Розробити лекцію на тему "Пошук акторів (actors) і варіантів використання розроблюваної системи". Мета: навчитись виявляти та описувати вимоги користувачів до програмної системи, визначати основних акторів і формулювати варіанти використання системи Розробка UML-діаграми акторів в розробці програмного забезпечення

Запит стосується розробки лекції з визначення акторів та варіантів використання в UML. Потрібно охопити ключові аспекти: виявлення вимог, ідентифікацію акторів, формулювання Use Case та створення діаграм.

Можна структурувати лекцію логічно: спочатку пояснити базові поняття акторів та Use Case, потім перейти до практичних кроків їх виявлення. Важливо підкреслити зв'язок між вимогами користувачів та цими елементами. Додати приклад діаграми буде корисно для наочності.

Враховуючи, що мета - навчити слухачів практичним навичкам, варто включити конкретні інструкції з опису акторів та варіантів використання. Приклад з системою управління бібліотекою ілюструє теоретичні концепції у знайомому контексті.

Лекція на тему: "Пошук акторів (actors) і варіантів використання розроблюваної системи"

1. Вступ

Мета будь-якої програмної системи – задовольнити певні потреби користувачів. Але як точно зрозуміти, що саме має робити система? Як перекласти розмовні побажання замовника на чіткий технічний план для розробників? Однією з найефективніших методик для цього є **моделювання вимог за допомогою Use Case Diagram (діаграми варіантів використання)** в UML.

Мета лекції: Навчитися виявляти та описувати вимоги користувачів до програмної системи, визначати основних акторів і формулювати варіанти використання системи, а також створювати відповідні UML-діаграми.

2. Вимоги до системи: Від ідеї до специфікації

Насамперед, важливо розрізняти типи вимог:

- **Функціональні вимоги:** Відповідають на запитання "ЩО система повинна робити?". Це конкретні функції, які система надає.

    - *Приклад:* "Система повинна надавати можливість користувачу зареєструватися за допомогою електронної пошти та пароля".

- **Нефункціональні вимоги:** Відповідають на запитання "ЯК система це повинна робити?". Вони стосуються продуктивності, безпеки, надійності тощо.
    - *Приклад:* "Сторінка входу в систему повинна завантажуватися не повільніше ніж за 2 секунди".

Use Case підхід зосереджений насамперед на **функціональних вимогах**.

3. Основа Use Case моделі: Актори (Actors)

Актор – це роль, яку користувач або інша система відіграє у взаємодії з розроблюваною системою.

Ключові характеристики актора:

- Це **РОЛЬ**, а не конкретна людина або посада. Одна людина може виконувати кілька ролей (наприклад, адміністратор може також бути звичайним користувачем).
- Актор знаходиться **ЗОВНІ** системи і взаємодіє з нею.
- Актором може бути не тільки людина, але й:
    - **Зовнішня система** (наприклад, платіжний шлюз, сервер електронної пошти).
    - **Апаратне забезпечення** (наприклад, таймер, який ініціює певну подію).

Як виявляти акторів? Задайте собі питання:

1. Хто безпосередньо використовуватиме основні функції системи?
2. Хто буде адмініструвати, підтримувати систему?
3. З якими зовнішніми системами має взаємодіяти наша система?
4. Хто/що забезпечує системі інформацію?
5. Хто/що отримує інформацію від системи?

Приклад для системи "Онлайн-банк":

- Актор Клієнт
- Актор Адміністратор

- Актор Платіжна система (зовнішня система)

4. Варіанти використання (Use Cases)

Варіант використання – це послідовність дій, яку система виконує для отримання корисного результату для конкретного актора.

Ключові характеристики Use Case:

- Описує **ЩО** система робить, але не **ЯК**.
- Завжди має мету, яка є цінністю для актора.
- Назва Use Case зазвичай формулюється як **дієслівний зворо** (наприклад, "Оформити замовлення", "Переглянути баланс рахунку").

Як виявляти Use Cases? Задайте собі питання для кожного актора:

1. Які основні завдання актор повинен виконувати в системі?
2. Чи потрібно актору читати, створювати, змінювати або видаляти дані в системі?
3. Чи повинен актор повідомляти системі про зміни у зовнішньому середовищі?
4. Чи потрібно системі повідомляти актора про певні події?

Приклад для актора Клієнт **в "Онлайн-банку":**

- Увійти в систему
- Переглянути баланс
- Створити платіжний переказ
- Переглянути історію транзакцій

5. Зв'язки між елементами на Use Case Diagram

UML дозволяє відображати різні типи зв'язків:

- **Асоціація (Association):** Зв'язок між актором і варіантом використання (пряма лінія). Показує, що актор бере участь у цьому сценарії.
- **Включення (Include):** Зв'язок між двома Use Cases (стрілка з пунктирною лінією та стереотипом <<include>>). Показує, що один Use Case обов'язково включає в себе поведінку іншого.

    - *Приклад:* Use Case Створити платіжний переказ **включає** (include) Use Case Підтвердити операцію через SMS. Без підтвердження платіж неможливий.

- **Розширення (Extend):** Зв'язок між двома Use Cases (стрілка з пунктирною лінією та стереотипом <<extend>>). Показує, що один Use Case може (але не обов'язково) розширити поведінку іншого за певних умов.
    - *Приклад:* Use Case Оформити кредитну картку **розширює** (extend) Use Case Переглянути інформацію про рахунок. Клієнт переглядає рахунок і *може*, за бажанням, натиснути кнопку "Оформити кредитку". Це опціональна поведінка.

6. Практичний приклад: Система "Бібліотека"

Крок 1: Визначаємо акторів.

- Бібліотекар (адмініструє систему)
- Читач (основний користувач)
- Касовий апарат (зовнішня система для сплати штрафів)

Крок 2: Визначаємо варіанти використання.

- Для Бібліотекаря: Додати нову книгу, Зареєструвати нового читача, Видати книгу, Прийняти книгу, Створити звіт.
- Для Читача: Пошук книги, Забронювати книгу, Переглянути свій профіль.
- Для Касового апарату: Сплатити штраф.

Крок 3: Уточнюємо зв'язки.

- Видати книгу **включає** (include) Зареєструвати нового читача (якщо читач не зареєстрований).
- Сплатити штраф **розширює** (extend) Прийняти книгу (якщо книга повернута простроченою).

Крок 4: Створюємо діаграму.

(Уявіть собі наочну UML-діаграму, яка будується в реальному часі):

text

[Читач] ---- (Пошук книги)

[Читач] ---- (Забронювати книгу)

[Читач] ---- (Переглянути свій профіль)

[Бібліотекар] ---- (Додати нову книгу)

[Бібліотекар] ---- (Зареєструвати нового читача)

[Бібліотекар] ---- (Видати книгу)

[Бібліотекар] ---- (Прийняти книгу)

[Бібліотекар] ---- (Створити звіт)

[Касовий апарат] ---- (Сплатити штраф)

(Видати книгу) .> (Зареєструвати нового читача) : include

(Прийняти книгу) < . (Сплатити штраф) : extend

7. Алгоритм пошуку акторів та варіантів використання

1. **Ознайомлення з предметною областю:** Проведіть інтерв'ю з замовниками та майбутніми користувачами.
2. **Визначення меж системи:** Чітко уявіть, де закінчується ваша система і починається зовнішнє середовище.
3. **Ідентифікація акторів:** Використовуйте список питань з розділу 3.
4. **Ідентифікація варіантів використання:** Для кожного актора визначте його цілі та завдання (розділ 4).
5. **Пріоритезація:** Визначте, які Use Cases є найважливішими для першої версії продукту.
6. **Деталізація:** Для кожного ключового Use Case створіть текстовий опис у форматі:
    - **Назва:** (напр., "Видати книгу")
    - **Актор:** (напр., "Бібліотекар")

    - **Основна послідовність:** (кроки успішного виконання)
    - **Альтернативні послідовності:** (що робити в разі помилок)
7. **Побудова діаграми:** Візуалізуйте знайдені елементи за допомогою Use Case Diagram.

8. Висновок

Моделювання за допомогою акторів і варіантів використання – це потужний інструмент для:

- **Чіткого формулювання вимог** мовою, зрозумілою для всіх учасників проекту.
- **Виявлення "білих плям"** у функціоналі на ранніх етапах.
- **Створення надійної основи** для подальшого проектування архітектури та написання тестів.

Правильно побудована Use Case модель – це запорука того, що буде розроблено саме та система, яка потрібна користувачеві.