**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ І ЗВ’ЯЗКУ**

**Кафедра інженерії програмного забезпечення**

**МОДЕЛЮВАННЯ ТА АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

**Методичні вказівки до практичних та лабораторних занять**

**з дисципліни «Моделювання та аналіз програмного**

**забезпечення» для студентів спеціальності 121–Інженерія**

**програмного забезпечення**

**\**

**Одеса 2021**

УДК 004.43 План НМВ 2021 р.

**Рецензент – д.т.н., доцент кафедри інженерної механіки Військової академії (м. Одеса) – Ісмаілова Н.П.**

**Моделювання та аналіз програмного забезпечення: методичні вказівки до практичних та лабораторних занять. / Укладач: Л.В. Глазунова – Одеса:ДУІТЗ, 2021., с. 92**

Методичні вказівки містять короткі теоретичні відомості з понять і принципів аналізу та моделювання програмного забезпечення. Згідно з навчальними планами дисципліни «Моделювання та аналіз програмного забезпечення» в методичних вказівках для закріплення теоретичного матеріалу також надані приклади застосування методів аналізу та моделювання різноманітних аспектів представлення розроблюваної 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 приділяється створенню діаграми класів для відображення об’єктів системи, ії структурування в рамках взаємодії та поведінки об’єктів. Для закріплення практичних навичок студенти виконують аналіз та моделювання власної автоматизованої інформаційної системи. Методичні вказівки будуть корисні студентам для закріплення лекційного матеріалу, а також при підготовці до практичних занять та виконанні лабораторних робіт.

Методичні вказівки призначено студентам зі спеціальності 121 – Інженерія програмного забезпечення.

СХВАЛЕНО ЗАТВЕРДЖЕНО

на засіданні кафедри методичною радою

інженерія програмного забезпечення ДУІТЗ.

та рекомендовано до друку. Протокол № 1

Протокол № 6 від 11 березня 2021 р.

від 22 січня 2021 р.

© Глазунова Л.В., 2021

© ДУІТЗ, 2021

**Тема 1. АНАЛІЗ ВИМОГ ТА ПОБУДОВА ЦІЛЬОВОЇ 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**

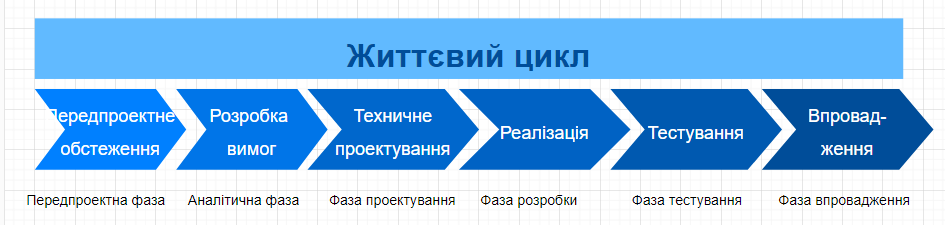
Перш ніж розглядати докладно документ вимог коротко розглянемо місце і роль цього документа в усьому процесі розробки програмного забезпечення (ПЗ). Життєвий цикл розробки програмного забезбечення (ПЗ) у спрощеному вигляді виглядає так, як це показано на рис. 1.1

Рисунок 1.1 - Спрощене уявлення життєвого циклу розробки ПЗ

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

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

З рис. 1.1 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 є своєрідним "коренем" і породжує ціле «дерево» інших проектних документів: документ архітектури ПЗ, опис UI, модель бази даних, плани реалізації, плани тестування, керівництво 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 збільшення меж проекту і, нарешті, домогтися більшої задоволеності замовників.

***Верхньорівнева структура документа вимог***  визначається структурою вимог, виявлення яких є необхідною умовою успішного проектування автоматизованої інформаційної системи (АІС) рис. 1.2

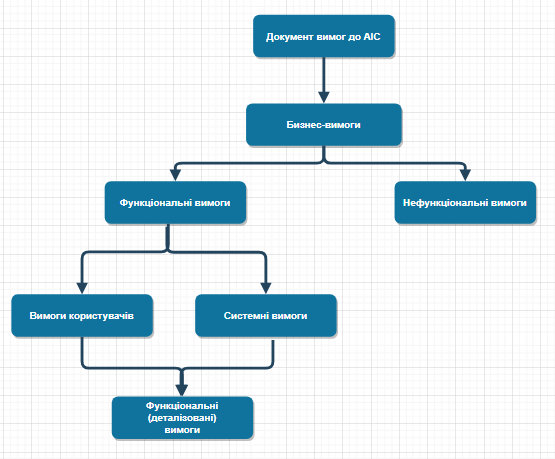


Рисунок 1.2 – Верхньорівнева структура документа вимог до АІС

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 є певної мірою 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 певною мірою завданнями, які повинна вирішувати АІС для досягнення цілей свого створення.

Як видно зі схеми, яка показана на рис. 1.2, бізнес-вимоги визначають функціональні і нефункціональні вимоги. На практиці може статися так, що функціональні і нефункціональні вимоги можуть "заходити" за межі 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 бере 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 би проблематично. Як приклад, сюди можна віднести вимоги зі зручності у використанні, продуктивності, масштабованості, вимоги з документування, з організаційного та методичного забезпечення, з інтеграції із зовнішніми системами.

Функціональні вимоги включають у себе вимоги користувачів і системні вимоги. ***Вимоги користувачів*** (Customer requirements) ̶ це ті вимоги верхнього рівня, які вимагають від системи майбутні бізнес-користувачі. Іншими словами, ті завдання, які система повинна вирішувати з точки зору бізнесу Замовника. Як приклад можна навести 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.н. Ще раз необхідно підкреслити, що ***вимоги користувачів описують верхньорівневую функціональність системи, не вдаючись у подробиці її технічної реалізації***. ***Системні вимоги*** (System requirements) мають рівно такий самий сенс, що і призначені для користувача вимоги, з тією різницею, що більшою мірою 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 самий 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 у світ клієнта.  Вони оформлюються у вигляді ***сценарієв використання***.

Хоча сценарії використання є одним з найпопулярніших методів для опису функціональності систем, вони також використовуються для вивчення їх нефункціональних характеристик. Найпростіший спосіб це зробити ̶ зафіксувати їх у рамках самих сценаріїв використання. Наприклад, поєднати вимоги з продуктивністю та часом між конкретними кроками сценарію використання або перерахувати рівень очікуваного обслуговування для сценарію використання як частину його самого.

В ***основної діючої особи*** є мета, система повинна допомоrати 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 діючої особи містить набір можливих cцeнарієв для досягнення мети, які повинні задовольняти наступним критеріям:

* всі взаємодії мають відношення до однієї і тієї ж мети, однієї і тєї ж основної діючої особи;
* варіант використання починається з ініціюючої події і продовжується, доки мета не досяrнута або її досягнення не перервано і система звільняється від своїх зобов'язань по відношенню до даної взаємодії.

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

* взаємодія між двома діючими особами (щоб сприяти досягненню мети);
* перевірка достовірності (щоб захистити учасника);
* зміна внутрішнього стану (від імені учасника).

***Модель «Учасники та інтереси»*** вносить лише невеликі зміни в загальну процедуру створення варіанта використання: перелік учасників та їх інтересів і застосування цього переліку в якості подвійної перевірки для rарантії, що ніщо не пропущено в тексті варіантів використання. Ця зміна суттєво впливає на якість варіанта використання.

***Функціональна область дії*** відноситься до послуг, які надає ваша система і які врешті-решт будуть зафіксовані у 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 сплетені. Список «Всередині / Поза» допомоrає вирішити ці завдання, тому що дозволяє провести межу між тим, що всередині області дії, і тим, що поза цією областю. Два інших інструменти ̶ це список «Дійова особа / Мета» і короткий опис вapіaнтів використання.

Кожен варіант використання повинен мати позначку йоro області дії в проектуванні. Припустимо, що компанія MyTelCo проектує систему NewApp, яка включає підсистему Searcher. Нижче надані імена областей дії проектування:

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

Користувацькі історії та концептуальні сценарії потрібні для розуміння основних мотивів користувачів і занурення у світ клієнта. Конкретні сценарії і варіанти використання вже можуть використовуватися для проектування інформаційної архітектури та інтерфейсів, а також при проведенні тестів і досліджень зручності використання.

Існують три рівні цілей:

* узагальнені;
* призначені для користувача;
* підфункції.

***Шаблон варіанта використання***:

* передумова;
* мета;
* з точки зору;
* область дії;
* бізнес процес;
* альтернативний процес.

***Передумова*** оголошує варіанти використання, виконання якої умови гарантує система перед тим, як дозволити запуск цього варіанта використання. Тому що умова виконується системою, і відомо, що вона істинна, то вона не буде перевірятися знову в період виконання варіанта використання. Загальновідомий приклад: користувач уже увійшов у систему і система ідентифікувала його.

Найбільшим інтересом є ***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 технолоrії 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 у систему» не вважаються цілями користувача. Онлайнові аукціони вимагають кілька днів і тому не проходять односеансовий тест. Вхід у систему 32 рази поспіль не відповідає (як правило) посадовим обов'язкам окремих людей або меті застосування системи. «Зареєструвати новоro клієнта» і «Купити книry» цілком можуть бути цілями користувачів. Завдання реєстраціі 32 нових клієнтів не позбавлена ​​сенсу для менеджера з продажу. Покупку книrи можна зробити за один сеанс.

Система характеризується підтримкою цілей користувача. Ось одна така мета: «Ви службовець, що сидить на робочому місці. Дзвонить телефон, ви берете трубку. Хтось на іншому кінці rоворить: «...». 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, відвертаєтеся від комп'ютера, прощаєтеся і кладете 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 зв'язаних цілей.
* формують перелік варіантів використання більш низькоrо рівня.

Узагальнені варіанти використання зазвичай виконуються протягом годин, днів, тижнів, місяців або років. Тут наданий основний сценарій «довrоrраючого» варіанта використання, завдання якого поєднати користувацькі варіанти використання, розкидані за роками.

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

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

***Набір варіантів використання ̶̶̶̶̶ це історія досягнення основними дійовими особами своїх цілей, яка безперервно розгортається***. Кожен варіант використання має розгалужену сюжетну лінію, що показує, як система забезпечує досягнення мети або відкидає її. Ця сюжетна лінія надана ​​основним сценарієм і набором фрагментів сценарію як розширення до нього. Кожен сценарій або фрагмент стартує при виконанні умови запуску, що вказує, коли відбувається запуск, і залишається активним, поки сценарій не завершиться або мета не буде відкинута. 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.***

Спочатку ми створюємо опис одного простого для розуміння і ти­пового сценарію, в якому досягається бізнес-мета і задовольняються інтереси всіх учасників, та виявляємо процеси, які треба віднести до розроблюваної інформаційної системи. Це основний сценарій.

***Образ проекту:*** для фірми «РемПобТех», яка виконує ремонт побутової техніки і в якої потрібно оптимізувати процес виконання замовлень. АІС «RepairТеch», який заснований на промисловій СКБД та високонадійний, на відміну від існуючого механізму на основі електронних таблиць, наш продукт оптимізує та прискорює виконання ремонту п/т.

***Вхідні дані***: замовлення клієнта, запасні частини, оплата.

***Вихідні дані:*** виконане замовлення.

***Передумова****:* клієнт приносить поломану п/т на фірму з ремонту «РемПобТех».

***Мета:*** якісно і своєчасно відремонтувати п/т клієнта для збільшення прибутку фірми на 30%.

***З точки зору:*** бізнесмена (господаря бізнесу).

***Область дії:*** фірма «РемПобТех».

***Бізнес процес (основний сценарій):***  всередині +/за ̶

* 1. Адміністратор оформляє заяву клієнта на ремонт п/т +
  2. Адміністратор приймає п/т і видає квитанцію клієнту +
  3. Адміністратор передає п/т Майстру з діагностики та тестування +

1. Майстер з діагностики встановлює причину поломки -
2. Майстер з діагностики оцінює вартість ремонту та термін ремонту, створює накладну на ремонт +
3. Майстер з діагностики передає п/т Майстру з ремонту після узгодження вартості ремонту з клієнтом. +
4. Майстер з ремонту замовляє на Складі необхідні запчастини +
5. Майстер з ремонту виконує ремонт п/т -
6. Майстер з ремонту відремонтовану п/т передає Майстру з

діагнастики для тестування +

1. Майстер з діагностики виконує тестування п/т -
2. Майстер з діагностики передає п/т Адміністратору +
3. Адміністратор приймає оплату за ремонт п/т +
4. Адміністратор передає ії клієнту по квітанції, пепевіряючи ії працездатність -
5. Адміністратор фіксує термін виконання +

***Альтернативна поведінка:***

6. Клієнт не згоден з ціною, Адміністратор повертає п/т клієнту +

7. На Складі немає деталей для ремонту

7.1 Адміністратор відкладає ремонт на термін до отримання +

деталей

7.2 Адміністратор повертає п/т клієнту +

11. Тестування не пройдено, Майстер з діагностики

повертає п/т Майстру з ремонту +

1. Термін виконання збільшено, Адміністратор нараховує штраф на

Майстра з ремонту +

**1.3 Визначення цілей та варіантів використання для користувачів АІС**

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

* Адміністратор – обслуговування клієнта;
* Майстер з діагностики – діагностування та тестування;
* Майстер з ремонту – ремонт п/т.

Цілі користувачів можливо зафіксувати у вигляді табл. 1.1 «Задачі користувачів»

Таблиця 1.1 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 п/т і видача квитанцію клієнту | Номер замовлення | Квитанція |
| 1.3 Підтвердження вартості ремонту | Номер замовлення | Статус=ремонт |
| 1.4 Перегляд списку заяв на ремонт п/т по фільтру | проміжок часу/кліент/ тип приладу/статус | список заяв |
| 1.5 Перегляд заяв, виконання яких було затримано |  | список заяв |
| 1.6 Видача відремонтованої техніки клієнту | Номер замовлення  Статус  Дата виконання |  |
| 1.7 Визначення прибутку фірми | Проміжок часу | прибуток фірми |
| 2 Майстер з діагностики | | |
| 2.1 Створення накладної на ремонт п/т | Заява  Тип поломки  Дата виконання  вартість ремонту |  |
| 2.2 Складення звіту про тестування | Накладна  Результат | звіт |
| 2.3 Призначення Майстра з ремонту | Номер накладной  Статус |  |
| Задача | Вхідна інформація | Вихідна інформація |
| 2.4 Переглядає свого активного списоку накладних на ремонт | статус≠ «готова» |  |
| 3 Майстер з ремонту | | |
| 3.1 Створення списку деталей для ремонту | Накладна  Назва деталі  Кількість  Ціна |  |
| 3.2 Зміна статусу в накладній з ремонту | Накладна  Статус |  |
| 3.3 Переглядає свої активні списоки накладних на ремонт | Майстер  Накладна | список накладних |

Приклади опису варіантів використання рівня системи для АІС «RepairТеch» в рамках основного сценарію за шаблоном.

|  |  |
| --- | --- |
| Специфікація варіанта використання | Оформлення замовлення |
| Контекст використання | Прийомп/т від клієнта для ремонту |
| Дійові особи | Адміністратор |
| Передумова | Вхід у систему |
| Тригер | Клієнт приніс поломану п/т |
| Сценарій | 1. Створити Замовлення (клієнт, тип п/т, назва, номер, що не працює, дата), 2. Додати інформацію, 3. Призначити Майстра з діагностики за правилами фірми, 4. Зберегти Замовлення, 5. Роздрукувати квитакцію. |
| Постумова | Створено Замовлення від клієнта з необхідною інформацією |

|  |  |
| --- | --- |
| Специфікація варіанту використання | Підтверження вартості ремонту |
| Контекст використання | Отримати підтвердження клієнта на суму ремонту |
| Дійові особи | Адміністратор |
| Передумова | Вхід у 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 змінено Статус на «ремонт» |

|  |  |
| --- | --- |
| Специфікація варіанта використання | Видача п/т клієнту |
| Контекст використання | Видача п/т клієнту після ремонту |
| Дійові особи | Адміністратор |
| Передумова | Вхід у систему  Квітанція на п/т |
| Тригер | Значення статусу в Замовленні є «готова»  Клієнт прийшов за п/т |
| Сценарій | 1. За квитанцією ввести номер Замовлення, 2. Прийняти оплату та змінити Статус на «видана», 3. Зафіксувати дату видачі, 4. Зберегти. |
| Постумова | У Замовленні змінений Статус на «видана» |

|  |  |
| --- | --- |
| Специфікація варіанту використання | Оформлення Накладної |
| Контекст використання | Виявлення поломок п/т |
| Дійові особи | Майстер з діагностики |
| Передумова | Вхід у систему |
| Тригер | Замовлення зі Статусом «діагностика» |
| Сценарій | 1. Перегляд нових Замовлень, 2. Створити Накладну (id замовлення, причина поломки, вартість ремонту, дата виконання), 3. Призначити Майстра з ремонту, тобто змінити статус Замовлення на «ремонт», 4. Зберегти Накладну. |
| Постумова | Створено Накладну на ремонт |

|  |  |
| --- | --- |
| Специфікація варіанту використання | Передача п/т на ремонт |
| Контекст використання | Передати п/т на ремонт |
| Дійові особи | Майстер з діагностики |
| Передумова | Вхід у систему |
| Тригер | Погоджена вартість ремонту з клієнтом |
| Сценарій | 1. Ввести Номер замовлення, 2. Змінити Статус на «ремонт», 3. Зберегти Накладну. |
| Постумова | Змінено значення Статусу на «ремонт» |

|  |  |
| --- | --- |
| Специфікація варіанта використання | Тестування п/т |
| Контекст використання | Протестувати роботоздатність п/т |
| Дійові особи | Майстер з діагностики |
| Передумова | Вхід у систему |
| Тригер | Значення Статусу замовлення «тестування» |
| Сценарій | 1. Ввести номер Замовлення. 2. Змінити Статус на «готова». 3. Зберегти Замовлення. |
| Постумова | Змінено значення Статусу на «готова» |

|  |  |
| --- | --- |
| Специфікація варіанта використання | Замовлення запчастин |
| Контекст використання | Замовити запчастини для ремонту п/т |
| Дійові особи | Майстер з ремонту |
| Передумова | Вхід у систему |
| Тригер | Накладна з результатами діагностики |
| Сценарій | 1. Створити новий список запчастин. 2. Ввести значення. 3. Зберегти список. |
| Постумова | Створено список запчастин |

|  |  |
| --- | --- |
| Специфікація варіанта використання | Передача на тестування |
| Контекст використання | Передати відремонтовану п/т на тестування |
| Дійові особи | Майстер з ремонту |
| Передумова | Вхід у систему |
| Тригер | Завершення ремонту |
| Сценарій | * 1. Ввести Номер замовлення.   2. Змінити Статус на «тестування».   3. Зберегти . |
| Постумова | Змінено Статус на «тестування» |

**Питання для самоконтролю**

1. Надати визначення каскадної, спіральної та гнучкої моделей ЖЦ.
2. Надати приклади застосування каскадної та гнучкої моделей ЖЦ.
3. Яка структура вимог для АІС?
4. Що таке користувацькі вимоги та варіант 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 сценарію користувача.

**Завдання до лабораторної роботи №1**

1. Обрати та обгрунтувати основати модель ЖЦ для своєї АІС таблиця 1.2
2. Створити концептуальну модель для своєї АІС.
3. Створити таблицю задач користувачів для своєї АІС.
4. Створити варіанти використання для користувачів АІС (3 найбільш складних).
5. Оформити протокол до лабораторної роботи.

Таблиця 1.2 Варіанти предметних областей для АІС

| № вар. | Завдання |
| --- | --- |
|  | Розробити АІС для центру занятості. |
|  | Розробити АІС для інспектора відділу кадрів університету. |
|  | Розробити АІС для працівників автостоянки. |
|  | Розробити АІС для інвестиційної компанії. |
|  | Розробити АІС для працівників автобусного парку. |
|  | Розробити АІС для мережі готелей. |
|  | Розробити АІС для фірми по автопрокату. |
|  | Розробити АІС для працівників складу мережі комп’ютерних магазинів. |
|  | Розробити АІС для співробітників фітнес клубу. |
|  | Розробити АІС туристичного агенства. |
|  | Розробити АІС для копіювального центру. |
|  | Розробити АІС для співробітників інтернет-клубу. |
|  | Розробити АІС електроної картотеки для пацієнтів полікліники. |
|  | Розробити АІС агенства нерухомості. |
|  | Розробити АІС мережі АЗС. |
|  | Розробити АІС шкільний «Щоденик». |
|  | Розробити АІС для центру дозвілля. |
|  | Розробити АІС обліку сімейного бюджету. |
|  | Розробити АІС для логістичної компанії (морських вантажоперевезень). |
|  | Розробити АІС для міні пекарні. |
|  | Розробити АІС розклад занять факультету ВФН. |
|  | Розробити АІС «Гуртожиток». |
|  | Розробити АІС «Домашній кухар». |
|  | Розробити АІС букмекерської контори. |
|  | Розробити АІС для мережі магазинів побутової техніки. |
|  | Розробити АІС для клубу знайомств. |
|  | Розробити АІС пошуку попутчиків (типу blablaсar). |
|  | Розробити АІС «Міський транспорт». |
|  | Розробити АІС «Мультімідійна бібліотека» (фото, відео, аудіо). |
|  | Розробити АІС «Ресторан». |
|  | Розробити АІС для співробітників телеканалу. |
|  | Розробити АІС для ОСББ. |

**ТЕМА 2. СТУКТУРНЕ МОДЕЛЮВАННЯ ПЗ. МЕТОДОЛОГІЇ ER. НОТАЦІЯ БАХМАНА.**

**Мета:** набути практичних навичок моделювання інформаційних аспектів подання 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 усіх інших екземплярів 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 імен, 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.

Сутність, яка є незалежною від інших сутністей, називається батьківською, а сутність, яка залежить від іншої сутності ̶ нащадок.

***Екземпляр сутності***  ̶ це конкретний представник даної суті. Наприклад, представником суті «Співробітник» може бути «Співробітник Іванов». Екземпляри сутностей повинні бути помітні, тобто суті повинні мати деякі властивості, унікальні для кожного екземпляра цієї сутності.

***Атрибут сутності***  ̶ це іменована характеристика, що є деякою властивістю сутності. Найменування атрибута повинно бути виражено іменником в однині (можливо, який характеризується прикметником). 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 зверху списку і виділяються знаком "\*" (див. рис. 2.1).

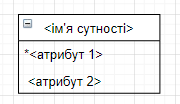


Рисунок 2.1 ̶ Графічне зображення сутності.

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

Приклад. Сутність ̶ «Співробітник». Атрибути: Табельний номер, Прізвище, Ім'я, По батькові, Посада, Зарплата. Ключ сутності ̶ табельний номер (рис. 2.2).

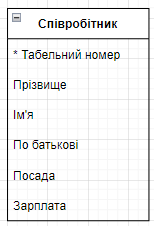


Рисунок 2.2 ̶ Приклад створення сутності «Співробітник»

Зв'язок (Relationship) ̶ це деяка асоціація між двома сутностями. Одна сутність може бути пов'язана з іншою сутністю або сама з собою. У зв'язках може даватися ім'я, яке виражається граматичним оборотом дієслова і поміщується 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 утворена з'єднанням імені батьківській сутності, імені зв'язку, виразу ступеня та імені сутності-нащадка. Кожен зв'язок може мати один із наступних типів зв'язку: один-до-одного 1:1, один-до-багатьох 1:N, багато-до-багатьох M:N (рис. 2.3).

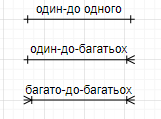


Рисунок 2.3 – Типи зв’язків

Розглянемо методи виявлення сутностей предметної області на прикладі розробки інформаційній системи з замовлень деякої оптової торгової фірми. Бізнес сценарій для проектованої системи з точки зору менеджера 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 декілька.

***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*** - явний кандидат на нову сутність.

***(?) Наявність товару***  ̶ це, швидше за все, атрибут, але атрибут якої 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). Кожну накладну потрібно виписувати на одного покупця. Кожна накладна повинна містити кілька товарів (не буває пустих накладних). Кожен товар, у 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-до-багатьох). 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).

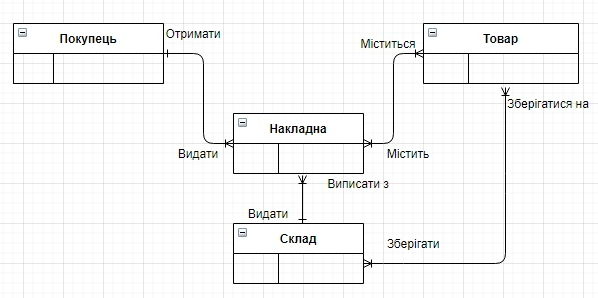


Рисунок 2.4 ̶ Модель ER для сутностей Покупець, Товар, Накладна, Склад

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

* Кожен покупець є юридичною особою і має найменування, адресу, 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 раніше ̶ 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 у накладній» все досить зрозуміло. Сутності «Накладна» і «Товар» пов'язані одна з одною відношенням типу багато-до-багатьох. Такий зв'язок повинен бути розділений на два зв'язки типу один-до-багатьох. 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 у накладних»,

«Кожен запис зі списку товарів у накладній повинен бути пов'язаний рівно з одним товаром».

  Атрибути «Кількість товару в накладній" і «Ціна товару в накладній» є атрибутами сутності «Список товарів у 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 своя.

Тепер можна внести все це в діаграму (рис. 2.5):

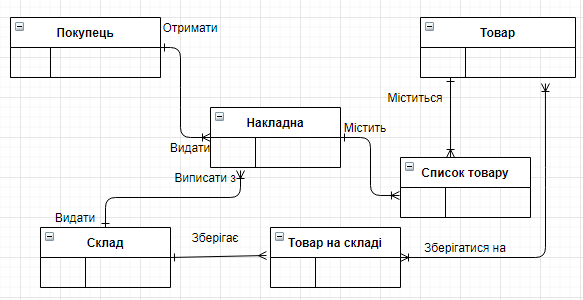


Рисунок 2.5 ̶ Модель ER для АІС «Продаж товару»

**2.2 Приклад створення інформаційної моделі ER для АІС «RepairTech»**

Створюємо ***Словник предметної області*** на основі моделі бізнес-процесу***:***

***Адміністратор, Майстер 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.

***Квитанція*** – документ, який видається клієнту для підтверження передачі п/т.

***Накладна*** – документ, в якому Майстер з діагностики фіксує причину поломки, вартість, плановану дату завершення ремонту.

***Склад запчастин*** – місце зберігання складових частин п/т.

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

Таблиця 2.1 ̶ Таблиця сутностей

| Сутність | Атрибути | Ключі |
| --- | --- | --- |
| Співробітник | ПІБ  Телефон  Адреса  Дата народження  Дата прийняття на роботу  Посада | ПК  ЗК |
| Посада | Назва  Оклад | ПК |
| Замовлення | ПІБ клієнта  Адміністратор  Причина поломки  Майстер з діагностики  Назва п/т  Номер приладу  Дата | ПК  ПК  ПК |
| Накладна\_ремонту | Замовлення  Мастер з ремонту  Вартість ремонту  Дата початку ремонту  Дата заверш ремемонту | ПК, ЗК |
| Склад запчастин | Номер  Назва запчастини  Кількість  Вартість | ПК |
| Список | Накладна\_ремонту  Склад запчастин  Кількість | ЗК  ЗК |

Створюємо модель ER (рис. 2.6) за допомогою додатка ***draw.io***

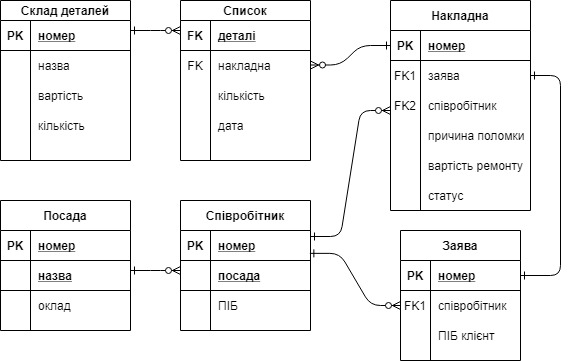


Рисунок 2.6 ̶ Модель ER для АІС «RepairTech»

**Питання для самоконтролю**

1. Дати визначення структурного моделювання.
2. Назвати і дати визначення методологій структурного моделювання.
3. Описати інформаційну модель даних сутність-зв’язок ER, назвати елементи цієї моделі.
4. Описати елементи нотації Бахмана (пташина лапка).
5. Назвати 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 сутностей.
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 подання системи в рамках структурного підходу.

**3.1 Короткі теоретичні відомості за методологію IDEF0**

***IDEF0 ̶***  методологія функціонального моделюванняполягає в тому, що за допомогою наочної графічної мови IDEF0 система, що розробляється, постає перед проектувальниками у вигляді набору взаємозв'язаних функцій (функціональних блоків ̶ у 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 поділяють складний об'єкт на складові частини, які надані у вигляді блоків. Деталі кожного з основних блоків показані у вигляді блоків на інших діаграмах. Кожна детальна діаграма є декомпозицією блока з більш загальної діаграми. На кожному кроці декомпозиції більш загальна діаграма називається батьківською 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 й асоційовані з ними відносини. Методологія може містити чотири типи діаграм:

1) Контекстна діаграма (діаграма верхнього рівня), що є вершиною деревовидної структури діаграм, показує призначення системи (основну функцію) і її взаємодію з зовнішнім середовищем. У кожній моделі може бути тільки одна контекстна діаграма. Після опису основної функції виконується функціональна декомпозиція, тобто 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 створених діаграм. Знайдені невідповідності усуваються, після чого починають подальшу деталізацію 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 у системі процесів (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 позначок і найменувань (відсутність повторюваних імен);
* синтаксичні правила для графіки (блоків і дуг);
* поділ входів та керувань (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 (декомпозировані) на більш прості дії, операції, функції. Іншими словами, складна бізнес-функція може бути показана ​​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, надані 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 податковий інспектор бачать організацію абсолютно по-різному. Контекстна діаграма відіграє ще одну роль у функціональній моделі. Вона фіксує межу в якій моделюється система, визначаючи те, як модельована система взаємодіє зі своїм оточенням. Це досягається за рахунок опису дуг, з'єднаних з блоком, що являє головну бізнес-функцію.

Основні елементи:

1. ***Функціональний блок*** (Activity Box) графічно зображується у вигляді

прямокутника й уособлює собою деяку конкретну функцію в рамках даної системи. Усередині кожного блока міститься його ім'я і номер. Номер блока розміщується в правому нижньому кутку. За вимогами стандарту назва кожного функціонального блока має бути сформульована глагольним способом, тобто бути дієсловом або дієслівним оборотом (наприклад, "виробляти послуги", а не «виробництво послуг»).

2. ***Интерфейсна дуга*** (Arrow) відображає елемент системи, який обробляється функціонального блока або впливає на функцію, відображену даними функціональним блоком. Залежно від того, до якої зі сторін підходить дана інтерфейсна дуга, вона носить назву «вхідної», «вихідної»,«керуючої» або «механізмом». Дуги (рис. 3.1) можуть складатися тільки з вертикальних або горизонтальних відрізків; відрізки, спрямовані по діагоналі не допускаються. Кінці стрілок повинні стосуватися зовнішньої межі функціонального блока, але не повинні перетинати її. Дуги повинні приєднуватися до блока на його сторонах.

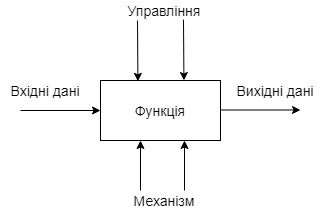


Рисунок 3.1 ̶ Елементи діаграми IDEF0: функція та інтерфейсні дуги

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

В основі методології IDEF0 лежать наступні правила:

* функціональний блок (або функція) перетворює інформацію «вхідну» у «вихідну», «керування» визначає, коли і як це перетворення може або повинно відбутися. Виконавці безпосередньо здійснюють цей захід;
* з дугами пов'язані написи (або позначки) природною мовою, що описують дані, які вони представляють;
* дуги показують, як функції між собою взаємопов'язані, як вони обмінюються даними і здійснюють керування одна одною;
* «виходи» однієї функції можуть бути «входами», «керування» або «механизми» для іншої;
* дуги можуть розгалужуватися і з'єднуватися;
* функціональний блок, який подає систему як єдиний модуль, деталізується на іншій діаграмі за допомогою декількох блоків, з'єднаних між собою інтерфейсними дугами;
* ці блоки є основними підфункціями (підмодулями) єдиного вихідного модуля;
* декомпозиція виявляє повний набір підмодулей, кожен з яких

поданий як блок, межі якого визначено інтерфейсними дугами. Кожен із цих підмодулей може бути декомпозований подібним чином для більш детального подання;

* кожна підфункція може містити тільки ті елементи, які входять у 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 у блок і виходять з нього на діаграмі верхнього рівня, є точно тими ж самими, що і дуги, що входять у діаграму нижнього рівня і виходять з неї, тому що блок і діаграма є однією і тією ж частиною системи.

4***. Контекстна діаграма.*** Модель IDEF0 завжди починається з подання системи як єдиного цілого ̶ одного функціонального блока з інтерфейсними дугами, що тягнуться за межі даної області. Така діаграма з одним функціональним блоком називається контекстною діаграмою, і позначається ідентифікатором «А-0». Оскільки єдиний блок являє собою всю систему як єдине ціле, ім'я, вказане в блоці, є спільним. Це вірно і для інтерфейсних дуг ̶ вони також є повним набором зовнішніх інтерфейсів системи в цілому.

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

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 для АІС «RepairТеch»

Таблиця 3.1 ̶ Опис елементів контекстної діаграми IDEF0

|  |  |
| --- | --- |
| Назва процесу | Діяльність фірми з ремонту побутової техніки |
| Мета процесу | Якісно і вчасно відремонтувати п/т для отримання прибутку |
| З токи зору | З позиції господаря фірми |
| Вхідні дані | * п/т клієнкта * оплата * запчастини |
| Вихідні дані | * п/т клієнкту * звіти |
| Управління | * правила та процедури фірми |
| Механізми | * співробітники фірми |

Створимо контекстну діаграму «А-0» для АІС «RepairТеch» засобами draw.io, рис. 3.2.

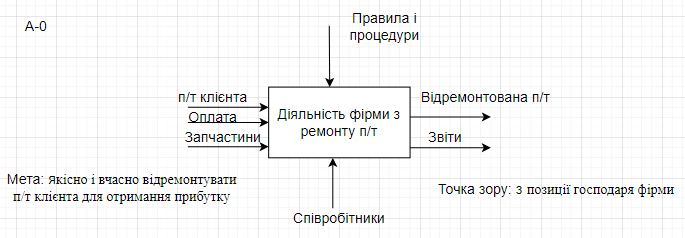


Рисунок 3.2 ̶ Контекстна діаграма **«А-0»** IDEF0 для АІС «RepairТеch»

**5.** ***Діаграми декомпозиції***

Блок, який являє систему як єдиний модуль, деталізується на іншій діаграмі за допомогою декількох блоків, з'єднаних інтерфейсними дугами. Ці блоки є основними підфункціями вихідної функції. Дана декомпозиція виявляє повний набір підфункції, кожна з яких подана ​​як блок, межі якої визначено інтерфейсними дугами. Кожна з цих підфункцій може бути декомпозована подібним чином для більш детального надання.

Кожна підфункція 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, надана ​​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 блока. Дочірня діаграма, створювана при декомпозиції, охоплює ту ж область, що і батьківський блок, але описує її більш детально. Таким чином, дочірня діаграма начебто вкладена у свій батьківський блок.

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

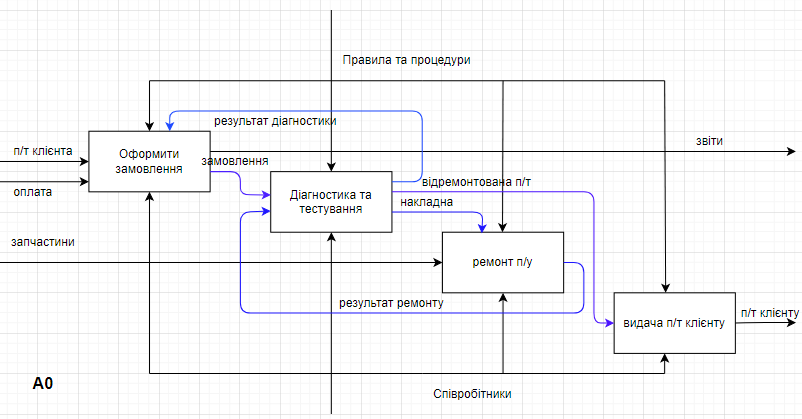
*Межеві стрілки*. На звичайній (НЕ контекстній) діаграмі межеві стрілки є входами, керуванням, виходами або механізмами батьківського блока діаграми. Джерело або споживач межевих стрілок можна виявити, тільки вивчаючи батьківську діаграму. Всі межеві стрілки на дочірній діаграмі (за винятком стрілок, поміщених у тунель, повинні відповідати стрілкам батьківського блока.

*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 дочірній діаграмі. Стрілка, вміщена в тунель там, де вона приєднується до блока, 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 декомпозиції «А0» для АІС «RepairТеch». Побудуємо діаграму декомпозиції «А0» контекстної діаграми «А-0», яка представляє функціїї першого рівня ієрархії функцій системи, тобто є декомпозіцією основної функції АІС.

Перший рівень декомпозиції містить чотири блоки:

* Оформлення замовлення на ремонт.
* Діагностика та тестування п/т.
* Ремонт п/т.
* Передача п/т клієнту.

Рисунок 3.3 ̶ Діаграма декомпозиції «А0» методології IDEF0 для АІС **«**RepairТеch»

**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 подає 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 - СКБД);
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 межеві стрілки на діаграмі DFD;
* створити відповідні зовнішні сутності і сховища даних;
* створити внутрішні стрілки, що починаються з зовнішніх сутностей замість межевих стрілок;
* стрілки на діаграмі IDEF0 затунелювати.

***Елементи графічної нотації DFD Гейне-Сарсона***

***Зовнішня сутність*** (термінатор) (рис. 3.4) являє собою матеріальний об'єкт або фізичну особу, що виступають як джерело або приймач інформації (наприклад, замовники, персонал, програма, склад, інструкція). Зовнішні сутності на DFD за змістом відповідають керуванню і механізмам, що відображається на контекстній діаграмі IDEF0. Визначення деякого об'єкта, суб'єкта або системи як зовнішньої сутності вказує на те, що вона знаходиться за межами границь 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 надана ​​як зовнішня сутність

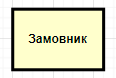


Рисунок 3.4 ̶ Зовнішня сутність Рисунок 3.5 ̶ Процес

***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 допустимої швидкості». Номер процесу служить для його ідентифікації і ставиться з урахуванням декомпозиції. На відміну від IDEF0 вкладеність процесів позначається через крапку (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 за наступною формулою: Назва роботи = Дія + Об'єкт, над яким дія 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 короткою (не більше 50 символів) і складатися з двох або з трьох слів.

У складних випадках також рекомендується для кожної короткої назви роботи зробити її докладний опис, який помістити у 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 поміщення і вилучення можуть бути будь-якими. Вони 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 у межах усієї моделі (усього набору діаграм). Ім'я накопичувача вибирається з міркування найбільшої інформативності для розробника. Наприклад, якщо як накопичувачі є 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 надання накопичувачів у подальшому полегшить побудову інформаційної моделі системи (рис. 3.6).

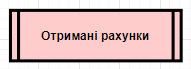


Рисунок 3.6 ̶ Накопичувач Рисунок 3.7 ̶ Потік даних

***Потік даних*** (рис. 3.7) визначає інформацію (матеріальний об'єкт), передану через деякі з'єднання від джерела до приймача. Реальний потік даних може бути інформацією, переданою 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. Напрям стрілки показує напрям потоку даних. Іноді інформація може рухатися в одному напрям, 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 межі зовнішньої сутності, процесу або накопичувача даних.

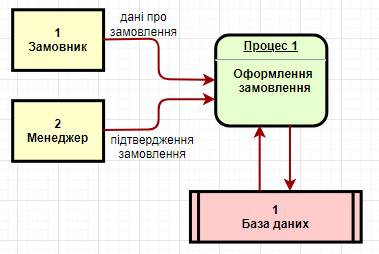


Рисунок 3.8 ̶ Приклад нотації Гейне-Сарсона

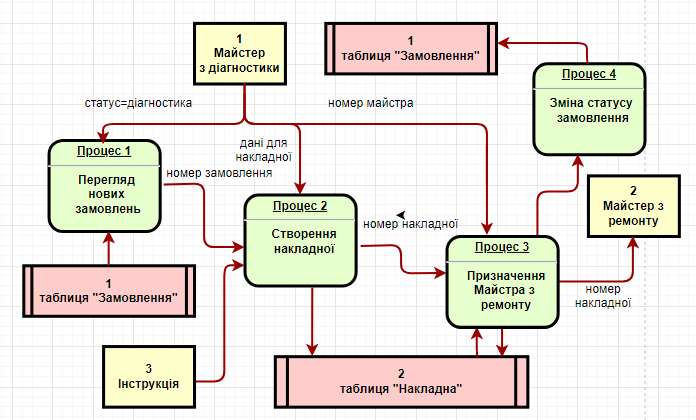
**3.3 Приклад створення нотації Гейне-Сарсона для функції «Діагностика та тестування» АІС «RepairТеch»**

Декомпозиція блока «Діагностика та тестування» (рис.3.8) має три процеса:

* Перегляд замовлень зі статусем «діагностика».
* Створення накладної на ремонт п/т.
* Призначення майстра з ремонту.
* Зміна статусу.

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

Накопичувачі: таблиця БД «Замовлення», 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 ії елементі.
3. У чому полягають правила методології IDEF0?
4. Надати визначення методології DFD та описати основні ії елементи.
5. Які нотації є методологією 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 з самоконтролю.
2. Створити для власної АІС контекстну діаграму «А-0» та діаграму декомпозиції «А0» згідно з методологією IDEF0.
3. Створити для власної АІС нотацію Гейна-Сарсона для одного із 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- орієнтованого моделювання для виявлення межі системи та взаєїмодії користувачів з системою за допомогою моделі прецедентів.

* 1. **Короткі теоретичні відомості з методології об’єктно орієнтованного моделювання**

**4.1.1 Принципи об'єктно-орієнтованої методології**

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

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

* об'єктно-орієнтований аналіз (OOA),
* об'єктно-орієнтоване проектування (OOD),
* об'єктно-орієнтоване програмування (OOР).

**ООА** методологія аналізу сутностей реального світу на основі понять класу й об'єкта, для розуміння і пояснення того, як сутності взаємодіють між собою. ***У предметній області виділяються класи об'єктів, які, якщо це необхідно, поділяються 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 структури системи.

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

Говорячи про об'єктно-орієнтовані розробки, маємо на увазі: об'єктно-орієнтовані методології (технології) розробки програмних систем; інструментальні засоби, що підтримують ці технології. На сьогоднішній день в якості основного підходу 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 ін.

**4.1.2 Методологія RUP для розробки ПЗ у рамках об'єктно-орієнтованого підходу**

Методологія RUP (Rational Unified Process) застосовує уніфіцирований процес (UP) розробки ПЗ у рамках об'єктно-орієнтованого підходу. UP є модифікацією ітеративної моделі ЖЦ, де розробка ПЗ виконується у вигляді декількох короткострокових міні-проектів фіксованої тривалості (наприклад, 4 тижня), які називаються фазами. Кожна фаза включає свої власні етапи аналізу вимог, проектування, реалізації і завершується тестуванням та створенням робочої системи. На кожній фазі система доповнюється новими можливостями, які узгоджуються з уже створенним ядром, та контролюється завдяки зворотному зв’язку з замовником та майбутніми користувачами.

Перевоги ітеративної розробки:

* своєчасне усвідомлення можливих технічних ризиків, розуміння вимог, задач проекту та зручність використання системи;
* швидкий та помітний прогрес;
* ранній зворотний зв’язок з користувачами та адаптація системи;
* керування складністю;
* отриманий на кожній ітерації досвід може методично використовуватися для покращення організації процесу розробки.

Також особливостю UP є: побудова базової архітектури 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 декілька дисциплін, наприклад, бізнес-моделювання (business modeling), вимоги (requirement), проектування (design), артефакти яких будуть використовуватися 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 артефактів для фаз UP на різних ітераціях

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Дисципліна | Артефакт  Ітерація | Початок  I1 | Розвиток  E1 …En | Конструювання  C1….Cn | Передача  T1….Tn |
| Бізнес-моделювання | Модель предметної області |  | п |  |  |
| Вимоги | Модель прецедентів | п | р |  |  |
| Бачення системи | п | р |  |  |
| Допоміжна спеціфікація | 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 функціювання системи та ії оточення***, тобто ця модель фіксує верхньо рівневі прецеденти, які були розглянуті раніше у вигляді текстового опису прецедентів, діаграм варіантів використання (прецедентів) та послідовтості взаємодії користувачів з системою мовою 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, інші будуть описані на ітераціях розвитку.

***Бачення системи*** – це документ, який описує основну мету та проблеми проекту, назіває коло зацікавленних осіб та їх потреби, осносні ідеї розробки проекту, тобто містить основу для більш детального опису технічних вимог до проекту.

***Допоміжна специфікація*** – це документ, який містить вимоги, обмеження та іншу інформацію, що не увійшла до опису прецедентів та у словник термінів, наприклад, такі як вимоги якості та спеціальні вимоги. До цього документа можно додавати наступну інформацію:

* вимоги згідно з моделлю 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 довідкова інформація;
* згода на ліцензування та інші згоди.

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

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

1. Створити короткий варіант документа «Бачення системи».
2. Визначити задачі користувачів і відповідні прецеденти.
3. Описати деякі прецеденти і почати створювати документ «Допоміжна специфікація».
4. На підставі отриманої інформації уточнити документ «Бачення системи».

У межах даної дисципліни зупинимося на створенні «Моделі прецедентів» на основі варіантів використання, яки були розглянуті раніше, 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 у 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 «потік подій» (flow of events). Сценарій детально документує процес взаємодії дійової особи з системою, що реалізується в рамках варіанта використання. Основний потік подій описує нормальний хід подій (за відсутності помилок). Альтернативні потоки описують відхилення від нормального перебігу подій (помилкові ситуації) та їх обробку. Переваги моделі варіантів використання:

* визначає користувачів і межі 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 полягає в наступному: проектована система надана у вигляді безлічі сутностей або акторів, які 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 модульовану систему так, як визначить сам розробник.

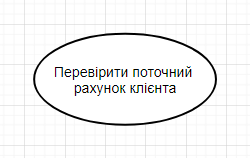
***Добре структурована діаграма варіантів використання має такі властивості:***

* акцентує увагу лише на одному аспекті динаміки системи;
* містить тільки такі прецеденти і акторів, 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 проектованою системою при взаємодії її з відповідним актором.



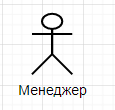


Рисунок 4.1 – Варіант використання Рисунок 4.2 – Актор

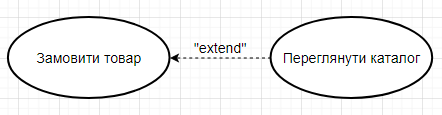
***Актор*** (actor), який зображається фігуркою людини, (рис.4.2) ̶ це безліч логічно пов'язаних ролей, виконуваних при взаємодії з прецедентами або сутностями (системою, підсистемою або класом).

Актори використовуються для моделювання зовнішніх по відношенню до проектованої системи 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 надані трьома різними способами, а саме за допомогою відношенню 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 варіанта 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).

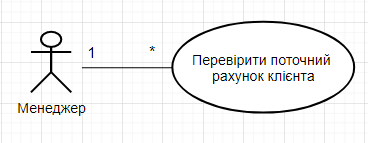


Рисунок 4.3 ̶ Відношення асоціяції Рисунок 4.4 – Відношення розширення

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

У мові UML відношення розширення є залежністю, спрямованою до базового варіанта використання і з'єднаною з ним у так званій точці розширення. Відношення розширення між варіантами використання позначається пунктирною лінією зі стрілкою (варіант відношення залежності), спрямованою від того варіанта використання, який є розширенням для початкового варіанта використання. Дана лінія зі стрілкою позначається ключовим словом **«extend»**, як показано на рис. 4.4. Один із 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) є вказівкою того факту, що деякий дочірній варіант використання може бути узагальнено до варіанта використання батька. Слід підкреслити, що нащадок успадкує всі властивості і поведінку свого батька, а також може бути доповнений новими властивостями й особливостями поведінки. Графічно дане відношення позначається суцільною лінією зі стрілкою у формі незафарбованого трикутника, яка вказує на батьківський варіант використання (рис. 4.5). Ця лінія зі стрілкою має спеціальну назву ̶ стрілка «узагальнення».

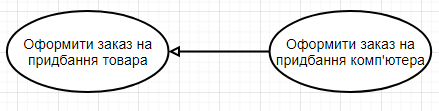


Рисунок 4.5 – Відношення узагальнення

Відношення узагальнення між варіантами використання застосовується в тому випадку, коли необхідно відзначити, що дочірні варіанти використання мають усі атрибути й особливості поведінки батьківських варіантів. При цьому, один варіант використання може мати кілька батьківських варіантів. В цьому випадку реалізується множинне успадкування властивостей і поведінки відношенню батьків. З іншого боку, один варіант використання може бути батьком для декількох дочірніх варіантів, що відповідає таксономічному характеру відношенню узагальнення.

Між окремими акторами також може існувати відношення узагальнення. Дане відношення є спрямованим і вказує на факт спеціалізації одних акторів щодо інших. Наприклад, відношення 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 відношення 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 у послідовність поведінки іншого варіанта використання.

Семантика цього відношення визначається наступним чином, коли екземпляр першого варіанта використання в процесі свого виконання досягає точки включення в послідовність поведінки екземпляра другого варіанта використання, екземпляр першого варіанта використання виконує послідовність дій, що визначає поведінку екземпляра другого варіанта використання, після чого продовжує виконання дій своєї поведінки. При цьому передбачається, що навіть якщо екземпляр першого варіанта використання може мати кілька включень у себе екземплярів інших варіантів, виконувані ними дії повинні закінчитися до деякого моменту, після чого має бути продовжено виконання перерваних дій екземпляра першого варіанта використання відповідно до заданої для нього поведінки.

Один варіант використання може бути включений в кілька інших варіантів, а також включати у себе інші варіанти. Варіант використання, який включається, може бути незалежним від базового варіанта в тому сенсі, що він надає останньому деяку інкапсульовану поведінку, деталі реалізації якої приховані від останнього і можуть бути легко перерозподілені між декількома долученими варіантами використання.

Відношення включення, спрямоване від варіанта використання А («Оформити заказ на придбання комп'ютера») до варіанту використання В («Виписати рахунок на оплату комп'ютера»), вказує, що кожен екземпляр варіанта А включає у себе функціональні властивості, задані для варіанта В. Ці властивості спеціалізують поведінку відповідного варіанта А на даній діаграмі. Графічно дане відношення позначається пунктирною лінією зі стрілкою (варіант відношення залежності), спрямованої від базового варіанта використання до такого, що включається. При цьому дана лінія зі стрілкою позначається ключовим словом **«include»**, як показано на рис. 4.6.

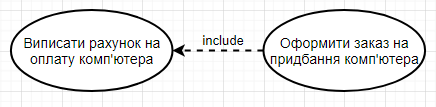


Рисунок 4.6 – Відношення включення

***Приклад створення діаграми варіантів використання UML для АІС «RepairТеch».***

Для побудови діаграми варіантів використання UML для АІС «RepairТеch» для користувачів (Адміністратор, Майстер з діагностики, Майстер з ремонту) будемо використовувати опис варіантів використання , який був показаний в темі №1. Діаграма варіантів використання UML для АІС «RepairТеch» показана на рис. 4.7

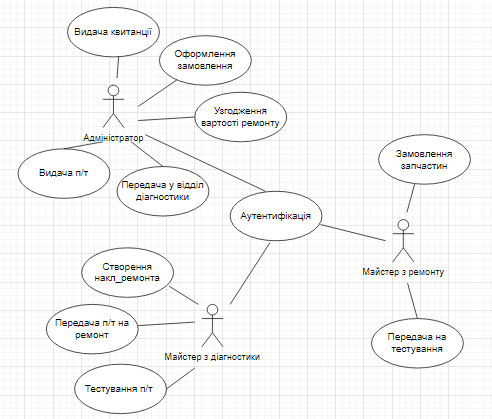


Рисунок 4.7 – Діаграма використання UML для АІС «RepairТеch»

Діаграми використання повинні обов’язково бути доповнені сценаріями для кожного прецедента за шаблоном, який розглянуто раніше в темі №1.

**4.3 Діаграма послідовності («sequence diagram»)**

***Діаграма послідовності***  ̶ діаграма взаємодії, в якій основний акцент спрямовано на впорядкування повідомлень у 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 відсутнє на діаграмі послідовності, то назівається тільки ім'я класу, а сам об'єкт вважається анонімним.

***Лінія життя об'єкта*** (object lifeline) ̶ це вертикальна пунктирна лінія, яка відображає існування об'єкта в часі. Якщо об'єкт існує протягом усієї взаємодії, його зображують у верхній частині діаграми, а його лінія життя проходить 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 керування***  («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 керування може тільки існуючий об'єкт, у якого в цей момент є лінія життя. Якщо ж певний об'єкт був знищений, то знову виникнути у системі він уже 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 у системі може бути актор або зовнішній користувач. В цьому випадку актор зображується на діаграмі послідовності найпершим об'єктом зліва зі своїм фокусом керування. Найчастіше актор і його фокус керування 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 іншому. При цьому прийом повідомлення ініціює виконання певних дій, спрямованих на вирішення окремого завдання тим об'єктом, з якого це повідомлення надіслано. На діаграмі послідовності всі повідомлення впорядковані за часом свого виникнення в модельованій системі. У такому контексті кожне повідомлення має напрямок від об'єкта, який ініціює та відправляє повідомлення до об'єкта, який його отримує.

***Стереотипи повідомлень:***

«call» (викликати) ̶ синхронні повідомлення (synchronous message), які 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. В усіх 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»), які не потребують відповіді і 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 потребувати явної 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:

{Час\_прійому\_повідомлення (час\_відправки\_повідомлення) <1 сек.}

{Час\_очікування\_відповіді <5 с.}

{Час\_передачі\_пакета <10 с.}

{Об'єкт\_1. час\_подачі\_сигнала\_тревоги> 30 с.}

Елементи діаграми послідовності показані на рис. 4.8.

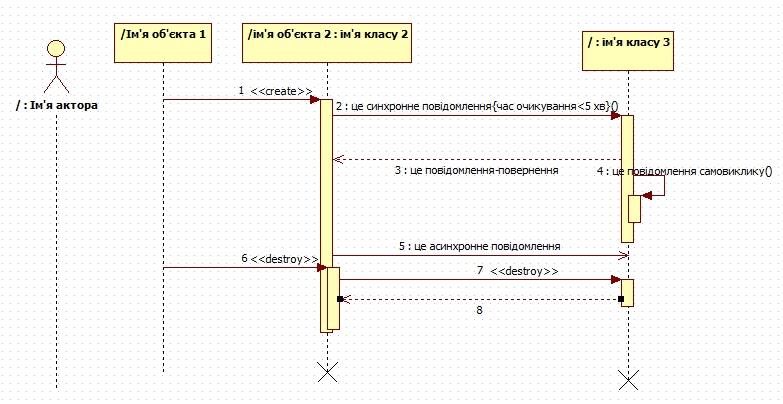


Рисунок 4.8 – Елементи діаграми послідовності

В межах «Моделі прецедентів» створюються діаграми послідовності, які демонструють обмін повідомленнями 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. Призначення цієї діаграми ̶ відображення подій, переданих виконавцями системи через ії 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 АІС «RepairТеch», використовючи його основний сценарій.

***Прецедент «Оформлення замовлення»***. Основні об’єкти: актор «Адміністратор», «Система». Повідомлення «Адміністратор», (рис. 4.9): «makeOrder» (створити замовлення), «addMaster» (призначити Майстра з діагностики), «changeStatus» (змінити статус), «Save» (зберегти). Повідомлення «Система»: «masterDiag» (номер Майстра з діагностики), «Receipt» (квитанція).

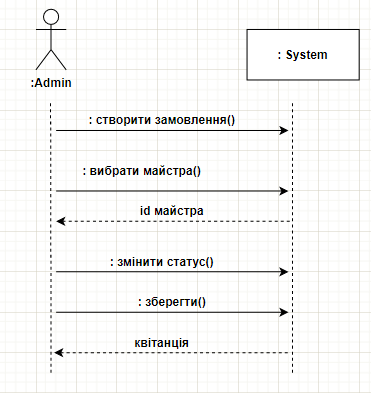


Рисунок 4.9 – Діаграма послідовності для прецедента «Оформлення замовлення»

**Питання для самоконтролю**

1. Які основні принципи об’єктно - орієнтованого моделювання ПЗ?
2. Надати визначення методології RUP. Фази та артифакти UP.
3. За допомогою яких 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 взаємодії (послідовності).

**Завдання до лабораторної роботи** **№ 4**

1. Відповісти на питання для самоконтролю.
2. На основіопису варіантів використання, які були створені в лабораторній роботі №1, створити для власної АІС діаграму прецедентів UML.
3. За шаблоном детально описати три найбільш складних прецедентів (основний та альтернативний потік).
4. Створити для всіх прецедентів діаграми послідовності взаємодії 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.

* 1. **Короткі теоретичні відомості з призначення та елементів діаграми класів UML**

Діаграма класів використовується 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 й інших елементів, що працюють спільно для забезпечення деякої кооперативного поведінки, більш значущої, 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 виявляютьсяпри розгляді трьох стереотипів: сутність («entity»), межа («boundary») і керування («control»). Клас-сутністьмістить інформацію, яка зберігається постійно, використовується для моделювання даних і поведінки об'єктів з довгим життєвим циклом. Вони можуть надавати інформацію про предметну область, а можуть надавати елементи самої системи. В UML клас зображається прямокутником з трьома рівнями (рис.5.1), де на першому рівні записується стереотип класу та його ім’я.

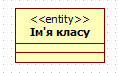


Рисунок 5.1 – Просте зображення класу в UML

***Атрибут*  ̶** це іменована 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, що відповідають деякій властивості класу. Кожне слово в імені атрибута, крім самого першого, зазвичай пишеться з великої літери.

Атрибут має наступні параметри: видимість, ім’я, тип даних, значення за замовуванням, властивості. Крім ім’я, інші параметри не обов’язкові.

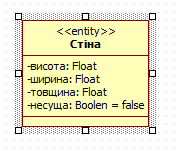


Рисунок 5.2 – Надання атрибутів в елементі класу UML

***Операція класу (operation****)* являє собою деякий сервіс, що надається для кожного екземпляра класу за певною вимогою. Операції записуються на третьому рівні елемента класу. Кожна операція класу в UML має наступні параметри: 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 екземпляра, тоді клас стає службовим (Utility), що містить лише атрибути і операції з областю дії класу;
* рівно один екземпляр, такий клас називають синглетним (Singleton);
* задано число екземплярів;
* довільне число екземплярів - варіант за замовчуванням.

Кількість екземплярів класу називається його кратністю. У мові UML кратність класу задається виразом, написаним у 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. показані варіанти імен об’єктів класів.

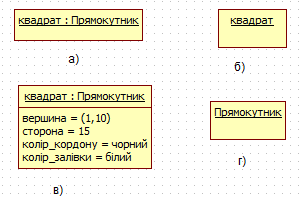


Рисунок 5.3 – Створення елемента об’єкта класу

Між класами у мові UML можна створювати базові відносини або зв'язки:

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

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

 a)

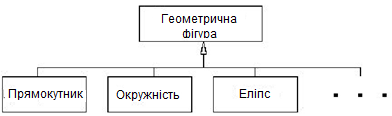
 b)

Рисунок 5.4 – Зв’язок узагальнення

***Асоціація показує відношення між реалізованими об'єктами класу***. Асоціації мають навігацію: двосторонню або однонаправлену, 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).

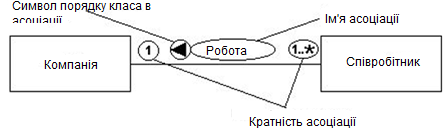


Рисунок 5.5 ̶ Зв’язок асоціації

Спеціальною формою або окремим випадком відношення асоціації є ***відношення агрегації*** (з'єднання частин), яке, у свою чергу, теж має спеціальну форму ̶ ***відношення композиції***. В UML ***агрегація*** відображає зв'язок класів, коли ***об'єкт одного класу є атрибутом іншого.*** Це 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).

 a)

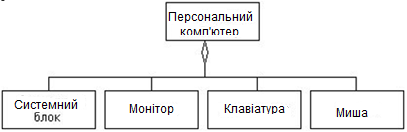
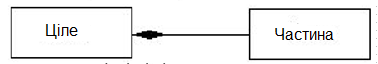
 b)

Рисунок 5.6 ̶ Зв’язок агрегації

***Відношення композиції*** служить для виділення спеціальної форми відношення «частина-ціле», за якої 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 можна.

 a)

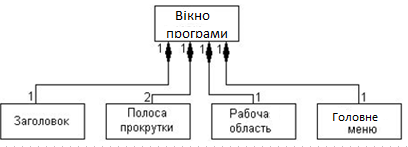
 б)

Рисунок 5.7 ̶ Зв’язок композиції

***Відношення реалізації* *показує відношення клас - об'єкт***. На діаграмі реалізація показана пунктирною лінією з незафарбованим трикутником на кінці (рис. 5.8).

***Відношення*** ***залежності*** в загальному випадку вказує деяке семантичне відношення між двома елементами моделі або двома множинами таких елементів, яке ***не є відношенням асоціації, узагальнення або реалізації****.* Зображується пунктирною лінією зі стрілкою на кінці. Стрілка направленя від класу-клієнта залежності до незалежного класу або класу-джерела, рис 5.8.

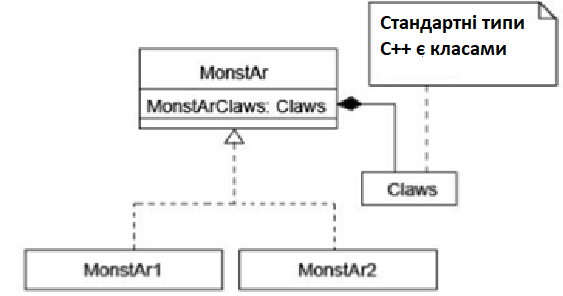
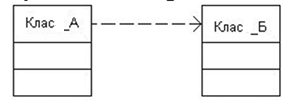


Рисунок 5.8 ̶ Зв’язки реалізації та залежності

**5.2 Модель предметної області в межах методології UP**

***Модель предметної області***  ̶ це найважливіша модель об'єктно-орієнтованого аналізу. Вона відображає основні 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, що відносяться до рівня предметної області. Основні поняття цієї моделі відображаються у словнику термінів.

Ідентифікація набору концептуальних класів ̶ основне завдання об'єктно-орієнтованого аналізу. На початкових ітераціях фази розвитку 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 відграють важливу роль у даній предметній області.

У мові UML Модель предметної області надається у вигляді набору діаграм класів, на яких не визначені жодні операції. Модель предметної області може відображати наступне (рис. 5.9):

* 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 Jаvа або С #. Отже, в Моделі предметної області не використовуються наступні елементи:

* артефакти програмування на зразок вікон або бази даних, якщо тільки розроблювана 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 за визначенням відносяться дані, що зберігаються на постійному носії, оскільки невідомо, чи потрібно зберігати в базі даних інформацію про класи. Концептуальні класи можуть взагалі не мати атрибутів і грати в предметній області чисто поведінкову, а не інформаційну роль.

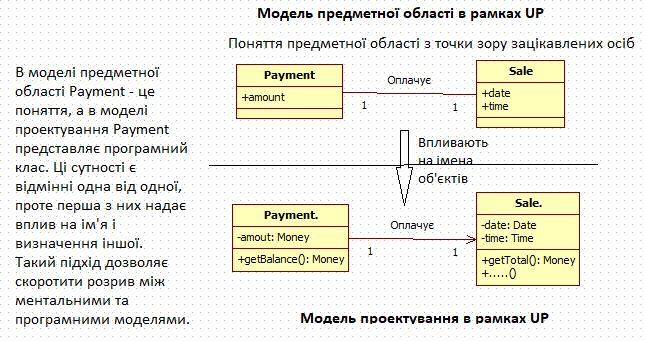


Рисунок 5.9 – Модель предметної області в межах UP

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

* виділити концептуальні класи (дивись нижченадані процедури з рекомендаціями);
* додати їх в Моделі предметної області у вигляді класів на діаграмі UML;
* додати необхідні асоціації й атрибути.

Існує три способи виявлення концептуальних класів:

1. Повторне використання або модифікація існуючих моделей. Це самий перший, найкращий і звичайно, найпростіший підхід. У 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 надані стандартні категорії, які зазвичай мають важливе значення для предметної області системи торгівлі, резервування авіаквитків і гри "Монополія".

Таблиця 5.1 – Категорії концептуальних класів

| Категорія | 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 запису 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 у транзакції | Cashier (Касир),  Customer (Покупець),  Store (Магазин),  MonopolyPlayer (Ігрок),  Pilot (Пілот),  Passenger (Пасажир) |
| ***Місця транзакцій*** | Store (Магазин),  Airport (Аеропорт),  Plane (Літак),  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 програмним системам, які призначені для керування або моделювання | Register (Реєстр),  Airplane (Самоліт),  Item (Товар),  Board (Дошка),  Die (Гральна кістка) |
| ***Опис об’єктів*** | ProductDescription (Специфікація товару),  FlightDescription (Опис польоту) |
| ***Каталоги ̶*** опис часто надає  у каталозі | ProductCatalog (Каталог товарів),  FlightCatalog (Каталог рейсів) |
| ***Контейнери інших об’єктів*** (фізичних або інформаційних) | Store (Магазин), Bin (Бункер),  Airplane (Самоліт), Board (Дошка) |
| ***Вміст контейнерів*** | Item (Елемент),  Square (Клітина на дошці),  Passenger (Пасажир) |
| ***Інші системи, зовнішні по відношенню до даної системи*** | CreditAuthorizationSystem (Система авторизації  кредитних платежів),  AirTrafficControl (Система керування рухом) |
| ***Записи фінансової, трудової, юридичної***  ***та іншої діяльності*** | Receipt (Чек), Ledger (Гроссбух),  MaintenanceLog (Журнал обслуговування) |
| ***Фінансові 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 вибір в якості кандидатів у концептуальні класи або атрибути.

Розглянемо застосування цього методу на прикладі опису сценарію для прецеденту «Оформити замовлення». Виділимо іменники та категорії.

1. Створити **замовлення** (**клієнт**, тип **п/т**, назва, номер, що не працює, дата).
2. Додати інформацію в замовлення.
3. Призначити **майстра з діагностики** за **правилами фірми.**
4. Призначення **статусу.**
5. Роздрукувати **квитанцію.**

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 *Х* у реальному світі не є числом або текстом, 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 являє реальну сутність, організацію, що займає деякий місце. Отже, Store потрібно розглядати як концептуальний клас.

Можно виділити наступні категорії для виявлення концептуальних класів прецедента «Оформлення замовлення»: Транзакції (замовлення), Товари або служби, пов'язані з транзакціями або їх елементами (п/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 (п/т) , Опис об’єктів (Специфікація п/т), Записи фінансової, трудової, юридичної та іншої діяльності (Квитанція), Керівництва, документи на які посилаються в процесі роботи (правила).

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

* Order (замовлення);
* Client (клієнт);
* Diagnostic master (майстер);
* Admin (адміністратор);
* Appliance (п/т).

ApplianceDescription (Специфікація п/т) – потрібна, 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 више, 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 у залежності від конкретного контексту.

***Лінії асоціацій в моделі предметної області не описують потоки даних, зовнішні ключі баз даних, зміні екземпляра класу або зв'язок між об'єктами в програмній 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 у 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. 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).

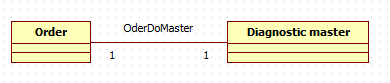


Рисунок 5.10 – Концептуальні класи та асоціація

***Пошук асоціацій за допомогою списку стандартних асоціацій***

Приступимо до додавання асоціацій з використанням списку, наданого у табл. 5.2. В ньому названі 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 |
| --- | --- |
| ***А*** є транзакцією, яка пов'язана  з іншою транзакцією ***В*** | CashPayment-Sale (Платіж готівкою -Продаж)  Reservation-Cancellation (Замовлення квитка-Скасування замовлення) |
| ***А*** є елементом транзакції | SalesLineitem-Sale (Елемент продажу-Продаж) |
| ***А*** є товаром чи пуслугою для  транзакції ***В*** | Item-SalesLineitem (Елемент-Елемент продажу),  Flight·Reservation (Рейс-Резервування) |
| ***А*** є роль, яка пов’язана з транзакцією ***В*** | Customer·Payment (Покупець-Платіж),  Passenger-Ticket (Пасажир-Квиток) |
| ***А*** є фізичною або логічною  частиною ***В*** | Drawer-Register (Пристрій друку торгових чеків-Реєстр),  Seat-Airplane (Місце-Літак),  Square-Board (Клітина-Дошка) |
| ***А*** фізично або логічно міститься у ***В*** | Register-Store (Реєстр-Магазин),  Item-Shelf (Товар-Полка),  Square-Board (Клітина-Дошка),  Passenger-Airplane (Пасажир-Літак) |
| ***А*** є описом ***В*** | ProductDescription-Item (Опис товару-Товар),  FlightDescription-Flight (Опис польоту-Політ) |
| ***А*** відомий / зареєстрований / записаний / включений у ***B*** | Sale-Register (Продаж-Реєстр),  Piece-Square (Об’єкт-Клітина),  Reservation-FlightManifest (Замовлення квитка-Декларація) |
| ***А*** є членом ***В*** | Cashier-Store (Касир-Магазин),  Player-MonopolyGame (Ігрок-Гра "Монополія"),  Pilot-Airline (Пілот-Авіарейс) |
| ***А*** є організаційною одиницею ***В*** | Department-Store (Відділ-Магазин),  Maintenance·Airline (Послуга підтримки-Авіарейс) |
| ***А*** використовує, керує або володіє ***В*** | Cashier-Register (Касир-Реєстр)  Player-Piece (Ігрок-Об’єкт)  Pilot-Airplane (Пілот-Літак) |
| ***А*** слідує за ***В*** | SalesLineitem-SalesLineitem (Назва товару ̶ Назва товару),  Сity-City (Місто-Місто),  Square-Square (Клітина-Клітина) |

Визначемо асоціації для прецеденту «Оформлення замовлення» з застосованням таблиці стандартних асоціацій.

***А*** є товаром чи пуслугою для транзакції ***В*** (Order записує прийом Appliance для ремонту).

***А*** є роль, яка пов’язана з транзакцією ***В*** (Client ініціїрує Order, Admin оформлює Order).

***А*** є членом ***В*** (Diagnostic master призначається для виконання Order).

***А*** є описом ***В*** (ApplianceDescription описує Appliance).

***А*** фізично або логічно міститься у ***В*** (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 умовного зображення класу. Додатково може бути вказано також тип атрибута або інша необов'язкова інформація. Рекомендується описувати вимоги до атрибутів у словниках термінів, який в майбутньому використовується як словник даних.

У Моделях предметної області атрибути повинні бути типами даних («data type»). До стандартного типу атрибутів відноситься логічне значення, дата, номер, рядок (текст), час та інші часто використовувані типи, наприклад, Адреса (адреса), Колір (колір), Геометрія (точка, прямокутник) (Геометричні фігури: точка, прямокутник), Номер телефону (номер телефону), Номер соціального страхування (номер страхового поліса), Універсальний код товару (UPC) (універсальний код товару), SKU, ZIP або поштовий індекс (поштовий індекс), перераховані типи. Стандартна помилка ̶ це моделювання складного поняття предметної області у формі атрибута, наприклад, зовнішній ключ сутності, який потрібно відображати через асоціації.

Тип даних, особливо який розглядається як число або рядок, можуть бути надані у вигляді нового типу даних у Моделях предметних областей в наступних випадках:

* якщо він складений з окремих частин (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 задання 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 є типом даних (унікальна тотожність не використовується в перевірці на рівність), його можна помістити в розділ атрибутів відповідного класу. Якщо же клас типу даних є новим типом й у нього є власні атрибути та асоціації, то більш корисним він буде в якості окремого концептуального класу.

Більшість кількісних сутностей неможно надати у вигляді кількох типів даних. Візьмемо, як приклад, 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 у себе окремий концептуальний клас Quantity (Кількість), з яким буде асоціюватися клас Unit (Одиниця виміру). Так як кількість розглядається як дані типу, відповідний атрибут можна розмістити в розділі атрибутів. Зазвичай кількість вимірюється в деяких одиницях. Гроші ̶ це кількість, одиницею вимірювання якої служить тип валюти. Вага - це кількість, яка вимірюється в килограмах або фунтах.

Розглянемо атрибути концептуальних класів для прецеденту «Оформлення замовлення», які відповідають інформаційним вимогам першої ітерації сценарію. Опис сценарію прецеденту дозволяє виявити наступні атрибути, які пов’язані з необхідністю друку квитанції – 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 прецеденту «Оформлення замовлення», рис. 5.11

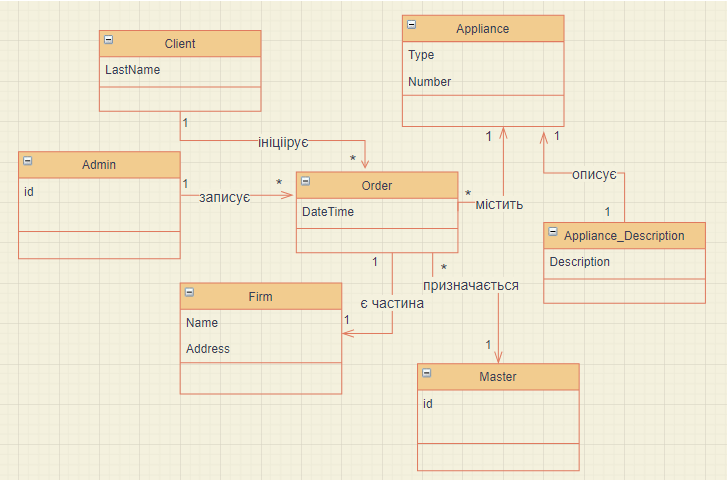


Рисунок 5.11 – Діаграма моделі предметної області для прецеденту «Оформлення замовлення»

При ітеративній розробці Модель 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 взаємозв’язки між ними. Наприклад, на першій ітерації системи «RepairТеch» розглядається лише спрощений сценарій прецеденту «Оформлення замовлення», без друку квитанції замовлення, тому модель предметної області, яка створюється на цій ітерації, відображає лише поняття, необхідні для даної ітерації сценарію (рис. 5.11).

***Виявлення системних операцій концептуальних класів***

Аналіз необхідних операцій в межах прецеденту 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. 4.9). Вхідні системні події вимагають обробки за допомогою системних операцій. Вони обробляються за допомогою об'єктно-орієнтованих методів.

***Шаблон опису опрерації виглядає так:***

***Операція*** – ім’я операції та ії параметри;

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

***Передумова*** - пропозиції про стан систем або об'єктів моделі предметної області до виконання операції. 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 екземпляра,
* формування або розрив асоціацій,
* зміна атрибута.

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

Рекомендації по створенню опису системних операцій:

1. Визначте системні операції за допомогою діаграми послідовності.

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

3. При опису постумови використовуйте категорії.

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

* (добрий варіант) *створено екземпляр SalesLineItem (sli)*,

а не

* (поганий варіант) *створення екземпляра SalesLineItem (sli)*.

1. Які атрибути нових чи вже існуючих об’єктів повинні бути модифицировані,

коли касир вводить код товару *itemID* і кількість його одиниць *quantity*? Повинна бути встановлена кількість одиниць для товару *SalesLineItem*. Таким чином, атрибуту *sli.quantity* присвоєно значення *quantity* (модификація атрибуту).

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 поточним продажею *Sale*. Тому однією із постумов цієї операції є наступний стан:

* екземпляр *sli* пов’язан з поточним екземпляром класу *Sale* (формування асоціації);
* экземпляр *sli* пов’язан з класом *ProductDescription* за допомогою

ідентифікатора товару *itemID* (формування асоціації).

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 недостатьо описані у сценарії прецеденту, тобто такий опис системної операції можна розглядати як ще один механізм формулювання вимог.

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

1. Операція – *makeOrder()*;

Посилання – прецеденти: «Оформлення замовлення».

Передумова – відсутня

Постумова –

* створено екземпляр *ord* класу *Order* *(створення екземпляру);*
* екземпляр класу *Order* пов’язаний з класом *AppliensDiscription* *(формування асоціації);*
* атрибуту *ord.datetime* присвоєно значення *DateTime* *(модифікація атрибута).*

2. Операція – *getMaster()*;

Посилання - прецеденти: «Оформлення замовлення».

Передумова – ініційовано замовлення на ремонт (створено екземпляр класу *Order*).

Постумова –

* створено список *ListOrders[]* екземплярів класу *Order* *(створення екземпляра);*
* екземпляр *ListOrders[i]* пов’язаний з класом екземпляром класу *Order* *(формування асоціації);*
* атрибуту *ord.id\_master* присвоєно значення *ListOrders[min].id\_master* *(модифікація атрибута)*
* атрибуту *ord.status* присвоєно значення *String* “diagnostic” *(модифікація атрибута).*
  1. ***Логічна архітектура***

***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 архітектурою, оскільки не визначає способи розгортання цих елементів у різних операційних системах або на фізичних комп'ютерах у мережі (це відноситься до архітектури розгортання (deployment architecture)). Логіка структури системи, як правило, визначається в документі «Додаткові вимоги». В інформаційних системах зазвичай використовується гнучка багатошарова архітектура (шаблон Layers), за якої 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 шаблона Layers дозволяє вирішити наступні можливі проблеми:

* зміна вихідного коду тягне за собою корегування всіх елементів системи, оскільки всі частини системи тісно пов'язані однf з одною;
* логіка додатка переплітається з інтерфейсом користувача, тому в додатку не можливо змінити інтерфейс або принципи реалізації логіки додатка;
* 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 шаблона 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 додатка ̶ рівней, підсистем, пакетів. Кожен рівень можливо показати у вигляді пакета, наприклад, рівень інтерфейсу користувача можна моделювати за допомогою пакета UI.

Діаграма пакетів забезпечує один зі способів угрупування 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 пакета UML. Ім'я пакета розташовується на корінці, якщо на діаграмі відображається внутрішній вміст пакета, або на самому позначенні пакета.

Найчастіше зображують залежності між пакетами, щоб розробникам був зрозумілий взаємозв'язок елементів системи. Для цього використовується лінія залежності UМL (dependency line), що являє собою пунктирну лінію зі стрілкою, спрямовану в бік 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 й інші позначення вкладених пакетів. Іноді вкладені пакети графічно розміщуються всередині більш загального пакета. Різні позначення вкладених пакетів показані на рис. 5.12.

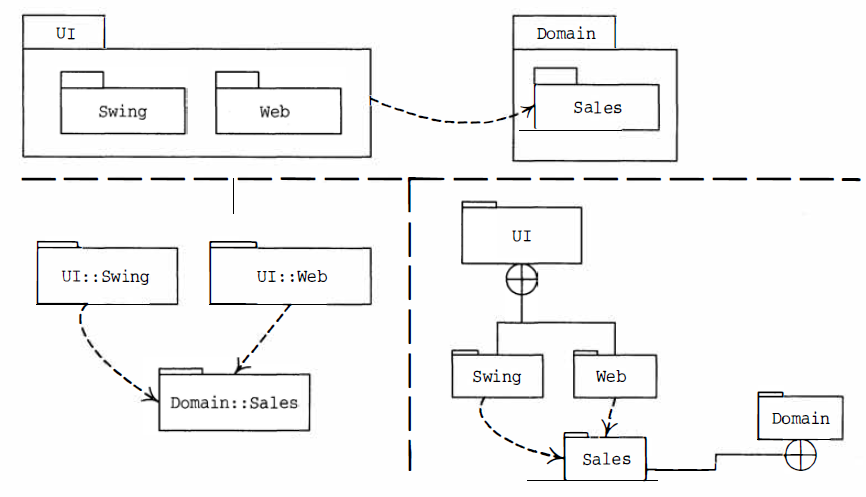
****

Рисунок 5.12 – Зображення шарів на пакетної діаграмі UML

* 1. **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 всі 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 з реалізації логіки додатка. Наприклад, в реальному світі існує РОS-система, яка обробляє 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 являє 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 шару предметної області на основі відповідної моделі, ми скорочуємо розрив уявлення (representational gap) між ріальним світом і його програмною реалізацією. Наприклад, концептуальний клас *Sale* з моделі предметної області стає прототипом програмного класу *Sale* шару 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 шару уявлення. Наприклад, об'єкт *Register* або *Sale* не повинен безпосередньо відправляти повідомлення про зміну кольору або про закриття об'єкта вікна *ProcessSaleFrame*. Деяке «пом'якшення» цього принципу забезпечує шаблон Observer, згідно з яким об'єкти рівня предметної області відправляють повідомлення об'єктам інтерфейсу користувача, що розглядаються тільки як з’єднані об'єкти між шарами (*PropertyListener*).

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

Побудову пакерної діаграми для АІС «RepairTech», див. рис. 5.13.

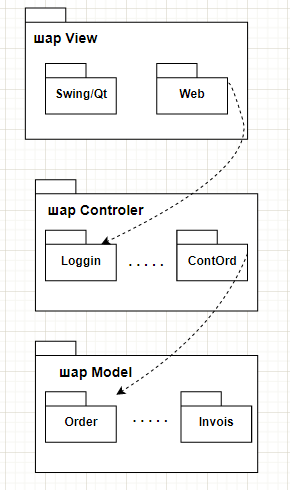


Рисунок 5.13 – Діаграма пакетів для прецедента «Оформлення замовлення»

**Питання для самоконтролю**

1. Перерахувати статичні діаграми UML. З якою метою створюються діаграми классів?
2. Призначення, зображення та правила найменування елементів діаграми класів: клас, атрибут, операція.
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 зображується 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. Призначення діаграми пакетів UML. Елементи діаграми пакетів.
9. Що таке логична архитектура програмної системи? Який архитектурний шаблон застосовується для розробки інформаійної системи? Призначення шару предметноі області.

**Завдання до лабораторної роботи** **№5**

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

**ТЕМА 6. ПОБУДОВА ДІАГРАМ ВЗАЄМОДІЇ ДЛЯ ВИЯВЛЕННЯ РОЗПОДІЛУ ОБОВ’ЯЗКІВ МІЖ ОБ’ЄКТАМИ**

**Мета:** набути практичних навичок розподілу обов’язків між об’єктами програмної системи на основі базових принципів об’єктно-орієнтованого моделювання (шаблони GARSP) за допомогою діаграм взаємодії.

**6.1 Принципи розподілу обов’язків між об’єктами програмної системи (шаблони GARSP)**

Після створення основних документів та моделей (артефактів) початкової фази розробки програмної системи (Модель прецедентів, Словник термінів, Додаткова специфікація, 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 діаграми класів системи необхідно визначити, які операції будуть забезпечувати взаємодію об'єктів системи для виконання ії фунціональності. Тобто треба здійснити розподіл обов'язків між взаємодіючими об'єктами, використовуючи різні принципи об'єктно-орієнтованого проектування, наприклад шаблони GRASP або GoF (Gang-ofFour).

Найчастіше проектування програмних об'єктів і великомасштабних компонентів описують у 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;
* керування діями інших об'єктів і їх координування;
* обов'язки, що пов'язані з знанням об'єкта;
* наявність інформації про закриті інкапсульованих даних;
* наявність інформації про зв'язані об'єктах;
* наявність інформації про наслідки або обчислювані величини.

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

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

У RDD існує ідея коопераціі (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 розподілі обов’язків. Ці шаблони називають також шаблонами розподілу обов'язків (pattern of assigning responsibllities).

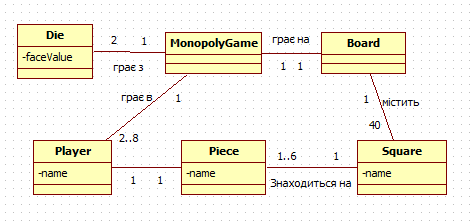
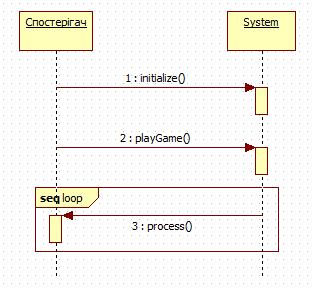
Всього до складу GRASP входить дев’ять шаблонів, але визначимо перших шість найбільш важливих: Creator, Information Expert, Low Coupling, Controller, High Cohesion.

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

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

Назва Creator

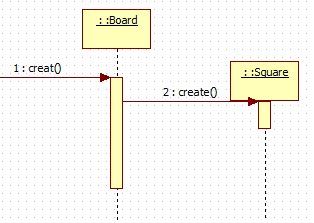
Проблема: xто відповідає за створення нового екземпляра деякого класу А?

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 проектування програмної системи *MonopolyGame,* рис. 6.1. Звернемося до моделі предметної області і зафіксуємо для себе, що 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 агрегуються об'єктом *Board*, тому останній керує їх створенням і руйнуванням.



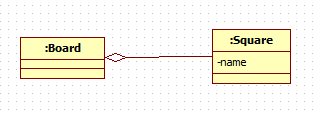


Рисунок 6.2 – Діаграми послідовності та класів , які відображають шаблон ***Creator***

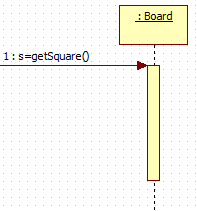
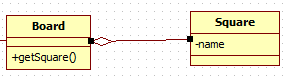
Нагадаємо, що в контексті гнучкого проектування статичні і динамічні моделі створюються паралельно. Отже, це проектне рішення необхідно одночасно відобразити і на діаграмі класів, і на діаграмі взаємодії, рис. 6.2. З рис. 6.2 видно, що при створенні об'єкта *Board* відразу ж створюються й об'єкти *Square*. Для спрощення на цій діаграмі не показаний цикл створення 40 екземплярів 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. У даному випадку кожна конкретна клітина на полі може бути об'єктом, що володіє інформацією про всі клітини. 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 шаблона Expert у 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 ступінь зв'язування одного елемента з іншими. Під зв'язуванням розуміється володіння інформацією або будь-якою залежністю. 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. 6.1 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 шаблона Low Coupling. Якщо клас *Dog* містить метод *getSquare* (), це означає, що він повинен взаємодіяти з об'єктом *Board* для отримання інформації про колекцію об'єктів *Square*. Ймовірно, ці дані зберігаються в об'єкті-колекції *Мар*, 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'єкт *Square*. Таким чином, ступінь зв’язування стає вище тої, що 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 модифікації програмної системи. В зв’язку з цим, можно підкреслити, що шаблон Expert підтримує принцип мінімізації зв’язування.

Як було зазначено раніше, згідно з принципом 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 являє 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 l,a. Клас являє всю систему в цілому або кореневой об'єкт. В ролі такого об'єкта може виступати *MonopolyGame*.

Варіант l,б. Клас являє пристрій, на якому працює дана програмна система. Цей варіант відноситься до спеціалізованих апаратних засобів, наприклад, телефону або касового апарата (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 l,a, який використовується у випадку, коли у системі є лише кілька системних операцій. В якості контролера обираємо об’єкт *MonopolyGame,* який буде реалізовуватисистемнуоперацію *PlayMonopoly* (), тобто повідомлення від об’єкта вікна форми інтерфейсу користувача перетворюється у системну операцію *PlayMonopoly*.

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

Щоб забезпечити можливість підтримки інформації про стан прецеденту, для обробки всіх системних подій в рамках одного прецеденту повинен використовуватися один і той самий клас контролера. Така інформація може знадобитися, наприклад, для ідентифікації моменту порушення послідовності системних випадків (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 делегувати 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 ступінь зв'язування і сфокусованості обов'язків класу. Вважається, що елемент має високий ступінь 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.ін.

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:

* складності розуміння;
* cкладнощі при повторному використанні;
* cкладнощі підтримки;
* ненадійність, постійна схильність до змін.

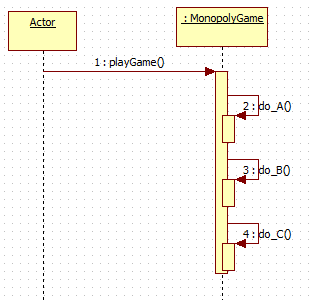
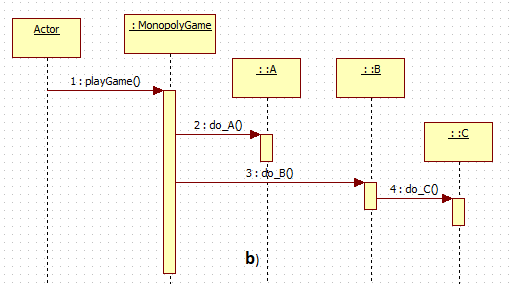
 Неформально: зачеплення (cohesion) в контексті проектування програмних систем являє ступінь зв'язування операцій одного програмного елемента і обсягом роботи, що ним виконується.

Рисунок 6.3 – Приклад використання шаблона ***HighCohesion*** реалізації системної операції *playGame()*

У даному ліворуч проектному рішенні рис. 6.3 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 шаблона ***HighCohesion.***

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

**6.3 Приклад розподілу обов’язків серед об’єктів програмної системи «RepairTech», прецедент «Оформлення замовлення»**

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

* клас, який являє всю систему в цілому, Register, FirmSystem

пристрій або підсистему;

* клас, який представляє одержувача або штучний *ProcessOrderHandle* обробник усіх 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 буде цим прецедентом *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 потрібно замінити на декілько контролерів, рис. 6.4.

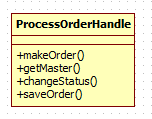


Рисунок 6.4 – Клас-Controller *ProcessOrderHandle* для програмної системи«RepairTech» прецедент «Оформлення замовлення»

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, застосовуючи правила шаблона Creator, згідно з яким обов'язок по створенню нових екземплярів делегується класу, який містить, агрегує або записує інформацію про створювані класи. Проаналізувавши модель предметної області, приходимо до висновку, що об’єкта, який відповідає за створення об’єктів *Order* немає. В цій 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*.

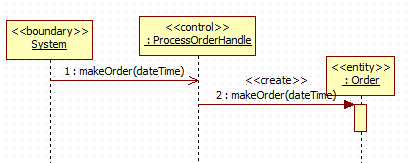


Рисунок 6.5 – Діаграма послідовності для створення об’єкта *Order*

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

* створений список екземплярів *ListOrders*[] об’єктів *Order* *(створення екземпляра);*
* екземпляр ListOrders[i] об’єкта Order пов’язаний з класом Order *(формування асоціації);*
* атрибуту ord.id\_master присвоєно значення ListOrders[min].id\_master *(модифікація атрибуту);*
* атрибуту ord.status присвоєно значення String “diagnostic” *(модіфікація атрибута).*

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 чи 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 й 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. 6.6*).*

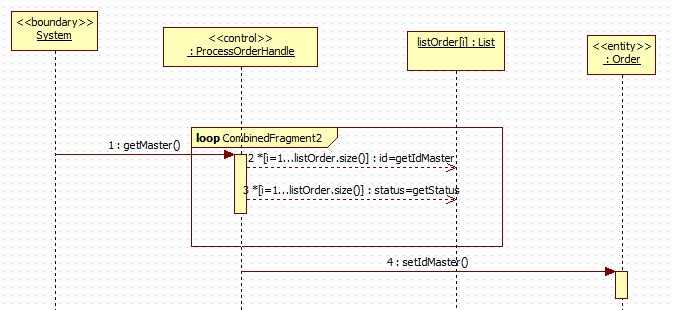


Рисунок 6.6 – Діаграма послідовності для операції *getMaster*()

Наступна системна операція *changeStatus*(), яка змінює статус замовлення. При створенні нового замовлення можливо атрибут «статус» ініціювати за умовчуванням «діагностика», але у бізнес-сценарії 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 стати *ProcessOrderHandle*, який отримає значення статусу з інтерфейсу користувача після визначення Майстра з діагностики, рис.6.7.

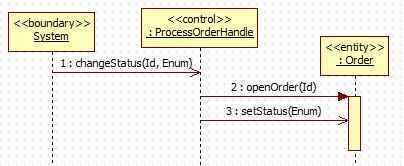
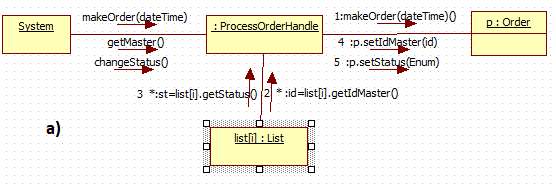


Рисунок 6.7 – Діаграма послідовності для реалізації операції *changeStatus*()

Таким чином, розглянуто обов’язки різних об’єктів у реалізації сценарію прецедента «Оформлення замовлення» на першій ітерації фази розвитку. Підсумуємо результати аналізу у вигляді діаграми кооперації, рис. 6.8,a, яка більш компактна ніж діаграма послідовності, та діаграми класів рис. 6.8,б для цього прецеденту.

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



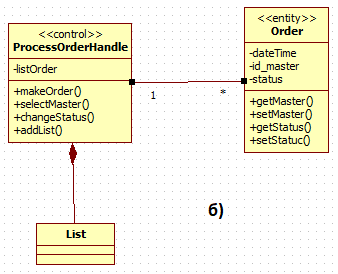


Рисунок 6.8 – a) діаграма кооперації; б) діаграма класів для прецеденту «Оформлення замовлення»

Як уже визначалось раніше, 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.6.8 показує, що на даній ітерації розподіл обов’язків між об’єктами прецеденту «Оформлення замовлення» немає альтернативного рішення і тому відповідає низькому зв'язуванню.

Шаблон ***HighCohesion*** аналізує ступінь зв'язаності і сфокусованості обов'язків класу. Вважається, що елемент має високий ступінь 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.ін. 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. 6.8,б, мають високий ступінь зачеплення, тому що всі операції зв’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? Назвіть і опишіть їх призначення.
3. Для виявлення яких обов’язків застосовується шаблон ***Creator***? Правила його використання. Дайте приклад використання цього шаблона.
4. Для виявлення яких обов’язків застосовується шаблон ***Information Expert***? Які правила його використання? Дайте приклад використання цього шаблона.
5. Для виявлення яких обов’язків застосовується шаблон ***Controller***? Які правила його використання? Дайте приклад використання цього шаблона.
6. Що таке принцип зв’язування 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’єктами одного прецеденту.
3. Результат аналізу надати у вигляді діаграм послідовності, кооперації та класів для кожного шаблона та загалом для прецеденту.
4. Написати коди реалізації класів на C#.
5. Оформити протокол лабораторної роботи.

**Тема 7. ПОБУДОВА ДІАГРАМ СТАНА 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 стана та діяльноті.

* 1. **Використання діаграми стана (activity diagram)**  **UML для аналізу послідовності дій**

Діаграми станів визначають можливі послідовності станів і переходів, у 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 екземпляра певного класу, тобто моделює всі можливі зміни у стані конкретного об'єкта. При цьому зміна стана об'єкта може бути викликана зовнішніми впливами з боку інших об'єктів або ззовні. Діаграма станів по суті є графом спеціального виду, який являє певний автомат.

***Елементи діаграми.***

***Стан.*** У мові UML під станом розуміється абстрактний метакласом, який використовується для моделювання окремої ситуації, протягом якої має місце виконання деякої умови. Стан може бути задано у вигляді набору конкретних значень атрибутів класу або об'єкта, при цьому зміна їх окремих значень буде відображати зміну стана 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 як ім’я використовувати дієслова в теперішньому часі. Формат запису дій: <позначка-дії / вираз-дії>. Позначка дії вказує на обставини або умови за яких буде виконуватися діяльність.

***Початковий та кінцевий стан***. Початковий стан є окремим випадком стана, який не містить ніяких внутрішніх дій (псевдостана). У цьому стані знаходиться об'єкт за замовчунням у початковий момент часу. Він служить для вказівки на діаграмі станів графічної області, від якої починається процес зміни станів. Графічно початковий стан у мові UML позначається у вигляді зафарбованого кола, з якого може тільки виходити стрілка, відповідна переходу. Кінцевий стан являє собою окремий випадок стана, який також не містить ніяких внутрішніх дій (псевдостан). У цьому стані буде перебувати об'єкт за замовчунням після завершення роботи автомата в кінцевий момент часу і служить для вказівки на діаграмі станів графічної області, в якій завершується процес зміни станів або життєвий цикл даного об'єкта. Графічно кінцевий стан у мові UML позначається у вигляді зафарбованого кола, який міститься в іншому колі і в який може тільки входити стрілка переходу.

***Перехід (transition)***  ̶ відношення між двома станами, яке вказує на те, що об'єкт у першому стані повинен виконати певні дії і перейти в другий стан. Перехід здійснюється при настанні деякої події або виконання певної умови, називаною сторожовою умовою. У цьому випадку говорять, що перехід спрацьовує. До спрацьовування переходу об'єкт знаходиться в попередньому від нього стані, називаним 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> '[' <сторожова умова> ']' <вираз дії>. Види переходів: ***тригерний перехід***, який специфікує подія-тригер, пов'язаний із зовнішніми умовами оскільки він розглядався станом. Тригерний перехід обов'язково повинен мати ім'я події, яке він специфікує. ***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 нетрігерний перехід.

***Складений стан і підстан.***  Складений стан (composite state) ̶ такий складний стан, який складається з інших вкладених у 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 стана. Складений стан може містити два або більше паралельних підавтомати або кілька послідовних підстанів. Послідовні підстани (sequential substates) використовуються для моделювання такої поведінки об'єкта, під час якої в кожен момент часу об'єкт може знаходитися в одному і тільки одному підстані. Поведінка об'єкта в цьому випадку являє собою послідовну зміну підстанів.

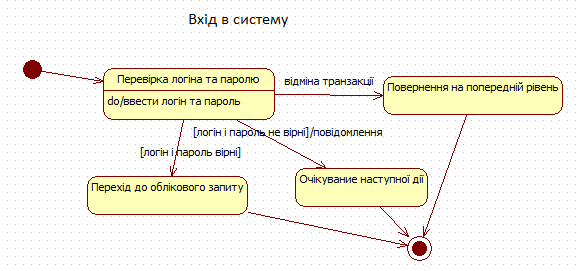
На рис. 7.1 показано приклад діаграми станів, яка створена за допомогою CASE засобу StarUML.

Рисунок 7.1 – Діаграма стана для системи регістрації користувача. У квадратних лапках для нетригерної події зазначені сторожові умови

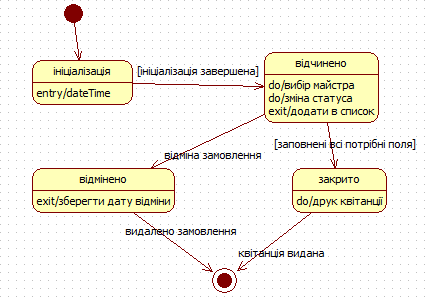
Для прикладу збудуємо діаграму стана для прецеденту «Оформлення замовлення», на якій зобразимо такі стани: ініціалізація замовлення, заповнення полів, при збереженні замовлення – друк квитанції, відміна замовлення, рис.7.2.

Рисунок 7.2 – Діаграма стану для прецеденту «Оформлення замовлення»

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

Діаграма діяльності показує потік переходів від однієї діяльності до іншої, при цьому увага фіксується на результаті діяльності. Сам же результат може привести до зміни стана 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 керування переходить від однієї діяльності до іншої.

Діаграми діяльності найбільш часто застосовуються:

* для опису поведінки, що включає значну кількість паралельних процесів;
* при паралельному програмуванні, оскільки можна графічно зобразити всі гілки і визначити, коли їх необхідно синхронізувати;
* для опису потоків подій у 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 надати 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 складаються зі станів діяльності і станів дії, переходів.

***Стан дії (action state)*** є спеціальним випадком стана 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 стана дії полягає в моделюванні одного кроку виконання алгоритму (процедури) або потоку керування. Кожна діаграма діяльності повинна мати єдиний початковий і єдиний кінцевий стани, як і діаграма стана. 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), який повинен бути унікальним у межах однієї діаграми діяльності. Ніяких додаткових або неявних обмежень при запису дій не накладається.

***Переходи***  ̶ при побудові діаграми діяльності використовуються тільки ***нетригерні переходи***. Нетригерні переходи ̶ це переходи, які спрацьовують одразу після завершення діяльності або виконання відповідної дії. Цей перехід переводить діяльність у 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 непозначеним. ***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 одині з них. У цьому випадку для кожного з таких переходів має бути явно записано сторожову умову в прямих дужках. Графічне розгалуження на діаграмі діяльності позначається невеликим ромбом, усередині якого немає ніякого тексту. В цей ромб може входити тільки одна стрілка від стана дії. Вихідних стрілок може бути дві або більше, але для кожної з них явно вказують відповідну сторожову умову у формі бульового виразу. Допускається використовувати замість сторожової умови слово «інакше», яке вказує на той перехід, який повинен спрацювати в разі невиконання сторожової умови розгалуження.

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

Надамо приклад застосування діаграми дії для бізнес-процесу фірми «РемПобТех», рис. 7.3.

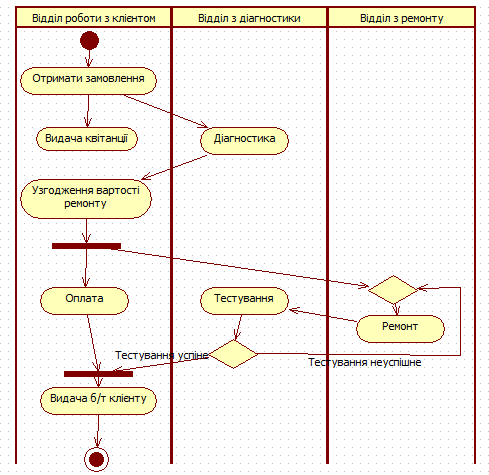


Рисунок 7.3 – Діаграма діяльносяті для бізнес процесу фірми «РемПобТех»

**Питання для самоконтролю**

1. В яких випадках застосовується діаграма стана?
2. Які елементи використовуються на діаграмах стана? Визначити, що таке «стан» і «перехід» та як вони позначаються .
3. Визначити тригерні та нетригерні переходи, єдиний та розгалужений перехід. Як позначаються ці переходи на діаграмах стана?
4. Що таке складений стан та підстан? Як підстан позначається на діаграмах?
5. В яких випадках застосовується діаграма діяльності?
6. Основні елементи діаграми діяльності та їх зображення.
7. Як позначаються паралельні обчислення на діаграмах діяльності?

**Завдання до лабораторної роботи** **№ 7**

1. Відповісти на питання для самоконтролю.
2. Створити діаграму стана для класу-транзакції.
3. Створити діаграму активності для класу-транзакції.
4. Оформити протокол лабораторної роботи.

**Література**

1. Ларман К. Применение UML 2.0 и шаблонов проектирования: практическое руководство. 3-е изд., Москва: ИД "Вильяме" , 2012. 673 c.
2. Петрик М.Р., Петрик О.Ю. Моделювання програмного забезпечення : науковометодичний посібник. Тернопіль : Вид-во ТНТУ імені Івана Палія, 2015. 200 с.
3. Алонцева Е. Н., Анохин А. Н., Саакян С. П. Структурное моделирование процессов и систем: учеб. Пособ. по курсу «CASE и CALS технология». Обнинск: ИАТЭ НИЯУ МИФИ, 2015. 72 c.
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. Марка Д.А., МакГоуэн К. 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**

1. АНАЛІЗ ВИМОГ ТА ПОБУДОВА ЦІЛЬОВОЇ МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ
   1. Короткі теоретичні відомості з аналізу вимог до АІС. . . . . . . . . . . . . . . . 3
   2. Аналіз предметної області: бізнес та користувацькі вимоги. . . . . . . . . . . 9
   3. Визначення цілей та варіатів використанн для користувачів АІС . . . . . 11
2. СТУКТУРНЕ МОДЕЛЮВАННЯ ПЗ. МЕТОДОЛОГІЇ ER. НОТАЦІЯ БАХМАНА
   1. Короткі теоретичні відомості за методологію ER. . . . . . . . . . . . . . . . . . 17
   2. Приклад створення інформаційної моделі ER для АІС «RepairTech» . . 22
3. СТУКТУРНЕ МОДЕЛЮВАННЯ ПЗ. МЕТОДОЛОГІЇ ФУНКЦІОНАЛЬНОГО МОДЕЛЮВАННЯ IDEF0 та DFD
   1. Короткі теоретичні відомості за методологію IDEF0. . . . . . . . . . . . . . 25
   2. Короткі теоретичні відомості за методологію DFD. . . . . . . . . . . . . . . . 32
   3. Приклад створення нотації Гейне-Сарсона для функції «Діагностика та тестування» АІС «RepairТеch». . . . . . . . . . . . . . . . . . . . 35
4. ПОБУДОВА ДІАГРАМ ВАРІАНТІВ ВИКОРИСТАННЯ (МОДЕЛЬ ПРЕЦЕДЕНТІВ)
   1. Короткі теоретичні відомості з методології

об’єктно-орієнтованого модлювання . . . . . . . . . . . . . . . . . . . . . . . . . . 37

* + 1. Принципи об'єкт-орієнтованої методології. . . . . . . . . . . . . . . . . . . 37
    2. Методологія RUP для розробки ПЗ у рамках об'єктно-орієнтованого підходу. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
  1. Діаграма UML варіантів використання (use case diagram). . . . . . . . . . . . 41
  2. Діаграма послідовності (sequence diagram). . . . . . . . . . . . . . . . . . . . . . . . 46

1. ПОБУДОВА ДІАГРАМИ КЛАСІВ ТА ПАКЕТІВ В UML. МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ
   1. Короткі теорітичні відомості з призначення та елементів діаграми класів UML. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
   2. Модель предметної області в межах методології UP. . . . . . . . . . . . . . . . 55
   3. Логічна архітектура. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
   4. Діаграми пакетів UML. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
   5. Шар предметної області або логіки додатку. . . . . . . . . . . . . . . . . . . . . . . 69
2. ПОБУДОВА ДІАГРАМ ВЗАЇМОДІЇ ДЛЯ ВИЯВЛЕННЯ РОЗПОДІЛУ ОБОВ’ЯЗКІВ МІЖ ОБ’ЄКТАМИ
   1. Принципи розподілу обов’язків між об’єктами програмної системи (шаблони GARSP). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
   2. Шаблони проектування GRASP. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
   3. Приклад розподілу обов’язків серед об’єктів програмної системи «RepairTech» прецедент «Оформлення замовлення». . . . . . . . . . . . . . . . 80
3. ПОБУДОВА ДІАГРАМ СТАНУ ТА ДІЯЛЬНОСТІ ЯК ІНСТРУМЕНТ АНАЛІЗУ СКЛАДНИХ ТРАНЗАКЦІЙ ТА ОПЕРАЦІЙ КЛАСІВ ПРИ ПРОЕКТУВАННІ СИСТЕМИ
   1. Використання діаграми стану (activity diagram) UML для аналізу послідовності дій. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84
   2. Використання діаграми діяльності (activity diagram) UML для аналізу послідовності дій**.** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

ЛІТЕРАТУРА . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91