Основи об'єктно-орієнтованого програмування
Лабораторна робота № 1
Моделювання з використанням UML

Частина 1. Conceptual/Domain Modeling

Змоделювати з використанням UML роботу користувача з популярними сервісами (конкретний сервіс залежно від варіанту). Створити глосарій та описати предметну область, основні рішення з архітектури та проектування у вигляді UML діаграм. Код писати не потрібно (хоча можна, за бажання).

Бажано описати загалом основну функціональність (діаграми Use Case, Class, Component, Deployment, Package) і вибрати кілька (3-5) сценарії використання для більш детального опису (діаграми Object, Composite Structure, Sequence, Communication, Timing, Activity, Interaction Overview, State, Profile).

Бажано використовувати різноманітні діаграми UML 2.x – щоб зрозуміти, в чому між ними відмінність та де їх використовують. Якщо якийсь з 14 типів діаграм не використовувався – треба пояснити, чому. Для деяких типів діаграм варто реалізувати кілька діаграм одного типу, наприклад для ілюстрації різних сценаріїв використання чи різних компонентів.

Для отримання **додаткових балів** можна подумати, якої функціональності не вистачає в обраному сервісі, чи що можна було б покращити, і спроектувати реалізацію цієї функціональності (з використанням UML діаграм).

Кожен варіант описує конкретний сервіс. Можна обрати аналогічний сервіс замість того, який вказано у списку (наприклад, замість Google Maps можна змоделювати Bing Maps чи OpenStreetMap). Проте не варто моделювати якісь абстрактні сервіси (систему мап взагалі) – краще обрати конкретний сервіс і описувати його.

Варіанти:

1. Web search (e.g. Google)
2. Web mail (e.g. GMail)
3. Collaborative document editing (e.g. Google Docs)
4. Cloud file storage and file sharing (e.g. Dropbox)
5. Web maps (e.g. Google Maps)
6. Social network (e.g. Facebook)
7. Messenger (e.g. Telegram, WhatsApp, Viber, Skype, …)
8. Twitter
9. Blog service (e.g. Medium)
10. Code repository (e.g. GitHub, Bitbucket or GitLab)
11. Project management/issue tracking system (e.g. Jira, Trac, Bugzilla, …)
12. IDE (e.g. Visual Studio, Eclipse, Qt Creator, …)
13. UML modeling / diagram tool (e.g. StarUML, PlantUML, yEd, …)
14. Інші аналогічні сервіси, на вибір студентів.

Частина 2. Design/Implementation Modeling

Використати UML діаграми для опису структури існуючого коду та його рефакторінгу. Виконання цієї частини складається з наступних кроків:

1. Обрати існуючу програму/проект, з якою планується працювати. Це може бути навчальний проект з минулого семестру, чи одна з лабораторних робіт, чи код з інших предметів, чи з якихось онлайн курсів, чи щось подібне. Також це може бути невеличкий open-source проект. Можна взяти кілька невеликих програм з

метою їх подальшого об’єднання. Код має бути досить складним – тобто не рівня Hello world чи реалізації одного нескладного алгоритму (хоча це може бути кілька схожих чи якось пов’язаних алгоритмів, і на подальших кроках можна буде створити для них спільний програмний інтерфейс).

2. Реалізувати unit tests, шо описують функціональність обраної програми. (Якщо такі тести вже існують – їх можна доповнити, або залишити як є)
3. Побудувати UML діаграми, що описують обрану програму. Варто описати сценарії використання (Use Case), структуру коду (Class, Component, Object, Composite Structure, Deployment, Package, Profile), логіку та поведінку програми (Sequence, Communication, Timing, Activity, Interaction Overview, State). Для побудови деяких діаграм можна використати автоматичну генерацію діаграм з коду; але при цьому діаграми мають бути зрозумілими. Наприклад, взяти 100 класів і кинути їх усі на одну діаграму класів – мабуть, не найкращий варіант ☺
4. Запропонувати якісь зміни в структурі/інтерфейсі/реалізації програми. Це може бути кращий object-oriented design, кращий поділ на компоненти чи відокремлення різних аспектів (наприклад, логіки програми від графічного інтерфейсу), використання якихось патернів проектування, можливість вибору різних варіантів реалізації і т.д. Бажано використовувати побудовану модель програми для опису запропонованих змін. Запропоновані зміни треба узгодити з викладачем.
5. Реалізувати запропоновані зміни.
6. Перевірити, що нова версія програми не вносить зміни в логіку/алгоритми (якщо це не було заплановано). Використати для цього реалізовані раніше unit tests і аналогічні тести, які будуть реалізовані для нової версії.
7. Порівняти попередню та оновлену версії програми за часом виконання окремих алгоритмів/функцій, обсягом коду і т.д.

Мета цієї частини – покращити структуру коду, зробити його більш гнучким та розширюваним. Як частину перетворень, можна реалізувати нову функціональність чи виправити недоліки в попередній – але це має бути додатково до покращень object-oriented design, а не замість нього.

Частина 3. Peer Review (Architecture/Design)

В цій частині студенти переглядають моделі (зокрема, UML діаграми), що були реалізовані іншими студентами (в попередніх частинах 1 та 2), описують свої враження від них і пропонують покращення.

Під час перегляду моделей варто відповісти на такі питання:

1. Наскільки моделі є зрозумілими, наскільки вони описують предметну область, структуру та поведінку відповідної системи?
2. Чи є якісь аспекти, які видаються важливими, але не відображені в моделі (на діаграмах)?
3. Чи є в моделі щось зайве, якісь аспекти описані занадто детально?
4. Наскільки доцільно використані різні типи діаграм?
5. Наскільки коректно використана нотація UML, різні елементи та конектори?
6. Наскільки вдалим є глосарій? Чи всі важливі поняття предметної області описано? Чи немає неоднозначностей?
7. Чи всі важливі сценарії використання описано в моделі? Наскільки зрозумілі різні сценарії, зв’язки між ними?
8. Наскільки доцільним є поділ системи на частини/компоненти/модулі/…?
9. Наскільки доцільними є зв’язки між компонентами/класами/об’єктами? Чи

немає занадто тісно зв’язаних компонентів?

10. Наскільки object-oriented design відповідає загальним принципам?

Студенти-автори моделей отримають доступ до коментарів студентів, які робили review, і матимуть змогу покращити моделі – чи пояснити, чому вони не погоджуються із знайденими недоліками/запропонованими покращеннями.

Варто зазначити, що студентам-авторам моделей не будуть знижуватись бали за те, що інші студенти знайшли недоліки в їх моделях (за умови, що ці недоліки будуть виправлені). Тому не варто приховувати знайдені проблеми в моделях своїх колег – це не допоможе авторам моделей, а лише знизить бали, отримані за цю 3 частину студентами-reviewers.

Для цієї частини дедлайни мають особливу структуру – на review виділяється певний час (орієнтовно 2 тижні?) після того, як автори моделей надали ці моделі для review. Проте варто розуміти, що якщо автори моделей затримуються з виконанням 1 та 2 частин лабораторної – то це створює незручності для інших студентів, які будуть робити review. Тому reviewers можуть заохочувати авторів відповідних моделей до більш швидкого завершення 1 та 2 частин.