Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

**КУРСОВА РОБОТА**

з дисципліни «Проектування інформаційних систем»

Тема «Проектування програмного забезпечення для автоматизації вирішення задач продажу транспортних засобів автосалону»

Виконав: студент 3 курсу

групи КС-32

Безрук Юрій Русланович

Перевірив:

доц. Гамзаєв Р.О.

Харків – 2020

**ЗМІСТ**

[ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ 4](#_Toc59655499)

[ВСТУП 5](#_Toc59655500)

[РОЗДІЛ 1 ОГЛЯД ОСНОВНИХ ПРИНЦИПІВ СУЧАНОЇ ПРОГРАМНОЇ ІНЖЕНЕРІЇ. ПОСТАНОВКА ЗАДАЧІ. 6](#_Toc59655501)

[1.1 Дослідження основних положень сучасної програмної інженерії 6](#_Toc59655502)

[1.1.1 Ядро знань SWEBOK 6](#_Toc59655503)

[1.1.2 Стандарти ISO / IEC 12207, ISO/IEC 9126 7](#_Toc59655504)

[1.1.3 Основи методології RUP 12](#_Toc59655505)

[1.2 Постановка задачі курсової роботи 14](#_Toc59655506)

[РОЗДІЛ 2 РЕАЛІЗАЦІЯ ЕТАПУ RUP/INCEPTION 15](#_Toc59655507)

[2.1 Розробка повної специфікації системних вимог. Опис розгорнутого сценарію за стандартом RUP 15](#_Toc59655508)

[2.2 Розробка UML-діаграм концептуального (архітектурного) рівня проектування ПЗ 19](#_Toc59655509)

[2.2.1 Use Case Diagram 19](#_Toc59655510)

[2.2.2 Robustness Diagrams 20](#_Toc59655511)

[2.2.3 Sequence Diagrams 22](#_Toc59655512)

[РОЗДІЛ 3 РЕАЛІЗАЦІЯ ЕТАПУ RUP/ELABORATION 24](#_Toc59655513)

[3.1 Проектування компонентних програмних рішень 24](#_Toc59655514)

[3.2 Розробка статичних UML-діаграм для логічного проектування компонентних програмних рішень. 24](#_Toc59655515)

[3.2.1 Class Diagram 24](#_Toc59655516)

[3.2.2 Package Diagram 25](#_Toc59655517)

[3.3 Розробка динамічних UML-діаграм для логічного проектування компонентних програмних рішень. 25](#_Toc59655518)

[3.3.1 Activity Diagram 25](#_Toc59655519)

[3.4 Розробка UML-діаграм для фізичного проектування компонентних програмних рішень. 26](#_Toc59655520)

[3.4.1 Component Diagram 26](#_Toc59655521)

[3.4.2 Deployment Diagram 27](#_Toc59655522)

[РОЗДІЛ 4 ФОРМУВАННЯ ПОВНОГО КОМПЛЕКТУ ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ЦІЛЬОВОЇ ВЕРСІЇ СИСТЕМИ 28](#_Toc59655523)

[4.1 Застосування метрик якості UML-діаграм 28](#_Toc59655524)

[4.2 Визначення специфікацій необхідних ресурсів, апаратно-програмної конфігурації (операційного середовища) і програмного інструментарію для реалізації проекту на подальших RUP-етапах 29](#_Toc59655525)

[ВИСНОВКИ 30](#_Toc59655526)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 31](#_Toc59655527)

# ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

IT (Information Technologies) – Інформаційні технології.

MVC (Model-View-Controller) – шаблон проектування Модель-Представлення-Контролер.

RUP (Rational Unified Process) – ітеративний процес розробки програмного забезпечення створений Rational Software — підрозділом IBM з 2003.

UML (Unified Modeling Language) – Уніфікована мова моделювання.

БД – База даних.

ПЗ – Програмне забезпечення.

ПС – Програмна система.

ЖЦ – Життєвий цикл.

# ВСТУП

Сучасні компанії і організації функціонують в умовах великого обсягу постійно змінюється інформації, яку необхідно оперативно аналізувати і приймати правильні рішення, бурхливо розвивається обчислювальна техніка та інформаційні технології. Важко знайти зараз компанію, не що займається розвитком інформаційних технологій. Сучасні керівники фірм повністю віддають собі звіт в тому, що в даний час успішність і прибутковість компанії повністю залежать в тому числі, і від рівня розвитку IT-технологій, швидкості і якості обробки інформації, обґрунтованості та виваженості прийнятих рішень. Потрібна постійна серйозна робота не тільки IT-фахівців, а й топ-менеджерів по погодженням або точніше - синхронізації всіх зусиль зі стратегічного розвитку компанії і її інформаційних систем [1]. Великою помилкою є позиція керівників компаній, які не використовують масштабне планування і проектування вимог і специфікацій ПЗ.

Тож у цій роботі буде розглянуто процес проектування и написання текстових вимог до програмного забезпечення на прикладі обраної предметної області, створення відповідних діаграм для візуалізації процесу і розгляд основних понять, пов’язаних з специфікаціями розробки системи.

# РОЗДІЛ 1 ОГЛЯД ОСНОВНИХ ПРИНЦИПІВ СУЧАНОЇ ПРОГРАМНОЇ ІНЖЕНЕРІЇ. ПОСТАНОВКА ЗАДАЧІ.

## 1.1 Дослідження основних положень сучасної програмної інженерії

### 1.1.1 Ядро знань SWEBOK

Для перетворення програмної інженерії в спеціальність світова комп'ютерна громадськість створила професійні комітети, які регламентують аспекти процесу програмування: ядро знань SWEBOK, етичний кодекс програміста, навчальні курси з підготовки фахівців в області програмної інженерії, навчання спеціальності і сертифікація спеціалістів. Ядро знань SWEBOK - нормативний документ, розроблений IEEE, є основоположним науково-технічним документом, який відображає думку фахівців в області програмної інженерії та узгоджується з сучасними регламентованими процесами життєвого циклу ПО (ЖЦ ПЗ), стандарту ISO / IEC 12207. У цьому ядрі знань міститься опис десяти областей, кожна з яких представлена загальною схемою опису, що включає визначення понятійного апарату, методів і засобів, а також інструментів підтримки інженерної діяльності. У кожній з цих десяти областей описується певний запас знань, який повинен використовуватися у відповідних процесах життєвого циклу розробки програмних систем (ПС).

Всі області ядра знань діляться на п'ять основних областей, пов'язаних з проектуванням ПС і, п'ять додаткових областей, пов'язаних з інженерією управління проектуванням ПС (конфігурацією, проектами, якістю).

Основні галузі знань SWEBOK:

1. Інженерія вимог, що включає наступні ключові поняття, підходи та методи: виявлення вимог, аналіз вимог, специфікація вимог, перевірка вимог, управління вимогами.

2. Проектування ПЗ, що включає: ключові поняття, підходи, методи, базові концепції архітектури ПС, аналіз якості, нотації, стратегії і методи проектування.

3. Конструювання ПЗ, яке включає наступні ключові поняття: зниження складності, відхилення від стилю, перевірки, використання зовнішніх стандартів.

4. Тестування ПЗ, яке включає: концепції, рівні тестування, техніки, метрики, управління тестуванням.

5. Супровід ПЗ, що включає: концепції, супровід, ключові питання, специфікації вимог, процес супроводу.

Організаційні області знань SWEBOK:

6. Управління конфігурацією, що включає: управління процесами, ідентифікацію, контроль, облік статусу, аудит, управління версіями.

7. Управління проектами, що включає: організацію управління, управління процесами, управління проектом, інженерію планування, інженерію вимірювання, інженерію вартості.

8. Процес інженерії ПС, що включає: концепції, інфраструктуру, визначення, оцінку, кількісний аналіз, процес змін.

9. Методи і засоби програмної інженерії, що включають: методи об'єктного, структурного і кількісного підходів, інструменти інженерії (SADT, BPwin, ERwin, Rational Rose, Oracle і т.п.).

10. Інженерія якості ПС, що включає: концепції якості, діяльності та техніки, гарантії якості ПС, планування якості, вимірювання якості [2].

### 1.1.2 Стандарти ISO / IEC 12207, ISO/IEC 9126

ISO / IEC 12207: 2008 Systems and software engineering - Software life cycle processes - стандарт ISO, що описує процеси життєвого циклу програмного забезпечення.

Стандарт розроблений підкомітетом ПК 7 «Системна і програмна інженерія» (англ. SC 7 System and Software Engineering) Спільного технічного комітету №1 ISO / IEC «Інформаційні технології» (англ. ISO / IEC JTC 1 Information Technology).

Даний стандарт, використовуючи усталену термінологію, встановлює загальну структуру процесів життєвого циклу програмних засобів, на яку можна орієнтуватися в програмної індустрії. Стандарт визначає процеси, види діяльності та завдання, які використовуються при придбанні програмного продукту або послуги, а також при поставці, розробці, застосуванні за призначенням, супроводі та припинення застосування програмних продуктів.

У стандарті ISO / IEC 12207 описані п'ять основних процесів життєвого циклу програмного забезпечення:

1) процес придбання визначає дії підприємства - покупця інформаційної системи, програмного продукту або служби програмного забезпечення;

2) процес поставки визначає дії підприємства-постачальника з постачання покупця інформаційною системою, програмним продуктом або служби програмного забезпечення;

3) процес розробки визначає дії підприємства-розробника, який розробляє принципи побудови програмного вироби і власне програмний продукт;

4) процес функціонування визначає дії підприємства-оператора, обслуговуючого систему в цілому. Сюди входять консультація користувачів, отримання зворотного зв'язку і т.д .;

5) процес супроводу визначає дії персоналу, який забезпечує супровід програмного продукту, тобто управління модифікацією програмного продукту, підтримку поточного стану та функціональної придатності, установку і видалення.

Крім п'яти основних процесів, ISO / IEC 12207 обумовлює вісім допоміжних процесів, які є невід'ємною частиною всього життєвого циклу системи:

1) процес вирішення проблем;

2) процес документування;

3) процес управління конфігурацією;

4) процес забезпечення якості;

5) процес верифікації;

6) процес атестації;

7) процес спільної оцінки;

8) процес аудиту.

У стандарті ISO / IEC 12207 також визначаються чотири організаційних процесу:

1) процес управління;

2) процес створення інфраструктури;

3) процес удосконалення;

4) процес навчання.

У стандарті ISO / IEC 12207 є додатковий процес, що дозволяє адаптувати стандарт до умов конкретного проекту.

ISO / IEC 9126 - міжнародний стандарт, який визначає оціночні характеристики якості програмного забезпечення. Складається з 4 частин, що описують такі аспекти: модель якості; зовнішні метрики якості; внутрішні метрики якості; метрики якості у використанні.

Модель якості, встановлена ​​в першій частині стандарту ISO 9126-1, класифікує якість ПО в 6-ти структурних наборах характеристик, які в свою чергу деталізовані під-характеристиками (субхарактеристиками), такими як:

Функціональність – набір атрибутів характеризує, відповідність функціональних можливостей ПО набору необхідної користувачем функціональності. Деталізується наступними субхарактеристиками:

1. Придатністю для застосування
2. Коректністю (правильністю, точністю)
3. Здатністю до взаємодії (зокрема мережному)
4. Захищеністю

Надійність - Набір атрибутів, що відносяться до здатності ПО зберігати свій рівень якості функціонування в встановлених умовах за певний період часу. Деталізується наступними субхарактеристиками:

1. Рівнем завершеності (відсутності помилок)
2. Стійкістю до дефектів
3. Відновлюваністю
4. Доступністю
5. Готовністю

Практичність (застосовність) - Набір атрибутів, які відносяться до обсягу робіт, необхідних для виконання і індивідуальної оцінки такого виконання певних або передбачуваним колом користувачів. Деталізується наступними субхарактеристиками:

1. Зрозумілістю
2. Простотою використання
3. Вивченням
4. Привабливістю

Ефективність - Набір атрибутів, що відносяться до співвідношення між рівнем якості функціонування ПЗ і обсягом використовуваних ресурсів при встановлених умовах. Деталізується наступними субхарактеристиками:

1. Тимчасовою ефективністю
2. Використовуваністю ресурсів

Супроводженість - Набір атрибутів, які відносяться до обсягу робіт, необхідних для проведення конкретних змін (модифікацій). Деталізується наступними субхарактеристиками:

1. Зручністю для аналізу
2. Змінністю
3. Стабільністю
4. Тестованістю

Мобільність - Набір атрибутів, що відносяться до здатності ПО бути перенесеним з одного оточення в інше. Деталізується наступними субхарактеристиками:

1. Адаптованістю
2. Простотою установки (інсталяції)
3. Співіснуванням (відповідністю)
4. Замінюваністю

Субхарактеристика Відповідність не приведена в вищеописаному списку, але вона належить всім характеристикам. Ця характеристика повинна відображати відсутність протиріч з іншими стандартами або характеристиками. Наприклад відповідність надійності і практичності.

Кожна якісна субхарактеристика (наприклад адаптованість) надалі поділяється на атрибути. Атрибут - це сутність, яка може бути перевірена або виміряна в програмному продукті. Атрибути не визначені в стандарті через їх різноманітності в різних програмних продуктах.

У стандарті виділено модель характеристик якості у використанні. Основними характеристиками якості програмних засобів (далі ПС) у використанні рекомендуються:

Системна ефективність - 'Застосування програмного продукту за призначенням'.

Продуктивність - 'Продуктивність при вирішенні основних завдань ПС, що досягається при реально обмежених ресурсах в конкретній зовнішньому середовищі застосування'.

Безпека - 'Надійність функціонування комплексу програм і можливий ризик від його застосування для людей, бізнесу і зовнішнього середовища'.

Задоволення вимог і витрат користувачів відповідно до цілей застосування ПС.

Друга і третя частини стандарту ISO 9126-2,3 присвячені формалізації відповідно зовнішніх і внутрішніх метрик характеристик якості складних ПС. У ній викладені зміст і загальні рекомендації по використанню відповідних метрик і взаємозв'язків між типами метрик.

Четверта частина стандарту ISO 9126-4 призначена для покупців, постачальників, розробників, які супроводжують, користувачів і менеджерів якості ПС. У ній повторена концепція трьох типів метрик, а також анотовані рекомендовані види вимірювань характеристик ПС.

### 1.1.3 Основи методології RUP

RUP поєднує в собі найбільш позитивні риси інших методологій проектування, зокрема таких як каскадна, спіральна та інкрементна моделі життєвого циклу розробки ПЗ [3].

Основним принципом RUP є принцип ітеративної розробки в рамках якої розробка ведеться у вигляді короткочасних міні-проектів фіксованої тривалості (наприклад, по 3-4 тижні), які називаються ітераціями. Кожна ітерація складається із власних фаз аналізу вимог, проектування, реалізації, тестування, інтеграції та створенням робочої версії програмної системи (ПС).

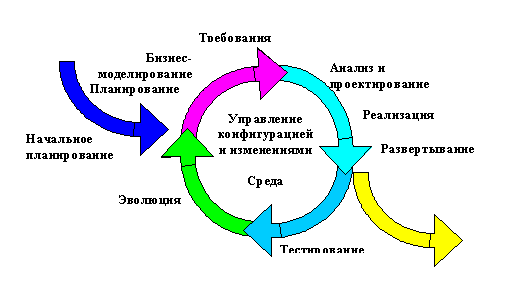


Рисунок 1 - Фази ітеративного процесу RUP

Такий ітеративний цикл базується на постійному розширенні і доповненні проекту ПС в процесі декількох ітерацій з періодичним зворотним зв’язком і адаптацією додаткових компонентів до існуючого ядра ПС, що розробляється.

Повний життєвий цикл розробки ПЗ складається з 4 етапів:

1.Початкова стадія (Inception)

У фазі початковій стадії:

Формуються бачення і межі проекту.

Створюється економічне обґрунтування (business case).

Визначаються основні вимоги, обмеження і ключова функціональність продукту.

Створюється базова версія моделі прецедентів.

Оцінюються ризики.

При завершенні початкової фази оцінюється досягнення етапу життєвого циклу мети, яке передбачає угоду зацікавлених сторін про продовження проекту.

2. Уточнення (Elaboration)

У фазі «Уточнення» проводиться аналіз предметної області та побудова виконуваної архітектури. Це включає в себе:

Документування вимог (включаючи детальний опис для більшості прецедентів).

Спроектовану, реалізовану і відтестовану виконувану архітектуру.

Оновлене економічне обґрунтування і більш точні оцінки термінів і вартості.

Знижені основні ризики.

Успішне виконання фази розробки означає досягнення етапу життєвого циклу архітектури

3. Побудова (Construction)

У фазі «Побудова» відбувається реалізація більшої частини функціональності продукту. Фаза Побудова завершується першим зовнішнім релізом системи і віхою початкової функціональної готовності.

4. Впровадження (Transition)

У фазі «Впровадження» створюється фінальна версія продукту і передається від розробника до замовника. Це включає в себе програму бета-тестування, навчання користувачів, а також визначення якості продукту. У разі, якщо якість не відповідає очікуванням користувачів або критеріям, встановленим у фазі Початок, фаза Впровадження повторюється знову. Виконання всіх цілей означає досягнення віхи готового продукту і завершення повного циклу розробки.

## 1.2 Постановка задачі курсової роботи

Дана робота спрямована на оволодіння навичками концептуального проектування програмних систем, детального дослідження предметної області та побудови відповідних діаграм.

У якості прикладу розглядається предметна область «автосалон». Тобто ідея полягає в створенні відповідного програмного забезпечення для обслуговування процесу продажу транспортного засобу робітником сервісу клієнтові. Отже, необхідно забезпечити функціонал реєстрації клієнта, зв’язок з базою даних для оновлення реєстру, пошук потрібного транспортного засобу за критеріями, покупку, яка є складним функціоналом-транзакцією, що складається з декількох кроків, таких як утримання і перечислення коштів та отримання даних на автомобіль, оновлення інформації в базі даних згідно реєстру додатка тощо.

Отже, об’єктом даної роботи є програмна система створена згідно специфікацій.

Предметом курсової роботи є програмне забезпечення для продажу транспортних засобів.

Виходячи з вищевикладеного, метою роботи є розробка текстових специфікацій, прецедентів та діаграм проектування для обраної предметної області згідно з вимогами вказаними в п. 1.1 цього розділу.

# РОЗДІЛ 2 РЕАЛІЗАЦІЯ ЕТАПУ RUP/INCEPTION

## 2.1 Розробка повної специфікації системних вимог. Опис розгорнутого сценарію за стандартом RUP

**1) Предметна область розробки ПЗ:** «Програмна система для продажу транспортних засобів в автосалоні».

**2) Перелік основних прецедентів використання цієї ПС:**

2.1 Продаж транспортного засобу клієнту автосалону користувачем сервісу.

2.2 Надання інформації користувачеві сервісу щодо транспортного засобу.

2.3 Внесення змін в базу даних.

**3) Розробка розгорнутого сценарію для прецеденту (2.1)**

(3.1) Зацікавлені особи прецеденту та їх вимоги:

* Клієнт: хоче швидко придбати транспортний засіб згідно його потреб.
* Робітник автосалону (далі можливо: користувач системи): повинен швидко і безпомилково обробити персональні дані клієнта, зареєструвати його в системі, виконати за допомогою ПЗ пошук потрібного транспортного засобу в базі даних та ініціювати продаж.

(3.2) Користувач відповідного ПЗ, тобто основний актор цього прецеденту – це представник автосалону, який проводить реєстрацію клієнтів та консультує їх в питаннях придбання транспортних засобів.

(3.3) Передумови прецеденту ( pre - conditions):

* ПС має бути активною.
* Представник автосалону має пройти процедуру аутентифікації в системі.

(3.4) Основний успішний сценарій:

1. Клієнт висловлює свої потреби користувачеві системи.

2. Робітник сервісу заносить дані клієнта в ПС.

3. Система завантажує дані з бази даних.

4. Система виконує пошук потрібного транспортного засобу.

5. Система виконує продаж транспортного засобу.

6. Клієнт отримує транспортний засіб.

(3.5) Розширення основного сценарію або альтернативні потоки.

Відсутність потрібного транспортного засобу у базі даних:

1. ПС повідомляє користувача системи про помилку.
2. Користувач системи повідомляє про відсутність товару клієнта.
3. Клієнт формулює нові побажання або залишає автосалон

Помилка транзакції:

1. ПС виконує відкат змін: повертає клієнту гроші (якщо були передані), повертає дані авто до сервісу (якщо були отримані).

2. ПС повідомляє робітника сервісу про помилку транзакції.

3. Робітник сервісу виконує діагностику і усуває проблему.

4. Робітник сервісу повторно ініціює транзакцію.

(3.6) Пост-умови прецеденту ( post - conditions):

* Клієнт отримує шуканий транспортний засіб.
* Сервіс отримує гроші за транспортний засіб.
* Дані щодо продажу записуються в реєстр
* Зміни в реєстрі записуються до бази даних.

(3.7) Спеціальні вимоги прецеденту:

* Необхідно забезпечити надійність та стійкість ПС, оскільки нею користуватимуться одночасно декілька представників сервісу.
* Необхідно забезпечити 100% надійність обробки всіх транзакцій.
* Необхідно забезпечити можливість локалізації інтерфейсу ПС.

(3.8) Список необхідних технологій та додаткових пристроїв:

* Необхідно забезпечити наявність монітору для користувача ПС.
* Необхідно забезпечити закритий доступ до ПС (вхід до системи лише за допомогою відповідного особистого ключа користувача).
* Необхідно забезпечити безперервний доступ до мережі Internet.

**3) Розробка розгорнутого сценарію для прецеденту (2.2)**

(3.1) Зацікавлені особи прецеденту та їх вимоги:

* Робітник автосалону: хоче оперативно отримати інформацію про шуканий транспортний засіб.

(3.2) Користувач відповідного ПЗ, тобто основний актор цього прецеденту — це представник автосалону, який проводить реєстрацію клієнтів та консультує їх в питаннях придбання транспортних засобів.

(3.3) Передумови прецеденту (pre - conditions):

* ПС має бути активною.
* Представник автосалону має пройти процедуру аутентифікації в системі.

(3.4) Основний успішний сценарій:

1. Робітник сервісу вносить необхідні для пошуку дані в систему

2. Система завантажує реєстр з бази даних.

3. Система виконує пошук потрібного транспортного засобу.

4. Робітник отримує потрібну інформацію.

(3.5) Розширення основного сценарію або альтернативні потоки.

Відсутність потрібного транспортного засобу у базі даних:

1. ПС повідомляє користувача системи про помилку.

2. Користувач системи формулює нові побажання або завершує спробу.

(3.6) Пост-умови прецеденту ( post - conditions):

* Робітник сервісу отримує шукану інформацію про транспортний засіб.

(3.7) Спеціальні вимоги прецеденту:

* Необхідно забезпечити надійність та стійкість ПС, оскільки нею користуватимуться одночасно декілька представників сервісу.
* Необхідно забезпечити можливість локалізації інтерфейсу ПС.

(3.8) Список необхідних технологій та додаткових пристроїв: Аналогічно прецеденту 2.1.

**3) Розробка розгорнутого сценарію для прецеденту (2.3)**

(3.1) Зацікавлені особи прецеденту та ïx вимоги:

* Робітник автосалону: повинен чітко і безпомилково внести нові дані стосовно реєстру в базу даних

(3.2) Користувач відповідного ПЗ, тобто основний актор цього прецеденту — це представник автосалону, який проводить реєстрацію клієнтів та консультує їх в питаннях придбання транспортних засобів.

(3.3) Передумови прецеденту ( pre - conditions):

* ПС має бути активною.
* Представник автосалону має пройти процедуру аутентифікації в системі.
* БД має бути доступна

(3.4) Основний успішний сценарій:

1. Система встановлює зв’язок з базою даних.

2. Робітник сервісу вносить дані до реєстру.

3. Система вивантажує реєстр у базу даних.

4. Робітник отримує інформацію про успіх процедури і завершує сеанс.

(3.5) Розширення основного сценарію або альтернативні потоки.

Помилка з’єднання з базою:

Робітник отримує повідомлення.

Робітник проводить діагностику мережі.

Робітник проводить повторне з’єднання.

Помилка вивантаження даних:

1. Робітник отримує повідомлення.

2. Робітник проводить відкат змін

3. Робітник проводить діагностику

4. Робітник усуває проблему

5. Робітник проводить повторний виклик операції

(3.6) Пост-умови прецеденту (post - conditions): Відсутні

(3.7) Спеціальні вимоги прецеденту: Аналогічно прецеденту 2.2.

(3.8) Список необхідних технологій та додаткових пристроїв: Аналогічно прецедентам 2.1 і 2.2.

## 2.2 Розробка UML-діаграм концептуального (архітектурного) рівня проектування ПЗ

На основі аналізу та текстової специфікації вимог створюємо відповідні діаграми для візуалізації ключових моментів проектування системи на концептуальному рівні. Реалізуємо ключові прецеденти специфікацій.

### 2.2.1 Use Case Diagram

Діаграма прецедентів (англ. Use Case Diagram) відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів, яка дозволяє описати систему на концептуальному рівні.

Формулюємо на ній основних акторів (сутностей, що виконують певні дії) та прецедентів (відповідних дій, операцій, побажань) ПС згідно розроблених специфікацій. Окремо виділяємо частину прецедентів яка відноситься безпосередньо до додатку, тобто реалізована в системі.

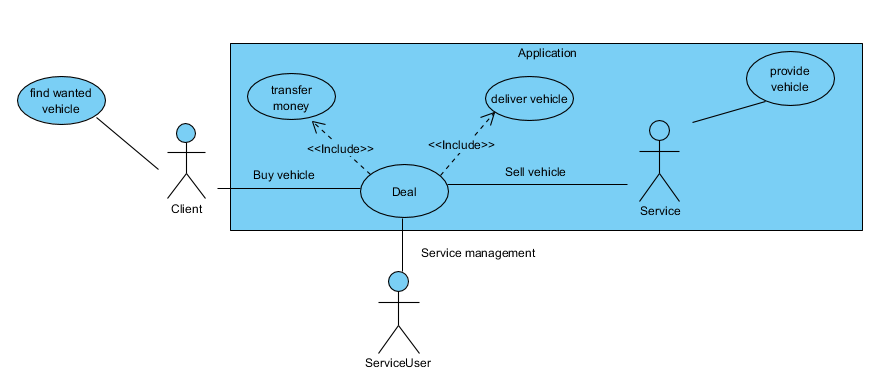


Рисунок 2 – Діаграма прецедентів

### 2.2.2 Robustness Diagrams

Прецеденти не можуть бути безпосередньо трансформовані в відповідні програмні об’єкти (класи). Для цього має бути використано певний проміжний рівень опису прецедентів, який дозволяє врахувати деякі особливості наступної програмної реалізації, наприклад, необхідність інтерфейсу користувача або наявність деяких функцій обробки бізнес-логіки. Одним з таких засобів подальшої деталізації моделі прецедентів є діаграма стійкості(ці діаграми належать до процесу проектування ПЗ за стандартом ICONIX [3]).

Діаграма стійкості розробляється на базі шаблону проектування MVC (Model-View-Controller) [2-3]. Тобто в системі, що розробляється, мають бути наявні три типи програмних об’єктів:

Model – це об’єкти, які моделюють дані предметної області;

View – це об’єкти, які реалізують відображення даних із моделі;

Controller – це об’єкти, які обробляють дані моделі для подальшого їх відображення.

Основна ідея шаблону MVC полягає у відокремленні даних від їх відображення. Таким чином, якщо в процесі розробки виникне потреба у змінені моделі даних, це ніяк не впливає на відображення (не треба буде нічого змінювати у компонентах View).

Згідно опису першого прецеденту, нам необхідні форми для внесення даних пошуку і придбання (View), відображення даних реєстру в програмі (Model), і додаткові операції над даними моделі, що будуть обробляти їх і надавати зовнішньому інтерфейсу (Controller).

Функціональність другого прецеденту реалізується за рахунок контролера і форми Search першої діаграми стійкості, тому будувати її для нього не має сенсу.

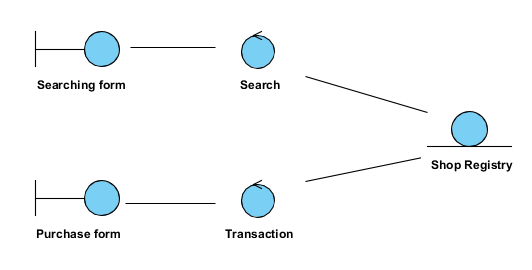


Рисунок 3 - Діаграма стійкості першого та другого прецедентів

Функціональність третього прецеденту в моделі MVC описана на наступній діаграмі стійкості:

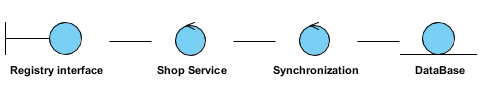


Рисунок 4 - Діаграма стійкості третього прецеденту

### 2.2.3 Sequence Diagrams

За проведеним текстовим аналізом вимог та діаграмою класів будуємо для основних прецедентів діаграми послідовностей, на яких для набору об'єктів на єдиній тимчасової осі показаний життєвий цикл об'єкта і взаємодія акторів інформаційної системи в рамках прецеденту. з описом ліній життя кожного актору чи частини системи і порядком викликів та відношень один до одного згідно їх основних сценаріїв.

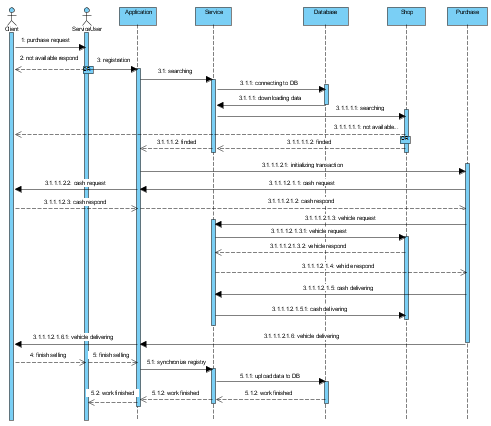


Рисунок 5 – Діаграма послідовності прецеденту придбання транспортного засобу

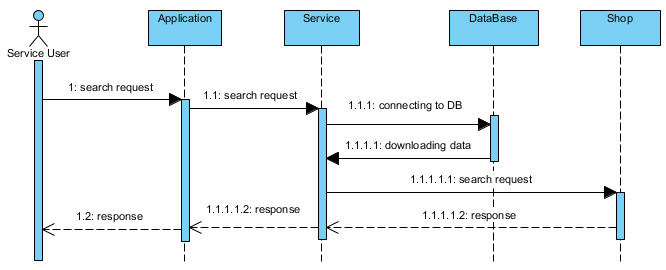


Рисунок 6 – Діаграма послідовності прецеденту отримання інформації

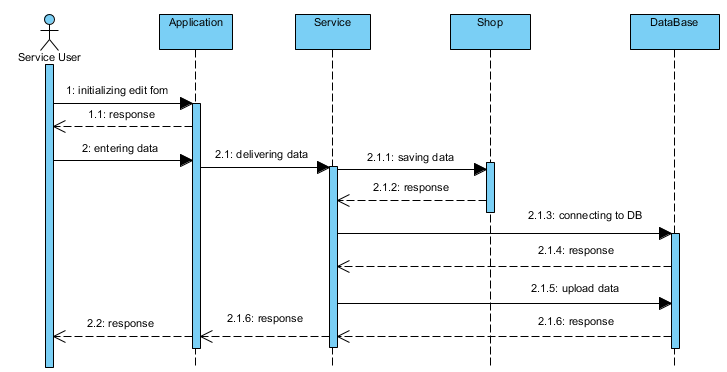


Рисунок 7 – Діаграма послідовності прецеденту внесення змін в реєстр

# РОЗДІЛ 3 РЕАЛІЗАЦІЯ ЕТАПУ RUP/ELABORATION

## 3.1 Проектування компонентних програмних рішень

Систему вирішено було ускладнити з точки зору компонентних рішень. Так, проведено класифікацію транспортних засобів. На етапі планування встановлені основні сутності, що мають бути реалізовані – як композиційні (на кшталт типів палива, моделей, типу кузову та ін.), так і асоційовані і розширювані. На відповідних діаграмах проектування подана інформація про можливу початкову версію програмного продукту.

## 3.2 Розробка статичних UML-діаграм для логічного проектування компонентних програмних рішень.

### 3.2.1 Class Diagram

Переходимо безпосередньо на рівень реалізації продукту – проектуємо необхідні сутності програми використовуючи діаграму класів, яка демонструє загальну структуру ієрархії класів системи, їх кооперацій, атрибутів, методів, інтерфейсів і взаємозв'язків між ними.

Повну структуру класів зв'язків між ними зображено на наступній схемі:

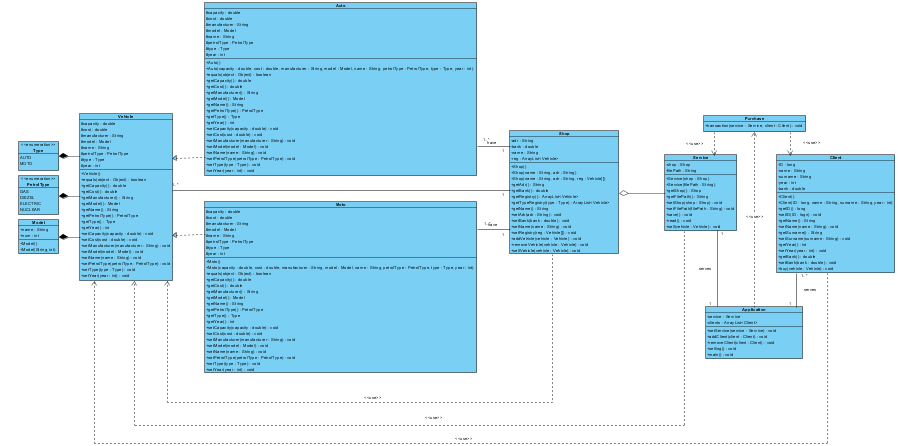


Рисунок 8 - Діаграма класів

### 3.2.2 Package Diagram

У процесі об’єктно-орієнтованої розробки ПС проводиться розподіл програмного рішення на окремі підсистеми, що уможливлює розподіл роботи між членами команди розробників і забезпечує подальше багаторазове використання програмного коду. Засобом моделювання в UML, який дає змогу агрегувати окремі компоненти в окремі групи (підсистеми), залежно від їх логічного призначення в системі, є поняття пакету (package), який вміщує в собі класи, біни та модулі проекту.

Обрану предметну область було вирішено розбити на такі пакети:

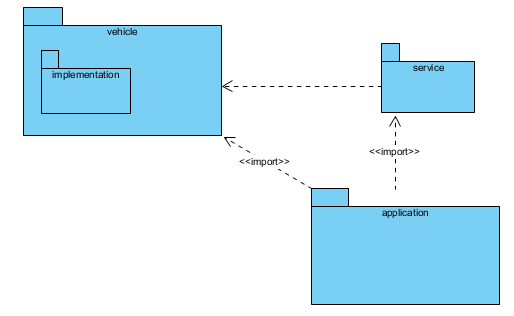


Рисунок 9 – Діаграма пакетів

## 3.3 Розробка динамічних UML-діаграм для логічного проектування компонентних програмних рішень.

Далі розгортаємо систему у динаміці: необхідно спланувати взаємодію вузлів, показати їх активності та виклики, що їх зв’язують. Використовуємо для цього діаграму діяльності.

### 3.3.1 Activity Diagram

Діаграми діяльності описують, як діяльність координується з метою надання послуги, яка може бути на різних рівнях абстракції. Як правило, подія повинна бути досягнута за допомогою деяких операцій, зокрема, коли операція призначена для досягнення ряду різних речей, які потребують координації, або як події в одному випадку використання співвідносяться між собою, зокрема, випадки використання, коли дії може перекриватися і вимагати координації. Це також підходить для моделювання того, як збірка випадків використання координується для представлення робочих процесів.

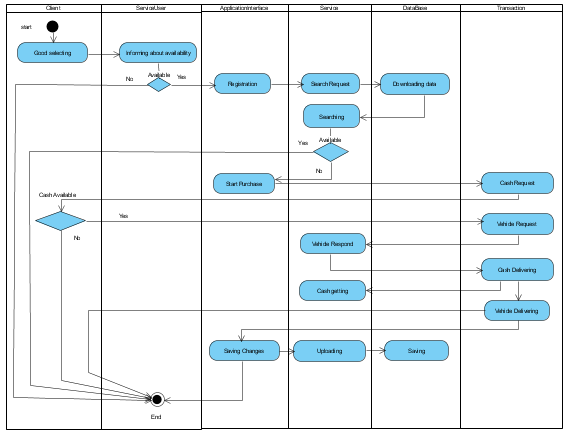


Рисунок 10 – діаграма діяльності

## 3.4 Розробка UML-діаграм для фізичного проектування компонентних програмних рішень.

### 3.4.1 Component Diagram

Діаграма компонентів – це діаграма, яка показує розбиття програмної системи на структурні компоненти та зв'язки (залежності) між компонентами, В якості фізичних компонентів виступають файли, бібліотеки, модулі, виконувані файли, пакети і т.д.

На наступній діаграмі зображена така взаємодія модулів програми через відповідні інтерфейси взаємодії і порти.

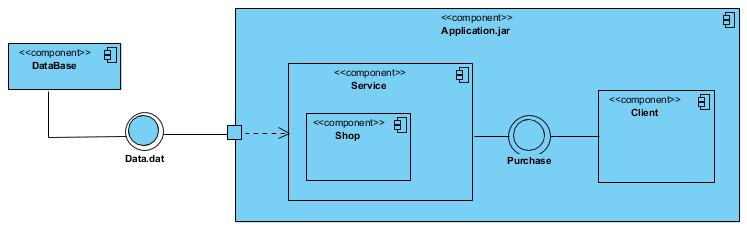


Рисунок 11 – Діаграма компонентів

### 3.4.2 Deployment Diagram

Готові програмні модулі та компоненти з артефактами слід правильно фізично розподілити на виконання на різних пристроях та застосунках.

Реалізовану систему продажу можна буде розгорнути на персональному комп’ютері, що буде встановлено в автосалоні. Додаткового використання отримує сервер з базою даних. Також можливе інтернет-з’єднання кишенькових персональних комп’ютерів з онлайн доступом до використання ресурсів системи.

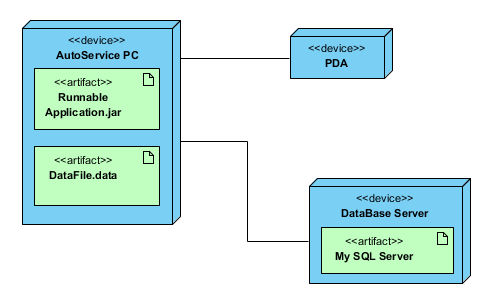


Рисунок 12 – Діаграма розгортання

# РОЗДІЛ 4 ФОРМУВАННЯ ПОВНОГО КОМПЛЕКТУ ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ЦІЛЬОВОЇ ВЕРСІЇ СИСТЕМИ

## 4.1 Застосування метрик якості UML-діаграм

У якості ефективної оцінки проекту в метриках UML-діаграм візьмемо за основу метрики оцінки якості об’єктно-орієнтованого підходу, оцінивши за допомогою них компоненти програми та діаграму класів в цілому.

Крім таких очевидних метрик, як загальне число класів, методів, атрибутів, середніх показників числа методів і атрибутів на клас та ін., застосовуються набагато більш комплексні метрики, за допомогою яких можна судити не тільки про обсяг вихідного коду проекту, але і про його складності, як, відповідно основним принципам ОО парадигми і т. д.

Висота дерева спадкування dit (Depth of Inheritance Tree)

DIT визначається як максимальна довжина шляху від листа до кореня дерева успадкування класів.

Відповідно, для окремого класу DIT, це довжина максимального шляху від даного класу до кореневого класу в ієрархії класів.

У міру зростання DIT ймовірно, що класи нижнього рівня будуть наслідувати багато методів. Це призводить до труднощів в прогнозі поведінки класу. Висока ієрархія класів (велике значення DIT) призводить до більшої складності проекту, так як означає залучення більшої кількості методів і класів.

В нашому випадку метрика DIT згідно діаграми класів становить 2.

Також вкажемо як метрику коцентрацію абстракцій (data abstraction coupling - DAC) – кількість атрибутів класу, «типом яких є інший клас». Для транспортного засобу, відповідно, 3, для Shop – 2, а для Application – 3. В середньому DAC для нашого ПЗ становить 2,(6).

Зважена насиченість класу 2 (WMPC2) – міра складності класу, заснована на тому, що клас з більшою кількістю методів і метод з більшою кількістю параметрів є більш складними (при обчисленні метрики батьківські класи не враховуються). Згідно з WMPC2, найскладнішими моделями є класи реалізації транспортного засобу, (авто, мото..) з кількістю методів 19 і максимальною кількістю атрибутів методу 10.

## 4.2 Визначення специфікацій необхідних ресурсів, апаратно- програмної конфігурації (операційного середовища) і програмного інструментарію для реалізації проекту на подальших RUP-етапах

У подальшій фазі побудови програмної системи потрібно розробити більшу частину функціоналу, а також зробити можливість зручного та швидкого релізу системи для дрібних виправлень і додання функціоналу. Для цього буде використана система CI/CD ( з англ. Continuous Integration – безперервна інтеграція, Continuous delivery – безперервна доставка) GitLab, яка поєднує ці два підходи, дозволяючи забезпечити послідовний і автоматизований спосіб збирання, упаковки та тестування додатків, зі сторони CI, а також, відразу після закінчення етапу CI розпочинається CD, що автоматизує розгортання додатків в різні оточення.

У фазі впровадження буде створено фінальну версію продукту, яка вже буде встановлена в програмне оточення ПК та PDA, що належать автосалону, а також буде проведено відповідне навчання працівників і через деякий час отримання зворотного зв’язку від них разом з звітом роботи програми, який буде зібраний через автоматичні форми. Це буде зроблено для переглядання якості продукту, а також для відповідності критеріям, заданим на початку розробки. В тому випадку, коли продукт буде не відповідати деяким вимогам, або ж буде не зовсім зручним для користувачів (продавці автосалону), будуть зроблені відповідні виправлення, і після цього наново запущена фаза впровадження.

# ВИСНОВКИ

Таким чином, під час виконання курсової роботи було досліджено можливості і засоби проектування програмних систем та програмного забезпечення, розробки текстових специфікацій вимог за методикою RUP та візуалізація різних фаз і підходів проекту за допомогою різноманітних UML-діаграм з відповідних груп діаграм.

В ході виконання роботи були розроблені вимоги до програмної системи за обраною предметною областю «автосалон». За цими специфікаціями були створені необхідні діаграми різних стадій розробки додатку за головними прецедентами його використання та основними сценаріями використання ПС акторами предметної області (клієнтом автосалону, робітником сервісу). Якість розробки оцінювалася за допомогою об’єктно-орієнтованих метрик Чидамбера и Кемерера, Лі і Генрі.на основі першої версії продукту та проектної документації.

Для виконання роботи використовувалось програмне забезпечення Visual Paradigm 16.2. Діаграми створювалися за стандартом мови UML 2.4.1., який прийнято у якості міжнародного стандарту ISO/IEC 19505-1, 19505-2 [4].

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. И.Ю. Коцюба, Чунаев А.В., А.Н. Шиков Основы проектирования информационных систем Учебное пособие Санкт-Петербург 2015
2. Анализ и характеристика областей знаний SWEBOK  
   // studepedia.org/index.php?vol=2&post=20736. Дата звернення: 21.12.2020
3. Кулямин В. В. Технологии программирования. Компонентный подход *//* Спец. курс на фак-те вычисл. техн. и кибернетики МГУ им. М.В. Ломоносова – М., 2006  // электр. ресурс http://panda.ispras.ru/~RedVerst/RedVerst/ Lectures%20and%20training%20courses/Software%20Development%20Technologies/All.pdf. Дата звернення: 12.12.2020
4. Д.В. Кознов «Визуальное моделирование: теория и практика» // http://www.intuit.ru/department/se/vismodtp/ Дата звернення: 12.12.2020
5. About the unified modeling language – Object Management Group // https://www.omg.org/spec/UML/ Дата звернення: 13.12.2020