МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

Системна інженерія програмного забезпечення

Методичні вказівки та завдання
до виконання практичних та лабораторних робіт (1–7)
для підготовки здобувачів другого (магістерського) рівня
вищої освіти спеціальності 121 «Інженерія програмного забезпечення»

Київ 2024

УДК 004.415/.416(075)
С – 40

Укладач О.Л. Соловей, канд. техн. наук, доцент.

Рецензент: О.А. Поплавський, канд. техн. наук, доцент.

Відповідальна за випуск Т.А. Гончаренко, канд. техн. наук, доцент.

Затверджено на засіданні кафедри інформаційних технологій, протокол № 9 від 27 березня 2024 року.

В авторській редакції.

С – 40 **Системна** інженерія програмного забезпечення: методичні вказівки та завдання до виконання практичних та лабораторних робіт (1–7) / уклад. Соловей О. Л. – Київ : КНУБА, 2024. – 36 с.

Містять теоретичні відомості і рекомендації щодо виконання практичних та лабораторних робіт та вимоги до оформлення звіту. з дисципліни "Інженерія програмного забезпечення". Спрямовані на організацію самостійної роботи студентів.

Призначено для здобувачів другого (магістерського) рівня вищої освіти спеціальності 121 «Інженерія програмного забезпечення» для використання під час виконання практичних та лабораторних робіт (1–7).

Зміст

Загальні положення

Лабораторні роботи є логічним продовженням лекційного курсу з дисципліни «Системна інженерія програмного забезпечення» і є перехідною ланкою від теоретичного курсу до набуття практичних навичок розробки програмного продукту відповідно до SWEBOK.

Під час виконання лабораторних робіт експериментально перевіряються ключові питання курсу програмування, набуваються практичні навички побудови та налагодження програм, перевіряється ступінь засвоєння основних положень предмету. Під час лабораторних робіт студенти знайомляться з типовими рішеннями деяких задач програмування.

Кожна лабораторна робота містить наступні види робіт:

- аналіз умови задачі;
- виконання задача;
- демонстрацію виконаного завдання;
- відповіді на контрольні запитання;
- складання і захист звіту.

Завершивши вивчення дисципліни, здобувач повинен вміти створювати документ про концепцію проєкту зі створення нового програмного продукту; виявляти вимоги до нового програмного продукту; проєктувати інтерфейс користувача програмного продукту; робити концептуальне та логічне проєктування структур даних програмного продукту; робити проєктування архітектури нового програмного продукту за допомогою UML діаграм розгортання; повинен вміти робити збереження ієрархічних структур даних у реляційній базі даних.

Лабораторна робота №1.
Створення документа про концепцію проєкту зі створення нового програмного продукту

Мета роботи: здобути навички визначення бізнес-цілей, проблем та критеріїв успіху для програмного продукту, створення документа про концепцію проєкту.

Теоретичні відомості

Вимоги до програмного продукту складаються з трьох рівнів, де перший рівень – це рівень бізнес-вимог.

Відправною точкою для формування бізнес-вимог у будь-якому проєкті чи ініціативі є розуміння цілей і завдань бізнесу – цілі зростання, розширення ринку, зниження витрат, підвищення задоволеності клієнтів тощо. Бізнес-цілі складаються з фінансових та нефінансових цілей. Наприклад, до фінансових цілей відносять:

- Освоїти X% ринку за Y місяців.
- Збільшити частку ринку в країні W з X% до Y% за Z місяців.
- Досягти обсягу продажів X одиниць або доходу в Y доларів за Z місяців.
- Отримати X% прибуток за інвестиціям протягом Y місяців.
- Досягти позитивного потоку коштів у цьому продукту протягом Y місяців.
- Заощадити X доларів на рік, які зараз витрачаються на обслуговування.
- Зменшити витрати на підтримку з X до Y доларів за Z місяців.
- Збільшити валову маржу для існуючого бізнесу з X до Y% протягом одного року.

До нефінансових цілей відносять:

- Досягти показника задоволення покупців, рівню X, протягом Y місяців від часу випуску продукту.
- Збільшити продуктивність обробки транзакцій на X% та знизити рівень помилок даних до величини трохи більше Y%.
- Розробити надійну платформу для сімейства пов’язаних продуктів.

- Розробити спеціальну базову технологічну основу для організації.
- Домогтися визнання продукту найкращим за надійністю в опублікованих оглядах продуктів до певної дати.
- Забезпечення виконання певних норм активів регулюючих органів.
- Отримати не більше X дзвінків у службу обслуговування з кожної одиниці товару та Y дзвінків за гарантією кожної одиниці товару протягом Z місяців після випуску продукту.
- Зменшити час виконання заявки до X годин на Y% дзвінків до служби підтримки.

Для виявлення бізнес-цілей необхідно визначити зацікавлені сторони, тобто осіб, які цікавляться результатом проєкту, їхні ролі та відповідальності – ці дані включаються в таблицю зацікавлених сторін (див табл. 1.1).

Таблиця 1.1

Роль та рівень впливу «зацікавлених сторін» для проєкту з розробки програмного продукту

Роль	Ім’я	Посада	Рівень впливу

Кожній ролі зацікавленої сторони відповідає діяльність:

1. Спонсори та зацікавлені особи (stakeholders): фінансують проєкт.
2. Власник продукту (product owner): у гнучких методологіях власник продукту несе відповідальність за визначення та розставляння пріоритетів для backlog. Вони представляють замовника та приймають рішення щодо того, які функції включено до програмного забезпечення.
3. Бізнес-аналітики: відповідають за виявлення, аналіз і документування вимог.
4. Системні архітектори: проєктують структуру високого рівня та компоненти системи програмного забезпечення. Вони гарантують, що вимоги відповідають загальній архітектурі та технічним обмеженням.

5. Розробники: використовують задокументовані вимоги як основу для розробки та впровадження програмного забезпечення.
6. Тестувальники: перевіряють, чи програмне забезпечення відповідає заданим вимогам.
7. Менеджери проєктів (PM): контролюють увесь проєкт розробки програмного забезпечення, включаючи процес вимог.
8. Кінцеві користувачі (end users): особи, які згодом використовуватимуть програмне забезпечення. Їхній внесок є цінним для перевірки вимог і забезпечення ефективного задоволення програмного забезпечення їхніх потреб.
9. Експерти з регулювання та відповідності: якщо програмне забезпечення має відповідати певним нормам або галузевим стандартам, можуть бути залучені експерти з питань регулювання та відповідності, щоб переконатися, що програмне забезпечення відповідає цим вимогам.
10. Експерти домену (SME): для проєктів, пов’язаних зі складними доменами або спеціалізованими галузями, експерти домену надають експертні знання з предмета, щоб гарантувати, що вимоги точно відображають нюанси домену.

Рівень впливу оцінюється за категоріями:

Responsible (R) – особа, яка бере безпосередню участь у визначенні вимог.

Accountable (A) – особа, яка засвідчує, що вимоги правильні.

Consulted (C) – окремі особи або групи, які надають додатку інформацію, пов’язану з вимогами.

Informed (I) – окремі особи або групи, яких необхідно інформувати про хід або статус роботи з вимогами. Вони не беруть безпосередньої участі у виконанні завдання, але повинні бути в курсі його розвитку.

Для визначення фінансових та нефінансові цілей використовують техніки:

1. Індивідуальні інтерв’ю із зацікавленими сторонами, щоб зібрати детальну інформацію про їхні потреби, уподобання та очікування.
2. Опитування та анкети, якщо є потреба ефективного збору великої кількості інформації.

3. Семінари з групою зацікавлених сторін, щоб сприяти мозковому штурму, обговоренню та генеруванню ідей. Можна використовувати різні формати семінарів.

Результатом визначених бізнес-цілей є діаграма бізнес-цілей, проблем та критеріїв успіху для програмного продукту (рис. 1.1).

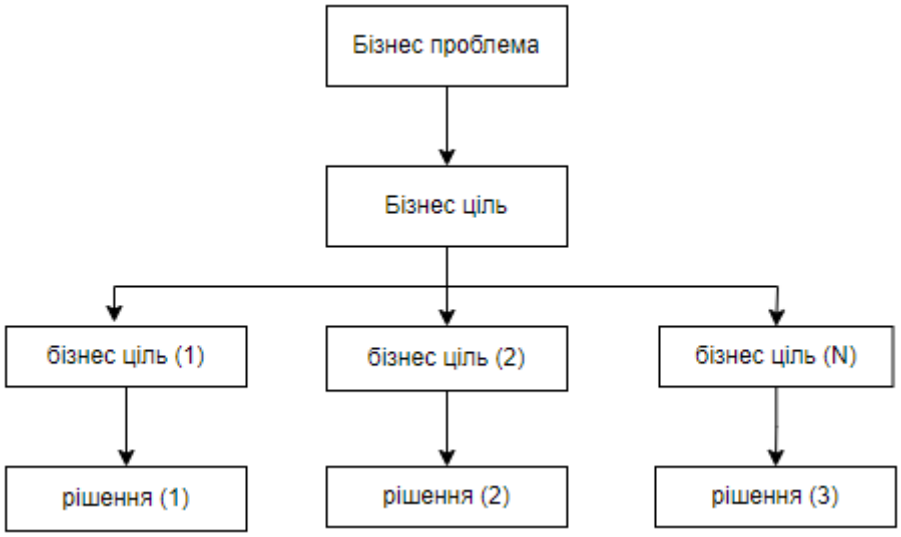

Рис. 1.1. Діаграма бізнес-цілей

Положення про концепцію проєкту узагальнює бізнес-цілі та призначення нового програмного продукту. Документ відображає концепцію і створюється на основі шаблону, який складається з ключових слів:

Для [кінцеві користувачі];

Який [положення про потреби чи можливості];

Ця (цей) [ім’я продукту];

Належить [категорія];

Який(а) [основні функції, ключова перевага, основна причина для покупки чи використання];

На відміну від [основний конкуруючий продукт або поточна система, або поточний бізнес-процес];

Наш продукт [положення про основну відмінність та перевагу нового продукту].

Бізнес-вимоги

Завдання

1. Вибрати проєкт для створення програмного продукту (теми вказані нижче), визначити проблему, яку допоможе вирішити автоматизація.

2. Визначити зацікавлені стороні та скласти таблицю «Роль та рівень впливу зацікавлених сторін».

3. Скласти запитання та відповіді для проведення консультацій із зацікавленими сторонами щодо виявлення бізнес-цілей (фінансових і нефінансових).

4. Скласти діаграму бізнес-цілей, проблем та критеріїв успіху для програмного продукту.

5. Скласти положення про концепцію проєкту, відповідно шаблону.

6. Підготувати звіт, який включить результати виконання кроків завдання. Відповіді на контрольні запитання.

Теми, що пропонуються для самостійного виконання

1. Програма, призначена для документування та збереження історичних будівель.

2. Програма, яка дозволяє архітекторам і клієнтам проводити рецензування дизайну дистанційно, з функціями анотування креслень, залишати відгуки та обговорювати зміни в реальному часі.

3. Програма, яка допомагає архітекторам і дизайнерам переконатися, що їхні проєкти відповідають місцевим будівельним вимогам і нормам.

4. Програма, призначена для управління будинком, що включає real-time моніторинг, як будинком витрачається енергія, та рекомендації щодо оптимізації. Для реалізації моніторингу рекомендовано використовувати обладнання типу «сенсор».

5. Програма, призначена для управління будинком, що включає real - time моніторинг технічного стану будинку, оцінку стану, автоматичного визначення обладнання, яке потребує технічного огляду. Для реалізації моніторингу рекомендовано використовувати обладнання типу «сенсор».

6. Програма, призначена для ефективного управління місцями для паркування на виділеній прибутковій площі, яка може змінюватись.

Контрольні запитання

1. Які 10 областей знань визначають теоретичні концепції, методи та засоби дисципліни «Програмна інженерія»?

2. Які підобласті знань включено в поняття «програмні вимоги»?

3. Які вимоги визначають три рівні «програмних вимог»?

4. У чому полягає різниця між функціональними і нефункціональними вимогами? Наведіть приклади.

5. Які існують «джерела програмних вимог»?

6. Які існують ролі для зацікавлених осіб?

7. Як розділяються відповідальності між особою, яка визначена «Responsible», та особою, яка визначена «Accountable»?

Лабораторна робота №2.
Виявлення вимог до нового програмного продукту

Мета роботи: здобути навички створення варіантів використання та складання діаграм використання.

Теоретичні відомості

Другому рівню ієрархії вимог до програмного продукту належать вимоги користувача. Основним джерелом для визначення вимог користувача виступають – «сценарії використання», в гнучких методологіях розробки програмного забезпечення це – «користувацькі історії». Варіант використання – це окрема, незалежна дія, яку чинна особа може виконати для отримання певного значущого результату. Один варіант використання може охоплювати кілька схожих дій з однією метою. «Користувацька історія» – це «короткий, простий опис функції з погляду людини, якій потрібна нова функція, зазвичай це користувач або клієнт системи». Таким чином, «сценарії використання» та «користувацькі історії» зміщають акцент на функції, які користувачам необхідно виконувати в системі, і мають за мету – описати завдання, які користувачам необхідно виконувати за допомогою системи.

Текстовий опис кожного «сценарію використання» подається відповідно до шаблону (див. табл. 2.1), в якому:

1. Ідентифікатор та назва: визначають мету варіанта використання та його унікальний номер.

Таблиця 2.1

Шаблон текстового опису сценарію використання програмного продукту

Ідентифікатор та назва	
Автор	
Дата створення	
Основна дійова особа	
Додаткова дійова особа	
Опис	
Тригер	
Попередні умови	
Нормальний напрямок подій	
Альтернативний напрямок подій	
Винятки	
Частота використання	
Бізнес-правила	
Спеціальні вимоги	
Припущення	

1. Основна та додаткова дійові особи: користувачі, які беруть участь у сценарії використання.

2. Опис: короткий текстовий опис варіанта використання.

3. Тригер: умова, яка ініціює виконання варіанта використання.

4. Попередні умови: список передумов, які мають бути виконані перед початком і після виконання варіанта використання.

5. Нормальний напрямок подій: послідовні кроки, які описують дії основних та додаткових дійових осіб сценарію.

6. Альтернативний напрямок подій: дії, які описують відхилення від кроків нормального напрямку подій.

7. Винятки: список особливих ситуацій, які можуть виникнути під час виконання сценарію.

8. Бізнес-правила: визначають додаткові вимоги, пов'язані зі сценарієм використання.

Створені варіанти використання об'єднуються у діаграму варіантів використання з метою візуалізації варіантів використання разом з відповідними користувачами. До елементів діаграми варіантів використання належать прямокутник, який показує межі варіанта використання; лінії, що з'єднують кожну дійову особу з варіантами використання (еліпсами), з якими ця особа взаємодіє. Стрілки вказують на асоціації між варіантами використання, для яких можливі відношення типів: розширення (extend) та включення (include).

Відомо, що життєвий цикл розробки програмного забезпечення (Software Development Life Cycle, SDLC) містить такі процеси: визначення вимог (Requirements), проєктування (Design), реалізація (Implementation), тестування (Testing), інтеграція (Integration), розгортання (Deployment) та супровід (Maintenance). У результаті виконання кожного етапу створюються нові артефакти, які використовуються в подальших фазах. Створені артефакти зберігаються в сховищі для спільної розробки програмного забезпечення.

GitHub – один із найбільших вебсервісів для спільної розробки програмного забезпечення. Базується на системі керування версіями Git.

Завдання

1. Визначити список варіантів використання нового програмного продукту (відповідно до вашого варіанта). Має бути визначено не менш 3-х типів користувачів і для кожного складено не менш 2-х варіантів використання.

2. Скласти текстовий опис кожного з визначених варіантів використання відповідно до шаблону (див. табл. 2.1).

3. Створити діаграму варіантів використання.

4. Документувати кроки 2–3 лабораторної роботи у Веб-сервісі GitHub системи керування версіями Git. У **новій гілці** Git-репозиторію у

файлі README.md мають бути рядки з урахуванням мови розмітки Markdown:

Київський національний університет будівництва і архітектури (md-заголовок 1-го рівня)

Кафедра: «Інформаційних технологій» (md-заголовок 2-го рівня)

Дисципліна: «Системна інженерія програмного забезпечення»

Проєкт: «Має бути назва вашого проєкту» (md-заголовок 3-го рівня)
Розробник:

ПІБ:

група:

Створити каталог «1-Requirements» з файлом README.md. у файлі README.md мають бути рядки: «Вимоги до програмного продукту». У каталозі «1-Requirements» створити підкаталог 1.1-UseCases в README.md має бути завантажено текстовий опис кожного з визначених варіантів використання. Створити підкаталог 1.2-UseCaseDiagram в README.md, де має бути завантажена діаграма варіантів використання.

5. Підготувати звіт, який включатиме результати виконання кроків завдання. Відповіді на контрольні запитання.

Контрольні запитання

1. Які підобласті знань в SWEBOK включено в поняття «Процес роботи з вимогами»?
2. Які завдання підобласті знань «Учасники процесу»?
3. Як класифікуються зацікавлені сторони відповідно до «матриці зацікавлених сторін»?
4. Які завдання підобласті знань «Якість та покращення процесів»?
5. Які дії включає процес «Banckmarking процесів роботи з вимогами»?
6. Які підобласті знань в SWEBOK включено в поняття «Виявлення вимог»?
7. Які джерела та методи для визначення вимог до програмного продукту вам відомі?

Лабораторна робота №3.
Документування функціональних та нефункціональних вимог до нового програмного продукту

Мета роботи: здобути навички збору та документування функціональних та нефункціональних вимог.

Теоретичні відомості

Міжнародний стандарт ISO/IEC/IEEE 29148:2011 «Systems and software engineering – Life cycle processes – Requirements engineering» містить наступний перелік вимог, яким мають відповідати якісні функціональні вимоги:

1. Атомарність (Singular) вимагає надавати опис лише однієї вимоги, а не об’єднувати декілька вимог.

2. Завершеність (Complete) вказує на те, що вимоги описують сутності та процеси у повному обсязі без подальшої деталізації.

3. Необхідність (Necessary) вказує на наявність такої вимоги, що коли її видалити, буде існувати недолік, який не можна задовольнити іншими можливостями продукту.

4. Однозначність (Unambiguous) вказує на єдине сприйняття термінів всіма зацікавленими особами проєкту, в іншому випадку треба розкрити терміни в окремому словнику термінів (глосарію).

5. Відокремленість від реалізації (Implementation Free) вказує на незалежність вимоги від особливостей подальшої реалізації, коли вимога визначається лише ЩО? потрібно виконати, а не ЯК?

6. Узгодженність (Consistent) вказує на відсутність протиріч між різними вимогами, наприклад, для однієї сутності різні вимоги надають різну структуру даних.

7. Здійсненність (Feasible) вказує на технічну досяжність реалізації вимог із прийнятним ризиком.

8. Перевіряємість (Verifiable) вказує на наявність чіткого алгоритму перевірки відповідності програмного продукту вимозі, наприклад, за наявності метричного показника, який можна виміряти.

9. Відстежуваність (Traceable) вказує на можливість відстежувати зв’язки між вимогами різних рівнів, наприклад, бізнес-вимоги, вимоги

користувача, або відстежувати різні версії вимог на етапі супроводу програмного продукту.

Умова «відстежуваність» виконується, надаючи вимогам унікальні ієрархічні ідентифікатори FRi, як показано в табл. 3.1.

Таблиця 3.1

Опис функціональних вимог

Номер за порядком	Опис
FR1	
FR1.1	
FR1.2	
FR2	
FR2.1	
…	
FR3	
…	

Нефункціональні вимоги до програмного продукту, за стандартом ISO/IEC/IEEE 29148:2011, представляються ієрархією (рис. 3.1).

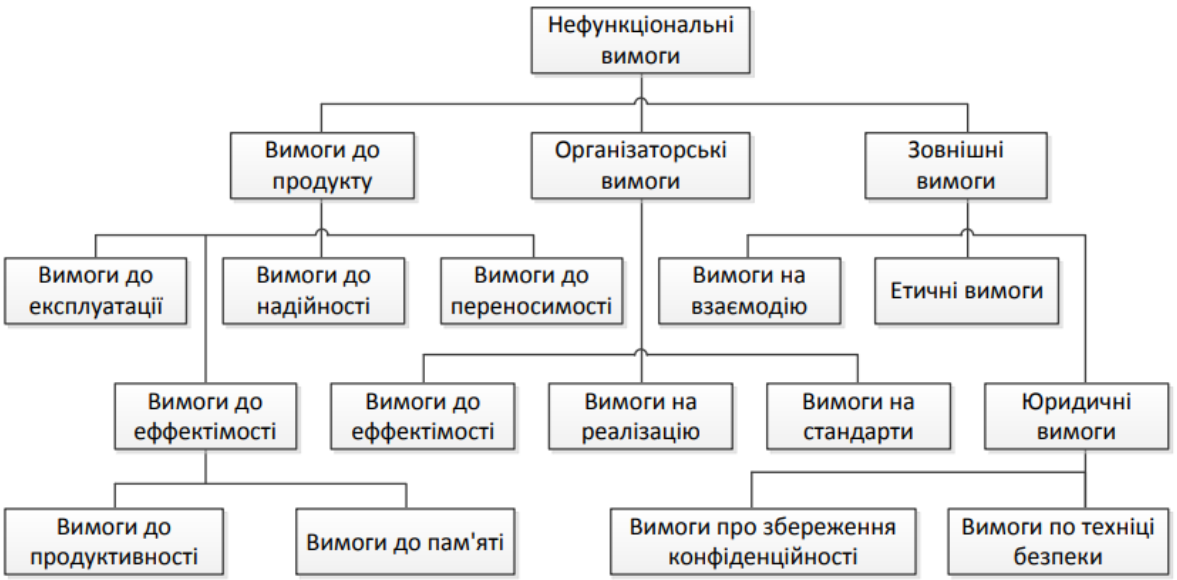

Рис. 3.1. Ієрархія нефункціональних вимог програмного продукту

Нефункціональні вимоги, показані на рис. 3.1, поділено на три великі групи.

1. Вимоги до продукту описують експлуатаційні властивості програмного продукту. Сюди належать вимоги до продуктивності системи, ємності необхідної пам’яті, надійності, підтримувані апаратні платформи та зручність експлуатації.

2. Організаційні вимоги відображають політику й організаційні процедури замовника та розробника ПЗ. Вони включають стандарти розроблення програмного продукту, вимоги до реалізації ПЗ (тобто до мови програмування та методів проєктування), вихідні вимоги, які визначають терміни виготовлення програмного продукту, і супутню документацію.

3. Зовнішні вимоги ураховують фактори, що визначають взаємодію проєктованої системи з іншими системами, юридичні вимоги, включення яких гарантує, що система буде розроблятись і функціонувати в межах існуючого законодавства, а також етичні вимоги. Останні повинні забезпечувати прийнятність системи для користувачів або замовника.

Нефункціональні вимоги виражаються кількісними показниками:

1. Швидкість – кількість виконаних транзакцій за секунду, час реакції на дії користувача, час відновлення екрана.

2. Розмір – кілобайти, кількість модулів пам’яті.

3. Простота експлуатації – час навчання персоналу, кількість статей у довідковій системі.

4. Надійність – середня тривалість часу між двома послідовними проявами помилок у системі, імовірність виходу системи з ладу, коефіцієнт готовності системи.

5. Стійкість до збоїв – час відновлення системи після збою, відсоток подій, що призводять до збоїв, імовірність псування даних у разі збоїв.

6. Сумісність – відсоток машинно-залежних операторів, кількість машинно-залежних підсистем.

Умова «відстежуваність» для зв’язку Non Functional Requirements (NFR) і Functional Requirements (FR) та ідентифікації NFR виконується відповідно до табл. 3.2.

Таблиця 3.2

Опис нефункціональних вимог

FR	NFR	Опис NFR
FR1	NFR 1	
FR1.1	NFR 2	
FR1.1	NFR 3	
FR1.1	NFR 4	

Завдання

1. Визначити функціональні вимоги і заповнити відповідно до табл. 3.1.
2. Визначити нефункціональні вимоги і заповнити відповідно до табл. 3.2.
3. Створити каталог «2-FuncNoFuncRequirements» з файлом README.md. У файлі README.md мають бути рядки: «Функціональні та нефункціональні вимоги до програмного продукту». Створіть підкаталог «2.1-FunctionalRequirements» у README.md, файл додайте табл. 3.1. Створіть підкаталог «2.2-NonFunctionalRequirements» у README.md, файл додайте табл. 3.2.
4. Підготувати звіт, який включатиме результати виконання кроків завдання. Відповіді на контрольні запитання.

Контрольні запитання

1. Які документи, відповідно до SWEBOK, включені в список рекомендованих для опису комплексних проєктів із розробки програмного забезпечення?
2. Який стандарт описує структуру документа «специфікація програмних вимог» ?
3. Які дії необхідно виконувати для «управління змінами» до вимог?
4. Що означає базова версія вимог?
5. Які варіанти документування змін до вимог ви знаєте?

Лабораторна робота №4.
Проєктування інтерфейсу користувача програмного продукту

Мета роботи: здобути навички проєктування каркасу та макету інтерфейсу користувача.

Теоретичні відомості

Інтерфейс (англ. interface – перегородка) надає засоби та правила взаємодії (керування, контроль тощо) між елементами системи. Проєктування інтерфейсу користувача може проводитися за трьома методами в порядку підвищення складності процесу та ефекту від результатів:

1) орієнтоване на завдання користувачів (Task Centered Design);
2) орієнтоване на користувачів (User Centered Design);
3) орієнтоване на мету, цілі користувачів (Goal Directed Design).

До основних принципів проєктування інтерфейсу користувача належать:

1. Орієнтація на знання користувача – використання термінів і понять, зрозумілих користувачам системи.

2. Узгодженість – різні частини інтерфейсу повинні бути узгодженими щодо однотипних операцій.

3. Мінімізація несподіванок – прогнозоване поводження системи.

4. Відновлюваність – інтерфейс повинен включати засоби, що дозволяють відновлення даних після помилкових дій.

5. Забезпеченість довідковою інформацією – інтерфейс повинен надавати довідкову інформацію на випадок помилок користувача і підтримувати засоби контекстно-залежної довідки.

6. Орієнтованість на можливості користувачів – інтерфейс повинен мати засоби для зручної взаємодії з користувачами різного рівня кваліфікації і можливостей.

Дуже важливим є принцип відновлюваності системи, бо користувачі завжди допускають помилки. Правильно спроєктований інтерфейс може зменшити кількість помилок користувача (наприклад, використання меню дозволяє уникнути помилок, які виникають з уведенням команд із клавіатури), однак всі помилки усунути неможливо. В

інтерфейсах повинні бути засоби, що запобігають, за можливості, помилкам користувача, а також дають змогу відновити інформацію після усунення помилок. Ці засоби бувають двох видів:

1. Підтвердження деструктивних дій. Якщо користувач вибрав потенційно деструктивну операцію, то він повинен ще раз підтвердити свій намір.

2. Можливість скасування дій. Скасування дій повертає систему в той стан, у якому вона перебувала до їх виконання. Не зайвим буде підтримання багаторівневого скасування дій, оскільки користувачі не завжди відразу розуміють, що зробили помилку.

Розробнику інтерфейсу користувача необхідно вибрати один з п’яти стилів, відповідно з яким користувач буде взаємодіяти з ПП:

1. Безпосереднє маніпулювання. Користувач взаємодіє з об’єктами на екрані. Наприклад, для видалення файлу користувач перетягує його в кошик.

2. Вибір команди зі списку пунктів меню. Обрана команда впливає лише на той об’єкт, що виділений (обраний) на екрані.

3. Заповнення форм. Користувач заповнює поля екранної форми. Деякі поля можуть мати своє меню (випадне або списки).

4. Командна мова. Користувач уводить конкретну команду з параметрами, щоб указати на її подальші дії. Щоб видалити файл, користувач уводить команду видалення з іменем файла як параметра цієї команди.

5. Природна мова. Щоб видалити файл, користувач може ввести команду «Видалити файл з іменем XXX».

Інтерфейс користувача має включати засоби подання даних. Приймаючи рішення щодо подання даних, розробник повинен ураховувати такі фактори:

1) що потрібно користувачу – точні значення даних або співвідношення між значеннями;

2) наскільки швидко відбуватимуться зміни значень даних; чи потрібно негайно показувати користувачу зміну значень;

3) чи належить користувачу вдаватися до якихось дії у відповідь на зміну даних;

4) чи потрібно користувачу взаємодіяти з відображуваною інформацією за допомогою інтерфейсу із прямим маніпулюванням;

5) інформація має відображатися в текстовому (описово) або числовому форматі;

6) чи важливі відносні значення елементів даних.

Основним принципом розробників інтерфейсів є обережне використання кольорів на екранах. Для виконання обмеження застосовують правила:

1. У вікні слід використовувати 4–5 різних кольорів, в інтерфейсі системи не повинно бути більше 7-ми кольорів.
2. Зміни у стані системи відображають різними кольорами.
3. Виділяють аномальні елементи кольором; якщо потрібно знайти подібні елементи, їх позначають однаковим кольором.

Проєктування інтерфейсу користувача включає етапи:

1. Проєктування каркасу (Wireframe Design) – структурно-інформаційний, але приблизний опис взаємодії користувача з програмним доданком;
2. Проєктування макету (Mockup Design) – точний, але статичний опис зовнішнього вигляду програмного доданку;
3. Проєктування прототипу (Prototype Design) – інтерактивний точний опис зовнішнього вигляду програмного доданку.

Завдання

1. Розробити проєкт каркасу – приблизний опис, як користувачі будуть взаємодіяти з програмним продуктом (відповідно до вашого варіанта), який включає каркаси вікон та переходи між ними. Проєкт каркасу виконати в Visio або draw.io.
2. Розробити проєкт макету – точний опис зовнішнього вигляду вікон програмним продуктом (відповідно до вашого варіанта). Проєкт макету виконати в Visio або draw.io. Кожен макет назвати відповідно до шаблону: «NFR_№».
3. Створити каталог «3-Requirements» з файлом README.md., у файлі README.md мають бути рядки: «Вимоги до програмного продукту». У каталозі «3-UserInterface» завантажити створений проєкт каркасу та проєкти макетів.

4. Підготувати звіт, який включатиме результати виконання кроків завдання. Відповіді на контрольні запитання.

Контрольні запитання

1. Сформулюйте принципи проєктування інтерфейсу користувача.
2. Наведіть п’ять різних стилів взаємодії користувача із програмними системами.
3. Назвіть стилі подання інформації. У яких випадках доцільне графічне зображення даних?
4. Які правила проєктування засобів підтримання користувача застосовуються у відомому програмному забезпеченні?
5. Які 4 типи дій може виконувати користувач через INPUT-засіб?
6. Які 6 типів відчуттів може використати користувач через OUTPUT-засіб?

Лабораторна робота №5.
Концептуальне та логічне проєктування структур даних програмного продукту

Мета роботи: здобути навички з концептуального та логічного проєктування структур даних програмного продукту з урахуванням функціональних вимог до програмного продукту.

Теоретичні відомості

Модель даних (Data model) – інструмент абстрагування об’єктів, їхніх властивостей та взаємозв’язків предметної області. Модель даних використовують для з’ясування та аналізу вимог, а також для підтримки впровадження та безперервного вдосконалення програмного продукту. Види моделей даних:

1. Концептуальна модель (схема) даних – це опис високого рівня інформаційних потреб програмного продукту, що визначає структуру модельованої системи, властивості її елементів і причинно-наслідкові зв’язки. Зазвичай вона включає лише основні поняття та зв’язки між ними.

Вона приховує внутрішні деталі фізичного сховища та цілі опису сутностей, типів даних, зв’язків та обмежень.

2. Логічна модель (схема) даних – модель даних, виражена незалежно від конкретного продукту керування базами даних або технології зберігання, але в термінах структур даних, таких як реляційні таблиці, об’єктно-орієнтовані класи чи теги XML. Логічна модель (схема) даних походить від концептуальної моделі (схема) даних.

3. Фізична модель (схема) даних – опис даних, який призначений для реалізації. Фізична модель (схема) даних походить від логічної моделі (схема) даних.

Порівняльна характеристика моделей даних наведена в табл. 5.1.

Таблиця 5.1

Порівняльна характеристика моделей даних

Концептуальна модель	**Логічна модель даних**	**Фізична модель даних**
Включає високорівневі конструкції даних	Включає сутності, атрибути та відношення (ключі)	Включає таблиці, колонки, ключі, типи даних, правила перевірки, тригери баз даних, збережені процедури, домени й обмеження доступу
Описує конструкції даних нетехнічними назвами	Описує сутності, атрибути та відношення (ключі) технічними назвами	Використовує назви, які будуть використані під час реалізації програмного продукту
Може не бути нормалізованою	Нормалізована до четвертої нормальної форми (4НФ)	Може бути денормалізована, щоб відповідати вимогам продуктивності, заснованим на характері бази даних. Якщо характер бази даних є онлайновою обробкою транзакцій (OLTP) або операційним сховищем даних, то вона зазвичай не де-нормалізується. Денормалізація є звичною у сховищах даних

Розрізняють два головних підходи до моделювання даних у процесі концептуального проєктуванні:

1. Семантичні моделі.
2. Об'єктні моделі.

Семантичні моделі головну увагу приділяють структурі даних. Найбільш поширеною семантичною моделлю є модель «сутність – зв'язок» (Entity Relationship model, ER-модель). ER-модель складається із сутностей, зав’язків, атрибутів, доменів атрибутів, ключів. Моделювання даних відображає логічну структуру даних, так само, як блок-схеми алгоритмів відображають логічну структуру програми.

Об’єктні моделі головну увагу приділяють поведінці об’єктів даних і засобам маніпуляції даними. Головне поняття таких моделей – об’єкт, тобто сутність, яка має стан і поведінку. Стан об'єкта визначається сукупністю його атрибутів, а поведінка об’єкта визначається сукупністю операцій, специфікованих для нього.

Сутність дозволяє моделювати клас однотипних об’єктів. Сутність має унікальне ім’я у межах заданої системи. Передбачається, що в системі існує багато екземплярів даної сутності. Об’єкт, якому відповідає сутність, має набір атрибутів, які характеризують його властивості. Цей набір повинен бути таким, щоб можна було розрізняти конкретні екземпляри сутності.

Слабкі та незалежні сутності. Якщо сутність може існувати незалежно від інших, то вона є незалежною. Слабкою сутністю називають таку, що задовольняє умовам:

1) залежність від існування сутності з якою вона зв’язана;
2) первинний ключ цієї сутності частково або повністю отримується з іншої сутності.

Атрибути являють собою властивості сутності. Значення кожного атрибута вибирають з відповідної множини значень, яка включає всі потенційні значення, що можуть бути присвоєні атрибуту. Ця множина значень називається доменом. Між сутностями встановлюються зв’язки, що вказують, яким чином сутності співвідносяться або взаємодіють між собою.

Стандартна графічна нотація візуалізації ER-моделі – діаграма «сутність–зв’язок» (Entity-Relationship diagram, ER-діаграма, ERD). ERD – може бути представлена в концепції Пітера Чена, або «Пташина лапка».

Логічна модель даних може бути представлена у вигляді ієрархії класів за наступними правилами:

1. Для кожної сутності створюється клас з відповідними атрибутами.
2. Зв’язки між сутностями в логічній моделі даних перетворюються на зв’язки між класами в моделі класів.
3. Первинні ключі в логічній моделі даних стають унікальними ідентифікаторами або атрибутами у відповідних класах у моделі класів.
4. Зовнішні ключі в логічній моделі даних представляють зв’язки між сутностями і можуть бути переведені в асоціації або посилання між класами в моделі класів.

Завдання

1. Виділити сутності та створити логічну модель опису даних типу «сутність–зв'язок» (ERD) для обраного вами програмного продукту за нотацією Чена, або «Пташина лапка». Логічну модель має включити не менше шести сутностей.
2. Створити таблицю «словник атрибутів об’єктів» за структурою, яку представлено в табл. 5.2.

Таблиця 5.2

Структура таблиці «словника атрибутів об’єктів»

Об’єкт	**Атрибут**	**Короткий опис**	**Тип**	**Обмеження**

3. Створити UML діаграми класів, які відповідатимуть розробленій логічній модель даних. Розробити ієрархію класів із використанням шаблонів проєктування типу «структурні» та «поведінкові».
4. Створити каталог «4-Design» в GitHub з файлом README.md. Додайте підкаталог «4.1-Logical Data model» з файлом README.md. У файлі README.md має бути ERD діаграма. Додайте підкаталог «4.2-

UMLConceptClasses» з файлом README.md. У файлі README.md має бути UML діаграма класів. Додайте підкаталог «4.3-AttributeVocabulary» з файлом README.md. У файлі README.md має бути поміщено «словник атрибутів об’єктів».

5. Підготувати звіт, який включатиме результати виконання кроків завдання; відповіді на контрольні запитання; посилання на гілку в GitHub.

Контрольні запитання

1. У чому різниця між концептуальною, логічною та фізичною моделями даних?
2. Опишіть базові структури мережевої моделі даних.
3. Які основні операції можна проводити над мережевою моделлю даних?
4. Опишіть базові структури ієрархічної моделі даних.
5. Які основні операції можна проводити над ієрархічною моделлю даних?

Лабораторна робота №6.
Проєктування архітектури нового програмного продукту за допомогою UML діаграм розгортання

Мета роботи: здобути навички створення діаграми розгортання.

Теоретичні відомості

Діаграма розміщення (розгортання) – друга із двох різновидів діаграм реалізації UML, що моделюють фізичні аспекти об’єктно-орієнтованих систем.

Цей тип діаграм застосовується для подання загальної конфігурації і топології розподіленої програмної системи і містить зображення розміщення компонентів в окремих вузлах системи. Крім того, діаграма розгортання вказує на наявність фізичних сполук – маршрутів передавання інформації між апаратними пристроями, задіяними в реалізації системи.

Призначення діаграми розгортання полягає у візуалізації елементів і компонентів програми, які наявні лише на етапі розміщення системи. При

цьому подаються лише ті компоненти програми, які здійснюються файлами або динамічними бібліотеками. Компоненти, які не використовуються на етапі виконання, на діаграмі розгортання не показуються. Компоненти з вихідними текстами програм можуть бути лише на діаграмі компонентів. На діаграмі розгортання вони не показуються. Діаграма розгортання містить графічні зображення процесорів, пристроїв, процесів і зв'язків між ними. На відміну від діаграм логічного подання, вона є єдиною для системи в цілому, оскільки відображає особливості її реалізації. Цією діаграмою завершується процес проєктування конкретної програмної системи та її розроблення останній етап специфікації моделі. Вона розробляється спільно системними аналітиками, мережевими інженерами і системотехніками.

Елементами діаграм розгортання є вузли та асоціації, які зображуються відрізками ліній без стрілок. Наявність такої лінії вказує на необхідність організації фізичного каналу для обміну інформацією між відповідними вузлами. Характер з'єднання може бути додатково специфікований приміткою, стереотипом, супроводжується значенням або обмеженням. Як і інші діаграми, діаграми розміщення можуть включати примітки й обмеження. Так, на рис. 6.1 показано фрагмент діаграми розгортання, на якій визначені:

- Вузол – фізично існуючий елемент системи, який може містити обчислювальний ресурс, що має пам'ять. Вузол зображується у формі куба з боковими гранями сірого кольору.
- Пристрій на діаграмі розгортання – це також вузол, який представляється кубом з боковими гранями білого кольору (див. рис. 6.1).

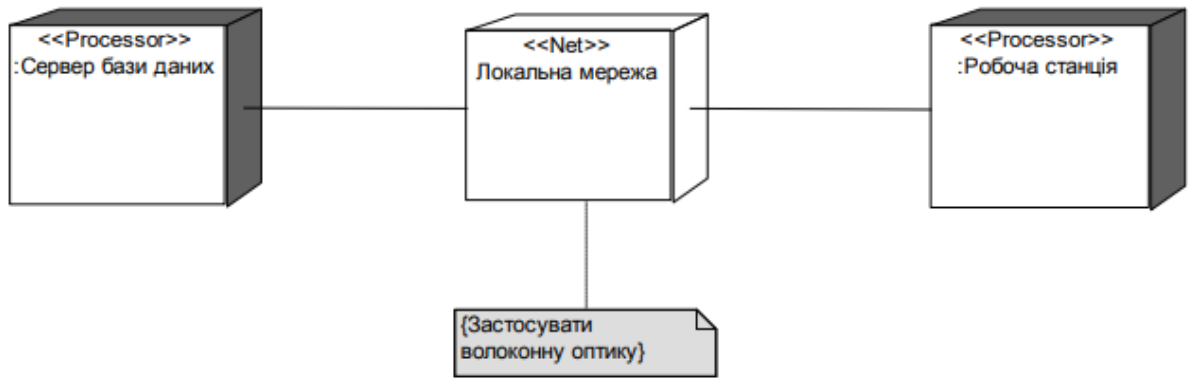

Рис. 6.1. Фрагмент діаграми розгортання

Розроблення діаграми розгортання починається з ідентифікації всіх апаратних, механічних та інших типів пристроїв, які необхідні для виконання системою всіх функцій. Спочатку специфікуються обчислювальні вузли системи, що містять процесор і пам'ять. Окремі вимоги до складу апаратних засобів можуть задаватись у формі обмежень і помічених значень. Подальша деталізація діаграми розгортання пов'язана з розміщенням усіх виконуваних компонентів діаграми у вузлах системи.

Діаграма розгортання створюється з урахуванням типу архітектури програмного продукту (ПП):

- *Rich Client Application (RCA)* – класичний інстальований ПП, який виконується на персональних комп'ютерах під управлінням однієї з платформ Windows/Linux /Mac/Qt/JVMJ2SE/.NET.
- *Mobile Application (MA)* – мобільний ПП, розроблений під одну з платформ Android/iOS/J2ME/Blackberry, який, як правило, виконується на пристроях з обмеженими апаратними ресурсами.
- *Service Application (SA)* – сервісний ПП, який виконує функції сервера застосунків, і представляє зовнішній інтерфейс за архітектурним стилем REST (Representational State Transfer), наприклад, протокол SOAP (Simple Object Access Protocol), RMI (Remote Method Invocation) й ін.
- *WEB Application (WA)* – класичний WEB ПП, який надає WWW інтерфейс за допомогою HTML/DHTML.
- *Rich WEB Application (RWA)* – спеціальний тип WEB ПП, який ґрунтується на HTML5+JavaScript, що виконується, як правило, на стороні клієнта і використовує його процесорний час і ресурси та досить часто складається з двох «половинок» – серверне та клієнтське ПЗ.
- *Calculating Application (CA)* – обчислювальний ПП, головною метою якого є демонстрація працездатності деякого алгоритму, його коректності, швидкості роботи або адекватності вихідних значень.
- *Framework or Library* – бібліотека для подальшого використання під час розробки ПП, яка не є самостійним виконуваним ПП.
- *Mixed Application* – змішаний архітектурний тип ПП, який використовує комбінацію з декількох типів.

На рис. 6.2 UML діаграми розгортання для ПП RWA-типу представлена 3-а рівнями: розділити на три рівня роботи:

1) рівень представлення – Presentation Level (PL), який містить компоненти зовнішнього інтерфейсу (програмний інтерфейс користувача);

2) рівень бізнес-логіки – Business Level (BL), який містить апаратно-програмні компоненти обробки даних;

3) рівень зберігання даних – Access Level (AL), який містить апаратно-програмні компоненти керування даними.

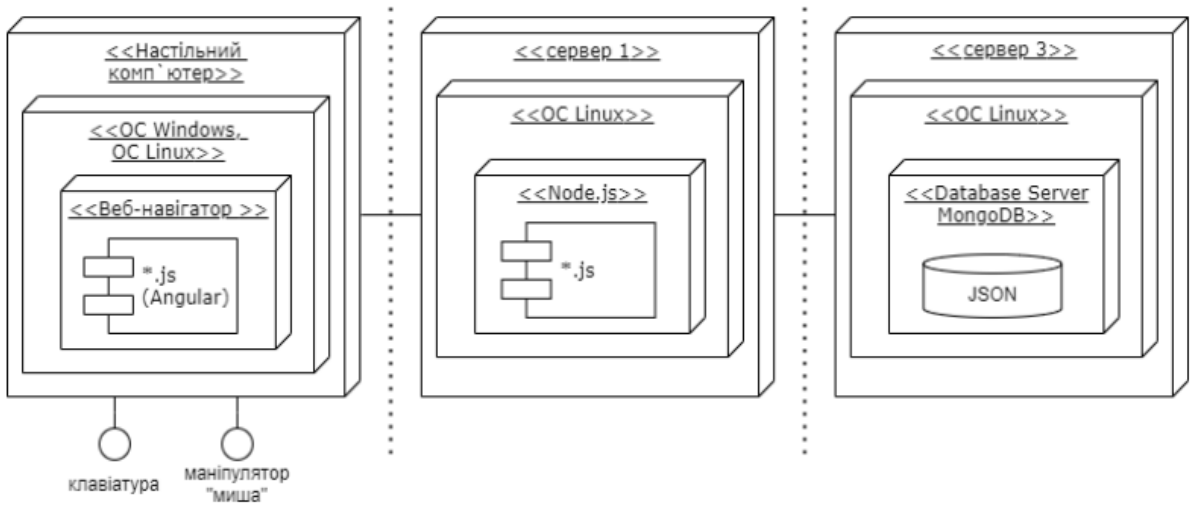

Рис. 6.2. Приклад UML діаграми розгортання для ПП RWA-типу

Завдання

1. Представили архітектуру нового програмного продукту (відповідно до варіанта) за допомогою UML діаграми розгортання. UML діаграма розгортання має відповідати обраному вами архітектурному типу: Rich Client Application (RCA), Mobile Application (MA), Service Application (SA), WEB Application (WA), Rich WEB Application (RWA). На діаграмах розгортання має бути визначено обрані вами типи платформи, протокол інтерфейсу: REST, SOAP, RMI, HTML. Модель представлення структур даних – реляційні, XML, JSON.

2. Створити каталог «5-Architecture» з файлом README.md. У файлі README.md мають бути рядки: «Проєктування програмного продукту». Додайте підкаталог «2.1-ArchitectureDesign» з файлом README.md. У файлі README.md мають бути UML діаграми розгортання.

3. Підготувати звіт, який включатиме результати виконання кроків завдання; відповіді на контрольні запитання; посилання на гілку в GitHub.

Контрольні запитання

1. Які архітектурні типи програмного продукту вам відомі?
2. Які три рівні представлення визначають під час опису архітектури?
3. Які основні принципи декомпозиції вам відомі?
4. Що означає, коли програмний модуль має Low Coupling?

Лабораторна робота №7.
Збереження ієрархічних структур даних у реляційній базі даних

Мета роботи: здобути навички збереження ієрархічних даних в таблиці.

Теоретичні відомості

Ієрархічні дані є набором елементів даних, пов’язаних між собою ієрархічними зв’язками. Ієрархічні зв’язки – це зв’язки, в яких один із елементів даних є батьком іншого елемента. Приклади ієрархічних даних, які зазвичай зберігаються у базах даних, включають таке:

- Організаційна структура.
- Файлова система.
- Група завдань у проєкті.
- Класифікація умовних термінів.
- Діаграма зв’язків між веб-сторінками.

Способи збереження ієрархічні даних в таблиці включають:

- Тип даних hierarchyid.
- Спосіб «батьки–нащадки».
- Інструкція OPENXML, тип даних XML.

Вбудований тип даних hierarchyid використовується для створення таблиць з ієрархічною структурою або для опису ієрархічної структури даних, які зберігаються в іншому розташуванні. Значення типу даних hierarchyid представляє позицію в деревоподібній ієрархії. Значення hierarchyid мають такі властивості:

- Компактність – середня кількість біт, необхідна для представлення вузла в деревоподібної структурі з n вузлів, залежить від

середньої кількості нащадків біля вузла. Для невеликих рівнів розгалуження (0 – 7) цей розмір дорівнює $6\log_A n$, де A – середній рівень розгалуження.

- Порівняння проводиться у порядку пріоритету глибини – якщо задано два значення hierarchyid – a і b тоді $a<b$ означає, що значення a з'являється раніше значення b, якщо проходити по дереву з пріоритетним напрямком у глибину. Індекси для типів даних hierarchyid розташовуються в порядку пріоритету глибини, а вузли, що зустрічаються поряд під час проходу по дереву з пріоритетним напрямком глибини, зберігаються поряд один з одним. Наприклад, нащадки деякого запису зберігаються поруч із цим записом.
- Кодування в типі даних hierarchyid обмежено 892 байтами. Отже, вузли, що мають занадто багато рівнів, щоб умістится в 892 байти, не можуть бути представлені типом даних hierarchyid.

Тип даних hierarchyid зберігає відомості про один вузол у дереві ієрархії, у вигляді шляху від кореня дерева до цього вузла. Такий шлях представлений у вигляді послідовності міток усіх відвіданих дочірніх вузлів, починаючи з кореня, таким чином шлях до кореня представлений однією косою рисою. Для рівнів нижче кореня кожна мітка кодується як послідовності цілих чисел, розділених точками.

У табл. 7.1 представлено SQL для створення таблиці EmployeeOrg, що включає дані про співробітників та їхнє ієрархічне підпорядкування. Для простоти ця таблиця містить лише 5 стовпців: OrgNode – це стовпець типу hierarchyid, в якому зберігаються ієрархічні зв'язки. OrgLevel – це обчислюваний стовпець, заснований на стовпці OrgNode, в якому зберігаються дані про рівень кожного вузла в ієрархії. Ці дані будуть використовуватися для створення індексу ширини. Стовпець EmployeeID містить типові ідентифікаційні номери працівників, які використовуються для таких завдань, як розрахунок заробітної плати. Стовпець EmpName містить ім'я співробітника. Стовпець Title містить посаду співробітника.

Таблиця 7.1

SQL для створення таблиці для збереження ієрархічних даних за допомогою типу hierarchyid

```
CREATE TABLE EmployeeOrg
(
OrgNode hierarchyid PRIMARY KEY CLUSTERED,
OrgLevel AS OrgNode.GetLevel(),
EmployeeID int UNIQUE NOT NULL,
EmpName varchar(20) NOT NULL,
Title varchar(20) NULL
)
```

У разі використання підходу «Батьки–нащадки» кожен рядок містить посилання на батьківський елемент. В табл. 7.2 представлено SQL для створення таблиці для зберігання рядків способом «батьки–нащадки».

Таблиця 7.2

SQL для створення таблиці для збереження ієрархічних даних способом «батьки–нащадки»

```
CREATE TABLE ParentChildOrg
 (
  BusinessEntityID int PRIMARY KEY,
  ManagerId int REFERENCES ParentChildOrg(BusinessEntityID),
  EmployeeName nvarchar(50)
 )
```

Метод «батьки–нащадки» є ефективнішим, коли структура нескінченних піддерев часто змінюється. В ієрархічній структурі «батьки–нащадки» зміна розташування рядка ієрархії стосується лише одного рядка. Якщо використовується тип hierarchyid для збереження, тоді зміна розташування рядка впливатиме на *n* рядків, де *n* – кількість вузлів у піддереві, що переміщується.

Інструкція OPENXML – це ключове слово Transact-SQL, яке дозволяє отримати доступ до XML-даних, схожим чином, як до реляційних наборів. Це робиться за допомогою представлення внутрішнього

відображення документа XML у вигляді набору рядків, записи в наборі рядків можуть зберігатись у таблицях баз даних.

Для виконання запитів до XML-документа з використанням OPENXML, необхідно спочатку викликати процедуру sp_xml_preparedocument. Таким чином, проводиться синтаксичний аналіз документа XML і повертається дескриптор для проаналізованого документа. Проаналізований документ є поданням дерева об'єктної моделі документа (DOM) різних вузлів у XML-документі. Дескриптор документа передається OPENXML. Потім інструкція OPENXML видає подання документа як набору рядків. Внутрішнє представлення XML-документа повинно бути видалено з пам'яті за допомогою виклику системної процедури sp_xml_removedocument для звільнення пам'яті.

В табл. 7.3 наведено SQL сценарій, в якому створюється таблиця для збереження даних документа XML за допомогою інструкції OPENXML; інструкція rowpattern (/ROOT/Customer) визначає вузли для обробки; параметр flags має значення 1, яке вказує на зіставлення з використанням атрибутивної моделі. В результаті XML-атрибути зіставляються зі стовпцями у наборі рядків, визначеному в елементі SchemaDeclaration.

Таблиця 7.3

Приклад збереження даних документа XML за допомогою інструкції OPENXML

```
-- Create tables for later population using OPENXML.
CREATE TABLE Customers (CustomerID varchar(20) primary key,
    ContactName varchar(20),
    CompanyName varchar(20));
GO
CREATE TABLE Orders( CustomerID varchar(20), OrderDate datetime);
GO
DECLARE @docHandle int;
DECLARE @xmlDocument nvarchar(max); -- or xml type
SET @xmlDocument = N'<ROOT>
<Customers CustomerID=«XYZAA» ContactName=«Joe»
CompanyName=«Company1»>
  <Orders CustomerID=«XYZAA» OrderDate=«2000-08-25T00:00:00»/>
```

```
<Orders CustomerID=«XYZAA» OrderDate=«2000-10-03T00:00:00»/>
</Customers>
<Customers CustomerID=«XYZBB» ContactName=«Steve»
CompanyName=«Company2»>No Orders yet!
</Customers>
</ROOT>';
 EXEC sp_xml_preparedocument @docHandle OUTPUT,
@xmlDocument;
 INSERT Customers
 SELECT *
 FROM OPENXML(@docHandle, N'/ROOT/Customers')
  WITH Customers;
 INSERT Orders
 SELECT *
 FROM OPENXML(@docHandle, N'//Orders')
  WITH Orders;
 SELECT * FROM OPENXML(@docHandle,
N'/ROOT/Customers/Orders')
  WITH (CustomerID nchar(5) '../@CustomerID',
 OrderDate datetime);
 EXEC sp_xml_removedocument @docHandle;
```

Завдання

1. Створити фізичну модель даних для збереження відповідно логічній структурі розробленій в лабораторній роботі №5.

2. Створити дані для тестування програмного продукту відповідно логічній структурі, розробленій в лабораторній роботі №5 у вигляді XML-документа.

3. Написати три SQL сценарії для збереження даних XML-документа в створеній таблиці трьома різними способами: тип даних hierarchyid, «батьки–нащадки», тип даних XML. Виконати кожен SQL сценарій.

4. Підготувати звіт, який включатиме результати виконання кроків завдання; відповіді на контрольні запитання.

Контрольні запитання

1. Під час збереження XML-документа за допомогою типу hierarchyid, за допомогою яких функцій є можливим перехід до батьківського вузла? До вузлів–нащадків?

2. Перечислить можливі варіанти збереження XML-документа в реляційній базі даних?

3. Опишіть, з якими параметрами виконується процедура sp_xml_preparedocument?

4. Яким чином можна зіставити поля таблиці з полями XML-документа під час виконання OPENXML?

Список літератури

1. Bourque, P. (2020, November). The SWEBOK Guide—More Than 20 Years down the Road. In 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T) (pp. 1-2). IEEE.

2. Laplante, P. A., & Kassab, M. (2022). Requirements engineering for software and systems. Auerbach Publications.

3. Johnson, J. (2020). Designing with the mind in mind: simple guide to understanding user interface design guidelines. Morgan Kaufmann.

4. Richards, M., & Ford, N. (2020). Fundamentals of software architecture: an engineering approach. O'Reilly Media.

5. Badia, A. (2020). SQL for data science: data cleaning, wrangling and analytics with relational databases. Springer Nature.

Навчально-методичне видання

Системна інженерія програмного забезпечення

Методичні вказівки та завдання
до виконання практичних та лабораторних робіт (1–7)
для підготовки здобувачів другого (магістерського) рівня
вищої освіти спеціальності 121 «Інженерія програмного забезпечення»

Укладач **Соловей** Ольга Леонідівна

Випусковий редактор *Л. С. Тавлуй*
Комп'ютерне верстання *К. А. Мавроді*

Підписано до друку 23.01.2025. Формат 60 х $84_{1/16}$
Ум. друк. арк. 2,09. Обл.-вид. арк. 2,25.
Електронний документ. Вид. № 131/ІІІ-24

Видавець і виготовлювач:
Київський національний університет будівництва і архітектури

Проспект Повітряних Сил, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб'єктів
видавничої справи ДК № 808 від 13.02.2002