно ідентифікують кожен екземпляр сутності;
- може володіти будь-якою кількістю зв'язків з іншими сутностями моделі.

Примірник сутності - це конкретний представник даної суті. Наприклад, представником суті "Співробітник" може бути "Співробітник Іванов". Примірники сутностей повинні бути помітні, тобто суті повинні мати деякі властивості, унікальні для кожного екземпляра цієї сутності.

Атрибут сутності - це іменована характеристика, що є деяким властивістю сутності. Найменування атрибута повинно бути виражено іменником в однині (можливо, з характеризують прикметниками). Прикладами атрибутів сутності "Співробітник" можуть бути такі атрибути як "Табельний номер", "Прізвище", "Ім'я", "По батькові", "Посада", "Зарплата" і т.п.

Атрибути, що визначають первинний ключ, розміщуються нагорі списку і виділяються знаком "*".

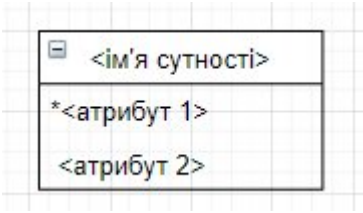


Рисунок 2.1. Графічне представлення сутності.

Ключ сутності (унікальний ідентифікатор) - це атрибут або сукупність атрибутів і / або зв'язків, призначена для унікальної ідентифікації кожного примірника даного типу сутності. Сутність може мати кілька різних ключів. Ключові атрибути зображаються на діаграмі підкресленням. У разі повної ідентифікації кожен примірник даного типу сутності повністю ідентифікується своїми власними ключовими атрибутами, в іншому випадку в його ідентифікації беруть участь також атрибути іншої сутності-батька.

Приклад. Сутність - співробітник. Атрибути: табельний номер, прізвище, ім'я, по батькові, посада, зарплата. Ключ сутності - табельний номер.

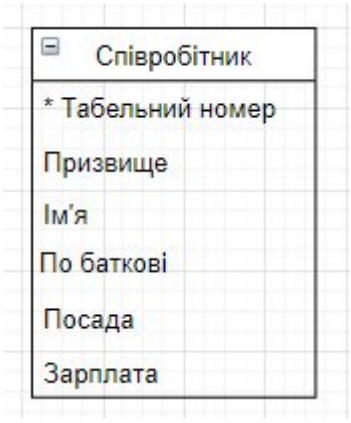


Рисунок 2.2. Приклад створення сутності «Співробітника».

Зв'язок (Relationship) - це деяка асоціація між двома сутностями. Одна сутність може бути пов'язана з іншого сутністю або сама з собою. Зв'язках може даватися ім'я, яке виражається граматичним оборотом дієслова і поміщається біля лінії зв'язку. Ім'я кожного зв'язку між двома даними сутностями повинно бути унікальним, але імена зв'язків у моделі не зобов'язані бути унікальними. Ім'я зв'язку завжди формується з точки зору батьків, так що пропозиція може бути утворене з'єднанням імені сущности-родителя, імені зв'язку, вирази ступеня та імені сутності-нащадка. Кожна зв'язок може мати один з наступних типів зв'язку: один-до одного 1:1, один-до-багатьох 1:N, багато-до-багатьох M:N, рис. 2.3.

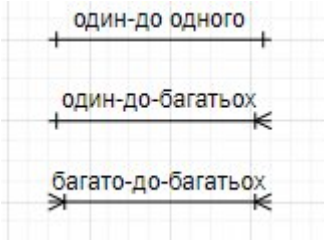


Рисунок 2.3 – Типи зв’язків

Кожна зв'язок може мати одну з двох модальностей зв'язку:

Розглянемо методи виявлення сущностей предметної області на прикладі розробки інформаційній системи з замовлень деякої оптової торгової фірми. Бізнес сценарій для проектованої системи з точки зору Менеджеру з продажу (менеджер) наступний:

1. Система повинна зберігати інформацію про покупців.
2. Друкувати накладні на відпущені товари.
3. Стежити за наявністю товарів на складі.
4. Складів на фірмі декілько.

Виділимо всі іменники (створення словника предметної області) в цих пропозиціях - це будуть потенційні кандидати на сутності й атрибути, і проаналізуємо їх (незрозумілі терміни будемо виділяти знаком питання):

Покупець - явний кандидат на сутність.

Накладна - явний кандидат на сутність.

Товар - явний кандидат на сутність

Склад - явно кандидат на нову сутність.

(?) Наявність товару - це, швидше за все, атрибут, але атрибут будь сутності?

Як зв'язати сутності: «накладна», «склад», «покупець» і «товар»? Покупці купують товари, отримуючи при цьому накладні, в які внесені дані про кількість і ціну купленого товару. Кожен покупець може отримати кілька накладних (зв'язок один-до-багатьох). Кожна накладна зобов'язана виписуватися на одного покупця. Кожна накладна повинна містити кілька товарів (не буває порожніх накладних). Кожен товар, в свою чергу, може бути проданий декільком покупцям через кілька накладних (зв'язок багато-до-багатьох). Крім того, кожна накладна повинна бути виписана з певного складу, і з будь-якого складу може бути виписано багато накладних (зв'язок один-ко-многим). Товар може зберігатися на різних складах, при цьому на кожному складі зберігається різний товар (зв'язок

багато-до-багатьох). Діаграма моделі ER буде виглядати наступним чином рис. 2.4:

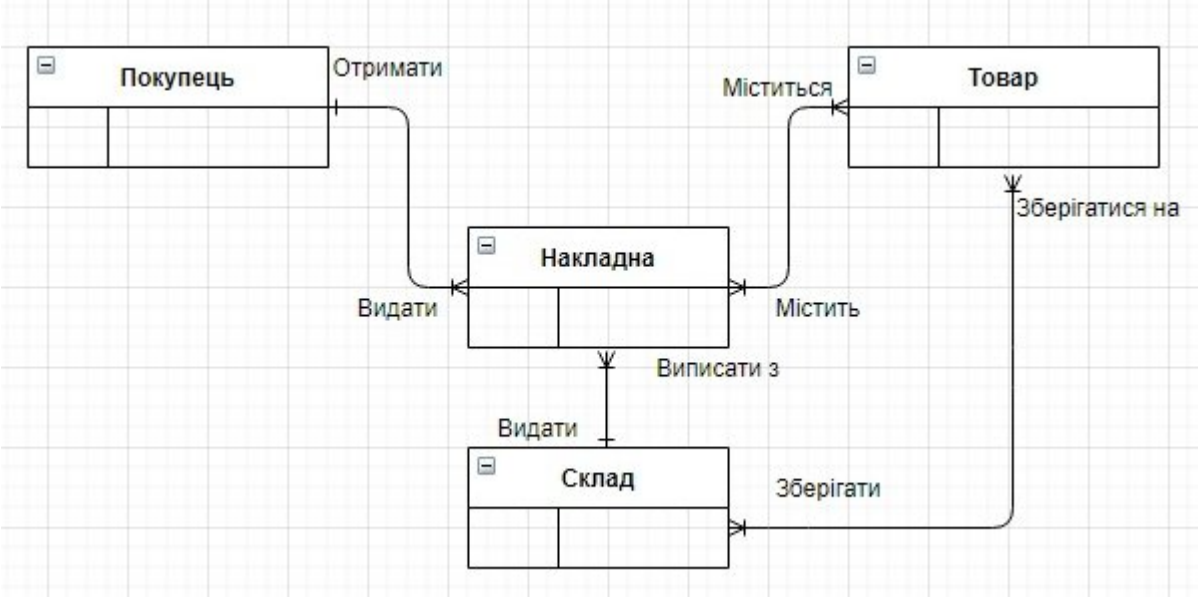


Рисунок 2.4 Модель ER для сутностей Покупець, Товар, Накладна, Склад

Тепер потрібно визначити атрибути сутностей. Розмовляючи з співробітниками фірми, ми з'ясували наступне:

• Кожен покупець є юридичною особою і має найменування, адреса, банківські реквізити.

• Кожен товар має найменування, ціну, а також характеризується одиницями виміру.

• Кожна накладна має унікальний номер, дату виписки, список товарів з кількостями і цінами, а також загальну суму накладної. Накладна виписується з певного складу і на певного покупця.

• Кожен склад має своє найменування.

Знову випишемо всі іменники (продовжуємо створювати словник), які будуть потенційними атрибутами, і проаналізуємо їх:

Юридична особа - термін риторичне, ми не працюємо з фізичними особами. Чи не звертаємо уваги.

Найменування покупця - явна характеристика покупця.

Адреса - явна характеристика покупця.

Банківські реквізити - явна характеристика покупця.

Найменування товару - явна характеристика товару.

(?) Ціна товару - схоже, що це характеристика товару. Чи відрізняється ця характеристика від ціни в накладній?

Одиниця виміру - явна характеристика товару.

Номер накладної - явна унікальна характеристика накладної.

Дата накладної - явна характеристика накладної.

(?) Список товарів в накладній - список не може бути атрибутом. Ймовірно, потрібно виділити цей список в окрему сутність.

(?) Кількість товару в накладній - це явна характеристика, але характеристика чого? Це характеристика не просто "товару", а "товару в накладній".

(?) Ціна товару в накладній - знову ж таки це має бути не просто характеристика товару, а характеристика товару в накладній. Але ціна товару вже зустрічалася вище - це одне й те саме?

Сума накладної - явна характеристика накладної. Ця характеристика не є незалежною. Сума накладної дорівнює сумі вартостей усіх товарів, що входять в накладну.

Найменування складу - явна характеристика складу.

В ході додаткової бесіди з менеджером вдалося прояснити різні поняття цін. Ціна одного і того ж товару в різних накладних, виписаних в різний час, може бути різною. Таким чином, є дві ціни - ціна товару в накладній і поточна ціна товару.

З поняттям "Список товарів в накладній" все досить ясно. Сутності "Накладна" і "Товар" пов'язані один з одним відношенням типу багато-до-багатьох. Такий зв'язок, повинна бути розщеплена на дві зв'язку типу один-ко-многим. Для цього потрібна додаткова сутність. Цією сутністю і буде сутність "Список товарів в накладній". Зв'язок її з сутностями "Накладна" і "Товар" характеризується наступними фразами -

"Кожна накладна повинна мати кілька записів зі списку товарів в накладній",

"Кожен запис зі списку товарів в накладній зобов'язан включатись рівно в одну накладну",

"Кожен товар може включатися в кілька записів зі списку товарів в накладних",

"Кожен запис зі списку товарів в накладній повинен бути пов'язан рівно з одним товаром".

Атрибути "Кількість товару в накладній" і "Ціна товару в накладній" є атрибутами сутності "Список товарів в накладній".

Точно теж саме зробимо зі зв'язком, що з'єднує сутності "Склад" і "Товар". Введемо додаткову сутність "Товар на складі". Атрибутом цієї сутності

буде "Кількість товару на складі". Таким чином, товар буде значитися на будь-якому складі і кількість його на кожному складі буде своє.

Тепер можна внести все це в діаграму:

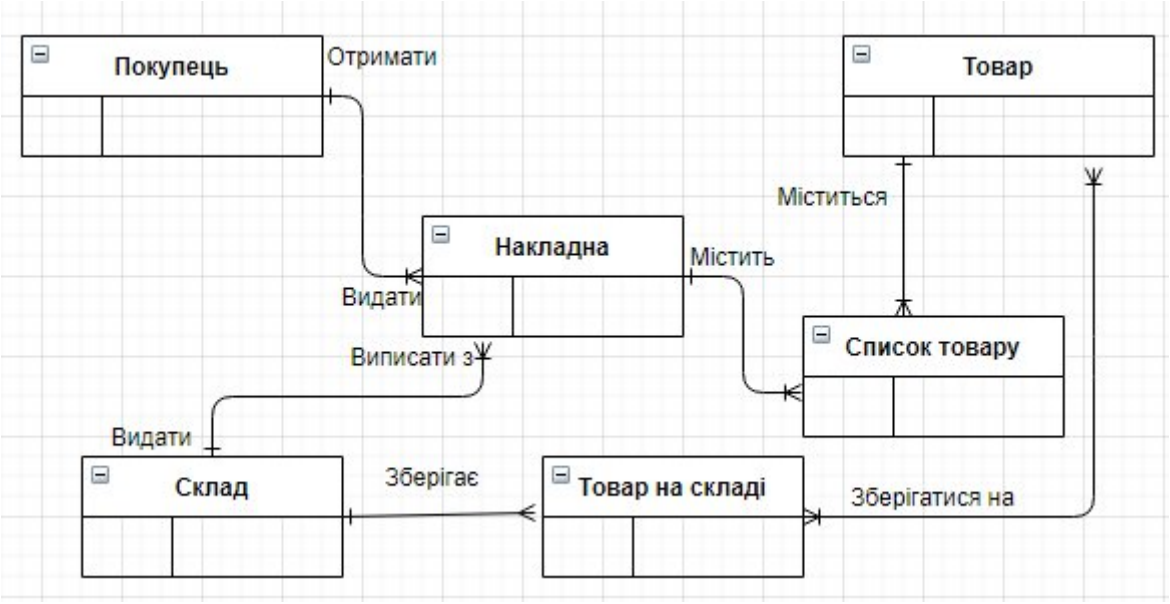


Рисунок 2.5 Модель ER для AIC «Продаж товару»

2.2 Приклад створеняя інформаційної моделі ER для AIC «RepairTech».

Створюємо ***Словник предметної області*** на основі моделі бізнес процесу (стр. 9-10)***:***

Адміністратор, Мастер з діагностики, Майстер з ремонту – співробітники фірми

Клієнт – людина, яка приносить поломану б/т для ремонту

Замовлення – документ, в якому адміністратор фіксує передачу б/т для ремонту

Квітанція – документ, який видається клієнту для підтверження передачі б/т

Накладна – документ, в якому майстер фіксує причину поломки, вартість, плановану дату завершення ремонту.

Склад запчастин – місто збереження складових частин б/т

На основі Словника створюємо таблицю сутностей з їх атрибутами (властивостями) та ключами (атрибути, які гарантують унікальність екземпляру сутності ПК, та які зв’язують сутності ЗК) табл. 2.1.

2.1 Таблиця сутностей

Сутність	Атрибути	Ключи
Співробітник	ПІБ Телефон Адреса Дата народження Дата приняття на роботу Посада	ПК ЗК
Посада	Назва Оклад	ПК
Замовлення	ПІБ клієета адміністратор Причина поломки Мастер по діагностики Назва б/т Номер приладу Дата	ПК ПК ПК
Накл_ремонту	Замовлення Мастер по ремонту Вартість ремонту Дата початку рем Дата заверш рем	ПК, ЗК
Склад запчастин	номер Назва запчастини Кількість вартість	ПК
Список	Накл_ремонту Склад запчастин Кількість	ЗК ЗК

Створюємо модель сутність-зв’язок ER на прикладі за допомогою додатка ***draw.io***

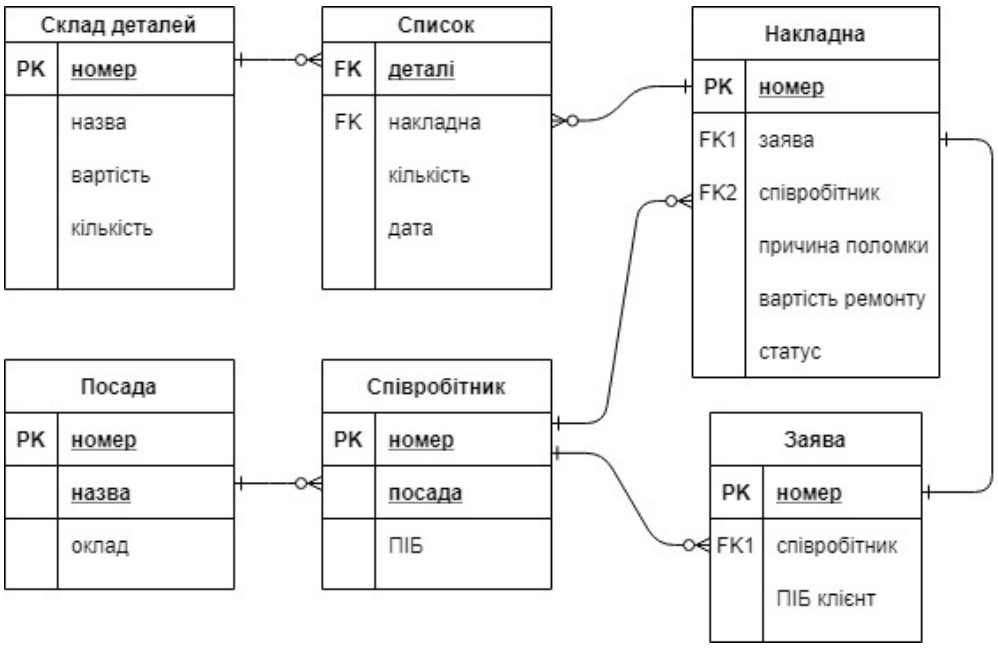


Рисунок 2.6 Модель ER для AIC «RepairTech»

Питання для самоконтролю

1. Дати визначення структурного моделювання.
2. Перерахувати і надати визначення методологій структурного моделювання.
3. Описати інформаційну модель даних сутність-зв’язок ER, перерахувати елементи цієї моделі.
4. Описати елементи діаграми ER (пташина лапка).
5. Етапи аналіза предметної області для створення інформаційної моделі.

Завдання до лабораторної роботи №2

1. Відповісти на питання.
2. Створити Словник для своєї предметної області.
3. Визначити необхідні атрибути сутністей.
4. Створити діаграму моделі ER для своєї предметної області.
5. Оформити протокол лабораторної роботи.

ТЕМА 3. СТУКТУРНЕ МОДЕЛЮВАННЯ ПЗ. МЕТОДОЛОГІЇ ФУНКЦІОНАЛЬНОГО МОДЕЛЮВАННЯ IDEF0 та DFD.

Мета: набути практичних навичок моделювання функціональних аспектів представлення системи в рамках структурного підходу.

3.1 Короткі теоретичні відомості за методологію IDEF0.

IDEF0 - методологія функціонального моделювання полягає у тому, що за допомогою наочної графічної мови IDEF0 система, що розробляється постає перед проектувальниками в вигляді набору взаємопов'язаних функцій (функціональних блоків - в термінах IDEF0). Як правило, моделювання засобами IDEF0 є першим етапом вивчення будь-якої системи в рамках структурного моделювання ПЗ.

Методологія IDEF0 являє собою серію діаграм із супровідною документацією, які розбивають складний об'єкт на складові частини, які представлені у вигляді блоків. Деталі кожного з основних блоків показані у вигляді блоків на інших діаграмах. Кожна детальна діаграма є декомпозицією блоку з більш загальної діаграми. На кожному кроці декомпозиції більш загальна діаграма називається батьківської для більш детальної діаграми.

IDEF0 - моделі складаються з трьох типів документів: графічних діаграм, тексту і глосарію. Ці документи мають перехресні посилання один на одного. Кожна діаграма є одиницею опису системи і розташовується на окремому аркуші.

Графічна діаграма - головний компонент IDEF0 моделі, що містить блоки, дуги, з'єднання блоків і дуг і асоційовані з ними відносини. Методологія може містити 4 типи діаграм:

1) Контекстна діаграма (діаграма верхнього рівня), будучи вершиною деревовидної структури діаграм, показує призначення системи (основну функцію) і її взаємодія з зовнішнім середовищем. У кожній моделі може бути тільки одна контекстна діаграма. Після опису основної функції виконується функціональна декомпозиція, т. Е. Визначаються функції, з яких складається основна.

2) Далі функції діляться на підфункції і так до досягнення необхідного рівня деталізації досліджуваної системи. Діаграми, які описують кожен такий фрагмент системи, називаються діаграмами декомпозиції. Після кожного сеансу декомпозиції проводяться сеанси експертизи - експерти предметної області

вказують на відповідність реальних процесів створеним діаграм. Знайдені невідповідності усуваються, після чого приступають до подальшої деталізації процесів.

3) Діаграма дерева вузлів показує ієрархічну залежність функцій (робіт), але не зв'язку між ними. Їх може бути скільки завгодно, оскільки дерево можна побудувати на довільну глибину і з довільного вузла.

4) Діаграми для експозиції будуються для ілюстрації окремих фрагментів моделі з метою відображення альтернативної точки зору на події в системі процеси (наприклад, з точки зору керівництва організації).

Правила IDEF0 моделі включають:

– обмеження кількості блоків на кожному рівні декомпозиції (правило 3-6 блоків);

– зв'язність діаграм (номери блоків);

– унікальність міток і найменувань (відсутність повторюваних імен);

– синтаксичні правила для графіки (блоків і дуг);

– поділ входів та управлінь (правило визначення ролі даних).

– відділення організації від функції, тобто виключення впливу організаційної структури на функціональну модель.

В IDEF0 реалізовані три базові принципи моделювання процесів:

Принцип функціональної декомпозиції являє собою спосіб моделювання типової ситуації, коли будь-яка дія, операція, функція можуть бути розбиті (декомпозіровани) на більш прості дії, операції, функції. Іншими словами, складна бізнес-функція може бути представлена у вигляді сукупності елементарних функцій.

Принцип обмеження складності. При роботі з IDEF0 діаграмами істотним є умова їх розбірливості і легкості читання. Суть принципу обмеження складності полягає в тому, що кількість блоків на діаграмі має бути не менше двох і не більше шести. Практика показує, що дотримання цього принципу призводить до того, що функціональні процеси, представлені у вигляді IDEF0 моделі, добре структуровані, зрозумілі і легко піддаються аналізу.

Принцип контекстної діаграми. Моделювання ділового процесу починається з побудови контекстної діаграми. На цій діаграмі відображається тільки один блок - головна бізнес-функція, що моделюється. Якщо мова йде про моделювання цілого підприємства або навіть великого підрозділу, головна бізнес-функція не може бути сформульована як, наприклад, "продавати

продукцію". Головна бізнес-функція системи - це "місія" системи, її значення в навколишньому світі.

Не можна правильно сформулювати головну функцію підприємства, не маючи уявлення про його стратегії. При визначенні головної бізнес - функції необхідно завжди мати на увазі мету моделювання і точку зору на модель. Одне і те ж підприємство може бути описано по-різному, в залежності від того, з якої точки зору його розглядають: директор підприємства і податкової інспектор бачать організацію абсолютно по-різному. Контекстна діаграма грає ще одну роль у функціональній моделі. Вона "фіксує" кордон в якому моделюється системи, визначаючи те, як моделюєма система взаємодіє зі своїм оточенням. Це досягається за рахунок опису дуг, з'єднаних з блоком, що представляє головну бізнес-функцію.

Основні елементи:

1. ***Функціональний блок*** (Activity Box) графічно зображується у вигляді прямокутника і уособлює собою деяку конкретну функцію в рамках даної системи. Усередині кожного блоку міститься його ім'я і номер. Номер блоку розміщується в правому нижньому кутку. За вимогами стандарту назва кожного функціонального блоку має бути сформульовано в глагольном способі, тобто бути дієсловом або дієслівним оборотом (наприклад, "виробляти послуги", а не "виробництво послуг").

2. ***Интерфейсная дуга*** (Arrow) відображає елемент системи, який обробляється функціональним блоком або впливає на функцію, відображену даними функціональним блоком. Залежно від того, до якої з сторін підходить дана інтерфейсна дуга, вона носить назву "входить", "вихідної" або "керуючої". Дуги можуть складатися тільки з вертикальних або горизонтальних відрізків; відрізки, спрямовані по діагоналі не допускаються. Кінці стрілок повинні стосуватися зовнішнього кордону функціонального блоку, але не повинні перетинати її. Дуги повинні приєднуватися до блоку на його сторонах.

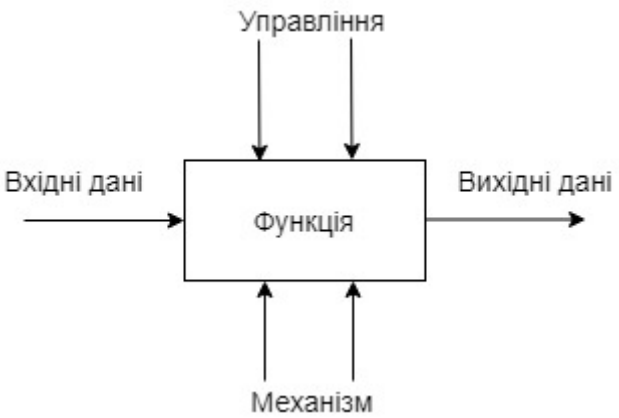


Рисунок 3.1 Елементи діаграми IDEF0: функція та інтерфейсні дуги.

3. ***Принцип декомпозиції*** (Decomposition) застосовується при розбитті складного процесу на складові його функції. При цьому рівень деталізації процесу визначається безпосередньо розробником моделі. Декомпозиція дозволяє поступово і структуровано представляти модель системи у вигляді ієрархічної структури окремих діаграм, що робить її менш перевантаженою і легко засвоюваній.

В основі методології IDEF0 лежать наступні правила:

- Функціональний блок (або Функція) перетворює Входи в Виходи (тобто вхідну інформацію в вихідну), Управління визначає, коли і як це перетворення може або повинно відбутися. Виконавці безпосередньо здійснюють цей захід.
- З дугами пов'язані написи (або мітки) на природній мові, що описують дані, які вони представляють.
- Дуги показують, як функції між собою взаємопов'язані, як вони обмінюються даними і здійснюють управління один одним.
- Виходи однієї функції можуть бути Входами, Управлінням або Виконавцями для іншої.
- Дуги можуть розгалужуватися і з'єднуватися.
- Функціональний блок, який представляє систему в якості єдиного модуля, деталізується на іншій діаграмі за допомогою декількох блоків, з'єднаних між собою інтерфейсними дугами.
- Ці блоки представляють основні підфункції (підмодулі) єдиного вихідного модуля.
- Дана декомпозиція виявляє повний набір подмодулей, кожен з яких
- представлений як блок, межі якої визначено інтерфейсними дугами. · Кожен з цих подмодулей може бути декомпозирован подібним же чином для більш детального уявлення.
- Кожна подфункция може містити тільки ті елементи, які входять в вихідну функцію. Крім того, модель не може опустити будь-які елементи, тобто, як уже зазначалося, батьківський блок і його інтерфейси забезпечують контекст. До нього не можна нічого додати, і з нього не може бути нічого видалено.
- Дуги, що входять в блок і виходять з нього на діаграмі верхнього рівня, є точно тими ж самими, що і дуги, що входять в діаграму нижнього рівня

і виходять з неї, тому що блок і діаграма представляють одну і ту ж частину системи.

4. ***Контекстна діаграма.*** Модель IDEF0 завжди починається з представлення системи як єдиного цілого - одного функціонального блоку з інтерфейсними дугами, що тягнуться за межі даної області. Така діаграма з одним функціональним блоком називається контекстної діаграмою, і позначається ідентифікатором "A-0". Оскільки єдиний блок представляє всю систему як єдине ціле, ім'я, вказане в блоці, є спільним. Це вірно і для інтерфейсних дуг - вони також представляють повний набір зовнішніх інтерфейсів системи в цілому.

Кожна модель повинна мати контекстну діаграму верхнього рівня, на якій об'єкт моделювання представлений єдиним блоком з граничними стрілками. Стрілки на цій діаграмі відображають зв'язку об'єкта моделювання з навколишнім середовищем. Оскільки єдиний блок представляє весь об'єкт, його ім'я загальне для всього проекту. Контекстна діаграма встановлює область моделювання та її кордон.

Контекстна діаграма A-0 також повинна містити короткі затвердження, що визначають точку зору посадової особи або підрозділу, з позицій якого створюється модель, і мета, для досягнення якої її розробляють.

Формулювання мети висловлює причину створення моделі, тобто містить перелік питань, на які повинна відповідати модель, що значною мірою визначає її структуру.

Найбільш важливі властивості системи зазвичай виявляються на верхніх рівнях ієрархії; в міру декомпозиції функції верхнього рівня і розбиття її на підфункції, ці властивості уточнюються.

Приклад створення контекстної діаграми IDEF0 A-0 для АІС «RepairTech»

Таблиця 3.1 Опис елементів контекстної діаграми IDEF0

Назва процеса	Діяльність фірми з ремонту побутовоїтехніки
Мета процесу	Якісно і вчасно відремонтувати б/т для отримання прибутку
З токи зору	З позиції господаря фірми
Вхідні дані	• б/т клієнкта • оплата • запчастини
Вихідні дані	• б/т клієнкту • звіти
Управління	• Законодавство • Правила та процедури фірми

Механізми	Співробітники фірми

Створимо контекстну діаграму A-0 для АІС «RepairTech» засобами draw.io рис. 3.2

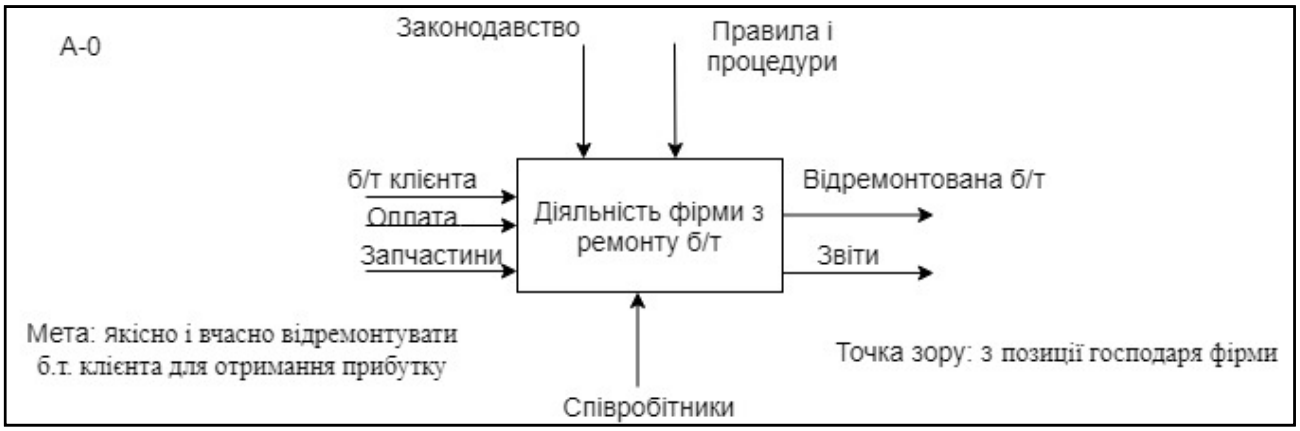


Рисунок 3.2 Контекстна діаграма **A-0** IDEF0 для АІС «RepairTech»

5. ***Діаграми декомпозиції***

Блок, який представляє систему в якості єдиного модуля, деталізується на іншій діаграмі за допомогою декількох блоків, з'єднаних інтерфейсними дугами. Ці блоки представляють основні підфункції вихідної функції. Дана декомпозиція виявляє повний набір підфункції, кожна з яких представлена як блок, межі якої визначено інтерфейсними дугами. Кожна з цих підфункцій може бути декомпозирована подібним чином для більш детального уявлення.

Кожна подфункция моделюється окремим блоком. Кожен батьківський блок детально описується дочірньою діаграмою на більш низькому рівні. Всі дочірні діаграми повинні бути в межах області контекстної діаграми верхнього рівня.

Дочірня діаграма. Єдина функція, представлена на контекстній діаграмі верхнього рівня, може бути розкладена на основні підфункції за допомогою створення дочірньої діаграми. У свою чергу, кожна з цих підфункцій може бути розкладена на складові частини за допомогою створення дочірньої діаграми наступного, більш низького рівня, на якій деякі або всі функції також можуть бути розкладені на складові частини. Кожна дочірня діаграма містить дочірні блоки і стрілки, що забезпечують додаткову деталізацію батьківського блоку. Дочірня діаграма, створювана при декомпозиції, охоплює ту ж область, що і батьківський блок, але описує її більш детально. Таким чином, дочірня діаграма як би вкладена в свій батьківський блок.

Батьківська діаграма - діаграма, яка містить один або більше батьківських блоків. Кожна звичайна (НЕ контекстна) діаграма є також дочірньої діаграмою, оскільки, за визначенням, вона детально описує деякий батьківський блок. Таким чином, будь-яка діаграма може бути як батьківської діаграмою (містити батьківські блоки), так і дочірньої (докладно описувати власний батьківський блок). Аналогічно, блок може бути як батьківським (докладно описуватися дочірньої діаграмою) так і дочірнім (що з'являється на дочірньої діаграмі).

Граничні стрілки. На звичайній (НЕ контекстної) діаграмі граничні стрілки представляють входи, управління, виходи або механізми батьківського блоку діаграми. Джерело або споживач граничних стрілок можна виявити, тільки вивчаючи батьківську діаграму. Всі граничні стрілки на дочірньої діаграмі (за винятком стрілок, поміщених в тунель повинні відповідати стрілкам батьківського блоку.

Тунель - круглі дужки на початку і / або закінчення стрілки. Тунельні стрілки означають, що дані, виражені цими стрілками, не розглядаються на батьківській діаграмі і / або на дочірньої діаграмі. Стрілка, вміщена в тунель там, де вона приєднується до блоку, означає, що дані, виражені цієї стрілкою, не обов'язкові на наступному рівні декомпозиції. Стрілка, що поміщається в тунель на вільному кінці означає, що виражені нею дані відсутні на батьківській діаграмі.

Приклад створення діаграми декомпозіції A0 для АІС «RepairTech». Для створення декомпозіції А0 контекстної діаграми А-0 будемо використовувати діаграму «Риб’я кість» рис.1.1, яка представляє функціїї першого рівня ієрархії функцій системи, тобто є декомпозіцією основної функції АІС.

Перший рівень декомпозіції містить 4 блока:

- Оформлення заяви на ремонт
- Діагностика та тестування б/т
- Ремонт б/т
- Передача б/т клієнту

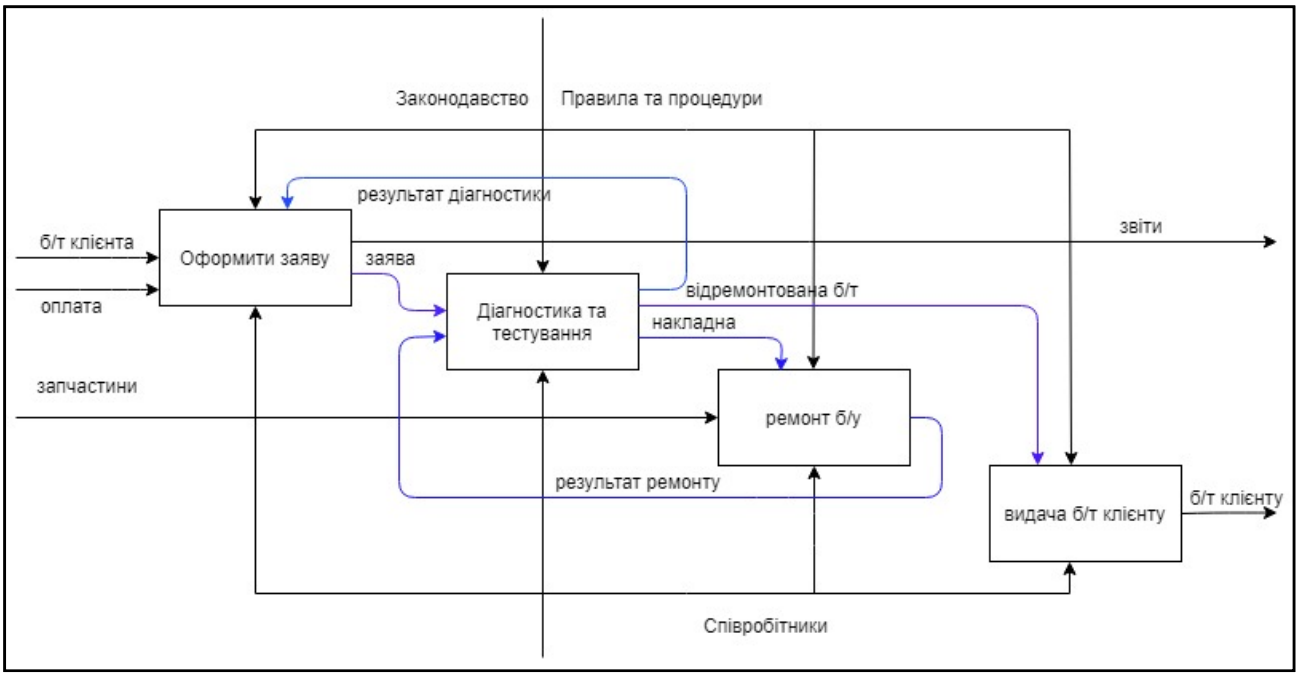


Рисунок 3.3 Діаграма декомпозії А0 методології IDEF0 для AIC RepairTech»

3.2 Короткі теоретичні відомості за методологію DFD.

DFD (Data flow diagram) - методологія графічного структурного аналізу. Діаграми потоків даних (DFD) використовуються для опису документообігу та обробки інформації. Подібно IDEF0, DFD представляє модельовану систему як мережу пов'язаних між собою робіт. Їх можна використовувати як доповнення до моделі IDEF0 для більш наочного відображення поточних операцій документообігу в корпоративних системах обробки інформації. ***Головна мета DFD - показати, як кожна робота перетворює свої вхідні дані у вихідні, а також виявити відносини між цими роботами.***

Створені моделі потоків даних організації можуть бути використані при вирішенні таких завдань, як:

1. визначення існуючих сховищ даних (текстові документи, файли, Системи управління базою даних - СУБД);

2. визначення та аналіз даних, необхідних для виконання кожної функції процесу;

3. підготовка до створення моделі структури даних організації, так звана ERD модель (IDEF1X);

4. виділення основних і допоміжних бізнес-процесів організації.

Будь-яка DFD-діаграма може містити роботи, зовнішні сутності, стрілки (потоки даних) і сховища даних. Існують дві нотації DFD: Йордана-Де Марко і Гейне-Сарсона. Декомпозиція роботи IDEF0 в діаграму DFD.

При декомпозиції роботи IDEF0 в DFD необхідно виконати наступні дії:

- видалити всі граничні стрілки на діаграмі DFD;
- створити відповідні зовнішні сутності і сховища даних;
- створити внутрішні стрілки, що починаються з зовнішніх сутностей замість граничних стрілок;
- стрілки на діаграмі IDEF0 затоннелювать.
-

Елементи графічної нотації DFD Гейне-Сарсона

Зовнішня сутність (термінатор) рис. 3.4 являє собою матеріальний об'єкт або фізична особа, що виступають як джерело або приймач інформації (наприклад, замовники, персонал, програма, склад, інструкція). Зовнішні сутності на DFD за змістом відповідають управлінню і механізмам, що відображається на контекстній діаграмі IDEF0. Визначення деякого об'єкту, суб'єкта або системи в якості зовнішньої сутності вказує на те, що вона знаходиться за межами кордонів проектованої інформаційної системи. У зв'язку з цим зовнішні сутності, як правило, відображаються тільки на контекстній діаграмі DFD. У процесі аналізу і проектування деякі зовнішні сутності можуть бути перенесені на діаграми декомпозиції, якщо це необхідно, або, навпаки, частина процесів (підсистем) може бути представлена як зовнішня сутність

Рисунок 3.4 Зовнішня сутність

Рисунок 3.5 Процес

Процес (в IDEF0 - функція, робота) рис. 3.5 являє собою перетворення вхідних потоків даних у вихідні відповідно до певного алгоритму. Кожен процес повинен мати ім'я у вигляді пропозиції з дієсловом у формі (обчислити, розрахувати, перевірити, визначити, створити, отримати), за яким слідують іменники в знахідному відмінку, наприклад: «Ввести відомості про клієнтів», «Розрахувати допустиму швидкість», «Сформувати відомість допускаються швидкостей» Номер процесу служить для його ідентифікації і ставиться з урахуванням декомпозиції. На відміну від IDEF0 вкладеність процесів

позначається через точку (наприклад, в IDEF0 - «236», в DFD - «2.3.6»). Перетворення інформації може показуватися як з точки зору процесів, так і з точки зору систем і підсистем. Якщо замість імені процесу «Розрахувати допустиму швидкість» написати «Підсистема розрахунку допустимої швидкості», тоді цей блок на діаграмі варто розглядати, як підсистему.

Вимоги до оформлення процесів (функцій):

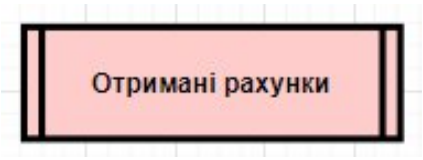
1. Кожна функція повинна мати ідентифікатор;

2. Назви роботи потрібно формулювати відповідно до наступне формулою: Назва роботи = Дія + Об'єкт, над яким дію здійснюється Наприклад, якщо ця робота пов'язана з дією з продажу продукції, то її потрібно назвати <Продаж продукції>.

3. Назва роботи має бути по можливості коротким (не більше 50 символів) і складатися з 2-3 слів.

У складних випадках також рекомендується для кожного короткого назви роботи зробити її докладний опис, який помістити в словник.

Накопичувач (сховище) даних являє собою абстрактний пристрій для зберігання інформації, яку можна в будь-який момент помістити в накопичувач і через деякий час витягнути, причому способи приміщення і вилучення можуть бути будь-якими. Воно в загальному випадку є прообразом майбутньої бази даних, і опис даних,які зберігаються в ньому, має відповідати інформаційній моделі (Entity-Relationship Diagram). Накопичувач даних може бути реалізований фізично у вигляді ящика в картотеці, області в оперативній пам'яті, файлу на магнітному носії і т. п. Накопичувачу обов'язково має надаватися унікальне ім'я і номер в межах всієї моделі (всього набору діаграм). Ім'я накопичувача вибирається з міркування найбільшої інформативності для розробника. Наприклад, якщо в якості накопичувачів виступають таблиці проектованої бази даних, тоді *як імена накопичувачів рекомендується використовувати імена таблиць*. Таким чином, накопичувач даних може являти собою всю базу даних цілком, сукупність таблиць або окрему таблицю. Таке уявлення накопичувачів в подальшому полегшить побудову інформаційної моделі системи.



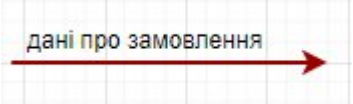


Рисунок 3.6 Накопичувач Рисунок 3.7 Потік даних

Потік даних рис. 3.7 визначає інформацію (матеріальний об'єкт), передану через деякий з'єднання від джерела до приймача. Реальний потік даних може бути інформацією, переданої по кабелю між двома пристроями, що пересилаються поштою листами, флешками або CD-дисками, які переносяться з одного комп'ютера на інший і т. Д. Кожен потік даних має ім'я, що відбиває його зміст. Напрямок стрілки показує напрямок потоку даних. Іноді інформація може рухатися в одному напрямку, оброблятися і повертатися назад в її джерело. Така ситуація може моделюватися або двома різними потоками, або одним - двонаправленим. На діаграмах IDEF0 потоки даних відповідають входам і виходам, на відміну від IDEF0 стрілки потоків на DFD можуть входити і виходити з будь-якої грані зовнішньої сутності, процесу або накопичувача даних.

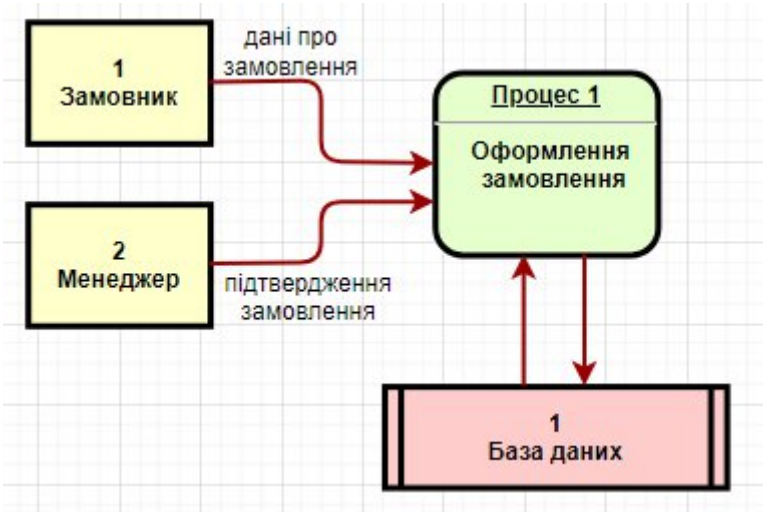


Рисунок 3.8 Приклад нотації Гейне-Сарсона

3.3 Приклад створення нотації Гейне-Сарсона для функції «Діагностика та тестування» АІС «RepairTech».

Декомпозіція блока «Діагностика та тестування» (див. рис.3.3) має 3 процеса:

- Перегляд замовлень зі статусем «діагностика» чи «тестування»
- Створення накладної на ремонт б/т
- Призначення майстра з ремонту

Зовнішні сутності : Мастер з діагностики, Мастер з ремонту, файл «Правила і процедури».

Накопичувачі: таблиця БД «Заява», таблиця БД «Накладна».

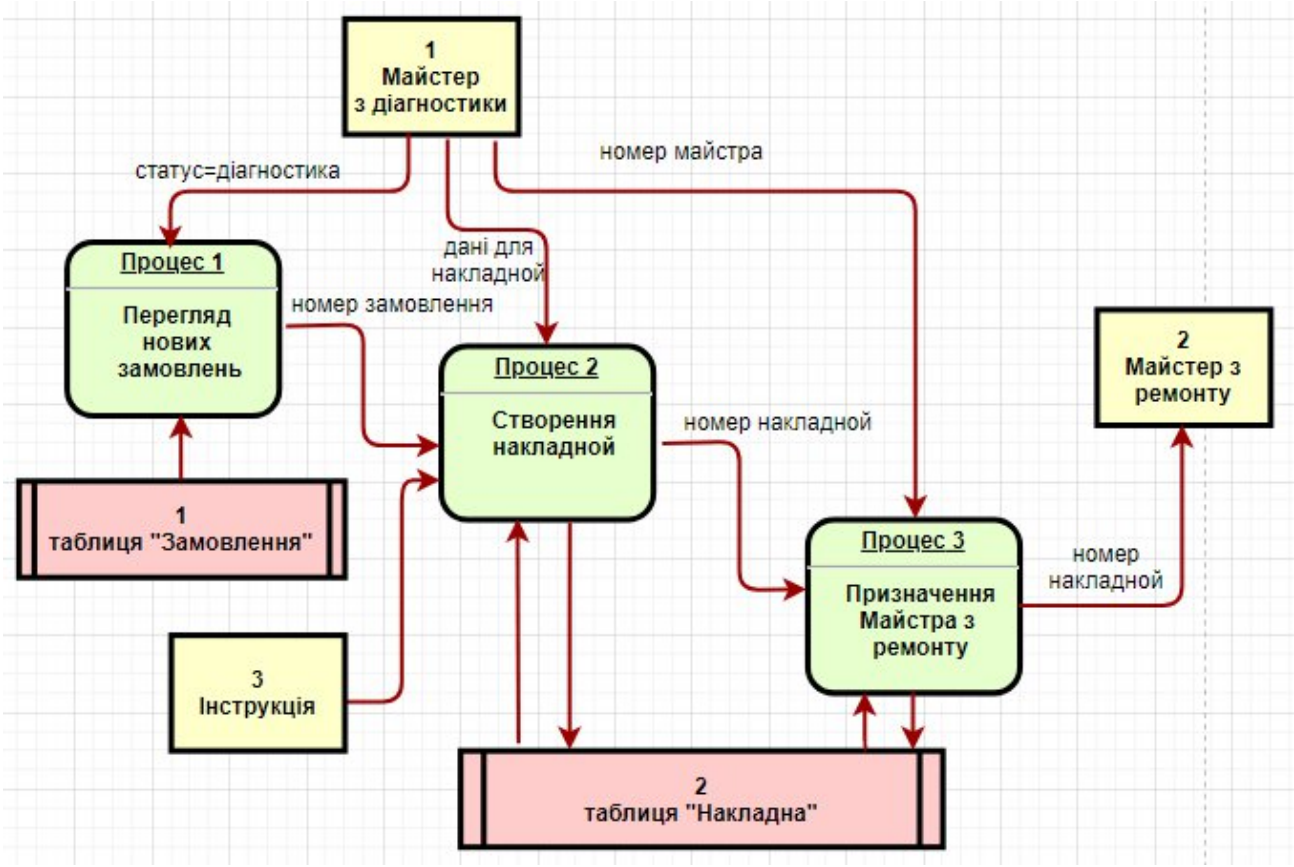


Рисунок 3.9 Нотація Гейна-Сарсона для блока « Діагностика та тестування» з використанням draw.io

Питання для самоконтролю

1. Які основні принципи структурного моделювання ПЗ?
2. Надати визначення методології IDEF0 та описати основні її елементі.
3. У чому полягають правила методології IDEF0?
4. Надати визначення методології DFD та описати основні її елементі.
5. Які нотації представляють методологію DFD? Описати одну із них.
6. Чим відрізняються методології IDEF0 та DFD?
7. Як перетворити діаграму IDEF0 в діаграму DFD?

Завдання до лабораторної роботи №3

1. Відповісти на питання для самоконтролю.
2. Створити для власної АІС контекстну діаграму А-0 та діаграму декомпозіції А0 згідно методології IDEF0.
3. Створити для власної АІС нотацію Гейна-Сарсона для одного з блоків діаграми А0 IDEF0.
4. Оформити протокол лабораторної роботи.

ТЕМА 4. ПОБУДОВА ДІАГРАМ ВАРІАНТІВ ВИКОРИСТАННЯ (МОДЕЛЬ ПРЕЦЕДЕНТІВ).

Мета: набути практичних навичок застосування методології об’єктно орієнтірованного моделювання для виявлення межі системи та взаїмодії користувачів з системою за допомогою моделі прецедентів.

4.1 Короткі теорітичні відомості з методології об’єктно орієнтірованного моделювання.

4.1.1 Принципи об'єктно-орієнтованої методології

Об'єктно-орієнтована методологія (ООМ) спеціалізується на створення великих систем, колективну їх розробку, подальший активний супровід при експлуатації і регулярної модифікації.

Об'єктно-орієнтована методологія (ООМ) складається з наступних частин:

- Об'єктно-орієнтований аналіз (ООА),
- Об'єктно-орієнтоване проектування (OOD),
- Об'єктно-орієнтоване програмування (ООР).

ООА - методологія аналізу сутностей реального світу на основі понять класу і об'єкту, для розуміння і пояснення того, як сутності взаємодіють між собою. ***У предметної області виділяються класи об'єктів, які, якщо це необхідно, розбиваються на підкласи. Кожен клас і його підклас аналізуються в три етапи: інформаційне моделювання, моделювання станів, моделювання процесів***. Аналіз вимог до системи зводиться до розробки моделей цієї структуру системи.

OOD - методологія проектування програмного продукту, що з'єднує в собі процес об'єктної декомпозиції, що спирається на виділення класів і об'єктів, і прийоми представлення моделей, що ***відображають логічну (структура класів і об'єктів) і фізичну (архітектура моделей і процесів).***

Говорячи про об'єктно-орієнтованої розробки, маємо на увазі: об'єктно-орієнтовані методології (технології) розробки програмних систем; інструментальні засоби, що підтримують ці технології. На сьогоднішній день в якості основного підходів для проектування програмного забезпечення при об'єктно-орієнтованому моделюванні є використання стандарту UML, який включений в методології і CASE засоби ARIS, Rational Rose, Oracle Designer, Together Control Center (Borland), AllFusion Component Modeler (Computer Associates), Microsoft Visual Modeler та інші).

4.1.2 Методологія RUP для розробки ПЗ в рамках об'єктно-орієнтованого підхода.

Методологія RUP (Rational Unified Process) застосовує уніфіцированний процес (UP) розробки ПЗ в рамках об'єктно-орієнтованого підхода. UP є модіфікацією ітеративної моделі ЖЦ, де розробка ПЗ виконується у вигляді декількох короткострокових міні-проектів фіксованої тривалості (наприклад, 4 тижня), які називаються ітераціями. Кожна ітерація включає свої власні етапи аналізу вимог, проектування, реалізації і завершуеться тестуванням та створенням робочої системиям. На кожній ітерації система доповнюється новими можливостями, які узгоджуються з вже створенним ядром, та контролюється завдяки зворотньому зв’язку з замовником та майбутніми користувачами.

Перевоги ітеративної розробки:

- своєчасне усвідомлення можливих технічних рисків, розуміння вимог, задач проекта та зручність використання системи,
- швидкий та помітний прогрес,
- раній зворотній зв’язок з користувачами та адаптація системи,
- керування складністю,
- отриманий на кожній ітерації досвід може методично використовуваться для покращення оргацізації процесу розробки.

Також особливостью UP є: оцінка рисків і ключевих моментів проекту на ранніх ітераціях; постійний відгук користувачів, зворотній зв’язок та модіфікація вимог; побудова базової архитектури на ранніх ітераціях; використання прецедентів (варіанти використання); візуалізація програмної моделі за допомогою мови UML; керування конфігурацією.

Фази UP:

1) початкова (inception) – визначення початкового бачення проблеми, прецедентів та оцінка складності задачі;
2) розвиток (elaboration) – фомування більш повного бачення проблеми, ітеративна реалізація базової архітектури, створення найбільш критичних компонентів , ідентифікація основних вимог, отримання більш реалістичних оцінок;
3) конструювання (construction) – ітеративна реалізація найбільш критичних та простих елементів, підготовка до розгортання;
4) передача (transition) – бета-тестування та розгортання.

В рамках UP існує декілько дисциплін, наприклад, Бізнес-моделювання (business modeling), Вимоги (requirement), Проектування (design), артефакти яких будуть використовуваться для реалізації ПЗ. Бізнес-моделювання – моделювання об'єктів предметної області. Вимоги – аналізу вимог для даного ПЗ, опис прецедентів та визначення нефункціональних вимог. Проектування – розробка загальної архітектури, об'єктів , баз даних та конфігурацій мереж.

Таблиця 4.1 – Приклади артифактів для фаз UP на різних ітераціях.

Дисципліна	Артефакт Ітерація	Початок I1	Розвиток E_1 …E_n	Конструювання C_1….C_n	Передача T_1….T_n
Бізнес-моделювання	Модель предметної області		п		
Вимоги	Модель прецедентів	п	р		
	Бачення системи	п	р		
	Допоміжна специфікація	п	р		
	Словник термінів	п	р		
Проектування	Модель проектування		п	Р	
	Опис архітектури		п	Р	
	Модель даних		п	Р	
Реалізація	Модель реалізації		п	р	р
Керування проектом	План розробки	п	р	р	р
Тестування	Модель тестування		п	Р	
Оточення	Документи	п	р		

Таким чином, основні документи, які потрібно створити на початку розробки це «Модель прецедентів», «Бачення системи», «Допоміжна специфікація» та «Словник термінів», які відносяться до дисципліни «Вимоги».

Модель прецедентів (варіантів використання) ***UP*** – модель функцюювання системи та її оточення, тобто ця модель фіксує верхньо рівневі прецеденти, які були розглянуті у темі 1 стр. у вигляді текстового опису прецедентів, діаграм варіантів використання (прецедентів) та послідовтості взаємодії користувачів з системою мови UML. В межах UP не всі прецеденти на початковій ітерації описуються детально, а тільки 10-20 % найбільш складних і

критичних з точки зору розробки системи, інши будуть описані на ітераціях розвитку.

Бачення системи – це документ, який описує основні цілі та проблеми проекту, вказує коло зацікавленних осіб та їх потреби, осносні ідеї розробки проекту, тобто містить основу для більш детального опису техничних вимог до проекту.

Допоміжна специфікація – це документ, який містить вимоги, обмеження та інша інформація, що не ввійшла до опису прецедентів та в словник термінів, наприклад, такі як вимоги якості та спеціальні вимоги. В цей документ можно додавати наступну інформацію:

- Вимоги згідно моделі FURPS+ - Functionality (функціональність); Usability (зручність застосування); Reliability (надійність); Performance (продуктивність); Supportability (підтримка); плюс обмеження проектування, розробки та інтерфейсу.
- Бізнес - правила.
- Звіти.
- Міжнародні стантарти.
- Документація та справочно інформація.
- Згода на ліцензування та інші згоди.

Словник термінів – це документ, який фиксує на початковій фазі важливі поняття термінів проекту та їх визначення для однозначності зрозуміння, а на фазі розвидку він перетворюється у словник даних. До атрибутів термінов відносяться – синоніми, опис, формат, взаїмозв’язки між іншими елементами, діапазон значень, правила перевірки коректності.

Артефакти, які створюються в рамках дисципліни «Вимоги», розробляються паралельно, однако можливо запропонувати наступну послідовність розробки цих артефактів:

1) Створити короткий варіант документу «Бачення системи»
2) Визначити задачи користувачів і відповідні прецеденти
3) Описати деякі прецеденти і приступити до створення документу «Допоміжна специікація»
4) На підставі отриманої інформації уточнити документ «Бачення системи»

В межах даної дисципліни зупинимося на створені Моделі прецедентів на основі варіантів використання, яки були розглянуті у темі 1, тобто потрібно

візуалізувати ці прецеденти за допомогою діаграм варіантів використання та послідовності UML.

4.2 Діаграма UML варіантів використання (use case diagram).

Діаграми варіантів використання показують взаємодії між варіантами використання і діючими особами, відображаючи функціональні вимоги до системи з точки зору користувача.

Мета побудови діаграм варіантів використання - документування функціональних вимог, які були описані в варіантах використання, в найзагальнішому вигляді, тому вони повинні бути обмежено прості.

Діаграма варіантів використання є найбільш загальним поданням функціональних вимог до системи. Для подальшого проектування системи потрібні більш конкретні деталі, які описуються в документі «сценарій варіанта використання» або «поток подій» (flow of events). Сценарій детально документує процес взаємодії дійової особи з системою, що реалізується в рамках варіанту використання. Основний потік подій описує нормальний хід подій (при відсутності помилок). Альтернативні потоки описують відхилення від нормального перебігу подій (помилкові ситуації) і їх обробку. Переваги моделі варіантів використання:

- визначає користувачів і кордони системи;
- визначає системний інтерфейс;
- зручна для спілкування користувачів з розробниками;
- є основою для написання документації користувача;
- добре вписується в будь-які методи проектування (як об'єктно-орієнтовані, так і структурні).

Діаграма варіантів використання є вихідним концептуальним поданням або концептуальною моделлю системи в процесі її проектування і розробки.

Суть даної діаграми 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 з зовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка може служити джерелом впливу на моделируемую систему так, як визначить сам розробник.

Добре структурована діаграма варіантів використання має такі властивості:

- акцентує увагу лише на одному аспекті динаміки системи;
- містить тільки такі прецеденти і актори, які важливі для розуміння цього аспекту;
- містить тільки такі деталі, які відповідають даному рівню абстракції, і тільки ті доповнення, які необхідні для розуміння системи;
- не так лаконічна, щоб ввести читача в оману щодо важливих аспектів семантики.

Варіант використання (use-case), який позначається на діаграмі еліпсом рис. 4.1, - опис окремого аспекту поведінки системи з точки зору користувача. Кожен варіант використання визначає послідовність дій, які повинні бути виконані проектованої системою при взаємодії її з відповідним актором.

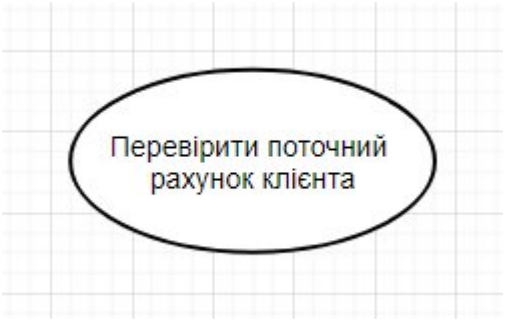


Рисунок 4.1 – Варіант використання

Рисунок 4.2 – Актор

Актор (actor), який зображається фігуркою людини рис.4.2, - це безліч логічно пов'язаних ролей, виконуваних при взаємодії з прецедентами або сутностями (системой, підсистемой або класом).

Актори використовуються для моделювання зовнішніх по відношенню до проектованої системі сутностей, які взаємодіють з системою і використовують її в якості окремих користувачів.

В UML є кілька ***стандартних видів відносин*** між акторами і варіантами використання:

- відношення асоціації (association relationship)
- відношення розширення (extend relationship)
- відношення узагальнення (generalization relationship)
- відношення включення (include relationship)

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме за допомогою відносин розширення, узагальнення і включення.

Відношення асоціації служить для позначення специфічної ролі актора в окремому варіанті використання. Це відношення встановлює, яку конкретну роль грає актор при взаємодії з екземпляром варіанту використання. На діаграмі варіантів використання, так само як і на інших діаграмах, відношення асоціації позначається суцільною лінією між актором і варіантом використання. Ця лінія може мати додаткові умовні позначення, такі, наприклад, як ім'я та кратність рис.4.3.

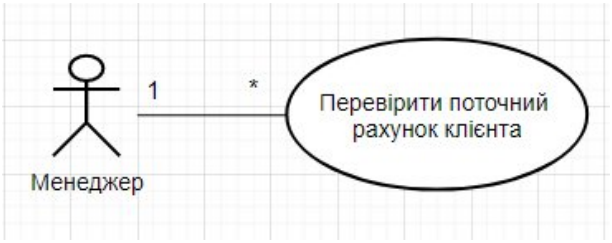


Рисунок 4.4 - Відношення асоціяції.

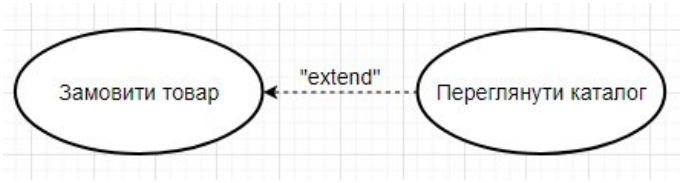


Рисунок 4.5 – Відношення розширенн.

Відношення розширення (extend) визначає взаємозв'язок базового варіанту використання з іншим варіантом використання, функціональне поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов. Так, якщо має місце відношення розширення від варіанту використання А до варіанту використання В, то це означає, що властивості екземпляра варіанту використання В можуть бути доповнені завдяки наявності властивостей у розширеного варіанту використання А.

У мові UML відношення розширення є залежністю, спрямованої до базового варіанту використання і з'єднаної з ним в так званій точці розширення. Ставлення розширення між варіантами використання позначається пунктирною лінією зі стрілкою (варіант відношення залежності), спрямованої від того варіанту використання, який є розширенням для початкового варіанту використання. Дана лінія зі стрілкою позначається ключовим словом **"extend"** ("розширює"), як показано на рис. 4.5. Один з варіантів використання може бути розширенням для декількох базових варіантів, а також мати в якості власних розширень кілька інших варіантів. Базовий варіант використання може додатково ніяк не залежати від своїх розширень.

Відношення узагальнення (успадкування) служить для вказівки того факту, що деякий дочірній варіант використання може бути узагальнено до варіанту використання предка. Слід підкреслити, що нащадок успадкує всі властивості і поведінку свого батька, а також може бути доповнений новими властивостями і особливостями поведінки. Графічно дане відношення позначається суцільною лінією зі стрілкою в формі незафарбовані трикутника, яка вказує на батьківський варіант використання (рис. 4.6). Ця лінія зі стрілкою має спеціальну назву - стрілка "узагальнення".

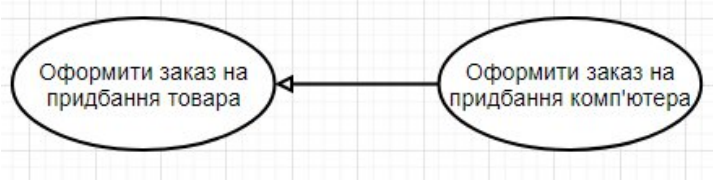


Рисунок 4.6 – Відношення узагальнення.

Відношення узагальнення між варіантами використання застосовується в тому випадку, коли необхідно відзначити, що дочірні варіанти використання мають всі атрибути і особливостями поведінки батьківських варіантів. При цьому, один варіант використання може мати кілька батьківських варіантів. В цьому випадку реалізується множинне успадкування властивостей і поведінки відносини предків: З іншого боку, один варіант використання може бути предком для декількох дочірніх варіантів, що відповідає таксономичному характеру відносини узагальнення.

Між окремими акторами також може існувати відношення узагальнення. Дане відношення є спрямованим і вказує на факт спеціалізації одних акторів щодо інших. Наприклад, ставлення узагальнення від актора А до актора В відзначає той факт, що кожен екземпляр актора А є одночасно екземпляром актора В і володіє всіма його властивостями.

Відношення включення (**include**) це різновид відносини залежності між базовим варіантом використання і його спеціальним випадком. Відношення включення між двома варіантами використання вказує, що деяка задана поведінка для одного варіанта використання включається як складовий компонент в послідовність поведінки іншого варіанту використання.

Семантика цього відношення визначається наступним чином, коли екземпляр першого варіанту використання в процесі свого виконання досягає точки включення в послідовність поведінки екземпляра другого варіанту використання, екземпляр першого варіанту використання виконує послідовність

дій, що визначає поведінку екземпляра другого варіанту використання, після чого продовжує виконання дій своєї поведінки. При цьому передбачається, що навіть якщо екземпляр першого варіанту використання може мати кілька включень в себе примірників інших варіантів, виконувані ними дії повинні закінчитися до деякого моменту, після чого має бути продовжено виконання перерваних дій примірника першого варіанту використання відповідно до заданого для нього поведінкою.

Один варіант використання може бути включений в кілька інших варіантів, а також включати в себе інші варіанти. Варіант використання, який включається, може бути незалежним від базового варіанту в тому сенсі, що він надає останньому деяку інкапсульовану поведінку, деталі реалізації якої приховані від останнього і можуть бути легко перерозподілені між декількома долучаємих варіантами використання. Більш того, базовий варіант може залежати тільки від результатів виконання включається в нього поведінки, але не від структури включаються в нього варіантів.

Відношення включення, спрямоване від варіанту використання A (“Оформити заказ на придбання комп'ютера”) до варіанту використання B (“Виписати рахунок на оплату комп'ютера”), вказує, що кожен екземпляр варіанту A включає в себе функціональні властивості, задані для варіанту B. Ці властивості спеціалізують поведінку відповідного варіанту A на даній діаграмі. Графічно дане відношення позначається пунктирною лінією зі стрілкою (варіант відношення залежності), спрямованої від базового варіанту використання до такого, що включається. При цьому дана лінія зі стрілкою позначається ключовим словом "include" ("включає"), як показано на рис. 4.7.

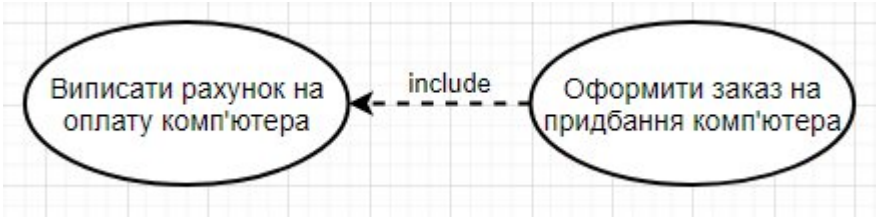


Рисунок 4.7 – Відношення включення

Приклад створення діаграми варіантів використання UML для АІС «RepairTech».

Для побудови діаграми варіантів використання UML для АІС «RepairTech» для користувачів (Адміністратор, Майстер з діагностики, Майстер з ремонту) будемо використовувати опис варіантів використання , який був представлений

на стр. 15-17. Діаграма варіантів використання UML для АІС «RepairTech» надана на рис. 4.8

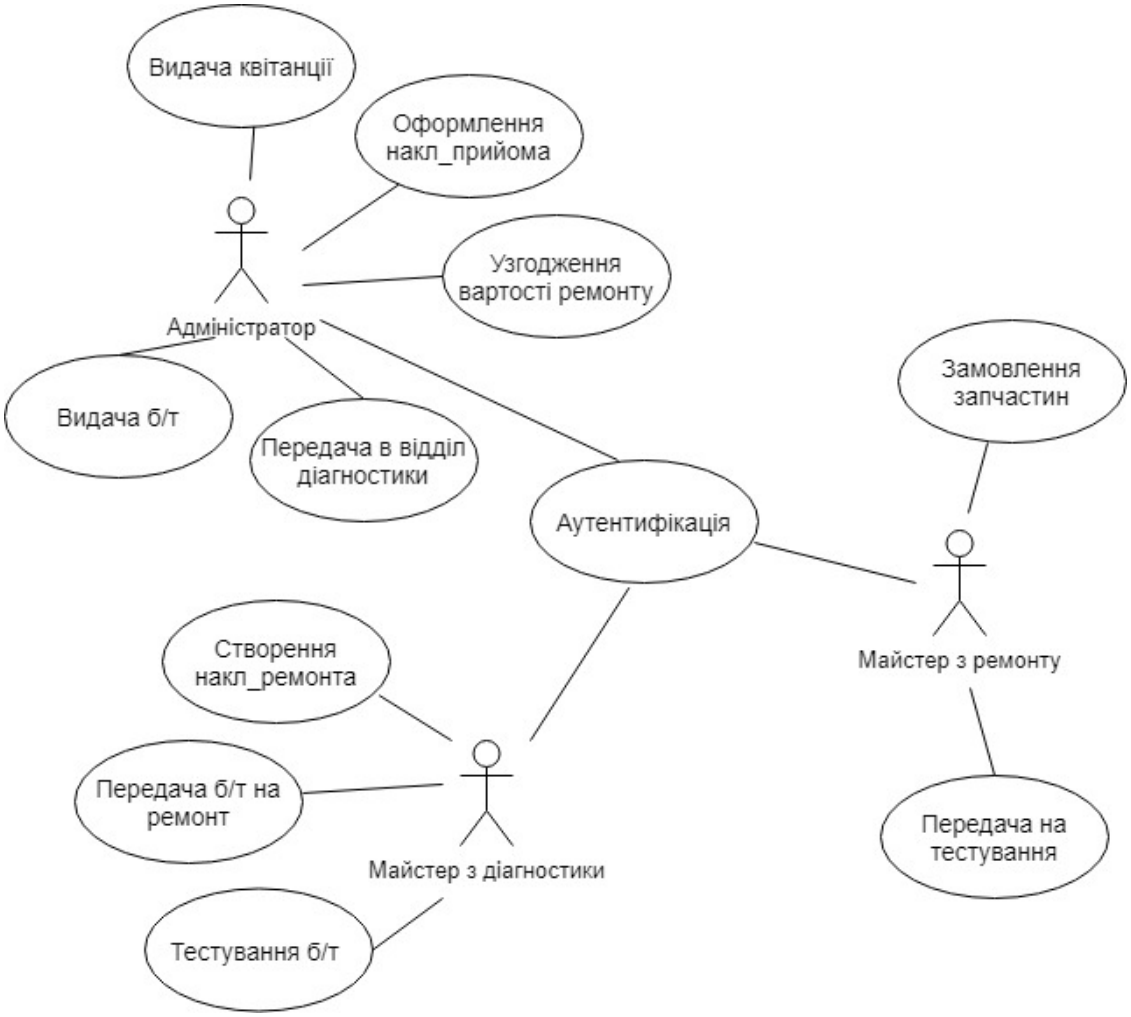


Рисунок 4.8 – Діаграма використання UML для АІС «RepairTech»

Діаграми використання повинні обов’язково бути доповнені сценаріями для кожного прецедента по шаблону,який розглянуто на стр. 13-16.

4.3 Діаграма послідовності (sequence diagram).

Діаграма послідовностей - діаграма взаємодії, в якій основний акцент зроблений на впорядкування повідомлень в часі.

Елементи діаграми.

Об'єкти: на діаграмі послідовності зображаються виключно ті об'єкти, які безпосередньо беруть участь у взаємодії і не показуються можливі статичні

асоціації з іншими об'єктами. Якщо ім'я об'єкта відсутня на діаграмі послідовності, то вказується тільки ім'я класу, а сам об'єкт вважається анонімним.

Лінія життя об'єкту (object lifeline) - це вертикальна пунктирна лінія, яка відображає існування об'єкта в часі. Якщо об'єкт існує протягом усього взаємодії, його зображують у верхній частині діаграми, а його лінію життя промальована від верху до низу. Якщо об'єкт був створений під час взаємодій, лінії життя починаються з отримання повідомлення зі стереотипом create. Якщо об'єкт знищуватися під час взаємодій, то його лінія життя закінчуються отриманням повідомлення зі стереотипом destroy, а в якості візуального образу використовується велика буква X, що позначає кінець життя об'єкта.

Фокус управління (focus of control) - витягнутий прямокутник, що показує проміжок часу, протягом якого об'єкт виконує будь-яку дію, безпосередньо або за допомогою підпорядкованої процедури. Періоди активності об'єкта (коли він виконує будь-яку дію) можуть чергуватися з періодами його пасивності або очікування. В цьому випадку у такого об'єкта є кілька фокусів управління. Отримати фокус управління може тільки існуючий об'єкт, у якого в цей момент є лінія життя. Якщо ж певний об'єкт був знищений, то знову виникнути в системі він вже не може. Замість нього лише може бути створений інший екземпляр цього ж класу, який, строго кажучи, буде іншим об'єктом.

В окремих випадках ініціатором взаємодії в системі може бути актор або зовнішній користувач. В цьому випадку актор зображується на діаграмі послідовності найпершим об'єктом зліва зі своїм фокусом управління. Найчастіше актор і його фокус управління активністю в ініціюванні взаємодій з системою. При цьому сам актор може мати власне ім'я або залишатися анонімним.

Повідомлення (message) являє собою закінчений фрагмент інформації, який відправляється одним об'єктом іншого. При цьому прийом повідомлення ініціює виконання певних дій, спрямованих на вирішення окремого завдання тим об'єктом, з яким це повідомлення надіслано. На діаграмі послідовності всі повідомлення впорядковані за часом свого виникнення в моделюється системі. У такому контексті кожне повідомлення має напрямок від об'єкта, який ініціює та відправляє повідомлення, до об'єкту, який його отримує.

Стереотипи повідомлень:

«Call» (викликати) -сінхронние повідомлення (synchronous message) - вимагають повернення відповіді.

«Return» (повернути) - повернення відповіді на синхронне повідомлення. Краще застосовувати зображення повернення тільки в тих випадках, коли це допоможе краще зрозуміти механізм взаємодії. У всіх інших випадках, варто опускати зображення повернень, тому що вони будуть вносити деяку плутанину. Просто, при використанні синхронного повідомлення, варто пам'ятати, що у нього завжди є повернення.

«Send» (надіслати) - асинхронні повідомлення (asynchronous message) - відповідь не тебується і викликає об'єкт може продовжувати роботу.

«Create» (створити) - створює об'єкт.

«Destroy» (знищити) - видаляє об'єкт.

Тимчасові обмеження на діаграмах послідовності. В окремих випадках виконання тих чи інших дій на діаграмі послідовності може зажадати явною специфікації тимчасових обмежень, що накладаються на сам інтервал виконання операцій або передачу повідомлень. У мові UML для запису тимчасових обмежень використовуються фігурні дужки.

Приклади запису обмежень:

{Время_пріема_сообщенія время_отправкі_сообщенія <1 сек.}

{Время_ожіданія_ответа <5 сек.}

{Время_передачі_пакета <10 сек.}

{Об'єкт_1. время_подачі_сігнала_тревогі> 30 сек.}

Елементи діаграми послідовності демонструються на рис. 4.9.

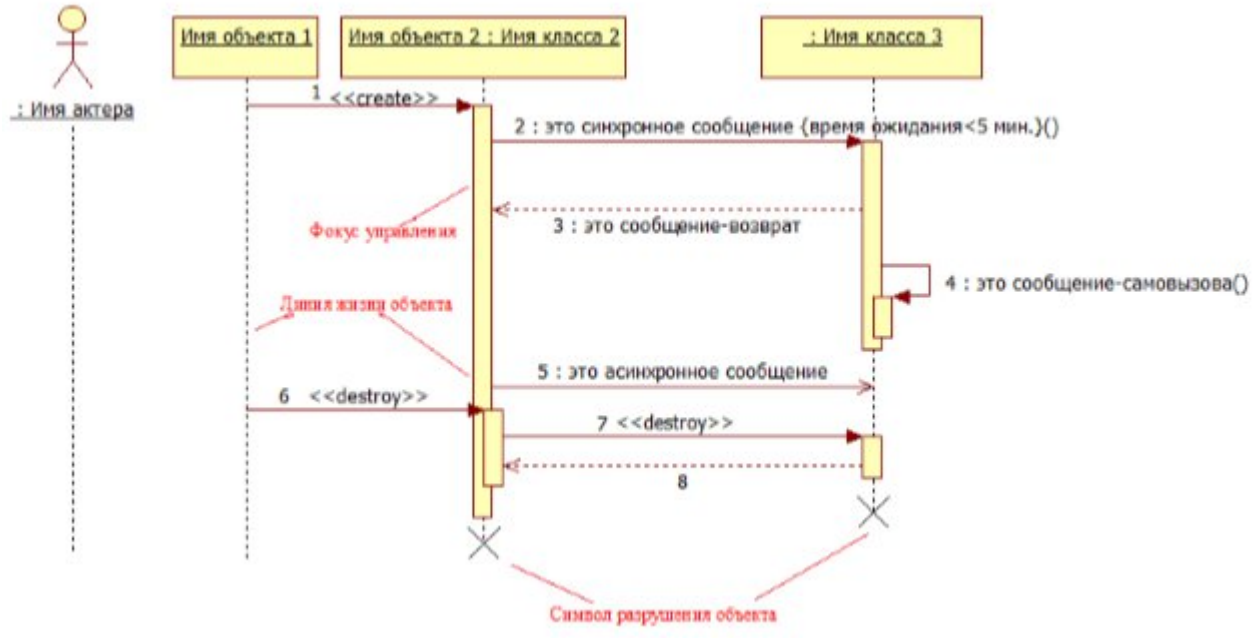


Рисунок 4.9 – Елементи діаграми послідовності.

В межах Моделі прецедентів створюються діаграми послідовності, які демострують обмін повідомленями акторів з системою для кожного прецедента, де система розглядається як «чорний ящик». *Системна діаграма послідовності* (system sequence diagram - SSD) – це схема, яка для певного сценарію прецедента показує події, які генеруються зовнішніми виконавцями, їх порядок, а також події, генеровані всередині самої системи. Назначення цієї діаграми - відображення подій, переданих виконавцями системі скрізь її межі. Дуже важливо знати, що слід розуміти під зовнішніми або системними подіями (системна подія). Вони є важливою частиною аналізу поведінки систем.

Створимо таку діаграму послідовності для прецедента «Оформлення замовлення» для AIC «RepairTech», використовуя його основний сценарій.

Прецедент «Оформлення замовлення». Основні об’єкти: актор «Адміністратор», «Система». Повідомлення «Адміністратор» рис. 4.10: «makeOrder» (створити замовлення), «addMaster» (призначити Майстра з діагностики), «changeStatus» (змінити статус), «printReceipt» (роздрукувати квітанцію), «Save» (зберегти). Повідомлення «Система»: «masterDiag» (номер Майстра з діагностики), «Receipt» (квітанція).

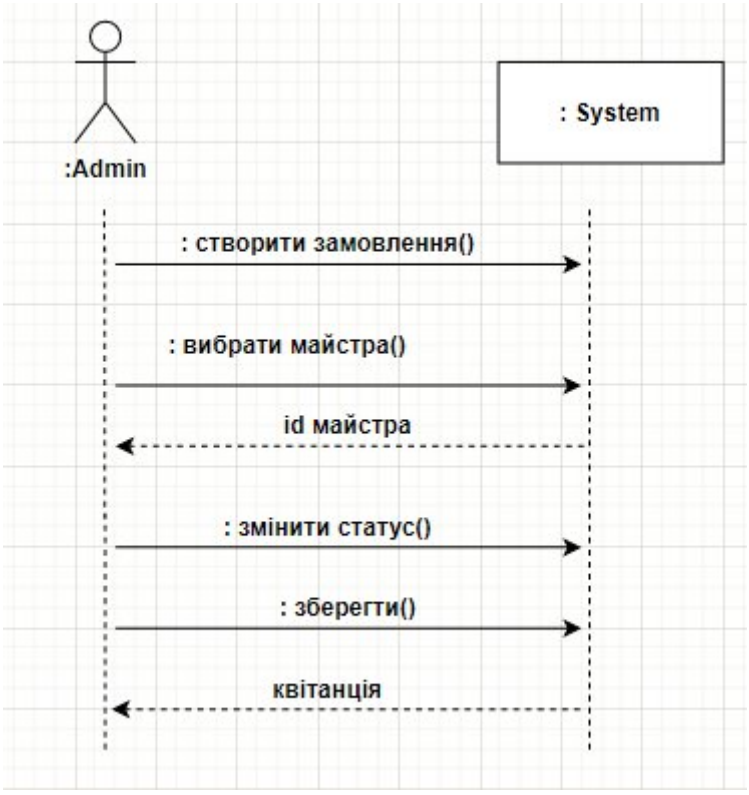


Рисунок 4.10 – Діаграма послідовності для прецедента «Оформлення замовлення»

Питання для самоконтролю

1. Які основні принципи об’єктно - орієнтованого моделювання ПЗ?
2. Надати визначення методології RUP. Фази та артифакти UP.
3. За допомогоюяких артефактів реалізується фаза Вимог? Визначити модель прецедентів.
4. Призначення мови UML. Структура та призначення діаграм UML.
5. Які мета та елементи діаграми прецедентів? Шаблон опису основного потоку подій.
6. Як використавується діаграма послідовності в моделі прецедентів? Мета та елементи діаграм взаїмодії (послідовності).

Завдання до лабораторної роботи №4

1. Відповісти на питання для самоконтролю.
2. На основіопису варіантів використання, які були створені влабораторної роботі 1, створити для власної АІС діаграму прецедентів UML.
3. За шаблоном детально описати 3 найбільш складних прецедентів (основний та альтернативний потік).
4. Створити для всіх прецедентів діаграми послідовності взаімодії Актора та Системи.
5. Оформити протокол лабораторної роботи.

ТЕМА 5. ПОБУДОВА ДІАГРАМИ КЛАСІВ В UML. МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ.

Мета: набути практичних навичок застосування методології об’єктно орієнтірованного моделювання для виявлення об’єктів системи за допомогою моделі предметної області та діаграми класів UML.

5.1 Короткі теорітичні відомості з призначення та елементів діаграми класів UML.

Діаграма класів використовуються для моделювання статичного виду системи з точки зору проектування.

Зазвичай діаграми класів використовуються для таких цілей:

– для моделювання словника системи;

– для моделювання простих кооперацій, **кооперація** - це співтовариство класів, інтерфейсів і інших елементів, що працюють спільно для забезпечення деякого кооперативного поведінки, більш значущого, ніж сума складових його елементів;

– для моделювання логічної схеми бази даних.

Класом (Class) називається опис сукупності об'єктів із загальними атрибутами, операціями, відносинами і семантикою. Можливі класи виявляються при розгляді трьох стереотипів: сутність (entity), межа (boundary) і управління (control). Клас-сутність містить інформацію, яка зберегається постійно. Використовується для моделювання даних і поведінки об'єктів з довгим життєвим циклом. Вони можуть представляти інформацію про предметну область, а можуть представляти елементи самої системи. У UML клас коротко познаяается двома способами рис.5.1

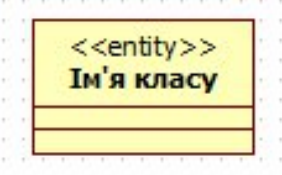


Рисунок 5.1 – Просте зображення класу в UML.

Атрибут - це іменоване властивість класу, що включає опис безлічі значень, які можуть приймати екземпляри цієї властивості. Клас може мати будь-яке число атрибутів або не мати їх зовсім. Атрибути поміщаються на

другий рівень елемента «клас» рис. 5.2. Для іменування атрибута використовують одне або кілька коротких іменників, відповідних деякому властивості класу. Кожне слово в імені атрибута, крім самого першого, зазвичай пишеться з великої літери. Атрибут маєнаступні параметри: видимість, ім’я, тип даних, значення по замовченню, властивості. Окрім ім’я, інші параметри не обов’язкові.

Рисунок 5.2 – Представлення атрибутів в елетенту класу UML.

Операція класу (operation) являє собою деякий сервіс, що надається для кожного екземпляра класу за певним вимогу. Операції записуються на третьому рівні елемента класу. Кожній операції класу в UML відповідає окремий рядок виду:

< видимість> <ім'я операції> (список параметрів): <вираз типу значення, що повертається> {рядок-властивість}

Приклади запису операцій UML:

display - тільки ім'я;

+ Display - видимість і ім'я;

set (n: Name, s: String) - ім'я і параметри;

getID (): Integer - ім'я і повертається значення;

restart () {guarded} - ім'я та властивість.

Часто виникає необхідність створити клас, у якого:

- немає жодного екземпляру - тоді клас стає службовим (Utility), що містить лише атрибути і операції з областю дії класу;
- рівно один екземпляр - такий клас називають синглетним (Singleton);
- заданий число екземплярів;
- довільне число екземплярів - варіант за замовчуванням.

Кількість екземплярів класу називається його кратністю. У мові UML кратність класу задається виразом, написаним в правому верхньому кутку його піктограми.

Об'єкт класу (object) є окремим екземпляром класу, який створюється на етапі виконання програми. Він має своє власне ім'я і конкретні значення атрибутів. На рис 5.3. представлені варіанти імен об’єктів класів.

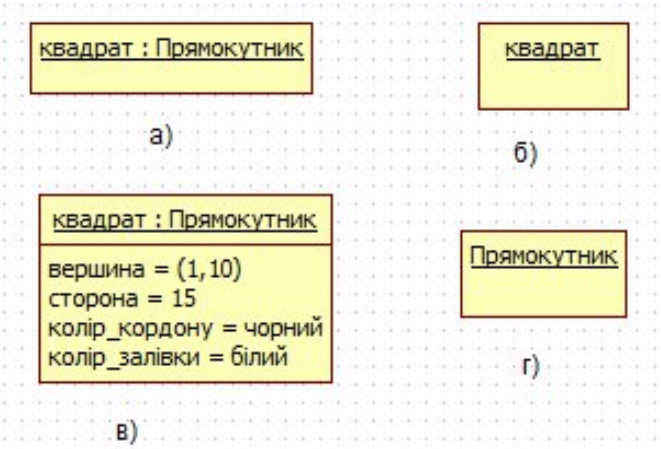


Рисунок 5.3 – Створення елементу об’єкт класу.

Між класами в мові UML можно створювати базовими відносинами або зв'язками:

- ***відносини асоціації*** (Association relationship)
  - Бінарна
  - N-арная
  - Агрегація
  - композиція
- ***ставлення узагальнення*** (Generalization relationship)
- ***ставлення реалізації*** (Realization relationship)
- ***ставлення залежності*** (Dependency relationship).

Ставлення узагальнення - ***це спадкування***. Ставлення узагальнення є звичайним відношенням між більш загальним елементом (батьком або предком) і більш приватним або спеціальним елементом (дочірнім або нащадком). На діаграмі класів зображається прямою лінією з пустим трикутноком на кінці рис. 5.4.

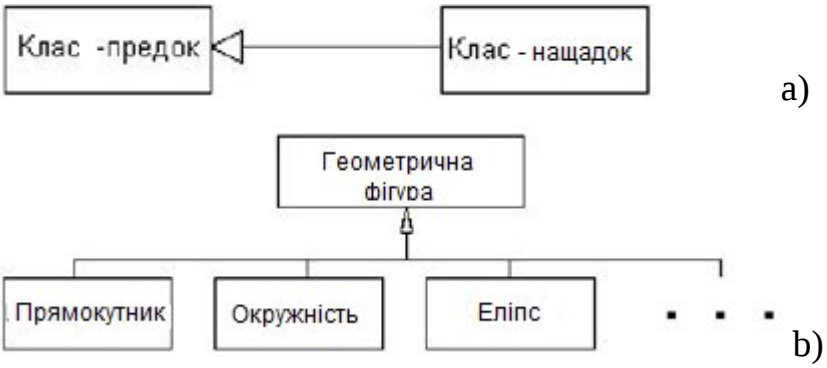


Рисунок 5.4 – зв’язок узагальнення.

Асоціація показує відносини між реалізованими об'єктами класу. Асоціації мають навігацію: двосторонню або односпрямовану, що вказує на напрям зв'язку. В якості додаткових спеціальних символів можуть використовуватися ім'я асоціації, а також імена і кратність класів-ролей асоціації рис. 5.5.

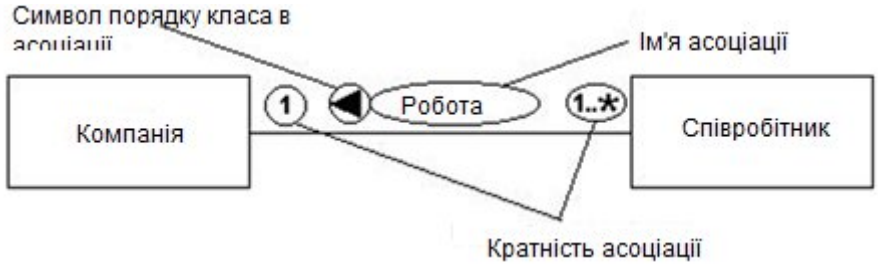


Рисунок 5.5 - Зв’язок асоціації.

Спеціальною формою або окремим випадком відносини асоціації є **відношення агрегації** (з'єднання частин), яке, в свою чергу, теж має спеціальну форму - **відношення композиції**. В UML ***агрегація*** відображає зв'язок класів, коли ***об'єкт одного класу є атрибутом іншого.*** Дане відношення має фундаментальне значення для опису структури складних систем, оскільки застосовується **для подання системних взаємозв'язків типу "частина-ціле"**. Розкриваючи внутрішню структуру системи, відношення агрегації показує, з яких компонентів складається система і як вони пов'язані між собою рис 5.6.

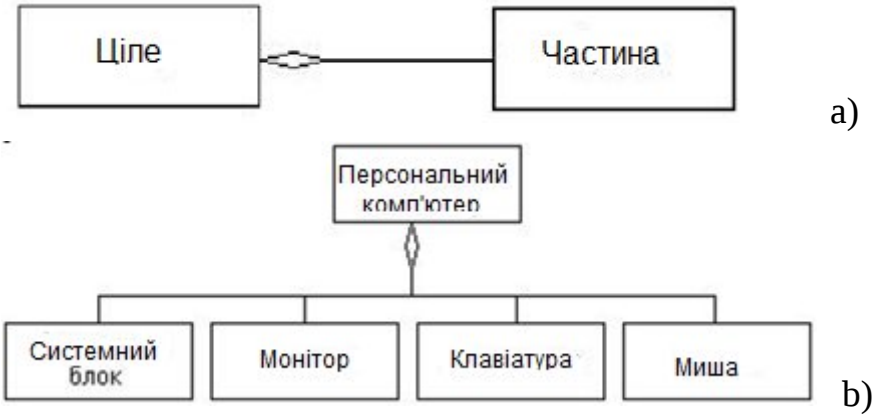
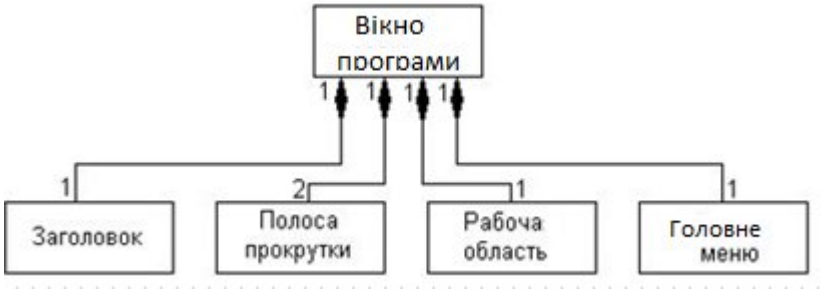


Рисунок 5.6 - Зв’язок агрегації.

Композиція служить для виділення спеціальної форми відносини "частина-ціле", при якій складові частини в деякому сенсі перебувають всередині цілого рис.5.7. 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. Різниця між композицією і агрегацією: компоненти зібрані агрегацией можна роз'єднати, а з композицією цього зробити не вийде.

a)



b)

Рисунок 5.7- Зв’язок композиції.

Реалізація показує відношення: клас - об'єкт. На діаграмі реалізація показується пунктирною лінією і незафарбовані стрілочкою.

Ставлення ***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**. Зображується пунктирною лінією зі стрілкою. Стрілка спрямована від класу-клієнта залежності до незалежного класу або класу-джерела рис 5.8.

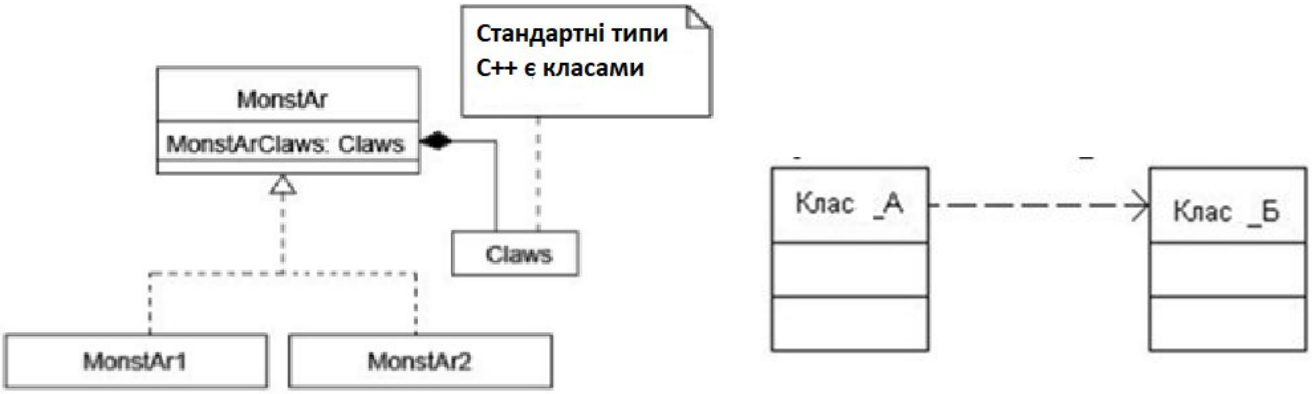


Рисунок 5.8 - Зв’язки реалізації та залежності.

5.2 Модель предметної області в межах методології UP.

Модель предметної області - це найважливіша модель об'єктно-орієнтованого аналізу. Вона відображає основні, тобто моделює класи понять (концептуальні класи) предметної області. Кожній ітерації відповідає своя модель предметної області, яка відображає реалізовані на даному етапі сценарії прецедентів. Таким чином, модель предметної області еволюціонує в процесі розробки системи. Модель предметної області пов'язана з моделлю проектування, особливо програмними об'єктами, відносяться до рівня

предметної області. Основні поняття цієї моделі відображаються в словнику термінів.

Ідентифікація набору концептуальних класів - основне завдання об'єктно-орієнтованого аналізу. На початкових ітераціях фази розвидку побудова моделі предметної області у досвідченого розробника може зайняти всього лише кілька годин, але на наступних етапах, коли вимоги до системи визначаються більш чітко, уточнення моделі предметної області потребує значно більше часу

Модель предметної області в рамках UP є конкретизацією більш загального поняття моделі бізнес-об’єктів (business object model- BOM), що забезпечує подання понять, що грають важливу роль в даній предметній області. Таким чином, подібна модель описує поняття однієї предметної області, наприклад, пов'язані з роздрібною торгівлею.

На мові UML Модель предметної області представляється у вигляді набору діаграм класів, на яких не визначені жодні операції. Модель предметної області може відображати наступне:

– Об'єкти предметної області або концептуальні класи.
– Асоціації між концептуальними класами.
– Атрибути концептуальних класів.

Модель предметної області можна розглядати як візуальний словник важливих абстракцій або словник предметної області. Таким чином, Модель предметної області - це результат візуалізації понять реального світу в термінах предметної області, а не програмних елементів, таких як класи Java або C #. Отже, в моделі предметної області не використовуються наступні елементи.

– Артефакти програмування на зразок вікон або бази даних, якщо тільки розробляється система не є моделлю програмного засобу, наприклад моделлю графічного інтерфейсу користувача.
– Обов'язки або методи.

Модель предметної області не є моделлю даних (data model), до якої по визначенню відносяться дані, що зберігаються на постійному носії, оскільки невідомо, чи потрібно зберігати в базі даних інформацію про класи. Концептуальні класи можуть взагалі не містити атрибутів і грати в предметної області чисто поведінкову, а не інформаційну роль.

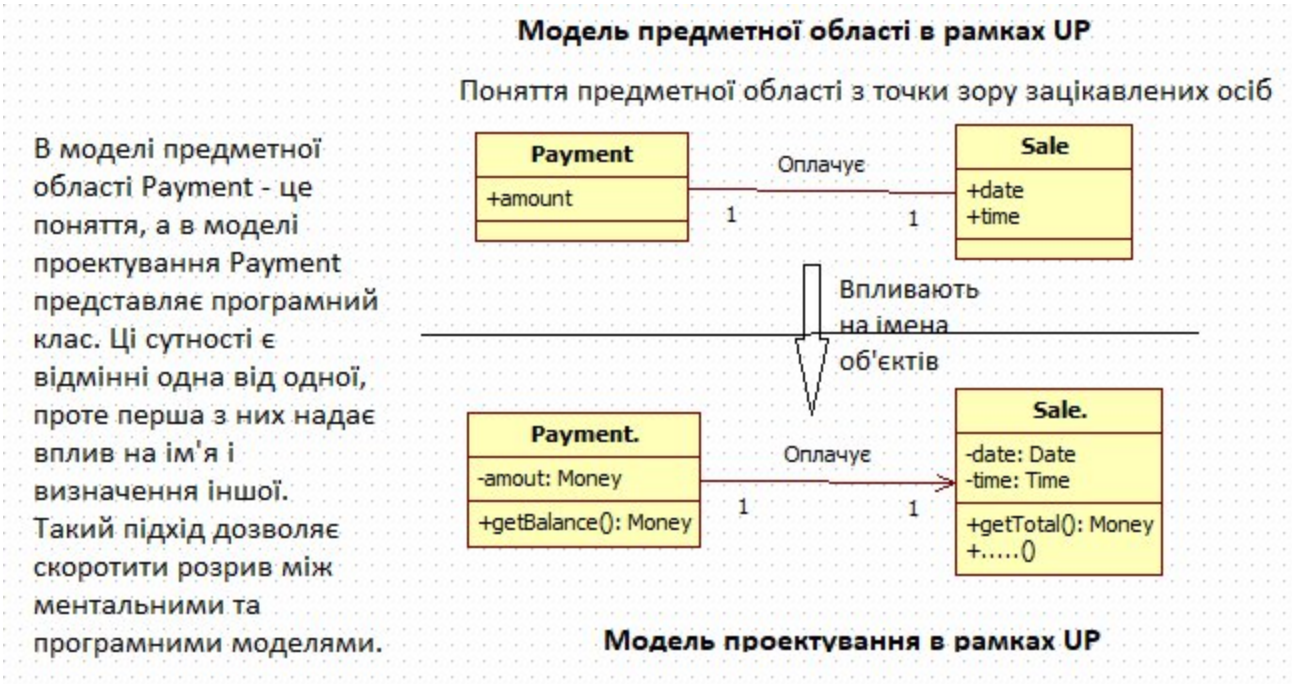


Рисунок 5.9 – Модель предметної області в межах UP.

Для створення Моделі предметної області на даній ітерації треба виконати наступні дії.

- Виділити концептуальні класи (дивись нижченаведені процедури з рекомендаціями).
- Додати їх в моделі предметної області у вигляді класів на діаграмі UML.
- Додати необхідні асоціації і атрибути.

Існує три способи виявлення концептуальних класів.

1. Повторне використання або модифікація існуючих моделей. Це самий перший, найкращий і звичайно, найпростіший підхід, з якого автор завжди намагається почати процес виділення концептуальних класів. В літературі описані моделі багатьох предметних областей і моделі даних, які можна трансформувати в моделі предметної області. Такі моделі існують для галузі фінансів, охорони здоров'я і т.п.
2. З використанням списку категорій концептуальних класів.
3. На основі виділення іменників.

Використання списку категорій концептуальних класів

Напочатку створення моделі предметної області доцільно скласти список кандидатів на роль концептуальних класів. У табл. 5.1 наведені стандартні категорії, які зазвичай мають важливе значення для предметної області системи торгівлі, резервування авіаквитків і гри "Монополія".

Таблиця 5.1 – Категорії концептуальних класів.

Категория	Приклад
Транзакції - ці класи особливо критичні, оскільки часто описують фінансові операції, тому процес виділення концептуальних класів слід почати саме з них	Sale (Продаж), Payment (Платіж), Reservation (Резервування)
Елементи транзакцій - транзакції часто складаються з елементів	SalesLineitem (Елемент продажу)
Товари або служби, пов'язані з транзакціями або їх елементами - транзакції виконуються над деякими елементами (товарами або службами)	Item (Елемент), Flight (Рейс), Seat (Місце)
Місця записи транзакцій - дуже важлива категорія	Register (Реєстр), FlightManifest (Розклад польотів)
Ролі людей або організації, які пов'язані з транзакціями, виконавці прецедентів - необхідно знати, хто бере участь в транзакції	Cashier (Касир), Customer (Покупець), Store (Магазин), MonopolyPlayer (Ігрок) Pilot (Пілот) Passenger (Пасажир)
Місця транзакцій	Store (Магазин), Airport (Аеропорт), Plane (Самоліт), Seat (Місце)
Важливі події, для яких необхідно зберігати час і місце	Sale (Продаж), Payment (Платіж), MonopolyGame (Монополія), Flight (Політ)
Фізичні об'єкти - такі об'єкти зазвичай відповідають програмних системам, які призначені для управління або моделювання	Register (Реєстр), Airplane (Самоліт), Item (Товар), Board (дошка), Die (Игральна кість)
Опис об’єктів	ProductDescription (Специфікація товара), FlightDescription (Опис польота)
Каталоги - опис часто наводиться у каталозі	ProductCatalog (Каталог товарів), FlightCatalog (Каталог рейсів)
Контейнери інших об’єктів (фізичних або інформаціоних)	Store (Магазин), Bin (Бункер), Airplane (Самоліт), Board (Дошка)
Вміст контейнерів	Item (Елемент), Square (Клітина на досці), Passenger (Пасажир)
Інші системи, зовнішні по відношенню до даної системі	CreditAuthorizationSystem (Система авторізації

Категория	Приклад
	кредитних платежів), AirTrafficControl (Система керування рухом)
Записи фінансової, трудової, юридичної та іншої діяльності	Receipt (Чек), Ledger (Гроссбух), MaintenanceLog (Журнал обслуговування)
Финансові інструменти	LineOfCredit (Кредитна лінія), Cash (Готівка), Check (Чек)
Керівництва, документи, статті, книги, на які посилаються в процесі роботи	DailyPriceChangeList (Бюлетень щоденної зміни цін), RepairManual (Керівництво про відновлення)

Визначення концептуальних класів на основі виділення іменників.

Ще один корисний (і дуже простий) прийом ідентифікації концептуальних класів реалізується на основі лінгвістичного аналізу текстових описів предметної області,той самий, який ми використовували при розробці моделі ER - виділення іменників та їх виборі в як кандидатів в концептуальні класи або атрібути.

Розгляне застосування цього методу на прикладі опису головного сценарію для прецедента «Оформити замовлення». Виделимо іменники та категорії.

1 Створити **замовлення** (**клієнт**, тип б/т, назва, номер, що не робить, дата)
2 Додати інформацію
3 Призначити **майстра з діагностики** за **правилами фірми**
4 Призначення **статуса**
5 Роздрукувати **квітакцію**

Можливо, найбільш типовою помилкою при створенні моделі предметної області є зарахування деякого об'єкта до атрибутів, в той час як він повинен відноситися до концептуальних класів. Щоб уникнути цієї помилки, слід дотримуватися простого правила, якщо деякий об'єкт *X* в реальному світі не є числом або текстом, значить, це скоріше концептуальний клас, ніж атрибут. Наприклад, чи є Store (магазин) атрибутом об'єкта Sale (Продаж) або окремим концептуальним класом Store? У реальному світі магазин не є числом або текстом, він представляє реальну сутність, організацію, що займає деякий місце. Отже, Store потрібно розглядати як концептуального класу.

Можно видилити наступні категорії для виявлення концептуальних класів прецедента «Оформлення замовлення»: Транзакції (замовлення), Товари або

служби, пов'язані з транзакціями або їх елементами (б/т) , Ролі людей або організації, які пов'язані з транзакціями, виконавці прецедентів (клієнт, Адміністратор, Майстер з діагностики), Місця транзакцій (Фірма), Важливі події, для яких необхідно зберігати час і місце (замовлення) , Фізичні об'єкти (б/т) , Опис об’єктів (Специфікація б/т), Записи фінансової, трудової, юридичної та іншої діяльності (квітанція), Керівництва, документи на які посилаються в процесі роботи (правила).

Як бачимо, кількість іменників та категорій значно різниться, тому що категорії вже враховує рекомендації для виявлення концептуальних класів для деяких предметних областей. Об’єкт Квитанція треба враховувати, якщо квітанція буде використовуватися при поверненні б/т в межах гарантії, інакше – це просто звіт про замовлення. Іменник «статус» є атрибутом, тому що це просто строка. Таким чином, можемо видилити наступні концептуальні класи для прецедента:

Order (замовлення)

Client (клієнт)

Diagnostic master (майстер)

Admin (адміністратор)

Appliance (б/т)

ApplianceDescription (Специфікація б/т) – потрібен, якщо існує необхідність опису елементів або служб незалежно від існування конкретних екземплярів цих об'єктів.

Firm (фірма)

Rules (правила)

Виявлення асоціацій між концептуальними класами в прецеденті «Оформлення замовлення».

Асоціація (association) - це відношення між класами (або точніше, екземплярами цих класів), що відображає деякий значущі і корисні зв'язки між ними. У мові UML, як це було описано вище, асоціації описуються як "семантичні взаємозв'язки між двома або кількома класифікаторами і їх екземплярами ". Заслуговують на увагу асоціації, що зазвичай містять знання про взаємозв'язок між об'єктами, які повинні зберігатися протягом деякого періоду. Цей період може вимірюватися в мілісекундах або роках в залежності від конкретного контексту.

Лінії асоціацій в моделі предметної області не описують потоки даних, зовнішні ключі баз даних, змінні примірників або зв'язку між об'єктами в програмної реалізації. Такі асоціації слід розглядати лише в концептуальному ракурсі - як зв'язок в реальній предметній області. Багато з цих зв'язків будуть реалізовані в програмному забезпеченні (як в моделі проектування, так і в моделі даних). Однак модель предметної області це не модель даних. Додані до неї ***асоціації дозволяють лише краще зрозуміти взаємозв'язок між об'єктами, але не відображають структури даних***.

Імена асоціацій повинні починатися з великої літери, оскільки асоціація зазвичай являє класифікатор зв'язків між екземплярами. У мові UML ім'я класифікатора починається з великої літери. Для імен асоціацій прийнято використовувати два формати: Records-Current (з дефісом) чи RecordsCurrent (без дефіса). Кожен кінець асоціації називається роллю (role). Роль додатково може мати такі характеристики: кратність, ім’я, напрямок зв’язку рис 5.10.

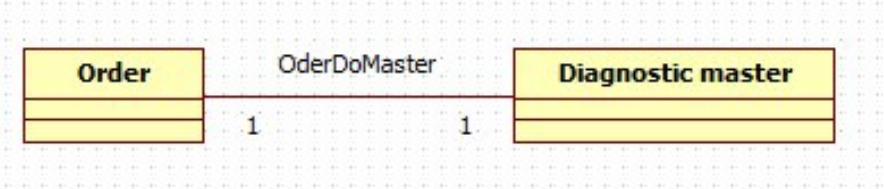


Рисунок 5.10 – Концептуальні класи та асоціація.

Пошук асоціацій за допомогою списку стандартних асоціацій.

Приступимо до додавання асоціацій з використанням списку, представленого в табл. 5.2. У ньому вказані стандартні категорії, якими зазвичай не слід нехтувати, особливо в комерційних інформаційних системах. Приклади асоціацій взяті з предметних областей резервування авіаквитків, роздрібної торгівлі та гри "Монополія".

Таблиця 5.2 – Стандартні асоціації.

Опис	Назва асоціації
A є транзакцією, яка пов'язана з іншого транзакцією ***B***	CashPayment-Sale (Платіж готівкою -Продаж) Reservation-Cancellation (Заказ білета-Скасування заказу)
A є елемент транзакції	SalesLineitem-Sale (Елемент продажі-Продаж)
A є товаром чи пуслугою для транзакції ***B***	Item-SalesLineitem (Елемент-Елемент продажу)

Опис	Назва асоціації
	Flight·Reservation (Рейс-Резервування)
A є роль, яка пов’язана з транзакцією ***B***	Customer·Payment (Покупець-Платіж) Passenger-Ticket (Пасажир-Білет)
A є физичною або логичною частиною ***B***	Drawer-Register (Пристрій друку торгових чеків-Реєстр) Seat-Airplane (Місце-Самоліт) Square-Board (Клітина-Дошка)
A физично або логично міститься у ***B***	Register-Store (Реєстр-Магазин) Item-Shelf (Товар-Полка) Square-Board (Клітина-Дошка) Passenger-Airplane (Пасажир-Самоліт)
A є описом ***B***	ProductDescription-Item (Опис товара-Товар) FlightDescription-Flight (Опис польоту-Політ)
A відомий / зареєстрований / записаний / включений в ***B***	Sale-Register (Продаж-Реєстр) Piece-Square (Об’єкт-Клітина) Reservation-FlightManifest (Замовлення білета-Декларація)
A є членом ***B***	Cashier-Store (Касир-Магазин) Player-MonopolyGame (Ігрок-Гра "Монополія") Pilot-Airline (Пілот-Авіарейс)
A є организацийною одиницею ***B***	Department-Store (Відділ-Магазин) Maintenance·Airline (Послуга підтримки-Авіарейс)
A використовує, керує або володіє ***B***	Cashier-Register (Касир-Реєстр) Player-Piece (Ігрок-Об’єкт) Pilot-Airplane (Пілот-Самоліт)
A слідує за ***B***	SalesLineitem-SalesLineitem (Назва товару - Назва товару) city-City (Місто-Місто) Square-Square (Клітина-Клітина)

Визначемо асоціації для прецедента «Оформлення замовлення», застосовуя таблицю стандартних асоціацій.

A є товаром чи пуслугою для транзакції ***B*** (Order записує прийом Appliance для ремонту)

A є роль, яка пов’язана з транзакцією ***B*** (Client ініціїрує Order, Admin оформлює Order),

A є членом ***B*** (Diagnostic master призначається для виконання Order),

A є описом ***B*** (ApplianceDescription описує Appliance),

A физично або логично міститься у ***B*** (Order належить Firm).

Виявлення атрибутів концептуальних класів.

В Моделі предметної області додаються атрибути, для яких визначені відповідні вимоги (наприклад, прецеденти) або для яких необхідно зберігати певну інформацію. Атрибути розміщуються у другому розділі условного зображення класу. Додатково може бути вказано також тип атрибута або інша необов'язкова інформація. Рекомендується описувати вимоги до атрибутамів у словниках термінів, який в майбутньому використовується як словник даних.

В Моделях предметної області атрибути повинні бути типами даних (data type). До стандартного типу атрибутів відноситься логічне значення, дата, номер, рядок (текст), час. Інші часто використовувані типи: Адреса (адреса), Колір (колір), геометрія (точка, прямокутник) (геометричні фігури: точка, прямоугольник), Номер телефону (номер телефону), Номер соціального страхування (номер страхового поліса), Універсальний код товару (UPC) (універсальний код товару), SKU, ZIP або поштовий індекс (почтовый индекс), перерахуємиє типы. Стандартна помилка - це моделювання складного поняття предметної області у формі атрибута, наприклад, зовнішній ключ сутності, які потрибно відображати через асоціації.

Тип даних, особливо розглядається як число або строка, може бути представлені у вигляді нового типу даних у моделях предметних областей у наступних випадках:

- Якщо він складений з окремих частей (Номер телефону, ім’я людини).
- Якщо з цим типом зазвичай асоціюються операції, такі як синтаксичний аналіз і перевірка (Номер страхового поліса).
- Якщо він містить інші атрибуги. Для льготної ціни можуть встановлюватися термінові дії (початок і кінець).
- Якщо цей тип використовується для завдання кількості з одиницями розмірності (Вартість товару вимірюється в деяких одиницях).
- Це абстракція одного або кількох типів. Ідентифікатор товару - це кілька об’єднань типів UPC (Universal Product Code) і EAN (European Article Number).

Якщо клас представляє тип даних (унікальна тождественність не використовується в перевірці на равенство), його можна помістити в розділ атрибутів відповідного класу. Якщо же клас типу даних є новим типом і у нього

є власні атрибути та асоціації, до більшого інтересу він буде представляти в якості окремого концептуального класу.

Більшість кількісних сутностей неможно представляти у вигляді кількох типов данных. Візьміть, до прикладу, вартість або швидкості. З цими кількісними сутностями пов'язані певні одиниці виміру, наприклад, якщо програмна система призначена для використання в різних країнах, тому вона повинна підтримувати різні типи валют. Рішення включає в себе окремий концептуальний клас Quantity (Кількість), з яким буде асоціюватися клас Unit (Єдиниця виміру). Наскільки кількість розглядається в якості даних типу, відповідний атрибут можна розмістити в розділі атрибутів. Зазвичай кількість вимірюється в деяких одиницях. Гроші - це кількість, одиниця вимірювання якої служить тип валюти. Вага - це кількість, яка вимірюється в килограмах або фунтах.

Розглянемо атрибути концептуальних класів для прецедента «Оформлення замовлення», які відповідають інформаційним вимогам першої ітерації сценарію. Опис сценарію предедента дозволяє виявити наступні атрибути, які пов’язані с необідностью друку квітанції – звіту про замовлення:

Order (datetime)

Client (lastname)

Diagnostic master (id)

Admin (id)

Appliance (type, numbur)

ApplianceDescription (descreption)

Firm (name, address)

Створемо діаграму моделі предметної області прецедента «Оформлення замовлення» рис. 5.3

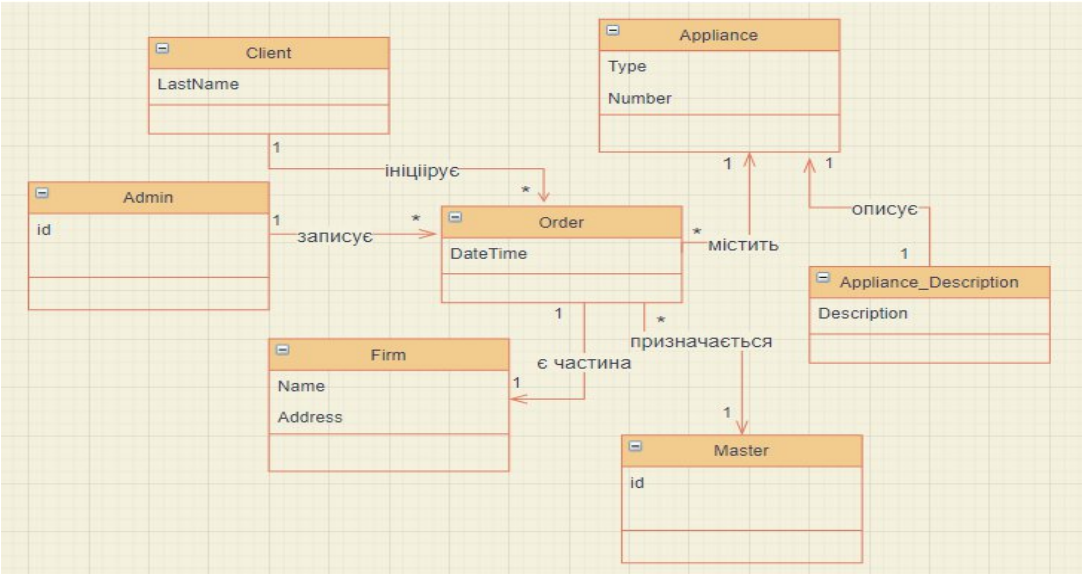


Рисунок 5.11 – Діаграма моделі предметної області для прецедента «Оформлення замовлення».

При ітеративній розробці моделі предметної області поступово еволюціонує протягом кількох ітерацій. На кожній ітерації будується модель, яка відноситься тільки до розглянутого сценарію та не охоплює всі можливі концептуальні класи та взаємосвязи між ними. Наприклад, на першій ітерації системи «RepairTech» розглядається лише непрофесійний сценарій прецедента «Оформлення замовлення», без друку квітанції замовлення, тому модель предметної області, яка створюється на цій ітерації, відображає лише поняття, необхідні для даної ітерації сценарія.

Виявлення системних операцій концептуальних класів.

Аналіз необхідних операцій в межах прецедента починається з детального опису системних операцій. Системними називаються операції, що знаходяться у відкритому інтерфейсі системи для обробки вхідних системних подій, які системи виконують як чорний ящик. Системні операції можна ідентифікувати на основі системних подій за допомогою системної діаграми послідовності (рис. 4.10). Вхідні системні події вимагають обробки з допомогою системних операцій. Вони обробляються за допомогою об'єктно-орієнтованих методов.

Шаблон опису опрерації виглядає так:

Операція – ім’я операції та її параметри;

Посилання - прецеденти, в рамках яких може виконуватися ця операція;

Передумова - пропозиції про стан систем або об'єктів моделі предметної області до виконання операцій. Виконання цих умов не перевіряється в рамках логіки виконання цієї операції, а передбачається, що вони справедливі;

Постумова - це найважливіший розділ, де декларуються зміна стану моделі предметної області.

Існують наступні категорії постумовного опису:

- створення екземпляру,
- формування або розрив асоціацій,
- зміна атрибуту.

Розрив асоціацій зустрічається дуже рідко. Наприклад, розрив асоціації відбуваються при виплаті кредиту або виходу людини з деякого товариства.

Рекомендації по створеню опису системних операцій.

1. Визначте системні операції за допомогою діаграми послідовності.

2. Складіть опис для складних системних операцій, результати яких не очевидні за описом прецедента.

3. При опису постумови використовуйте категорії.

4. Постуумова повинна описувати стан системи, а не дії, які вона виконує, тому іх треба сформулювати в минулому часі, щоб підкреслити зміни , які вже відбулися, наприклад:

- (добрий варіант) *створено екземпляр SalesLineitem*,
  а не
- (поганий варіант) *створення екземпляру SalesLineitem*.

1. Не забудьте встановити відносини між існуючими та новими об'єктами шляхом формування асоціацій. Наприклад, при виконанні операцій enteritem недостатньо просто створити новий екземпляр запису про покупки товару SalesLineitem. Після виконання операції цей екземпляр повинен бути пов'язаний з поточною продажем Sale. Тому однією із постумов цієї операції є наступний стан:
   - вновіть створений екземпляр SalesLineitem, пов'язаний з об'єктом Sale (сформована асоціація).

Опис операцій - чудовий засіб аналізу вимог або ООА, що дозволяє детально описати зміни, до яких приводять системні операції в термінах моделі предметної області, не вдаючись до деталей їх реалізації. Опис операцій потрібен тільки для складних подій, які не достатьо описані в сценарію прецедента, тобто такий опис системної операції можна розглядати як ще один механізм формулювання вимог.

Приклад опису подій Створити Замовлення та Призначити Майстра прецедента «Оформлення замовлення».

1. Операція – makeOrder();
   Посилання – прецеденти: «Оформлення замовлення».
   Передумова – відсутня
   Постумова -
   - створен екземпляр ord об’єкта Order *(створення екземпляру);*
   - екземпляр об’єкта Order пов’язан з класом AppliensDiscription *(формування асоціації);*
   - атрибуту ord.datetime присвоєно значення DateTime *(модіфікація атрибуту).*

2. Операція – getMaster();

Посилання - прецеденти: «Оформлення замовлення».

Передумова – ініціїровано замовлення на ремонт (створено екземпляр класу Order)

Постумова –

- створено список екземплярів ListOrders[] об’єктів Order *(створення екземпляру);*
- екземпляр ListOrders[i] об’єкта Order пов’язан з класом Order *(формування асоціації);*
- атрибуту ord.id_master присвоєно значення ListOrders[min].id_master *(модіфікація атрибуту)*
- атрибуту ord.status присвоєно значення String “diagnostic” *(модіфікація атрибуту).*

5.3 ***Логічна архітектура*** (logical architecture) описує систему в термінах її принципової організації у вигляді рівнів, пакетів (або просторів імен), програмних класів і підсистем. Вона називається логічною, оскільки не визначає способи розгортання цих елементів в різних операційних системах або на фізичних комп'ютерах в мережі (це відноситься до архітектури розгортання (dep1oyment architecture)). Логіка структури системи як правило визначається в документі «Додаткові вимоги». В інформаційних системах зазвичай використовується гнучка многошарова архітектура (шаблон Layers), при якій об'єкти можуть звертатися до декількох шарів, які розташовані нижче.

Шар (layer) - це великомасштабна група класів, пакетів або підсистем, які мають подібні обов'язки для більшості аспектів системи. До типових шарів об'єктно-орієнтованої системи відносяться наступні.

• Інтерфейс користувача (user interface).

• Рівень додатка або об'єктів предметної області (application logic and domain objects). До цього рівня відносяться програмні об'єкти, що представляють поняття предметної області (наприклад, програмний клас Sale) і забезпечують виконання вимог до системи (наприклад, обчислення загальної вартості покупки).

• Рівень технічних служб (technical services). Об'єкти і підсистеми загального призначення, що забезпечують підтримку взаємодії з базою даних або журналів реєстрації помилок. Ці служби зазвичай незалежні від додатка, і їх можна повторно використовувати в декількох системах.

Використання шаблону Layers дозволяє вирішити наступні можливі проблеми.

– Зміна вихідного коду тягне за собою корегування всіх елементів системи, оскільки всі частини системи тісно пов'язані один з одним.

– Логіка додатка переплітається з інтерфейсом користувача, тому в додатку не можливо змінити інтерфейс або принципи реалізації логіки додатки.

– Загальні технічні служби тісно пов'язані з бізнес-логікою додатка, тому їх не можна використовувати повторно, поширити на інші системи або змінити їх реалізацію.

– Через високий ступінь зв'язування складно модифікувати функції програми, масштабувати систему або переходити на нові технології.

Переваги використання шаблону Layers.

– В цілому, цей шаблон забезпечує поділ різних аспектів, високорівневих служб від низькорівневих, спеціалізованих функцій від загальних. Це знижує рівень зв'язування і залежності в додатку, підвищує ступінь зачеплення, збільшує потенціал повторного використання і вносить додаткову ясність.

– Складні елементи піддаються декомпозиції і підлягають інкапсуляції.

– Деякі рівні замінюються новими реалізаціями. У загальному випадку це не можливо для низькорівневих технічних служб і базового рівня (зокрема, java.util), однак цілком реально для шарів інтерфейсу, додатка і предметної області.

– Нижні шари містять повторно використовувані функції.

– Деякі шари (особливо шари предметної області та технічних служб) можуть бути розподіленими.

– Логічна сегментація забезпечує можливість роботи над додатком групи розробників.

5.4 Діаграми пакетів UML

Діаграми пакетів UML використовуються для ілюстрації логічної архітектури додатку - рівней, підсистем, пакетів. Кожен рівень можна представити у вигляді пакету, наприклад, рівень інтерфейсу користувача можна моделювати за допомогою пакета UI.

Діаграма пакетів забезпечує один із способів угруповання елементів. В одному пакеті UML можуть об'єднуватися різні елементи: класи, інші пакети, прецеденти і т.д. Найчастіше використовуються вкладені пакети. Пакет UML – це більш загальне поняття, ніж пакет Java або простір імен .NET, які можуть включатися до складу пакету UML. Ім'я пакета розташовується на корінці, якщо

на діаграмі відображається внутрішнє вміст пакету, або на самому позначенні пакету.

Найчастіше зображують залежності між пакетами, щоб розробникам була зрозуміла взаємозв'язок елементів системи. Для цього використовується лінія залежності UML (dependency line), що представляє собою пунктирну лінію зі стрілкою, спрямовану в сторону незалежного пакета. Пакет UML є простір імен (namespace), тому клас Date може бути визначений в двох різних пакетах. Якщо потрібно забезпечити повну кваліфікацію імен, використовується позначення виду java:: util:: Date, що описує вкладеність пакетів java і util.

У мові UML існують і інші позначення вкладених пакетів. Іноді вкладені пакети графічно розміщуються всередині більш загального пакету. Різні позначення вкладених пакетів показані на рис. 5.12.

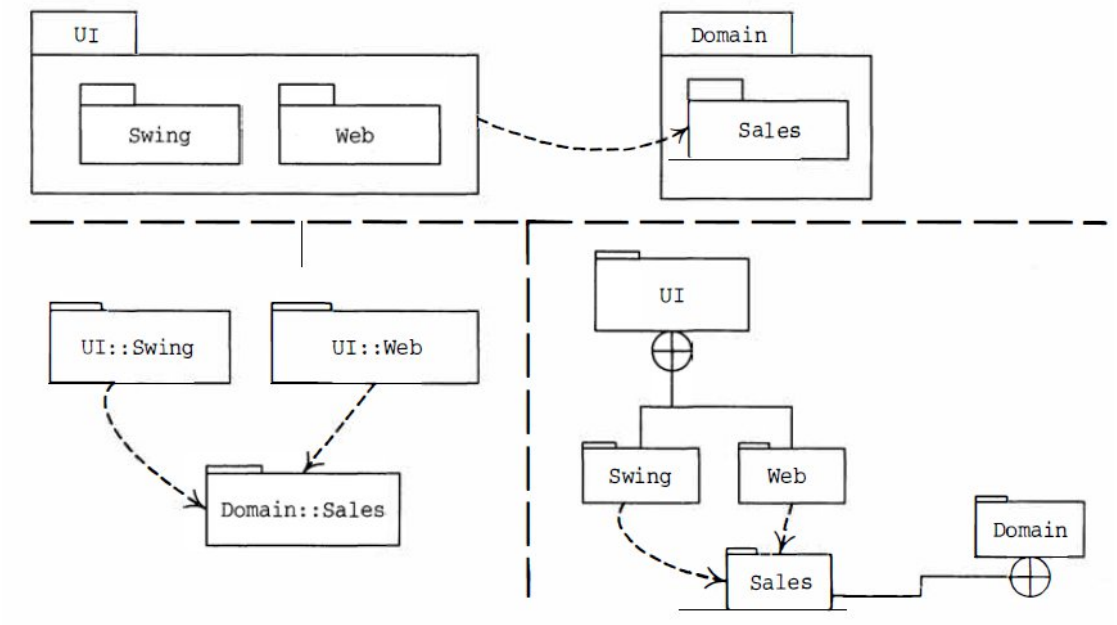


Рисунок 5.12 – Зображення шарів на пакетній діаграмі UML.

5.5 **Шар предметної області або логіки додатку.**

З точки зору аналізу та моделювання програмної системи детально будемо розглядати тільки шар предметної області, який є ядром проекту. Як спроектувати логіку додатка на основі об'єктного підходу?

Можна створити один клас XYZ і помістити в нього все методи, які реалізують логіку предметної області. Він буде працювати (хоча його дослідження виявиться справжнім кошмаром), однак це поганий спосіб.

Інакше вирішити цю проблему можно, якщо створити програмні об'єкти, імена яких відповідають сутностям предметної області, і присвоїти їм обов'язки

по реалізації логіки додатка. Наприклад, в реальному світі існує POScістема, обробна продажу і платежі. Тому має сенс створити класи Sale і Payrnent, призначивши їм відповідні обов'язки. Такі програмні об'єкти називаються об’єктами предметної області (domain object). Вони забезпечують реалізацію бізнес-логіки, наприклад, об'єкт Sale відповідає за обчислення вартості покупки. У цьому контексті шар логіки додатка слід називати шаром предметної області (domain layer).

Ще одне важливе питання: який є взаємозв'язок між шаром і моделлю предметної області? Модель предметної області (що представляє собою візуалізацію понять предметної області) служить для виділення імен класів шару предметної області. Шар предметної області є частиною програмної реалізації, а модель предметної області - частиною концептуального аспекту аналізу. І це різні речі. Однак, створюючи об'єкти шара предметної області на основі відповідної моделі, ми скорочуємо розрив уявлення (representational gap) між ріальним світом і його програмної реалізацією. Наприклад, концептуальний клас Sale з моделі предметної області стає прототипом програмного класу Sale шара логіки додатка в моделі проектування.

Відповідно до принципу Model-View Separation (реалізовано, наприклад, в архітектурному шаблоні MVC), об'єкти моделі (шар предметної області) не повинні безпосередньо взаємодіяти з об'єктами шара уявлення. Наприклад, об'єкт Register або Sale не повинен безпосередньо відправляти повідомлення про зміну кольору або про закриття об'єкту вікна ProcessSaleFrame. Деякий "пом'якшення" цього принципу забезпечує шаблон Observer, згідно якому об'єкти рівня предметної області відправляють повідомлення об'єктам інтерфейсу користувача, що розглядаються тільки в якості сполучних об'єктів між шарами (PropertyListener).

Системна операція, обробка якої показана на системній діаграмі послідовностей, забезпечує передачу запитів з рівня уявлення на рівень додатка або предметної області.

Побудова пакерної діаграми для АІС “RepairTech” рис. 5.13.

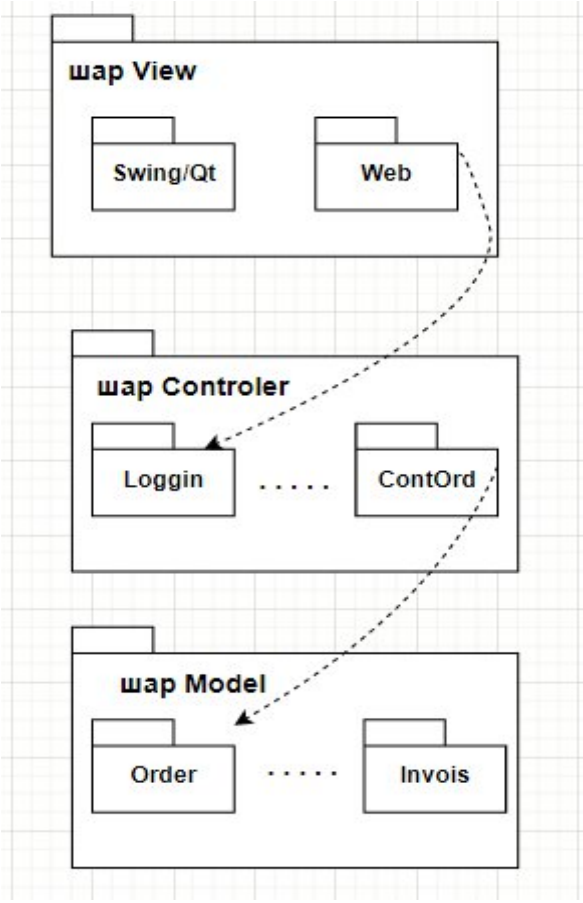


Рисунок 5.13 – Діаграма пакетів для прецедента «Оформлення замовлення».

Питання для самоконтролю

1. Перерахувати статичні діаграми UML. З якою метою створюються діаграми классів?
2. Призначення, зображення та правила наіменування елементів діаграми класів: клас, атрибут, опреація.
3. Які типи зв’язків підтримуються на діаграмах класів? Описати зображення основних типів зв’язків.
4. Надати визначення моделі предметної області (МПО) в межах методології UP. На якій фазі ЖЦ UP будується МПО? Що таке концептуальний клас МПО? Як зображається МПО засобами UML?
5. Які методи використовуються для виявлення концептуальних класів?
6. Які методи використовуються для виявлення асоціацій концептуальних класів?
7. Як визначити атрибути концептуальних класів?
8. Перерахувати статичні діаграми UMl. З якою метою створюються діаграми классів?

9. Призначення, зображення та правила наіменування елементів діаграми класів: клас, атрибут, опреація.
10. Які типи зв’язків підтримуються на діаграмах класів? Описати зображення основних типів зв’язків.
11. Надати визначення моделі предметної області (МПО) в межах методології UP. На якій фазі ЖЦ UP будується МПО? Що таке концептуальний клас МПО? Як зображається МПО засобами UML?
12. Які методи використовуються для виявлення концептуальних класів?
13. Які методи використовуються для виявлення асоціацій концептуальних класів?
14. Як визначити атрибути концептуальних класів?

Завдання до лабораторної роботи №5

1. Відповісти на питання для самоконтролю.
2. Розробити модель предметної області для власного варіанту концептуальні класи, асоціації, атрибути).
3. За шаблоном описати 3 найбільш складних операції одного з прецедентів власної АІС, застосовуя СДП (системна діаграма послідовності).
4. Оформити протокол лабораторної роботи.

ТЕМА 6. ПОБУДОВА ДІАГРАМ ВЗАЇМОДІЇ ДЛЯ ВИЯВЛЕННЯ РОЗПОДІЛУ ОБОВ’ЯЗКІВ ОБ’ЄКТІВ.

Мета: набути практичних навичок розподілу обов’язків між об’єктів програмної системи на основі базових принципів об’єктно орієнтованного моделювання (шаблони GARSP) за допомогою діаграм взаїмодії.

6.1 Принціпи розподілу обов’язків між об’єктів програмної системи (шаблони GARSP).

Після створення основних документів та моделей (артефактів) початкової фази розробки програмної системи (Модель прецедентів, Словник термінів, Додаткова спеціікація, Системна діаграма послідовності, Модель предметної області) переходимо до фази розвитку, де в рамках дисципліни об'єктно-орієнтоване проектування будемо моделювати та проектувати програмну систему. Для створення діарами класів системи необхідно визначити, які операції будуть забеспечувати взаїмодію об'єктів системи для виконання її фунціональності. Тобто треба здійснити розподіл обов'язків між взаємодіючими об'єктами, використовуючи різні принципи об'єктно-орієнтованого проектуваннявання, наприклад шаблони GRASP або GoF (Gang-ofFour).

Найчастіше проектування програмних об'єктів і великомасштабних компонентів описують в термінах обов'язків, ролей і кооперації. Це частина більш широкого підходу, що отримав назву проектування на основі обов'язків (Responsibllity-driven design- RDD). У RDD вважається, що програмні об'єкти мають певні обов'язки - абстракції, що реалізуються ними. В UML обов'язок (responsibllity) визначається як "Контракт або зобов'язання класифікатора". Обов'язки описують поведінку об'єкта в термінах його ролей. У загальному випадку можна виділити два типи обов'язків: ***знання*** (knowing) і ***дію*** (doing).

Обов'язки, які відносяться до дій об'єкта.

- Виконання деяких дій самим об'єктом, наприклад створення екземпляра або виконання обчислень.
- Ініціювання дій інших об'єктів.
- Управління діями інших об'єктів і їх координування.

Обов'язки, які пов'язані з знанням об'єкта.

- Наявність інформації про закриті інкапсульованих даних.
- Наявність інформації про пов'язаних об'єктах.
- Наявність інформації про наслідки або обчислюваних величинах.

Обов'язки присвоюються об'єктам в процесе об'єктно-орієнтованого проектування. Наприклад, можна сказати, що об'єкт *Firm* відповідає за створення примірника *Order* (дія) або що об'єкт *Appliance* відповідає за наявність інформації про битову техніку (знання).

Обов'язки, що відносяться до розряду "знань", часто випливають з моделі предметної області, оскільки вона ілюструє атрибути і асоціації. Наприклад, якщо в моделі предметної області сутність *Order* містить атрибут *date*, то з метою зменшення розриву в уявленнях програмний клас *Order* теж повинен "знати" дату відповідного замовлення.

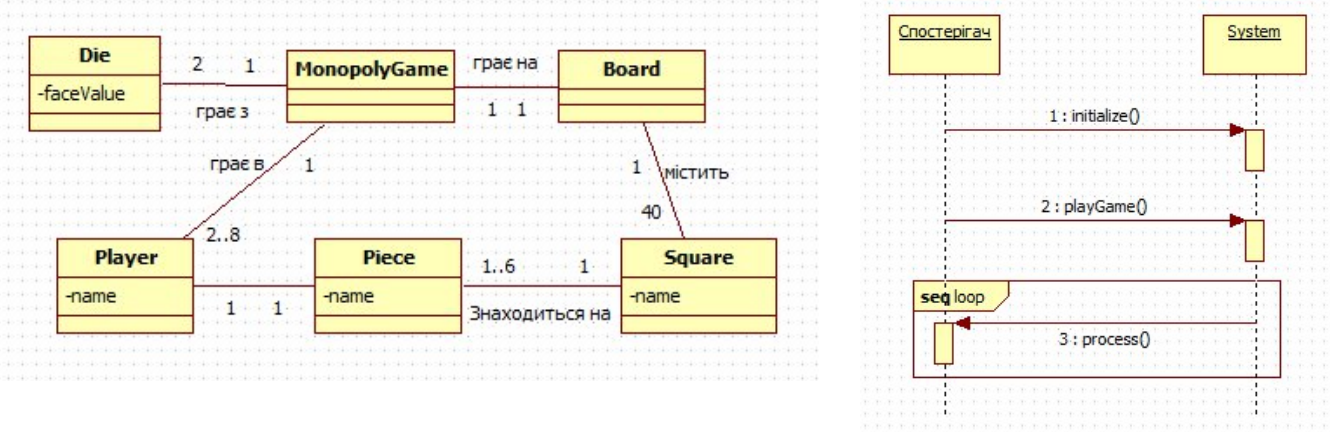


Рисунок 6.1 – Діаграми моделі предметної області та СДП для програмної системи *MonopolyGame.*

У RDD існує ідея кооперації (collaboration). Обов'язки реалізуються за допомогою методів, що діють або окремо, або у взаємодії з іншими методами і об'єктами. Наприклад, для класу *Firm* можна визначити один або кілька методів призначення Майстра з діагностики (скажімо, метод *getMaster*). Для виконання цього обов'язку об'єкт *Firm* повинен взаємодіяти з об'єктами класу *Order*, в тому числі передавати повідомлення *getMasterActive* кожному об'єкту *Order* про необхідність надання відповідної інформації цими об'єктами.

6.2 Шаблони проектування GRASP

Принципи об'єктного проектування відображені в шаблонах проектування GRASP (General Responsibllity Assignment Software Pattems), вивчення і застосування яких дозволить освоїти методичний підхід у розподілу обов’язків. Ці шаблони називають також шаблонами розподілу обов'язків (pattern of assigning responsibllities).

Всього до складу GRASP входить 9 шаблонів, але визначимо перших 6 найбільш важливих: Creator, Information Expert, Low Coupling, Controller, High Cohesion, Polymorphism, Pure Fabrication, Indirection, Protected Variations.

Шаблон ***Creator*** зводиться до наступного.

Назва Creator

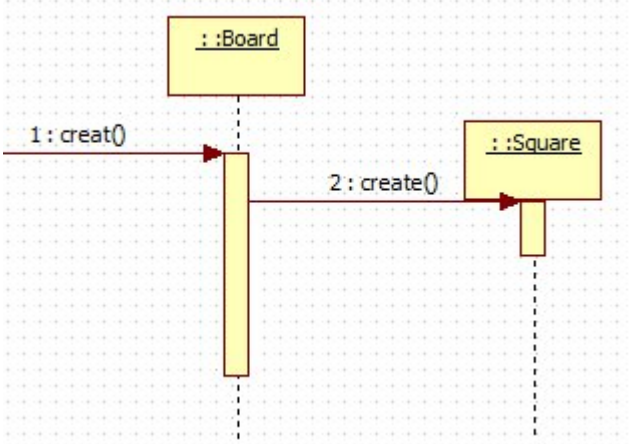
Проблема Хто відповідає за створення нового екземпляра деякого класу А?

Рішення (розглядається як порада) Призначити класу В обов'язок створювати екземпляри класу А, якщо виконується одна (або кілька) з наступних умов.

- Клас В містить (contains) або агрегує (aggregate) об'єкти А.
- Клас В записує (records) екземпляри об'єктів А.
- Клас В ахтівно використовує (closely uses) об'єкти А.
- Клас В володіє даними ініціалізації (has the initializing data) для
- об'єктів А.

Шаблони використовуються в процесі розподілу обов'язків. Розглянемо, як застосувати цей шаблон в реальній ситуації. По-перше, в даному випадку А і В є програмними об'єктами, а не об'єктами з моделі предметної області. Спочатку потрібно спробувати знайти програмний об'єкт В, що задовольняє перерахованим вище умовам. А що робити, якщо ніякі програмні класи ще не визначені? У цьому випадку необхідно звернутися до моделі предметної області.

Розглянемо використання шаблонів розподілу обов’язків на прикладі проектуванні програмної системи *MonopolyGame* рис. 6.1. Звернемося до моделі предметної області і зафіксуємо для себе, об'єкти *Square* містяться в об'єкті *Board*. Ця модель відображає концептуальний, а не програмний ракурс системи, проте її можна спроектувати на модель реалізації і припустити, що програмний об'єкт *Board* повинен містити програмні об'єкти *Square*. Тоді, згідно з шаблоном ***Creator***, об'єкт *Board* повинен створювати об'єкти *Square*. Об'єкти *Square* завжди агрегируются об'єктом *Board*, тому останній керує їх створенням і руйнуванням.



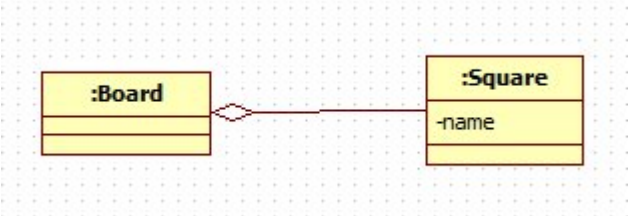


Рисунок 6.2 – Діаграми послідовності та класів , які відображають шаблон ***Creator***

Нагадаємо, що в контексті гнучкого проектування статичні і динамічні моделі створюються паралельно. Отже, це проектне рішення необхідно одночасно відобразити і на діаграмі класів і на діаграмі взаємодії (рис. 6.2). З рис. 6.2 видно, що при створенні об'єкта *Board* відразу ж створюються і об'єкти *Square*. Для простоти на цій діаграмі не показаний цикл створення 40 примірників клітин.

Шаблон ***lnformation Expert*** зводиться до наступного.

Назва Information Expert

Проблема Який базовий принцип розподілу обов'язків між об'єктами?

Рішення Призначити цей обов'язок тому класу, який володіє достатньою інформацією для її виконання. Для виконання цього обов'язку об'єкт повинен володіти інформацією про власний стан, його оточення, успадкованої інформації і т.п. В данному випадку кожну конкретну клітину на полі може представляти об'єкт, що володіє інформацією про всі клітинах. З рис. 6.2 видно, що таким об'єктом є *Board*, котрий агрегує всі програмні об'єкти *Square*. Отже, об'єкт *Board* володіє достатньою інформацією для виконання цього обов'язку. На рис. 6.3 показаний результат застосування шаблону Expert в даному контексті.

Шаблон ***Low Coupling*** зводиться до наступного.

Назва Low Coupling

Проблема Як зменшити вплив внесених змін на інші об'єкти?

Рішення Мінімізувати ступінь зв'язування в процесі розподілу обов'язків.

Ступінь зв'язування (coupling) - це міра пов'язаності одного елемента з іншими. Під зв'язуванням розуміється володіння інформацією або будь-яка залежність. Принцип Low Coupling використовується для оцінки існуючого проектного рішення або вибору рішення з кількох варіантів - за інших рівних умов слід віддавати перевагу проектному рішенню з більш низьким ступенем зв'язування. З рис. 7.1 видно, що об'єкт *Board* містить багато об'єктів *Square*. Чому не привласнити обов'язок знання об'єктів *Square* деякому класу *Dog*? Розглянемо ці альтернативи в термінах шаблону Low Coupling. Якщо клас *Dog* містить метод *getSquare* (), значить, він повинен взаємодіяти з об'єктом *Board* для отримання інформації про колекцію об'єктів *Square*. Ймовірно, ці дані

зберігаються в об'єкті-колекції *Map*, за допомогою якого можна отримувати інформацію по ключу. Тоді об'єкт *Dog* зможе отримати доступ до цих даних по ключу *name* і повернути інформацію про конкретний об'єкті *Square*. Таким чином, ступень звязування стає вище той, яка зображена на рис. 6.2.

Чим корисний принцип мінімізації зв'язування? Або, іншими словами, чому доцільно мінімізувати ступінь зв'язування об'єктів і вплив змін одних об'єктів на інші? Справа в тому, що цей принцип дозволяє заощадити час, зусилля і зменшити кількість похибок при модіфікації програмної системи. В зв’язку з цим, можно підкреслити, що шаблон Expert підтримує принцип мінімізації зв’язування.

Як було зазначено раніше, згідно принципу Model-View-Separation, об'єкти інтерфейсу не повинні реалізовувати бізнес-логіку додатка, наприклад, обчислювати ходи користувача. Вони повинні делегувати запит (надіслати його іншому об'єкту) на рівень об'єктів предметної області. Шаблон Controller (контролер) дозволяє відповісти на таке просте питання: який об'єкт за межами рівня інтерфейсу користувача (UI) повинен отримувати повідомлення від рівня UI?

Шаблон ***Controller*** можна описати таким чином.

Назва Controller

Проблема Хто повинен відповідати за отримання та координацію виконання системних операцій, що надходять від рівня інтерфейсу користувача?

Рішення Присвоїти цей обов'язок класу, який задовольняє одній з наступних умов.

- Клас представляє всю систему в цілому, кореневой об'єкт, пристрій або важливу підсистему (зовнішній контролер).
- Клас є сценарій деякого прецеденту, в рамках якого виконується обробка цієї системної операції (контралер прецеденту або контролер сеансу).

Розглянемо зазначені варіанти.

Варіант la. Клас представляє всю систему в цілому або кореневой об'єкт. В ролі такого об'єкта може виступати *MonopolyGame*.

Варіант lб. Клас представляє пристрій, на якому працює дана програмна система. Цей варіант відноситься до спеціалізованих апаратних засобів, наприклад, телефону або касового апарату (програмний клас *Phone* або *CachMachine*), і в даному випадку непридатний.

Варіант 2. Клас являє сценарій прецеденту або сеанс. Системна операція *playGame* виконується в рамках прецеденту *PlayMonopoly*. У ролі такого класу

може виступати програмний клас *PlayMonopolyGameHandler* (при використанні цієї версії об'єктного рішення до імен класів додається суфікс *Handle*r або *Session*).

Застосуємо для програмної системи *MonopolyGame* варіант la, який використовується у випадку, коли в системі є лише кілька системних операцій. В якості контролера обираємо об’єкт *MonopolyGame*, який буде реалізовувати системну операцію *PlayMonopoly* (), тобто повідомлення від об’єкта вікна форми інтерфейсу користувача перетворюється в системну операцію *PlayMonopoly*.

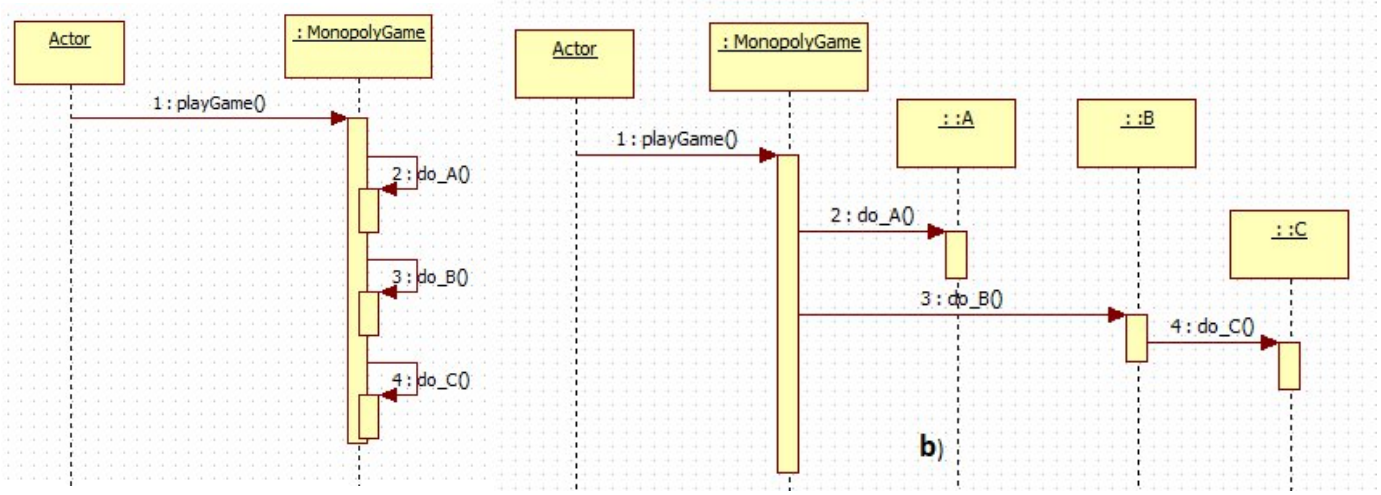


Рисунок 6.4 – Приклад використання шаблона ***HighCohesion*** реалізації системної операції *playGame()*.

У всіх випадках при застосуванні об’єктно-орієнтованного підхода для обробки зовнішніх подій використовуються контролери. Шаблон Controller забеспечує найбільш типичне проектне рішення для цього випадка. Контролер - це своєрідний вид інтерфейсу між рівнями предметної області та графічним представленням.

Щоб забезпечити можливість підтримки інформації про стан прецеденту, для обробки всіх системних подій в рамках одного прецеденту повинен використовуватися один і той же клас контролера. Така інформація може знадобитися, наприклад, для ідентифікації моменту порушення послідовності системних собьггій (наприклад, виконання операції makePayment перед виконанням операції endSale). Для різних прецедентів можна використовувати різні контролери.

Типовою помилкою при створенні контролерів є покладання на них занадто великого числа обов'язків. Зазвичай контролер повинен лише

делегіровшп' функції іншим об'єктам і координувати їх діяльність, а не виконувати ці дії самостійно.

Шаблон ***HighCohesion*** зводиться до наступного.

Назва HighCohesion

Проблема: як забезпечити сфокусованість обов'язків об'єктів, їх керованість і ясність та виконання принципу HighCohesion?

Рішення: розподіл обов'язків, що підтримує високий ступінь зачеплення.

У термінах об'єктно-орієнтованого проектування зачеплення (cohesion) (або, точніше, функціональне зачеплення) - це міра пов'язаності і сфокусированности обов'язків класу. Вважається, що елемент має високу ступенем зачеплення, якщо його обов'язки тісно пов'язані між собою і він не виконує непомірних обсягів роботи. У ролі таких елементів можуть виступати класи, підсистеми і т.д.

Цей принцип використовується для оцінки можливих альтернатив. Клас з низьким ступенем зачеплення виконує багато різнорідних функцій або незв'язаних між собою обов'язків. Такі класи створювати небажано, оскільки вони призводять до виникнення наступних проблем.

- Труднощі розуміння.
- Складнощі при повторному використанні.
- Складнощі підтримки.
- Ненадійність, постійна схильність до змін.

Неформально: зачеплення (cohesion) в контексті проектування програмних систем являє міру пов'язаності операцій одного програмного елемента і що виконується їм обсяг роботи.

У наведеному ліворуч проектному рішенні рис. 6.4 всю роботу системної операції *playGame()* виконує сам об'єкт *MonopolyGame*, а в ситуації, показаної праворуч, він делегує запит *playGame* іншим об’єктам та контролює його виконання, що є прикладом використання шаблону ***HighCohesion.***

Інші шаблони GARSP будуть розглянуті у дисципліні «Конструювання програмного забеспечення».

6.3 Приклад розподілу обов’язків серед об’єктів програмної систему «RepairTech» прецедент «Оформлення замовлення».

По перше, визначимо класи-контролери для реалізаціх системних операцій. Згідно шаблоном Controller, можливі наступні варіанти:

- клас, який представляє всю систему в цілому, Register, FirmSystem

пристрій або підсистему.

– клас, який представляє одержувача або штучний *ProcessOrderHandle* обробник всіх системних подій деякого сценарію *ProcessOrderSession* прецеденту.

Вибір найбільш гідного контролера визначається ще такими факторами, зокрема зачепленням і зв'язуванням. Про це більш детально розповідається нижче. На даній ітерації розробки прецедента створимо контролер, який буде представляти цій прецедент *ProcessOrderHandle* рис. 6.5 і відповідати за виконання системних операцій цього прецедента: *makeOrder*() - подія «створити замовлення», *getMaster*() – подія «призначити майстра», *changeStatus*() – подія «змінити статус», *saveOrder*() – подія «зберегти замовлення». Далі, якщо цей клас сильно збільшиться та перестане відповідати принципу ***HighCohesion,*** його можливо замінимо на декілько контролерів.

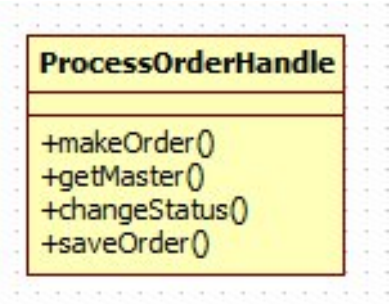


Рисунок 6.5 – Клас-Controller *ProcessOrderHandle* для програмної системи «RepairTech» прецедент «Оформлення замовлення».

Наступний крок – реалізувати системні операції класів рівня системи. Розглянемо, які класи будуть відповідати за реалізацію операції *makeOrder*() – створення об’єктів класу *Order*. Для цього будемо використовувати шаблон Creator. Сутність будемо обирати застосовуя правила шаблона Creator, згідно з яким обов'язок по створенню нових екземплярів делегується класу, який містить, агрегує або записує інформацію про створюваних клас. Проаналізувавши модель предметної області, приходимо до висновку, що об’єкту, який відповідає за створення об’єктів Order немає. В цієї ролі можливо на даному етапі проектування використавувати об’єкт-контролер *ProcessOrderHandle*, який отримає від інтерфейсу користувача інформацію для ініціалізації об’єкта *Order* (рис. 6.5), тобто володіє даними ініціалізації. Якщо екземпляри *Order* будуть створюватися об'єктом *ProcessOrderHandle*, то об'єкт *ProcessOrderHandle* буде містити посилання на поточний екземпляр *Order*.

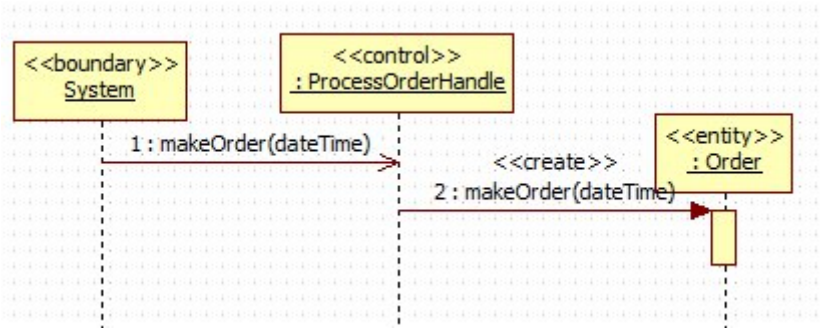


Рисунок 6.6 – Діаграма послідовності для створення об’єкта *Order*.

Реалізуємо системну операцію *getMaster*. Згідно з описом цієї опреції (стр. 74) нам необхідно перейти до стану, де реалізовано:

- створено список екземплярів *ListOrders*[] об’єктів *Order (створення екземпляру);*
- екземпляр ListOrders[i] об’єкта Order пов’язан з класом Order *(формування асоціації);*
- атрибуту ord.id_master присвоєно значення ListOrders[min].id_master *(модіфікація атрибуту)*
- атрибуту ord.status присвоєно значення String “diagnostic” *(модіфікація атрибуту).*

Для реалізації цієї операції будемо використаовувати шаблон ***Information Expert***, тобто знайдемо об’єкти, які мають інформацію для виконання цієї опреції.

Тепер виникає ключове питання: на основі якої моделі потрібно проналізувати інформацію - моделі предметної області або проектування? Модель предметної області ілюструє концептуальні класи з предметної області системи, а в моделі проектування показані програмні класи. Відповідь на це питання зводиться до наступного.

1) Якщо в моделі проектування є відповідні класи, в першу чергу, слід використовувати її.
2) В іншому випадку потрібно звернутися до моделі предметної області і постаратися уточнити її для полегшення створення відповідних програмних класів.

Інформацію про майстрів та кількість іх навантаження ми можемо отримати з об’єктів *Order*, дані про які має об’єкт *ProcessOrderHandle*, тобто опрецацію *getMaster()* буде реалізовувати цей об’єкт - визначати майстра з мінільним навантаженням та передавати його *id_master* у поточний об’єкт *Order* Для цього нам потрібно визначити для кожного майстра його навантаження,

тобто скільки замовлень розподілених на майстра мають статус «діагностика» і визначити майстра з мінімальним навантаженням (рис. 6.7*)*

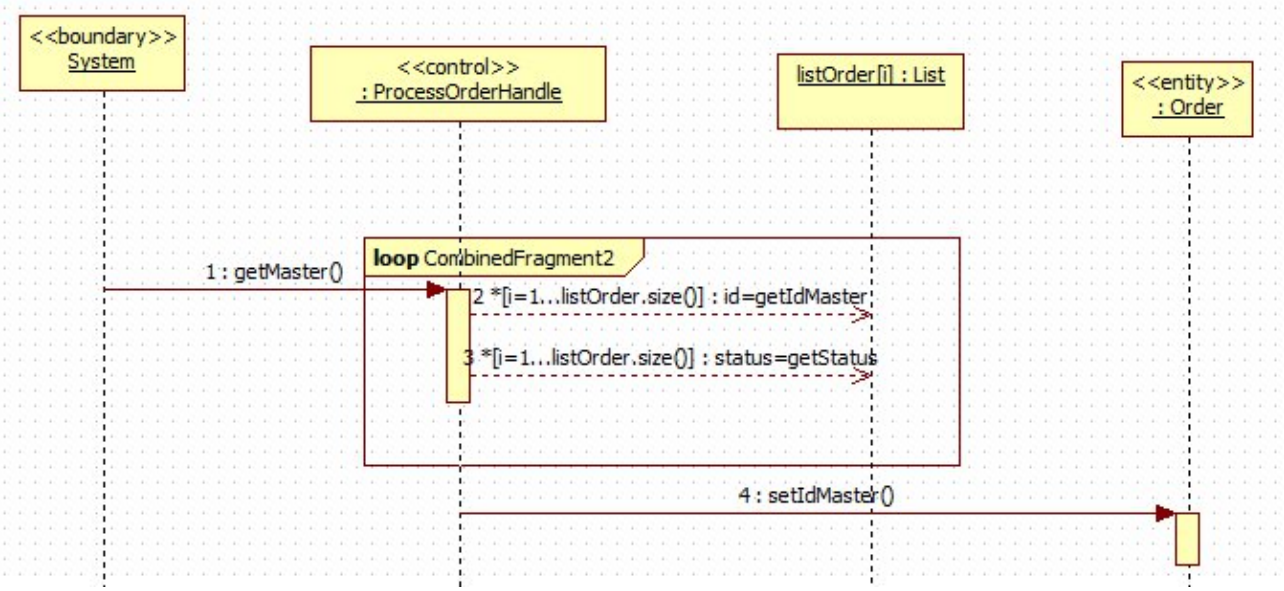


Рисунок 6.7 – Діаграма послідовності для операції *getMaster*().

Наступна системна операція *changeStatus*(), яка змінює статус замовлення. При створені нового замовлення можливо атрибут «статус» ініціювати за умовчаням «діагностика», але у бізнес процесі говориться про зміну статусу замовлення на різних етапах його виконання, тому необхідно створити операцію *changeStatus*(). Застосуємо знову шаблон ***Information Expert,*** експертом для зміни статусу може ствти *ProcessOrderHandle*, який отримає значення статусу з інтерфейсу користувача після визначення Майстра з діагностики рис.6.8.

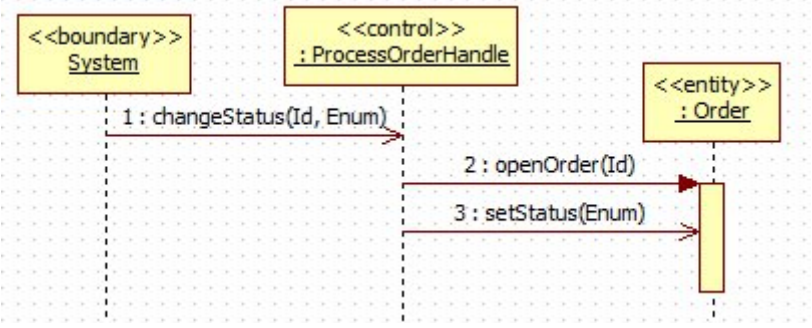


Рисунок 6.8 – Діаграма послідовності для реалізації операції *changeStatus*().

Таким чином, розглянуто обов’язкі різних об’єктів в реалізації сценарія прецедента «Оформлення замовлення» на першій ітерації фази розвитку. Підсумуємо результати аналізу у вигляді діаграми кооперації рис. 6.9 а, яка більш компактна чим діаграма послідовності, та діаграми класів рис. 6.9 б для цього прецедента.

На діаграмах не проаналізовані всі асоціації моделі предметної області на даной ітерації фази розвитку. Розглянемо використання шаблонів ***HighCohesion*** та ***Low Coupling*** в розподілу обов’язків.

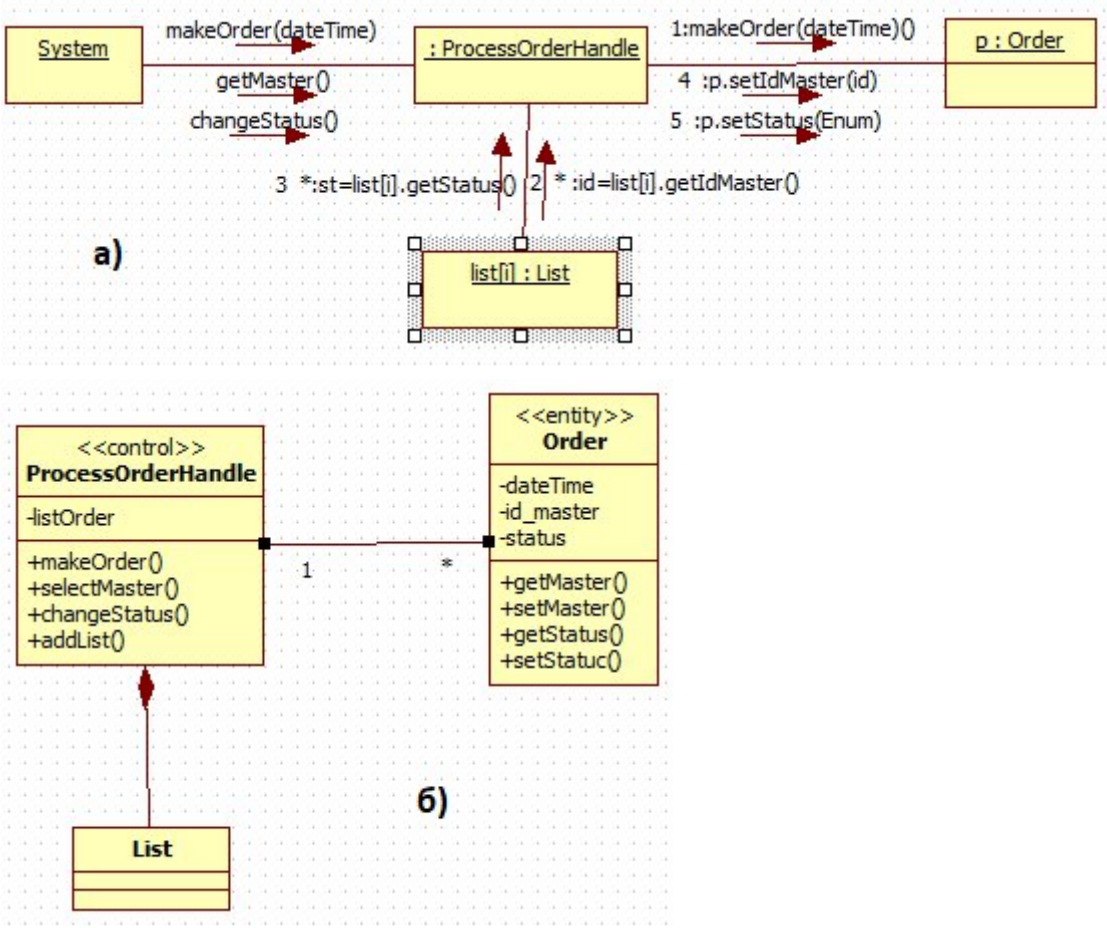


Рисунок 6.9 – а) діаграма кооперації, б) діаграма класів для прецедента «Оформлення замовлення»

Як було визначено вище, шаблон ***Low Coupling*** використовується для оцінки існуючого проектного рішення або вибору рішення з кількох варіантів - за інших рівних умов слід віддавати перевагу проектному рішенню з більш низьким ступенем зв'язування. Під зв'язуванням розуміється володіння інформацією або будь-яка залежність. Діаграма кооперації рис.6.9 показує, що на даній ітерації росподіл обов’язків між об’єктами прецедента «Оформлення замовлення» немає альтернативного рішення і тому відповідає низькому зв'язуванню.

Шаблон ***HighCohesion*** , як було сказано вище, аналізує міру пов'язаності і сфокусированности обов'язків класу. Вважається, що елемент має високий ступень зачеплення, якщо його обов'язки тісно пов'язані між собою і він не виконує непомірних обсягів роботи. У ролі таких елементів можуть виступати

класи, підсистеми і т.д. Клас з низьким ступенем зачеплення виконує багато різнорідних функцій або незв'язаних між собою обов'язків. Такі класи створювати небажано, оскільки вони призводять до виникнення різних проблем. Класи *ProcessOrderHandle* та *Order*, як показує діаграма класів рис. 6.9б, мають високу ступень зачеплення, тому що всі операції пов’язані з поведінкою саме цих класів і кількість операцій невелика.

Питання для самоконтролю

1. В чому суть підходу RDD для визначення операцій, за допомогою яких об’єкти взаїмодіють ?
2. Що таке шаблони GARSP? Перерахуйте та опишить іх призначення.
3. Для виявлення яких обов’язків застосовується шаблон ***Creator***? Правила його використання. Наведить приклад використання цього шаблону.
4. Для виявлення яких обов’язків застосовується шаблон ***Information Expert***? Які правила його використання? Наведить приклад використання цього шаблону.
5. Для виявлення яких обов’язків застосовується шаблон ***Controller***? Які правила його використання? Наведить приклад використання цього шаблону.
6. Що таке принцип звязування в проектуванні програмних систем. Який шаблон реалізує цей принцип? Правила його використання.
7. Що таке принцип зачеплення в проектуванні програмних систем. Який шаблон реалізує цей принцип? Правила його використання.

Завдання до лабораторної роботи №6

1. Відповісти на питання для самоконтролю.
2. Проаналізувати розподіл обов’язків за допомогою шаблонів GARSP між об’єктів одного прецедента.
3. Результат аналізу представити у вигляді діаграм послідовності, кооперації та класів для кожного шаблону та вцілому для прецедента).
4. Написати коди реалізації класів на C#.
5. Оформити протокол лабораторної роботи.

ТЕМА 7. ПОБУДОВА ДІАГРАМ СТАНУ ТА ДІЯЛЬНОСТІ ЯК ІНСТРУМЕНТ АНАЛІЗУ СКЛАДНИХ ТРАНЗАКЦІЙ ТА ОПЕРАЦІЙ КЛАСІВ ПРИ ПРОЕКТУВАННІ СИСТЕМИ.

Мета: набути практичних навичок аналізу складних транзакцій та операцій класів за допомогою діаграм стану та діяльноті.

7.1 Використання діаграми стану (activity diagram) UML для аналізу послідовності дій.

Діаграми станів визначають можливі послідовності станів і переходів, в яких може перебувати конкретний об'єкт. ***Діаграми станів застосовуються для моделювання поведінки класу, тільки якщо об'єкт класу може існувати в декількох станах і в кожному з них поводиться по-різному.***

На відміну від інших діаграм, діаграма станів описує процес зміни станів тільки одного класу, а точніше - одного примірника певного класу, тобто моделює всі можливі зміни в стані конкретного об'єкта. При цьому зміна стану об'єкта може бути викликано зовнішніми впливами з боку інших об'єктів або з зовні. Діаграма станів по суті є графом спеціального виду, який представляє певний автомат.

Елементи діаграми:

– ***Стан.*** У мові UML під станом розуміється абстрактний метаклассом, який використовується для моделювання окремої ситуації, протягом якої має місце виконання деякого умови. Стан може бути задано у вигляді набору конкретних значень атрибутів класу або об'єкта, при цьому зміна їх окремих значень буде відображати зміна стану модельованого класу або об'єкта. Не кожен атрибут класу може характеризувати його стан. Як правило, мають значення тільки такі властивості елементів системи, які відображають динамічний або функціональний аспект її поведінки. Стан на діаграмі зображується прямокутником із закругленими вершинами. ***Стан може мати ім'я і список дій***, причому список дій - не обов'язковий атрибут. Ім'я стану завжди записується з великої літери. Рекомендується в якості імені використовувати дієслова в теперішньому часі. Формат запису дій: <мітка-дії '/' вираз-дії>. Мітка дії вказує на обставини або умови, при яких буде виконуватися діяльність.

- ***Початковий та кінцевий стан***. Початковий стан є окремим випадком стану, який не містить ніяких внутрішніх дій (псевдосостоянія). У цьому стані знаходиться об'єкт за умовчанням в початковий момент часу. Воно служить для вказівки на діаграмі станів графічній області, від якої починається процес зміни станів. Графічно початковий стан в мові UML позначається у вигляді закрашеного круга, з якого може тільки виходити стрілка, відповідна переходу. Кінцевий стан являє собою окремий випадок стану, яке також не містить ніяких внутрішніх дій (псевдосостоянія). У цьому стані буде перебувати об'єкт за замовченням після завершення роботи автомата в кінцевий момент часу і служить для вказівки на діаграмі станів графічній області, в якій завершується процес зміни станів або життєвий цикл даного об'єкта. Графічно кінцевий стан в мові UML позначається у вигляді закрашеного круга, який міститься в колі, в яку може тільки входити стрілка, відповідна переходу.
- ***Перехід (transition)*** - відношення між двома станами, яке вказує на те, що об'єкт в першому стані повинен виконати певні дії і перейти в другий стан. Перехід здійснюється при настанні деякої події або виконання певної умови, званої сторожевою умовою. У цьому випадку говорять, що перехід спрацьовує. До спрацьовування переходу об'єкт знаходиться в попередньому від нього стані, званим вихідним станом, а після його спрацьовування об'єкт знаходиться в подальшому від нього стані (цільовому стані). На переході вказується ім'я події. Ім'я події повинно починатися з малої літери. Загальний формат імені події: <Сигнатура події> '[' <сторожову умову> ']' <вираз дії>. Види переходів: ***тригерний перехід***, який специфікує подія-тригер, пов'язане із зовнішніми умовами стосовно оскільки він розглядався станом. Тригерний перехід обов'язково повинен мати ім'я події, яке він специфицирует. ***Нетригерний перехід***, який відбувається по завершенні виконання діяльності в даному стані. Для них поруч зі стрілкою переходу не вказується ніякого імені події, а в початковому стані повинна бути описана внутрішня діяльність, після закінчення якої відбудеться той чи інший нетріггерний перехід.
- ***Складений стан і підстан.*** Складений стан (composite state) - такий складний стан, який складається з інших вкладених в нього станів. Останні будуть виступати по відношенню до першого як ***підстан*** (substate). Хоча між ними має місце відношення композиції, графічно всі вершини діаграми, які відповідають вкладеним станам, зображаються усередині символу складеного стану. Складений стан може містити два або більше паралельних підавтомат

або кілька послідовних підстанів. Послідовні підстани (sequential substates) використовуються для моделювання такої поведінки об'єкта, під час якого в кожен момент часу об'єкт може знаходитися в одному і тільки одному підстанів. Поведінка об'єкта в цьому випадку представляє собою послідовну зміну підстанів.

На рис. 7.1 зображен приклад діаграми станів, яка створена за допомогою CASE засобу StarUML.

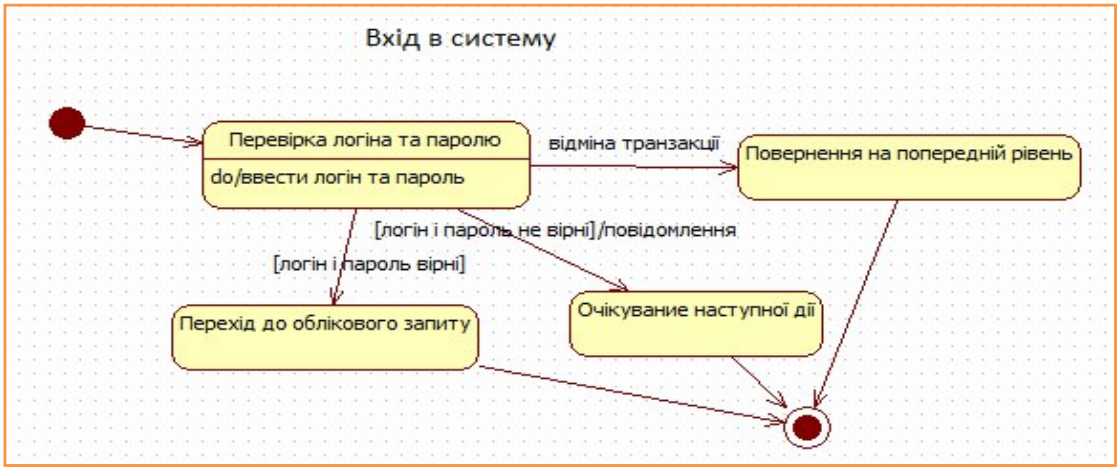


Рисунок 7.1 – Діаграма стану для системи регістрації користувача. У квадратних лапках для нетригерних події зазначені сторожеві умови.

Для прикладу збудуємо діаграму стану для прецедента «Оформлення замовлення», на якій відобразимо такі стани: ініціалізація замовлення, заповнення полів, при збережені замовлення – друк квітанції, відміна замовлення рис.7.2

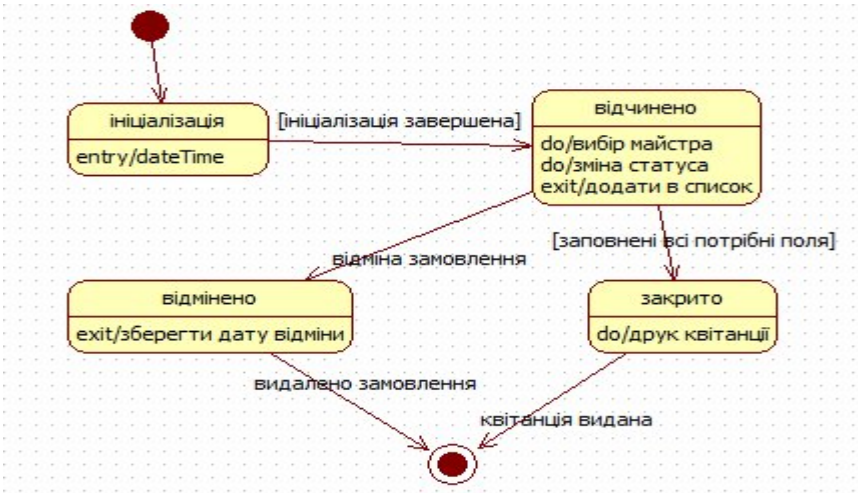


Рисунок 7.2 – Діаграма стану для прецеденту «Оформлення замовлення».

7.2 Використання діаграми діяльності (activity diagram) UML для аналізу послідовності дій.

Діаграма діяльності показує потік переходів від однієї діяльності до іншої, при цьому увагу фіксується на результаті діяльності. Сам же результат може привести до зміни стану системи або повернення деякого значення. Діяльність (Activity) - це триває в часі неатомарний крок обчислень. Діаграми діяльності застосовуються в UML для моделювання динамічних аспектів поведінки системи. По суті, блок-схема, яка показує, як потік управління переходить від однієї діяльності до іншої.

Діаграми діяльності найбільш часто застосовуються:

- Для опису поведінки, що включає велику кількість паралельних процесів.
- При паралельному програмуванні, оскільки можна графічно зобразити всі гілки і визначити, коли їх необхідно синхронізувати.
- Для опису потоків подій в варіантах використання. На відміну від текстового опису, вони надають ту ж інформацію, але в наочній графічній формі.
- При моделюванні бізнес-процесів.
- Для візуалізації особливостей реалізації операцій класів, коли необхідно представити алгоритми їх виконання. При цьому кожен стан може бути виконанням операції деякого класу або її частини, дозволяючи використовувати діаграми діяльності для опису реакцій на внутрішні події системи.

Діаграми діяльності можно вважати окремим випадком діаграм станів, тому що на діаграмах діяльності також присутні позначення станів і переходів.

Елементи діаграми діяльності в загальному випадку складається з: станів діяльності і станів дії, переходів.

Стан дії (action state) є спеціальним випадком стану з вхідною дією і принаймні одним переходом, який виходить з нього. Цей перехід передбачає, що вхідна дія вже завершилась. Стан дії не може мати внутрішніх переходів. Звичайне використання стану дії полягає в моделюванні одного кроку виконання алгоритму (процедури) або потоку управління. Кожна діаграма діяльності повинна мати єдиний початковий і єдиний кінцевий стани, як і діаграма стану. При цьому кожна діяльність починається в початковому стані і закінчується в кінцевому стані. Саму діаграму діяльності прийнято розташовувати таким чином, щоб дії слідували зверху вниз. Графічно стан дії зображується так само, як стан діаграми станів, де записується вираз дії (action-expression), яке повинно бути унікальним в межах однієї діаграми діяльності. Ніяких додаткових або неявних обмежень при запису дій не накладається.

Переходи - при побудові діаграми діяльності використовуються тільки ***нетріггерние переходи***. Нетріггерние переходи - це переходи, які спрацьовують одразу після завершення діяльності або виконання відповідної дії. Цей перехід переводить діяльність в подальший стан відразу, як тільки закінчиться дія в попередньому стані. На діаграмі такий перехід зображується суцільною лінією зі стрілкою. ***Єдиний перехід***, якщо зі стану дії виходить єдиний перехід, то він може існувати не позначений. ***Розгалуження*** - поділ діяльності на альтернативні гілки в залежності від значення деякого проміжного результату. Тобто розгалуження - це кілька переходів, які виходять зі стану дії, але спрацювати може тільки один з них. В цьому випадку для кожного з таких переходів має бути явно записано сторожову умову в прямих дужках. Графічно розгалуження на діаграмі діяльності позначається невеликим ромбом, усередині якого немає ніякого тексту. В цей ромб може входити тільки одна стрілка від стану дії. Вихідних стрілок може бути дві або більше, але для кожної з них явно вказують відповідну сторожову умову в формі булевского вираження. Допускається використовувати замість сторожової умови слово "інакше", яке вказує на той перехід, який повинен спрацювати в разі невиконання сторожової умови розгалуження.

Поділу і злиття паралельних обчислень або потоків управління. Символом поділу і злиття є прямий відрізок горизонтальної лінії, товщина якої трохи ширше основних суцільних ліній діаграми діяльності, аналогічно позначенню переходу в формалізмі мереж Петрі. При цьому поділ (concurrent fork) має один вхідний перехід і кілька виходять. Злиття (concurrent join) має кілька вхідних переходів і один вихідний.

Наведемо приклад застосування діаграми дії для бізнес процесу фірми «РемПобТех» рис. 7.3.

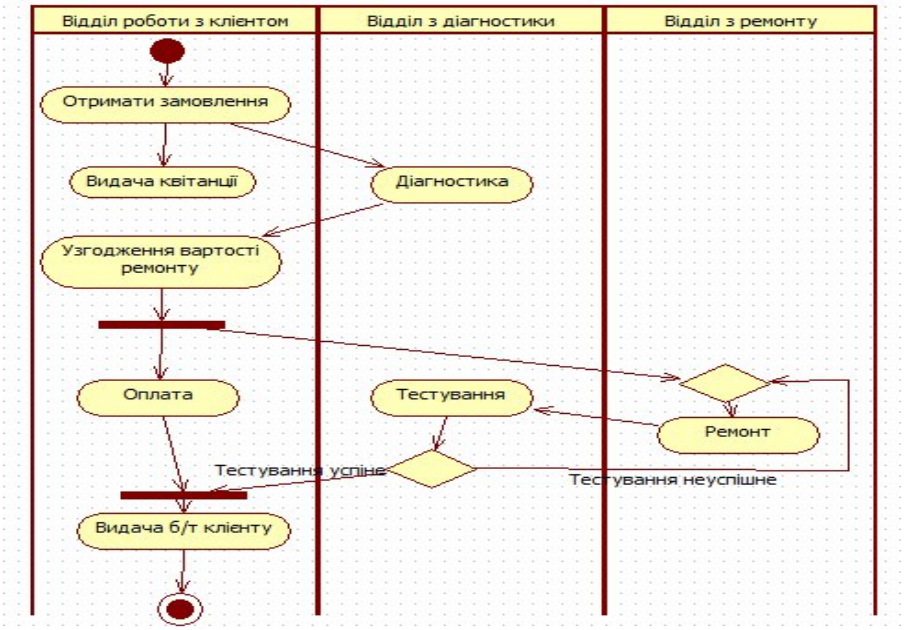


Рисунок 7.3 – Діаграма діяльнояті для бізнес процесу фірми «РемПобТех»

ЛІТЕРАТУРА

1. К. Ларман, Применение UML 2.0 и шаблонов проектирования. Практическое руководство. 3-е издание, ИД "Вильяме" г. Москва, 2012
2. Петрик М.Р. Моделювання програмного забезпечення : науковометодичний посібник / М.Р. Петрик, О.Ю. Петрик – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. – 200 с.
3. Алонцева Е. Н., Анохин А. Н., Саакян С. П. Структурное моделирование процессов и систем. Учебное пособие по курсу «CASE и CALS технология». – Обнинск: ИАТЭ НИЯУ МИФИ, 2015. – 72 с.
4. Дин Леффингуэлл, Дон Уидриг, Принципы работы с требованиями к программному обеспечению. Унифицированный подход. Вильямс 2002. – 448 с.
5. Методы ARIS. Весть – МетаТехнология, 2000.- 227 с.
6. Д. А. Марка, К. МакГоуэн, МЕТОДОЛОГИЯ СТРУКТУРНОГО АНАЛИЗА И ПРОЕКТИРОВАНИЯ SADT. - М.: ДМК Пресс ,1993. – 231 с.
7. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. – М.: ДМК Пресс, 2006. – 496 с.
8. StarUML. Інструкція користувача, http://staruml.sourceforge.net/docs/user-guide(ru)/user-guide.pdf

ЗМІСТ