МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

С. О. КАРПОВА

ПРОФЕСІЙНА ПРАКТИКА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Методичні вказівки до виконання лабораторних робіт

Рекомендовано Методичною радою НУК

ВИДАВНИЦТВО
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КОРАБЛЕБУДУВАННЯ
ІМ. АДМІРАЛА МАКАРОВА

2024

УДК 004.4`2
К26

Автор С. О. Карпова, старш. викладач

Рецензент О. М. Дудченко, канд. техн. наук, професор

Рекомендовано Методичною радою НУК

Карпова С. О.

К26 Професійна практика програмної інженерії : методичні вказівки до виконання лабораторних робіт / С. О. Карпова. – Миколаїв : НУК, 2024. – 52 с.

Вміщено методичні вказівки до виконання й оформлення лабораторних робіт з дисципліни «Професійна практика програмної інженерії», а також перелік літератури.

Призначено для студентів галузі знань 12 «Інформаційні технології» спеціальності 121 «Інженерія програмного забезпечення» (освітньо-професійної програми «Інженерія програмного забезпечення»).

УДК 004.4`2

ВСТУП

Програмна інженерія (*software engineering*) – наука побудови комп'ютерних програмних систем, що містить у собі теоретичні концепції, методи і засоби програмування, технологію програмування, системи та інструменти їхньої підтримки, сучасні стандарти, зокрема, процеси життєвого циклу, вимірювання, оцінювання якості розробки програмних продуктів.

Головне призначення програмної інженерії – побудова програмних продуктів, починаючи з аналізу предметної галузі й закінчуючи виготовленням вихідного коду для виконання на комп'ютері.

Фундаментальну основу побудови програмних продуктів становлять: теорія алгоритмів, математична логіка, теорія обчислень, теорія керування та інше. Колективна розробка великих програмних проєктів зумовила розвиток інженерних, технологічних методів і засобів регламентованого проєктування програмних систем з урахуванням організаційних процесів життєвого циклу: інженерії вимог, керування ризиком і якістю, планування і регулювання ресурсів, оцінювання процесів життєвого циклу та показників якості, вартості і строків виготовлення програмного продукту.

Метою вивчення навчальної дисципліни «Професійна практика програмної інженерії» є:

– здатність розв'язувати складні спеціалізовані завдання або практичні проблеми інженерії програмного забезпечення, що характеризуються комплексністю та невизначеністю умов, із застосуванням теорій та методів інформаційних технологій;

– здатність дотримуватися специфікацій, стандартів, правил і рекомендацій у професійній галузі під час реалізації процесів життєвого циклу;

– здатність накопичувати, обробляти та систематизувати професійні знання щодо створення і супроводження програмного забезпечення та визнання важливості навчання протягом усього життя.

Лабораторна робота № 1

ПЛАНУВАННЯ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОЇ СИСТЕМИ. РОЗРОБКА ТИМЧАСОВОЇ ДІАГРАМИ

Мета роботи: засвоїти основні поняття процесу розробки програмної системи. Розробити тимчасову діаграму Ґанта.

Короткі теоретичні відомості

Процес розробки програмного забезпечення (англ. *software development process, software process*) – сукупність низки послідовних дій, спрямованих на розробку програмного забезпечення (ПЗ) [8].

Існує кілька моделей, кожна з яких описує свій підхід, у вигляді завдань і/або діяльності, які мають місце в ході процесу. Вибір тієї або іншої моделі здійснюється відповідно до обраної методології розробки програмного забезпечення.

Методологія розробки програмного забезпечення (англ. *software development methodology*) – сукупність методів, застосованих на різних стадіях життєвого циклу розробки програмного забезпечення, що мають спільний філософський підхід, та відповідно до цього підходу дозволяють забезпечити найкращу ефективність процесів розробки.

Кожна модель розробки програмного забезпечення має свої унікальні особливості, переваги та недоліки. Визначити, яка з них краща – не можна, оскільки під різні завдання, продукти та ідеї обирається свій принцип розробки.

Основні методології розробки:

Waterfall Model (каскадна модель або «водоспад») – це класична модель, яка використовувалася на початку епохи розробки, але продовжує активно застосовуватися. Її принцип роботи досить простий: кожний наступний етап виконується лише тоді, коли повністю завершено попередній. Існує чітке розподілення етапів, й технологія реалізується ніби каскадом, поступово рухаючись від першого до останнього етапу.

Incremental Model (інкрементна модель) розробки програмного забезпечення підходить у тому випадку, якщо є чіткий план дій, але продукт потрібно запустити досить швидко, а зміни можна буде вносити потім. Її суть полягає в тому, що спочатку розробляється план дій та сегментується на невеликі завдання. Далі кожен блок розробляється за традиційною каскадною моделлю. Спочатку робиться «базовий» продукт з мінімальними, але важливими функціями. Поступово він доповнюється завдяки розробці інших компонентів, які називаються *інкрементами*. Процес зациклюється, доки не буде повністю зібраної єдиної системи.

Iterative Model (ітеративна або ітераційна модель) суть ітеративної методології розробки програмного забезпечення полягає в тому, щоб створити базовий функціонал та поступово його покращувати.

Agile Model (гнучка методологія розробки) – це чудове рішення для створення продукту, який не до кінця сформований у своїй ідеї. Особливість даного методу полягає в тому, що замовник може одразу спостерігати за змінами у розробці та коригувати дії. Це можливо завдяки визначенню спринтів – відрізків, у яких виконуються завдання. Під час обговорення ставляться цілі, визначається час відрізка, на якому розробники виконують завдання. Далі відбувається обговорення, вносяться зміни, призначається новий відрізок.

Spiral Model (спіральна модель) підходять для великих проєктів, де припущення помилок призводить до великих втрат та негативних наслідків. Акцент робиться на оцінці ризиків та опрацюванні конкретних бізнес-завдань. Принцип роботи наступний: робота йде по спіралі, на кожному витку якої здійснюється чотири основні дії – створення плану, аналіз ризиків, розробка та конструювання, оцінка результату та збирання відгуків. Якщо всі чотири етапи успішно пройдені, то технологія переходить на новий виток.

V-Model (*V*-подібна модель). Особливість підходу полягає в наголосі на тестування та перевірку працездатності систем

під час розробки. Тести виконуються одночасно із самим процесом створення продукту. Сам принцип успадковує базовий підхід під час каскадної розробки. Процес іде покроково, існує чіткий план дій, складається суворе технічне завдання. Але паралельно виконуються тести, у разі виявлення помилок вони одразу виправляються, незалежно від етапу розробки.

RAD Model (*rapid application development model* або швидка розробка додатків). Методологія розробки програмного забезпечення *RAD* підходить тим компаніям, які хочуть максимально швидко запустити продукт. Суть у тому, що етапи створення додатка розподіляються на кілька окремих блоків, з кожним із яких працює окрема команда розробників. Після цього робочі невеликі модулі збираються до єдиної системи та утворюють робочий прототип, який можна показати клієнту. Далі збираються відгуки та вносяться зміни.

Розробка якісного продукту починається з визначення його життєвого циклу. Життєвий цикл програмного забезпечення – це сукупність окремих етапів робіт, що проводяться в заданому порядку протягом періоду часу, який починається з вирішення питання про розроблення програмного забезпечення і закінчується припиненням використання програмного забезпечення [4].

Склад процесів життєвого циклу регламентується Міжнародними стандартами:

ISO / 1EC 12207:1995 – «*Information Technologe – Software Life Cycle Processes*» («Інформаційні технології – процеси життєвого циклу програмного забезпечення»).

ISO – *International Organization for Standardization* – Міжнародна організація із стандартизації.

IEC – *International Electrotechnical Commission* – Міжнародна комісія з електротехніки. Цей стандарт описує структуру життєвого циклу програмного забезпечення та його процеси.

Використання поняття життєвого циклу дозволяє вибрати найбільш ефективні підходи для задач певного етапу життя ПЗ.

Залежно від особливостей процесів розробки і супроводу програм існують різні моделі ЖЦ. Використання певної моделі ЖЦ дозволяє визначитися з основними моментами процесу замовлення, розробки і супроводу ПЗ навіть недосвідченому програмісту. Також використання моделей дозволяє чітко зрозуміти, у який період переходити від версії до версії, та які дії з удосконалення виконувати і на якому етапі.

Процес розробки (*development process*) складається з таких етапів:

- аналізу вимог – специфікації програмного забезпечення;
- проєктування програмного забезпечення;
- програмування;
- тестування програмного забезпечення;
- системної інтеграції;
- впровадженні програмного забезпечення (або установки програмного забезпечення);
- супроводі програмного забезпечення.

Процес розробки передбачає дії і завдання, що виконуються розробником, і охоплює роботи зі створення програмного забезпечення та його компонентів відповідно до заданих вимог, включаючи оформлення проєктної та експлуатаційної документації, а також підготовку матеріалів, необхідних для перевірки працездатності та відповідності якості програмних продуктів, матеріалів, необхідних для навчання персоналу і т. п.

Протягом усього життєвого циклу проєкту здійснюються процеси планування. Планування проєкту (*project planning*) – безперервний процес визначення найкращого способу дій для досягнення поставлених цілей проєкту з урахуванням обстановки, що складається. Планування є найбільш важливим процесом управління проєктом (управління проєктами (англ. *project management*) – галузь знань з планування, організації та управління ресурсами з метою успішного досягнення цілей та завершення завдань проєкту) – це процес формування рішень, що визначають порядок, у якому повинна відбуватися послідовність окремих заходів, дій та робіт за проєктом.

На стадії планування визначається організація, методи та засоби управління здійсненням проєкту як цілісної системи, так і в розрізі окремих її етапів та елементів.

Одним з методів планування та управління проєктами є діаграма Ґанта.

Діаграма Ґанта (також стрічкова діаграма, графік Ґанта) – діаграма, яка використовується для ілюстрації плану, графіка робіт за будь-яким проєктом (рис. 1.1)

Рис.1.1. Діаграма Ґанта «Процес розробки програмного забезпечення»

Діаграма Ґанта являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, складові плану, розміщуються по вертикалі. Початок, кінець і довжина відрізка на шкалі часу відповідають початку, кінцю і тривалості завдання. На деяких діаграмах Ґанта також показується залежність між завданнями.

Завдання

Розробити діаграму Ґанта у відповідності до отриманого індивідуального завдання та оформити її згідно з поданим нижче зразком та викладеними теоретичними відомостями.

Лабораторна робота № 2

РОЗРОБКА ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета роботи: засвоїти основні поняття про вимоги, що висуваються до програмного забезпечення. На основі аналізу предметної галузі сформувати вимоги до розроблюваної системи.

Короткі теоретичні відомості

Вимоги до програмного забезпечення – це набір вимог щодо властивостей, якості та функцій програмного забезпечення, що буде розроблено, або знаходиться у розробці. Вимоги визначаються в процесі аналізу вимог та фіксуються в специфікації вимог, діаграмах варіантів використання та інших артефактах процесу аналізу та розробки вимог [12].

Розробка вимог до програмної системи може бути поділена на декілька етапів:

- знаходження вимог (збір, визначення потреб зацікавлених осіб та систем);
- аналіз вимог (перевірка цілісності та закінченості;
- специфікація (документування вимог);
- тестування вимог.

Карл Вігерс визначає три рівні вимог до програмного забезпечення [3]:

Бізнес-вимоги (*business requirements*) – визначають призначення ПЗ, можуть описуватися в документі про бачення (англ. *vision*) та документі про межі проєкту (англ. *scope*).

Вимоги користувача (*user requirements*) – визначають набір завдань користувача, які повинна вирішувати програма, а також сценарії їхнього вирішення в системі. Ці вимоги можуть мати вигляд тверджень, варіантів використання, історій користувача, сценаріїв взаємодії.

Функціональні вимоги (*functional requirements*) – визначають «що» повинен робити програмний продукт. Ці вимоги описуються в документі «Специфікація вимог до програмного забезпечення».

Специфікація вимог до програмного забезпечення (англ. *software requirements specification* (*SRS*) – це повний опис поведінки системи, що розробляється. Вона включає в себе множину прецедентів, що описують усі взаємодії, які користувачі мають з програмним забезпеченням. Прецеденти також відомі як функціональні вимоги. На додачу до прецедентів *SRS* також включає в себе нефункціональні (чи додаткові) вимоги. Нефункціональні вимоги є вимогами, які накладають обмеження на проєкт чи реалізацію (такі як вимоги інженерії продуктивності, стандарти якості, чи обмеження проєктування).

Функціональні вимоги визначають функціональність ПЗ, яку розробники повинні забезпечити, щоб користувачі змогли виконати свої завдання в межах бізнес-вимог. Інколи їх називають вимоги поведінки (*behavioral requirements*), вони містять положення з традиційним «повинна». Наприклад, «Система повинна електронною поштою відправляти користувачу підтвердження замовлення». Функціональні вимоги описують, що розробнику необхідно реалізувати. Терміном «системні вимоги» (*system requirements*) позначають високорівневі вимоги до ПС, які містять багато підсистем.

До того ж кожна система характеризується нефункціональними вимогами. Модель на рис. 2.1 ілюструє спосіб представлення вимог, схематично показує організацію вимог. Овал – позначає тип інформації для вимог, прямокутник спосіб зберігання інформації (документи, діаграми, бази даних) [9].

Види вимог за характером:

Функціональний характер – вимоги до поведінки системи:

– бізнес-вимоги;

– вимоги користувача;

– функціональні вимоги.

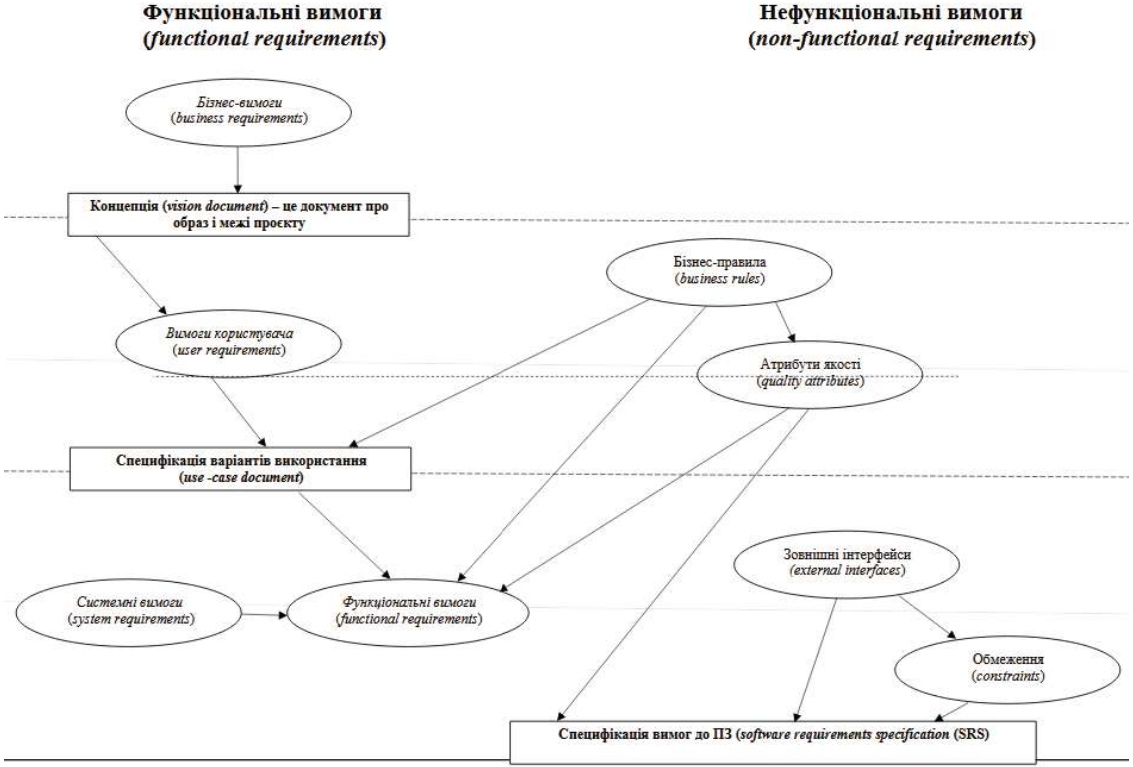


Рис. 2.1. Взаємозв’язки кількох типів інформації для вимог

Нефункціональний характер – вимоги до характеру поведінки системи:

– бізнес-правила (*business rules*) – визначають обмеження, що витікають з предметної галузі;

– системні вимоги (*system requirements*) – вимоги до програмних інтерфейсів, надійності, обладнання;

– атрибути якості (*quality attributes*);

– зовнішні системи та інтерфейси (*external interfaces*);

– обмеження (*constraints*).

Функціональні вимоги (*functional requirements*) – це вимоги до програмного забезпечення, які описують внутрішню роботу системи, її поведінку: обчислення даних, маніпулювання даними, обробку даних та інші специфічні функції, які має виконувати система. На відміну від нефункціональних вимог, які визначають якою система повинна бути, функціональні вимоги визначають, що система повинна робити. Функціональні вимоги до програмного забезпечення визначаються на першій стадії процесу розробки ПЗ – на етапі аналізу вимог [18].

Нефункціональні вимоги (англ. *non-functional requirements*) – це вимоги до програмного забезпечення, які задають критерії для оцінки якості його роботи [15]. Нефункціональні вимоги до програмного забезпечення визначаються стандартом ISO/IEC/IEEE 29148:2018 [29].

На відміну від функціональних вимог, які визначають, що система повинна робити, нефункціональні вимоги визначають якою система повинна бути. Нефункціональні вимоги до програмного забезпечення визначаються на першій стадії процесу розробки ПЗ – на етапі аналізу вимог.

Нефункціональні вимоги можна поділити на дві категорії:

• покращення (безпека, надійність, швидкодія, зручність у використанні та ін.);

• удосконалення (маштабування, відновлюваність та ін.) властивостей системи.

Види нефункціональних вимог:

Вимоги до інтерфейсу (*interface requirements*):

• апаратні інтерфейси (*hardware interfaces*) – апаратні інтерфейси необхідні для підтримки системи, включаючи в себе логічну структуру, фізичні адреси й очікувану поведінку;

• інтерфейси ПЗ (*software interfaces*) – інтерфейси програмного забезпечення з якими аплікація повинна взаємодіяти;

• комунікаційні інтерфейси (*communications interfaces*) – інтерфейси для комунікацій (взаємодії) з іншими системами та/або пристроями.

Апаратні та програмні вимоги (*hardware/software requirements*) – опис апаратної та програмної платформ, необхідних для роботи (і підтримки) системи.

Операційні вимоги (*operational requirements*):

• безпека та конфіденційність (*security and privacy*);
• надійність (*reliability*);
• відновлюваність (*recoverability*);
• продуктивність (*performance*);
• потенціал (*capacity*);
• збереження даних (*data retention*);
• керування помилками (*error handling*);
• правила перевірки (*validation rules*);
• узгоджені стандарти (*convention standards*).

Завдання

Розробити у відповідності до отриманого індивідуального завдання функціональні та нефункціональні вимоги.

Приклад 1. Розробка програмного модуля «Редактор математичних моделей».

Основні функції системи:

• можливість вибору математичної моделі;

• відображення математичної моделі на екран користувача;

• створення тесту, де система генерує вхідні змінні математичної моделі і змінну, значення якої потрібно знайти;

• відображення на рисунку математичної моделі тих параметрів, які обрала система випадковим чином;

• введення власних значень параметрів моделі;

• можливість відображення підказки для розв'язання конкретної задачі математичної моделі;

• розрахування відповіді на випадкове питання математичної моделі.

Функціональні вимоги

Система повинна:

• відображати перелік математичних моделей;

• надавати інформацію про конкретну модель, яка повинна містити її креслення, систему позначень, загальні формули для вирішення тестового завдання;

• генерувати кілька вхідних параметрів математичної моделі та параметр, значення якого потрібно знайти;

• обчислювати відповідь після введення користувачем значень випадкових параметрів;

• відображати підказки до розв'язання задачі після натискання іконки «?»;

• мати прозору навігацію та чітке розуміння для користувача значення іконок та кнопок на сторінці.

Нефункціональні вимоги

Система повинна:

• відкриватися не більше 3 секунд;

• бути доступною користувачам для використання в будь-який час;

• коректно відображатися та функціонувати в наступних браузерах: *Chrome, FireFox, Safari, Opera, Internet Explorer*;

• мати резервне копіювання і зберігатися на локальному комп'ютері або жорсткому диску.

Слід зазначити, що особливих вимог щодо пам'яті під час використання системи, немає.

Приклад 2. Розробка вимог до апаратного забезпечення сервера для коректної роботи вебзастосунку «CashJet» [20].

Основні функції системи

Користувач, зареєстрований у системі, відповідно до своєї посади та прав доступу має (може):

• свій акаунт, де збережена вся інформація про його рахунок, поточний стан, усі витрати, чеки, також інформація про самого користувача;

• можливість сортувати витрати за зростанням / спаданням, обрати період, за який відображати витрати;

• додати витрати за певний день;

• редагувати витрати, незалежно від дати додавання;

• можливість бачити графік ідеальних витрат, що формується системою і власний, для порівняння;

• змінити адресу електронної пошти та пароль;

• можливість створювати шаблонні операції;

• вибрати зручну мову за бажанням.

Функціональні вимоги:

1) код вимоги – цифровий / цифро-літерний уніфікований ідентифікатор вимоги;

2) автор вимоги – посада, прізвище, ім'я, по батькові;

3) опис вимоги – описується суть вимоги доступною для розуміння користувачів лексикою;

4) код сценарію використання – зазначається код сценарію використання для зв'язку реєстру зі специфікацією сценарію використання;

5) примітки – додаткова інформація, яка може бути корисна для розробки програмного забезпечення.

Таблиця 2.1. **Реєстр вимог**

Код вимоги	Автор вимоги	Опис вимоги	Код сценарію використання	Примітки
B1	Розробник серверної частини системи	Використання фреймворку *NodeJS* на мові програмування *JavaScript* для написання серверної частини програмного застосунку	L1-L10	Інформацію про зазначені інструменти розробки можна прочитати тут _
…	…	…	…	…
B3	Розробник серверної частини системи	Використання як сховище даних СКБД *SqlLite3*	S1-S2	Інформацію про зазначені інструменти розробки можна прочитати тут_

Таблиця 2.2. **Сценарії використання (описуються всі сценарії) L1**

Код сценарію використання	L1
Назва	Реєстрація користувача в системі
Розробив	ПІБ
Дата розробки	Число. Місяць. Рік
Актори	Користувач. Адміністратор
Дія	Користувач реєструється в системі
Попередні умови	Посилання на платформу
Подальші умови	Можливість використання всіх функцій системи
Нормальна послідовність дій	1. Введення свого імені. 2. Введення e-mail. 3. Введення пароля. 4. Перехід на сторінку «Про себе» уже як авторизований користувач
Альтернативний хід	Відсутній
Винятки	Залежно від прав користувача присутні різні рівні доступу

Продовж. табл. 2.2

Код сценарію використання	L1
Пріоритет	1
Частота використання	Використовується лише один раз під час створення користувача
Бізнес-правила	Інтуїтивно зрозумілий інтерфейс
Спеціальні вимоги	Цілодобовий доступ Захищеність даних про користувачів
Графічна модель	Див. рис. _

Таблиця 2.3. **Сценарії використання (описуються всі сценарії) L10**

Код сценарію використання	L10
Назва	Удосконалення системи
Розробив	ПІБ
Дата розробки	Число. Місяць. Рік
Актори	Розробники системи
Дія	Створення удосконаленої версії для користувача
Попередні умови	Система повинна мати потенційних покупців, що знають про систему
Подальші умови	Продовжують користуватися системою
Нормальна послідовність дій	1. Додавання нового функціоналу. 2. Випуск нової версії з новим функціоналом. 3. Удосконалення мобільної версії
Альтернативний хід	Відсутній
Винятки	Користувачі, які не мають доступу до системи
Пріоритет	10
Частота використання	Удосконалення системи не частіше, ніж 1 раз на місяць
Бізнес-правила	Створення системи згідно з побажаннями споживачів
Спеціальні вимоги	Проведення опитувань, аби сформувати список побажань
Графічна модель	Див. рис. _

Нефункціональні вимоги:

• вимоги до мереж;

• вимоги до серверів;

• вимоги до клієнтських комп'ютерів;

• вимоги до мобільних пристроїв.

Вимоги до мереж

Підтримувані протоколи передачі даних: HTTP/HTTPS.

Підключення до мережі може бути здійснено як за допомогою дротового способу, так і бездротовим дистанційним способом (наприклад, за допомогою Wi-Fi). Головне, щоб забезпечувався стабільний зв'язок з мережею «Інтернет» з такими параметрами:

• мінімальна пропускна здатність – 1 Мбіт/с;

• рівень втрати селевих пакетів – не більше ніж 4 % (1 пакетна втрата на 25 разів обміну пакетами).

Під час взаємодії з частинами сервісу, які передбачають роботу із завантаженням, скануванням чеків, рекомендовані параметри зв'язку з мережею «Інтернет»:

• Рекомендована пропускна здатність – 100 Мбіт/с.

• Рівень втрати селевих пакетів – не більше ніж 0,012 % (3 пакетних втрати на 250 разів обміну пакетами).

• Управління об'єктами мережі повинно бути організовано за допомогою доменів або робочих груп.

Вимоги до серверів

Головні завдання сервера:

• Регулювати роботу інтернет-провайдера. Саме він підключає користувача до мережі і постачає йому трафік, будучи свого роду основою.

• Зберігати інформацію. Це більш надійно, ніж збирати всі потрібні фотографії і документи на своєму ПК.

• Хостинг. Це місце, де зберігаються сайти і всі документи, пов'язані з ними.

Відповідно до головних завдань сформовано основні характеристики сервера:

• Цілодобовий доступ.

• Оперативна пам’ять (ОЗП) не менше 8ГБ.

• Постійна пам’ять (HDD) не менше 300 Mb вільного об’єму.

• Пропускна спроможність каналу зв’язку для кожного з користувачів: мінімальна 50 Кбіт/с, рекомендована – 100 Кбіт/с.

• Одночасна підтримка роботи з п’ятдесятьма користувачами, тобто рекомендована пропускна спроможність каналу зовнішніх підключень сервера – 5000 Кбіт/с.

• Початковий об’єм пам’яті БД – 4 ГБ з можливістю подальшого розширення.

Вимоги до клієнтських комп’ютерів

Клієнтський комп’ютер повинен повністю забезпечувати можливість доступу до мережі та здійснення операцій. Рекомендовані характеристики комп’ютера:

• Процесор: Core i3 або аналогічні.

• Оперативна пам’ять (ОЗП) – 4 ГБ.

• Операційна система: Windows, Linux, MacOS.

• Браузери Google, Chrome, Mozilla останніх версій.

Вимоги до мобільних пристроїв

Оскільки CashJet – WEB-додаток, тому мобільний пристрій може отримати доступ до системи у будь-який зручний момент. У MVP не реалізована можливість зручного доступу з мобільного пристрою, тому рекомендовані характеристики мобільних пристроїв зазначені нижче:

• Браузери Google Chrome, Mozilla останніх версій.

• Операційна система: Windows, Android, iOS.

• Оперативна пам’ять (ОЗП) – 1 ГБ.

• Кількість ядер – 4.

Лабораторна робота № 3

РОЗШИРЕННЯ ФУНКЦІОНАЛЬНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ, ДІАГРАМА КЛАСІВ, ДІАГРАМА ПОСЛІДОВНОСТЕЙ, ДІАГРАМА КОМПОНЕНТІВ)

Мета роботи: ознайомитись з основними компонентами побудови діаграм та видами специфікацій. Навчитися розширювати функціональність програмного забезпечення для існуючого чи нового.

Короткі теоретичні відомості

Функціональність (*functionality*) – це здатність програмного забезпечення в певних умовах розв'язувати задачі, потрібні користувачам. Визначає, що саме робить ПЗ, які задачі воно розв'язує.

Розроблення нової функціональності для існуючого програмного додатка є доволі складним завданням, оскільки завжди є відмінність між початковим дизайном та поточною реалізацією, а також між підходами до програмування різних програмістів або команд. Для визначення поточного стану розробки додатка існують різні архітектурні інструменти, одними з яких є Visual Studio, який дозволяє зрозуміти наявний додаток та прийняті в ньому рішення, спроєктувати необхідну нову функціональність і перевірити відповідність реалізації проєктуванню [21].

Після того, як отримано більш-менш повне уявлення про те, як працює існуючий додаток, можна розробити нову функціональність програмного забезпечення. Проєктування вимагає стандартизації, діаграми Unified Modeling Language (UML) дозволяють представити структуру додатка у зрозумілому для інших вигляді. Наприклад, можна побудувати діа-

грами компонентів або діаграми класів UML, які описують існуючі елементи структури додатка, а потім додати в діаграми нові елементи, щоб проілюструвати й задокументувати зміни. Діаграму класів (*class diagram*) використовують для подання статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування.

Діаграми варіантів використання (*use case diagrams*) – використовують для моделювання бізнес-процесів організації (вимог до системи). Суть даної діаграми полягає в наступному: проєктована система подається у вигляді безлічі сутностей або акторів, що взаємодіють із системою за допомогою так званих варіантів використання. До того ж актором (*actor*) або дійовою особою називається будь-яка сутність, яка взаємодіє із системою ззовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, що може слугувати джерелом впливу на модельовану систему так, як визначить сам розробник. Зі свого боку варіант використання (*use case*) слугує для опису сервісів, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконується системою під час діалогу з актором.

Діаграма класів (*class diagram*) слугує для подання статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відбивати, зокрема, різні взаємозв'язки між окремими сутностями предметної галузі, такими як об'єкти та підсистеми, а також описує їх внутрішню структуру та типи відносин. На даній діаграмі не вказується інформація про тимчасові аспекти функціонування системи. З цієї точки зору діаграма класів є подальшим розвитком концептуальної моделі проєктованої системи.

Діаграми послідовності (*sequence diagrams*) відображають потік подій, що відбуваються в рамках варіанта використання. На діаграмі послідовності зображуються виключно ті об'єкти, які безпосередньо беруть участь у взаємодії та не

показуються можливі статичні асоціації з іншими об'єктами. Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів у часі. До того ж діаграма послідовності має як би два виміри. Один – зліва направо у вигляді вертикальних ліній, кожна з яких зображує лінію життя окремого об'єкта, який бере участь у взаємодії. Другий вимір діаграми послідовності – вертикальна тимчасова вісь, спрямована зверху вниз. Початковому моменту часу відповідає сама верхня частина діаграми. До того ж взаємодії об'єктів реалізуються за допомогою повідомлень, які посилаються одними об'єктами іншим.

Діаграма компонентів (*component diagram*) дозволяє визначити архітектуру розроблюваної системи, установивши залежності між програмними компонентами, у ролі яких може виступати вихідний, бінарний та виконуваний код. Діаграма компонентів описує особливості фізичного представлення системи. Дана діаграма застосовується для моделювання статичного вигляду системи з точки зору реалізації. За своєю сутністю діаграми компонентів – не що інше, як діаграми класів, сфокусовані на системних компонентах [22].

Завдання

Для існуючого чи нового програмного забезпечення розширити функціональність програмного забезпечення, побудувавши діаграми варіантів використання, класів, послідовностей, компонентів.

Лабораторна робота № 4

РОЗРОБКА ПРОЄКТУ КОРИСТУВАЛЬНИЦЬКОГО ІНТЕРФЕЙСУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета роботи: узагальнити та поглибити на практиці знання та навички розроблення графічного інтерфейсу програмної системи.

Короткі теоретичні відомості

Інтерфейс користувача (скорочено ІК), (англ. *user interface, UI*) – це засіб зручної взаємодії користувача (людини) з інформаційною системою. Сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача; у графічних системах інтерфейс користувача, втілюється багатовіконним режимом, змінами кольору, розміру, видимості (прозорості, напівпрозорості, невидимості) вікон, їх розташуванням, сортуванням елементів вікон, гнучкими налагодженнями як самих вікон, так і окремих їх елементів (файлів, теків, ярликів, шрифтів тощо), доступністю багатокористувацьких налаштувань. [14]

Етапи розробки користувальницького інтерфейсу

Розробка користувальницького інтерфейсу включає в себе ті ж основні етапи, що й розробка програмного забезпечення:

1) постановка задачі – визначення типу інтерфейсу та загальних вимог до нього;

2) аналіз вимог і визначення специфікацій – визначення сценаріїв використання та користувальницької моделі інтерфейсу;

3) проєктування – проєктування діалогів та їхня реалізація у вигляді процесів введення-виведення;

4) реалізація – програмування та тестування інтерфейсних процесів.

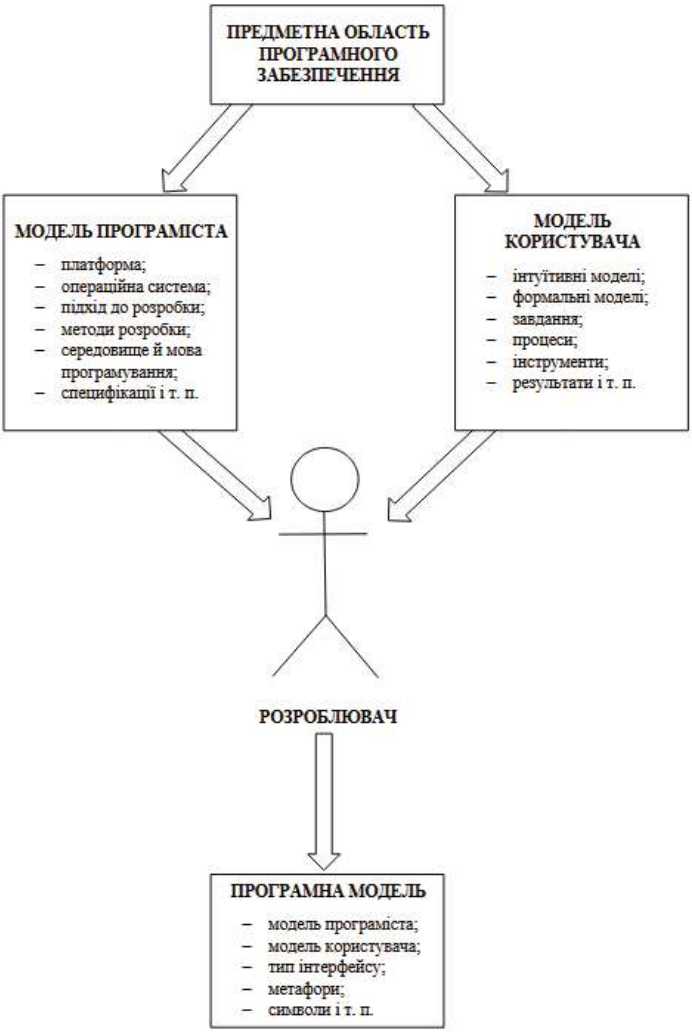


Рис. 4.1. Процес проєктування користувальницького інтерфейсу [25].

Процес проєктування інтерфейсу має орієнтуватися на користувача. Інтерфейс повинен взаємодіяти з користувачем на його «мові», бути логічним і послідовним. В інтерфейсі повинні бути довідкові засоби, які допомагають користувачам під час роботи із системою, та засоби відновлення після помилок.

На рис. 4.2 зображено ітераційний процес проєктування інтерфейсу користувача. Найбільш ефективним підходом до проєктування інтерфейсу користувача є розробка із застосуванням моделювання функцій користувача. На початку процесу прототипування створюються паперові макети інтерфейсу, потім розробляються екранні форми, що моделюють взаємодію користувача [11].

Графічний інтерфейс користувача (ГІК, англ. GUI, *Graphical user interface*) – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації [13].

Елементи інтерфейсу:

- кнопка (*button*);
- список (*list box*);
- випадаюче меню (*pull down menu*);
- список, що розкривається (*en:combo box, drop-down list*);
- прапорець / перемикач (*check box*);
- радіо-кнопка (*radio button*);
- поле редагування (*textbox, edit field*);
- значок (*icon*);
- панель інструментів (*toolbar*);
- стрічка стану (*status bar*);
- спливаюча підказка (*tooltip, hint*);
- полоса прокрутки (*scrollbar*);
- вкладка (*tab*);
- елемент для відображения табличних данних (*grid view*);
- меню (*menu*);
- головне меню вікна (*main menu*);
- контекстне меню (*popup menu*);
- вікно (*window*);
- панель (*panel*);
- діалогове вікно (*dialog box*);
- модальне вікно (*modal window*);
- дерево – елемент для відображення ієрархії (*tree view*).

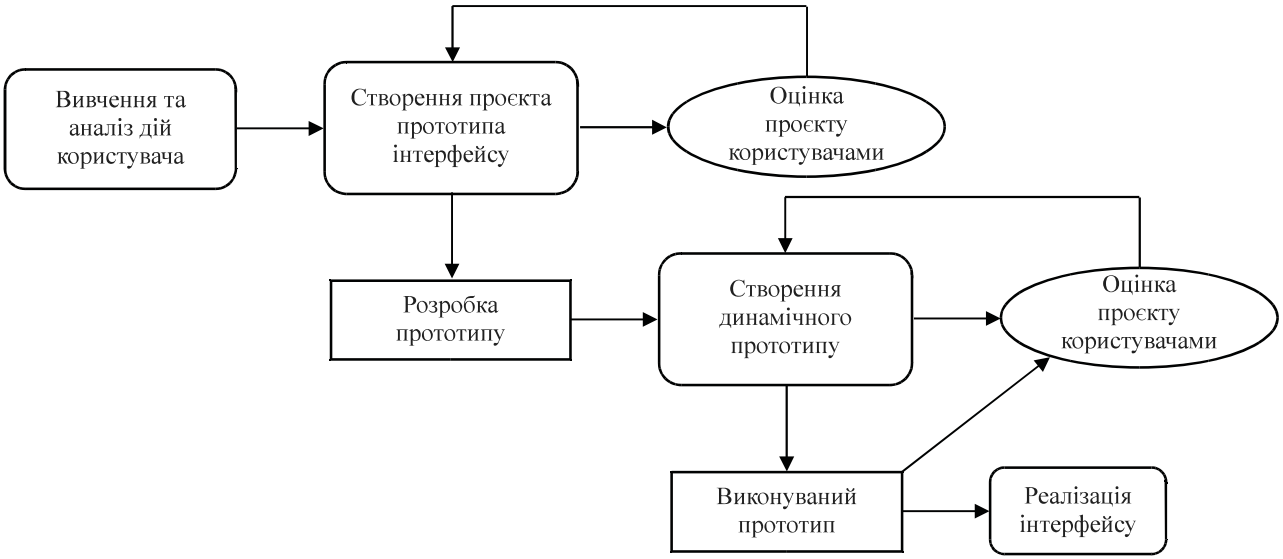


Рис. 4.2. Ітераційний процес проєктування інтерфейсу користувача

Є кілька доступних інструментів, за допомогою яких можна створити повний графічний інтерфейс одним клацанням миші. Деякі інструменти можуть бути вбудовані в програмне середовище (IDE – *integrated development environment*).

Інструменти реалізації графічного інтерфейсу надають потужний набір графічних елементів керування. Існують різні сегменти інструментів GUI відповідно до їх різного використання та платформи, мобільний графічний інтерфейс, комп'ютерний графічний інтерфейс, сенсорний екран тощо. Список кількох інструментів, які стають у нагоді для створення графічного інтерфейсу користувача:

- FLUID;
- AppInventor (Android);
- LucidChart;
- Wavemaker;
- Visual Studio.

Завдання

Розробити візуальний інтерфейс у відповідності до проєктування інтерфейсу.

Лабораторна робота № 5

РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета роботи: ознайомитись та набути практичних навичок застосування різних технік тестування.

Короткі теоретичні відомості

Тестування програмного забезпечення (англ. *software testing*) – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, у якому його мають використовувати. Техніка тестування також включає в себе як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки.

Тестування програмного забезпечення – це процес перевірки відповідності висунутих до продукту вимог і реально реалізованої функціональності, який здійснюють шляхом спостереження за його роботою у штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Тестування програмного забезпечення – це техніка контролю якості, що перевіряє відповідність між реальною й очікуваною поведінкою програми завдяки кінцевому набору тестів, які обираються певним чином. Поняття якості обмежується такими поняттями як коректність, надійність, практичність, безпечність, але може містити більше технічних вимог, котрі описані у стандарті ISO 9126. Склад та зміст супутньої документації процесу тестування визначається стандартом IEEE 829-1998 *Standard for Software Test Documentation*. Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів – це по суті дослідницький та творчий процес, а не тільки створення та виконання рутинної процедури.

Тестування – це одна з технік контролю якості, що охоплює:

• планування робіт (*test management*);
• проєктування тестів (*test design*);
• виконання тестування (*test execution*);
• аналіз отриманих результатів (*test analysis*).

Якість (*quality*) – ступінь, з якою компонент, система або процес відповідає зафіксованим вимогам і/або очікуванням та потребам користувача або замовника.

Дефект (*defect, bug*, помилка) – ключовий термін тестування, що означає відхилення фактичного результату від очікуваного. Для виявлення дефекту необхідно виконати три умови: знати фактичний результат; знати очікуваний результат; зафіксувати факт різниці між фактичним і очікуваним результатом.

Процес тестування як процес пошуку дефектів зводиться до наступної послідовності дій:

1. Дізнаємося очікуваного результату.
2. Дізнаємося фактичного результату.
3. Порівнюємо очікуваний і фактичний результати.

Джерелом очікуваного результату є специфікація – детальний опис того, як має працювати ПЗ. У загальному випадку будь-який дефект є відхилення від специфікації. Важливо виявити ці дефекти до того, як їх знайдуть кінцеві користувачі. Тестування можна класифікувати за дуже великою кількістю ознак. На рис. 5.1 наведено узагальнений список видів тестування за різними підставами [23].

Верифікація (*verification*) – це процес оцінки системи або її компонентів з метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проєкту, визначені на початку поточної фази.

Валідація (*validation*) – це визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи.

Рис. 5.1. Класифікація видів тестування залежно від об’єкта

План тестування (*test plan*) – це документ, що описує весь обсяг робіт з тестування, починаючи з опису об’єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків із варіантами їх вирішення.

Тест-дизайн (*test design*) – це етап процесу тестування програмного забезпечення, на якому проєктуються і створюються тестові випадки (тест-кейси), відповідно до визначених раніше критеріїв якості та цілей тестування.

Тестовий випадок (*тест кейс/test case*) – це документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини.

Баг-дефект репорт (*bug report*) – це документ, що описує ситуацію або послідовність дій (*steps*), що призвела до некоректної роботи об'єкта тестування (*misbehavior*), із зазначенням причин та очікуваного результату (*expected result*).

Тестове покриття (*test coverage*) – це одна з метрик оцінки якості тестування, що являє собою щільність покриття тестами вимог або коду, що виконується.

Деталізація тест-кейсів (*test case specification*) – це рівень деталізації опису тестових кроків і необхідного результату, за якого забезпечується розумне співвідношення часу проходження до тестового покриття.

Час проходження тест-кейса (*test case pass time*) – це час від початку проходження кроків тест-кейса до отримання результату тесту [17].

Тестування графічного інтерфейсу – це тип тестування програмного забезпечення, який перевіряє графічний інтерфейс користувача програмного забезпечення.

Завданням тестування графічного інтерфейсу користувача є виявлення помилок наступного характеру:

• помилки у функціональності за допомогою інтерфейсу;

• необроблені виключення під час взаємодії з інтерфейсом;

• втрата або перекручення даних, переданих через елементи інтерфейсу;

• помилки в інтерфейсі (невідповідність проєктної документації, відсутність елементів інтерфейсу).

Мета тестування графічного інтерфейсу користувача (GUI) полягає у забезпеченні функціональних можливостей програмних додатків відповідно до специфікацій шляхом перевірки екранів та елементів керування, таких як меню, кнопки, піктограми тощо.

Етапи тестування зручності використання інтерфейсу користувача:

• Дослідницьке – проводиться після формулювання вимог до системи та розробки прототипу інтерфейсу. Основна мета – провести високорівневе обстеження інтерфейсу та з'ясувати, чи дозволяє він з достатнім ступенем ефективності розв'язувати задачі користувача.

• Оціночне – проводиться після розробки низькорівневих вимог та деталізованого прототипу інтерфейсу. Поглиблює дослідницьке тестування та має ту саму мету. На даному етапі проводяться кількісні виміри характеристик користувальницького інтерфейсу: вимірюються кількість звернень до системи допомоги по відношенню до кількості здійснених операцій, кількість помилкових операцій, час усунення наслідків помилкових операцій і т. ін.

• Валідаційне – проводиться ближче до етапу завершення розробки. На цьому етапі проводиться аналіз відповідності інтерфейсу програмної системи стандартів, що регламентує питання зручності інтерфейсу (наприклад ISO 13407 [26], ISO/IEC 25010:2011 [27]), проводиться загальне тестування всіх компонентів для користувача інтерфейсу з точки зору кінцевого користувача. Під компонентами інтерфейсу розуміється як його програмна реалізація, так і система допомоги й керівництво користувача. Також на даному етапі перевіряється відсутність дефектів зручності використання інтерфейсу, виявлених на попередніх етапах.

• Порівняльне – проводиться на будь-якому етапі з метою порівняння декількох версій. У ході порівняльного тестування порівнюються два чи більше варіантів реалізації призначеного для користувача інтерфейсу.

Зазвичай під час тестування зручності використання користувальницького інтерфейсу використовуються деякі евристичні критерії і характеристики, які замінюють точні оцінки у класичному тестуванні програмних систем.

Якоб Нільсен у своїй роботі [10] виділив 10 евристичних характеристик зручного для користувача інтерфейсу, які

з його точки зору повинні перевірятися під час тестування зручності використання інтерфейсу:

1. *Visibility of system status* / Видимість статусу системи. Система повинна завжди інформувати користувача про те, що відбувається, шляхом належного та своєчасного зворотного зв'язку.

2. *Match between system and the real world* / Відповідність між системою та реальним світом. Система повинна спілкуватися з користувачем його мовою, знайомими йому словами, фразами та ідеями, а не термінами, орієнтованими на систему.

3. *User control and freedom* / Контроль та свобода користувача. Користувачі часто обирають системні функції помилково, тому потребуватимуть чітко означеного «аварійного виходу», щоб покинути небажаний стан без зайвих тривалих діалогів. Підтримуйте можливість скасування та відтворення дії.

4. *Consistency and standards* / Послідовність та стандарти. Користувачі не мають задумуватись, чи різні слова, ситуації або дії означають одне й те саме. Зберігайте відповідність до конвенцій платформи.

5. *Error prevention* / Запобігання помилок. Дбайливе запобігання виникненню проблеми як такої є кращим підходом, аніж наявність гарної системи повідомлень про помилку. Або усуньте вразливі до помилок умови, або пильнуйте за ними та давайте користувачеві можливість підтвердити свої дії.

6. *Recognition rather than recall* / Впізнавання, а не пригадування. Мінімізуйте навантаження на пам'ять користувача, робіть об'єкти, дії та опції видимими. Користувач не повинен запам'ятовувати інформацію між двома різними вікнами. За потреби інструкції з користування системою мають бути видимими, або легкодоступними.

7. *Flexibility and efficiency of use* / Гнучкість та легкість у користуванні. Прискорювачі – невидимі для ока новачка, часто можуть покращити взаємодію із системою для

досвідченого користувача (наприклад, «гарячі» комбінації клавіш). Дозвольте користувачам підлаштувати часті дії для свого комфорту.

8. *Aesthetic and minimalist design* / Естетичність та мінімалістський дизайн. Сторінки та діалогові вікна не повинні містити недоречної та зайвої інформації. Кожна додаткова одиниця інформації конкурує з потрібними одиницями інформації та знижує їхню відносну видимість.

9. *Help users recognize, diagnose, and recover from errors* / Допомога користувачеві в розпізнанні, діагностуванні та виправленні помилок. Повідомлення про помилки подавайте простою мовою, чітко вказуйте на проблему та пропонуйте конструктивне вирішення.

10. *Help and documentation* / Допомога та документація. Хоч і краще, коли системою можна користуватися без документації, може бути необхідним надати допомогу та документацію. Будь-яка інформація такого типу має бути легкодосяжною, фокусувати користувача на його завданнях, перераховувати конкретні кроки, необхідні для виконання, та не бути надмірно об’ємною.

Основним підходом до тестування програмних продуктів є складання тест-кейсів – набору вхідних даних, умов виконання та очікуваних результатів, розроблених з метою перевірки тієї чи іншої властивості або поведінки програмного продукту. У випадку тестування графічного інтерфейсу доцільніше використовувати тест-кейси, які можуть бути зведені до конкретних пунктів чек-ліста. Чек-ліст – це документ, який містить у собі варіанти ідей для покрокового тестування ПЗ, тому дозволяє вносити зміни у список, безпосередньо в ході виконання тестування. Детальність чек-ліста може залежати від кількості тестувальників, які інформують про всі події, під час роботи та складності готового програмного продукту. Чек-ліст може бути оформлений у вигляді:

1) списку, у якому послідовність пунктів не має значення (наприклад, список значень якогось поля);

2) списку, у якому послідовність пунктів важлива (наприклад, кроки в короткій інструкції);

3) структурованим (багаторівневим) списком, що дає змогу побачити ієрархію ідей.

Юзабіліті-тестування (*Usability testing*) – це процес взаємодії з тестованим продуктом, у якому вивчається рівень зручності, а також задоволеності під час використання. Проводиться як тестувальниками, так і простими користувачами, до того ж за підсумком усі результати сумуються та обробляються.

Основні критерії:

• швидкість завантаження сайта і сторінок (інтерфейсу ПЗ);

• адаптивність верстки;

• мова і система числення, формат часу представлені у звичному форматі (або можна швидко змінити);

• зручний шрифт і якість контенту;

• основний функціонал зрозумілий і його не потрібно шукати;

• відсутність горизонтальної смуги прокрутки;

• одноманітність інтерфейсу, є карта сайта;

• присутність інформації про компанії та є робоче лого;

• немає дратівливих елементів для користувачів (наприклад, реклама зі звуком);

• сторінка 404 повідомляє час і причину помилки, містить інформацію про основні розділи та контакти.

Крім цього дуже важливо не забувати про основні цілі юзабіліті-тестування – перевірки зручності використання і рівня задоволеності від продукту користувачем. Також необхідно перевірити, як люди використовують продукт і виявити проблемні області для покращення деяких функцій [28].

Чек-ліст юзабіліті-тестування:

1. Зручність користування функціоналом:

• миттєвий скролінг (кнопки вгору / вниз на довгих сторінках);

• варіанти вибору динамічно змінюються (на початку введення є список доступних результатів, також є можливість не вибирати зі списку, якщо немає відповідного варіанта);

• усе, що можна зробити програмно, робиться (підставляється місце доставки, виходячи з місця розташування користувача, але є можливість редагування);

• логотип клікабельний і веде на головну сторінку, а на головній не перевідкриває сторінку;

• присутній зручний пошук на сайті або в додатку;

• зручність кнопок (натискається сама кнопка, а не текст на ній або прилегла область).

2. Дизайн і ключові елементи:

• відсутні яскраві зухвалі кольори й кольоровий текст;

• колір залежить від сприйняття підсвідомого значення (червоний – помилка, зелений – усе добре);

• оптимальний розмір активних елементів (посилання, банери, кнопки);

• пояснення спливають, де це необхідно (наприклад, чому кнопка посилання неактивна, не потрібно їх зовсім приховувати);

• рядок пошуку, контакти, підзаголовки статей на сторінці розташовані за схемою у вигляді букви *F* (так ковзає погляд користувача);

• оформлення посилань в одноманітних кольорах і стилі по всьому додатку;

• єдине поле пошуку, допомога за відсутності результатів;

• присутній захист від спаму (у коментарях);

• можлива авторизація через соціальні мережі;

• є зручний зворотний зв’язок з підтвердженням питань, відправкою відповіді;

• чат підтримки з людиною, який також відображає час очікування відповіді;

• розбивка на тематичні категорії.

3. Реєстрація:

• для введення даних поля обов'язково повинні бути виділені;

• мінімальна, але необхідна кількість даних;

• одна колонка полів;

• простота заповнення (використання підказок);

• перевірка валідності введених даних до відправки форми;

• інформаційне повідомлення про помилку (також виділення потрібного поля);

• функція перегляду пароля;

• зручне редагування;

• підтвердження реєстрації;

• є можливість відмовитися від розсилок.

4. Зображення:

• однаковість у розмірах однотипних форм і кольорів;

• корисність (зображення з інформацією або елемент дизайну);

• висока якість.

Список популярних засобів тестування графічного інтерфейсу:

– Ranorex;

– Selenium;

– QTP;

– SilkTest;

– TestComplete;

– Squish GUI Tester.

Інструменти і програмне забезпечення для керування тестами [30]:

• Visure Requirements ALM Platform;

- Helix ALM;
- Jira;
- IBM Rational Team Concert;
- Microsoft Visual Studio;
- GitLab;
- HP ALM;
- TFS;
- VersionOne;
- Rally;
- Bamboo;
- CodeBeamer;
- QA Complete;
- TestRail;
- Zephyr;
- PractiTest.

Завдання

Протестувати програмне забезпечення одним із видів тестування. Створити тест-кейси для тестування графічного інтерфейсу програмного забезпечення.

Структура тест-кейсу(*test-case*) [31]:

1. **Ідентифікатор (номер) / ID (*number*):** унікальний ідентифікатор тесту допомагає ефективніше відстежувати його під час тестування. Його часто використовують для організації індексування та пошуку тестів у менеджерах тестових випадків. Такі ідентифікатори створюються автоматично в системі управління тестами.

2. **Назва / Name:** описова назва тесту, яка передає його суть і функціональність, що тестується. Вдало підібрана назва допомагає швидко зрозуміти зміст тесту.

3. **Передумови (умови та параметри) / Preconditions (*conditions and parameters*):** ці умови, розташовані в розділі Передумови, визначають початковий стан системи перед за-

пуском тесту. Вони вказують на стан системи, у якому можна правильно виконати тест.

4. **Кроки / Steps:** цей розділ містить послідовність дій, які має виконати тестувальник, щоб виконати певний тестовий сценарій. Кожен крок має бути чітко описаний і зрозумілий.

5. **Тестові дані / Test Data:** конкретні дані (наприклад, ім'я користувача та пароль), необхідні для проходження поточного тесту. Це також може бути зазначено в попередніх умовах або поточних кроках. Але треба зберігати тестові дані. Не кожен крок тестового прикладу потребує тестових даних.

6. **Очікуваний результат / Expected result:** визначає очікувану поведінку після кожного виконаного кроку тесту. Тест порівнює фактичні та очікувані результати для виявлення помилок.

7. **Постумови / Postconditions:** цей розділ може містити дії після завершення тесту для повернення системи до початкового стану. Це може бути корисним, якщо тест змінює дані або налаштування системи.

Тестові випадки повинні охоплювати різні позитивні та негативні сценарії та інші варіанти. Також рекомендується організувати тестові випадки в логічні групи (набори тестів і плани тестування) та додати посилання на відповідні вимоги, що забезпечує більшу прозорість і легкість відстеження вимог.

У прикладі представлено два тести: перший перевіряє правильність входу в систему (позитивний сценарій), а другий – негативний сценарій, коли вводиться неправильний пароль. Кожен тестовий приклад має свій унікальний ідентифікатор та ім'я, задані умови та параметри, необхідні для виконання тесту, кроки для виконання очікуваних результатів і можливі післяумови. Рекомендується вводити назву функції, яку потрібно перевірити, перед її назвою (щоб побачити групу тестів однієї функції).

Таблиця 5.1. **Тестовий приклад вебдодатка**

ID	Name / Ім'я	Preconditions / Передумови	Steps / Кроки	Data / Дані	Expected Result / Очікуваний результат
TC1	Увійдіть за допомогою дійсних облікових даних	Користувач повинен бути зареєстрований у системі	1. Відкрийте сторінку авторизації	https://site.com/authorization	Відобразиться сторінка «Авторизація»
			2. Введіть дійсний логін у поле «Увійти»	validlogin@example.com	Поле «Вхід» доступне для редагування, і введені дані відображаються
			3. Введіть дійсний пароль у поле «Пароль»	Дійсний_пароль	Поле «Пароль» редагується; введені дані відображаються зірочками
			4. Натисніть кнопку «Увійти»		Користувач авторизується та перенаправляється на «Домашню» сторінку
TC2	Увійти з неправильним паролем	Користувач повинен бути зареєстрований у системі	1. Відкрийте сторінку авторизації	https://site.com/authorization	Відобразиться сторінка «Авторизація»
			2. Введіть дійсний логін у поле «Увійти»	validlogin@example.com	Поле «Вхід» доступне для редагування, і введені дані відображаються
			3. Введіть недійсний пароль у поле «Пароль»	Недійсний_пароль	Поле «Пароль» можна редагувати, а введені дані відображаються зірочками
			4. Натисніть кнопку «Увійти»		Помилка «Помилка входу. Недійсне ім'я користувача або пароль». Над полем «Увійти», поля «Вхід» і «Пароль» виділені червоним кольором

Лабораторна робота № 6

ПРЕЗЕНТАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

Мета роботи: ознайомитися з основними правилами створення презентації.

Короткі теоретичні відомості

Презентація (англ. *presentation*) – процес ознайомлення слухачів з певною темою. Зазвичай це демонстрація, лекція чи промова з метою проінформувати чи переконати когось. Презентація може містити три компоненти:

• промова доповідача: те, що доповідач розповідає;

• слайди: те, що бачить аудиторія на екрані;

• роздаткові матеріали: те, що роздається кожному в аудиторії окремо, і містить деталі інформації, що презентується. Наприклад, список літератури та інших посилань чи статистичні таблиці.

Часто, щоб допомогти створити й передати зміст презентації, використовуються програми для створення і показу презентацій [16].

Програма підготовки презентацій – комп'ютерна програма, яка використовується для створення, редагування та показу презентацій на проєкторі на великому екрані. Програма підготовки презентацій дозволяють створювати слайди (кадри) презентації і наповнювати їх умістом, налаштовувати зовнішній вигляд презентації та можливі візуальні й анімаційні ефекти. Створювана презентація може включати в себе елементи інтерактивності, такі як кнопки для переміщення між слайдами та посилання на вебсторінки. Програми для створення мультимедійних презентацій дозволяють використовувати в презентації не тільки текстові та графічні зображення, а й аудіо- і відеоелементи. Застосунки для презентації:

• Apple iWork Keynote;

• Google Презентації;

• LibreOffice Impress – входить до складу вільного офісного пакета LibreOffice;
• Microsoft PowerPoint;
• OpenOffice.org Impress;
• Emaze;
• Prezi;
• ProPresenter;
• Selvery;
• Stages (програма);
• SoftMaker Presentations.

Microsoft PowerPoint (повна назва – Microsoft Office PowerPoint) – це застосунок для створення та відтворення презентацій, що є частиною Microsoft Office, і доступний у редакціях для ОС Microsoft Windowsi Mac OS.

Основними елементами презентації є слайди. За допомогою редактора Microsoft Office PowerPoint можна створювати слайди, у яких текст поєднується з таблицями, діаграмами, графічними об'єктами, картинками, рисунками, фотографіями, фільмами зі звуком, відеокліпами.

Кожен слайд презентації володіє властивостями, які впливають на його відображення під час демонстрації:

– розмір слайда;

– розмітка слайда (розташування заголовків, тексту й об'єктів на слайді);

– шаблон оформлення (дизайн слайда);

– ефект переходу від слайда до слайда.

Презентація програмного продукту повинна обов'язково містити такі елементи [24]:

– тему, виконавця, керівника;

– актуальність роботи;

– мету та завдання дослідження;

– об'єкт та предмет дослідження;

– методи дослідження;

– апробацію результатів досліджень та публікації;

– викладення результатів власних досліджень з висвітленням того нового, що пропонується;

– прийняті проєктні рішення з розробки ПЗ (моделі ПЗ та даних, алгоритми та інше);

– інтерфейс користувача (форми, вікна, екрани);

– висновки.

Усі слайди презентації повинні мати порядковий номер і легко читатися з екрана на відстані п’яти метрів. Останнім бажано мати слайд з подякою присутнім за увагу.

Завдання

Створити презентацію розробленого програмного продукту за допомогою будь-якого застосунка для презентації.

СПИСОК ЛІТЕРАТУРИ

1. Козак О. Л. Опорний конспект лекцій з курсу «Аналіз вимог до програмного забезпечення» для студентів напрямку підготовки «Програмна інженерія». Тернопіль, 2011. 56 с.

2. Табунщик Г. В., Каплієнко Т. І., Петров О. А. Проєктування та моделювання програмного забезпечення сучасних інформаційних систем : навч. посіб. Запоріжжя : ЗНТУ, 2016. 250 с.

3. Wiegers, Karl; Beatty, Joy (2013). Software Requirements (3rd ed.). Microsoft Press. ISBN 978-0-7356-7966-5.

4. Jacobson, Ivar; Grady Booch; James Rumbaugh (1998). The Unified Software Development Process. Addison Wesley Longman. ISBN 0-201-57169-2.

5. Jena Ajay Kumar, Das Himansu, Mohapatra Durga Prasad (eds.). Automated Software Testing: Foundations, Applications and Challenges. New York : Springer, 2020. 173 p.

6. Mili Ali, Tchier Fairouz. Software Testing: Concepts and Operations. Wiley, 2015. 400 p.

7. O'Regan O. Concise Guide to Software Testing. New York : Springer, 2019. 309 p.

8. https://docplayer.net/4985784-Engineering-software-eric-j-braude-michael-e-bernstein-modern-approaches-universitatsbibliothek-hannover-technische-informationsbibliothek.html.

9. https://www.booksfree.org/wpcontent/uploads/2022/03/Software_Requirements_3rd_Edition_compressed.pdf.

10. Nielsen, Jakob (2005),Ten usability heuristics https://media.nngroup.com/media/articles/attachments/Heuristic_Summary1-compressed.pdf.

11. https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf.

12. Вимоги до програмного забезпечення – https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D0%BC%D0%BE%D0%B

3%D0%B8_%D0%B4%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F.

13. Графічний інтерфейс користувача – https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0#:~:text=GUI%2C%20Graphical%20user%20interface)%20%E2%80%94,%D0%BD%D0%B0%D0%B1%D0%BE%D1%80%D1%96%20%D0%BA%D0%BE%D0-%BC%D0%B0%D0%BD%D0%B4%20%D1%82%D0%B0%20%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%96%D0%B9%20%D0%BD%D0%B0%D0%B2%D1%96%D0%B3%D0%B0%D1%86%D1%96%D1%97.

14. Інтерфейс користувача – https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0.

15. Нефункціональні вимоги – https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%96_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3%D0%B8.

16. Презентація – https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B7%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D1%96%D1%8F_(%D0%B2%D0%B8%D1%81%D1%82%D1%83%D0%BF)#cite_note-1.

17. Тестування програмного забезпечення – https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D

1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0-%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F.

18. Функціональні вимоги – https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%96_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3%D0%B8.

19. Конспект лекцій з дисципліни «Професійна практика програмної інженерії» для здобувачів вищої освіти першого (бакалаврського) рівня зі спеціальності 121 «Інженерія програмного забезпечення» усіх форм навчання / уклад. М. В. Бабенко. Кам'янське : ДДТУ, 2021. 88 с.

20. Люшенко Л. А., Хіцко Я. В. Навчальний посібник для студентів зі спеціальності 121 «Інженерія програмного забезпечення», освітньої програми «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем». КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,83 Мбайт). Київ : КПІ ім. Ігоря Сікорського, 2020. 63 с.

21. Олійник О. О., Олійник А. О., Льовкін В. М. Методичні вказівки до виконання лабораторних робіт з дисципліни «Професійна практика програмної інженерії» для студентів спеціальності 121 «Інженерія програмного забезпечення» (усіх форм навчання). Запоріжжя : ЗНТУ, 2016. 51 с.

22. Дудченко О. М., Карпова С. О. CASE-засоби розробки та тестування програмного забезпечення : методичні вказівки до виконання лабораторних робіт. Миколаїв : НУК, 2018. 84 с.

23. Коротенко Г. М., Коротенко Л. М., Шевцова О. С. Методичні вказівки до практичних робіт з курсу «Тестування та верифікація ПЗ». Дніпро : НТУ «ДП», 2020. 62 с.

24. Дудченко О. М., Приходько С. Б., Латанська Л. О., Ма-

карова Л. М. Методичні рекомендації щодо виконання кваліфікаційних (магістерських) робіт для студентів, що навчаються за освітньо-професійною програмою «Інформаційні управляючі системи та технології» спеціальності 122 «Комп'ютерні науки» галузі знань 12 «Інформаційні технології». Миколаїв : НУК, 2023. 38 с.

25. https://allcompositions.at.ua/temy_1-2.pdf.

26. https://www.iso.org/obp/ui/#iso:std:iso:13407:ed-1:v1:en.

27. https://www.iso.org/standard/35733.html.

28. https://training.qatestlab.com/blog/technical-articles/usability-testing-check-list-testing/.

29. https://standards.ieee.org/ieee/29148/6937/.

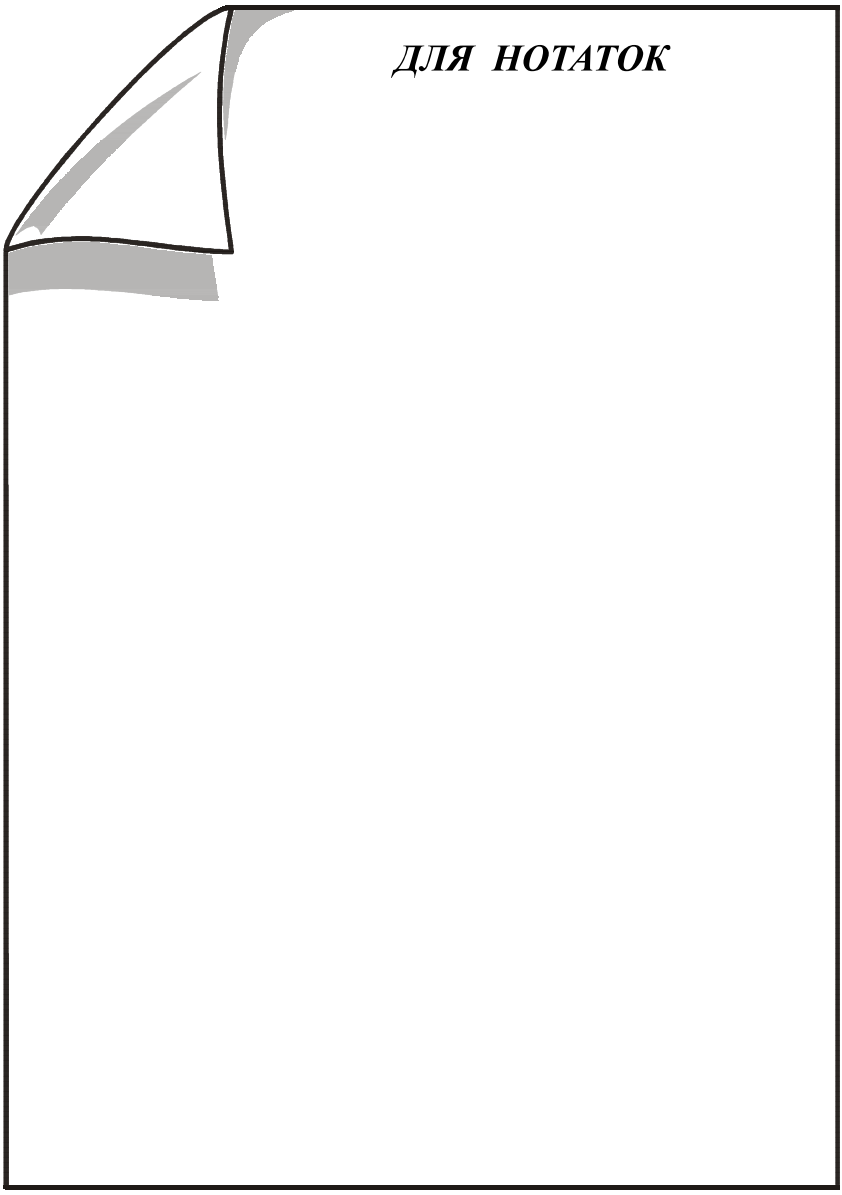
30. https://visuresolutions.com/uk/blog/best-test-management-tools-and-softwares/.

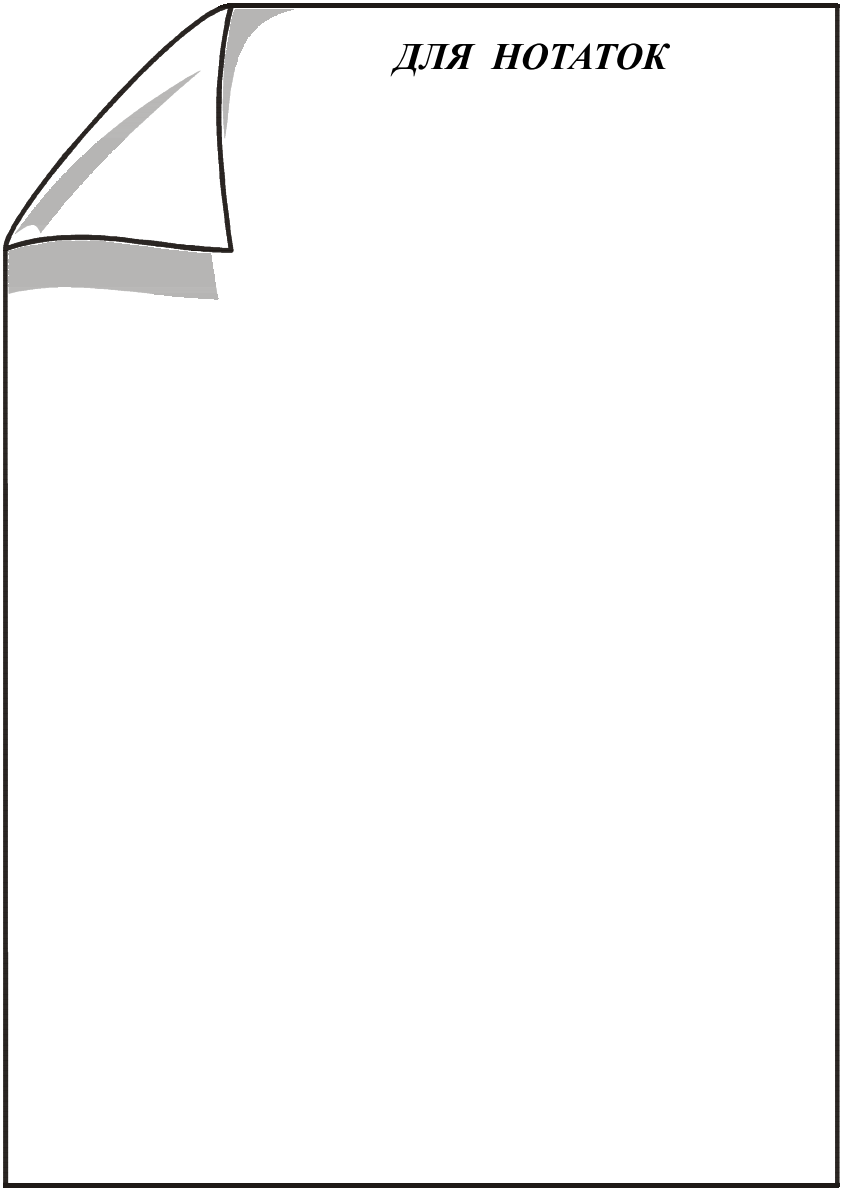
31. https://luxequality.com/blog/how-to-write-test-cases-for-software-testing/.

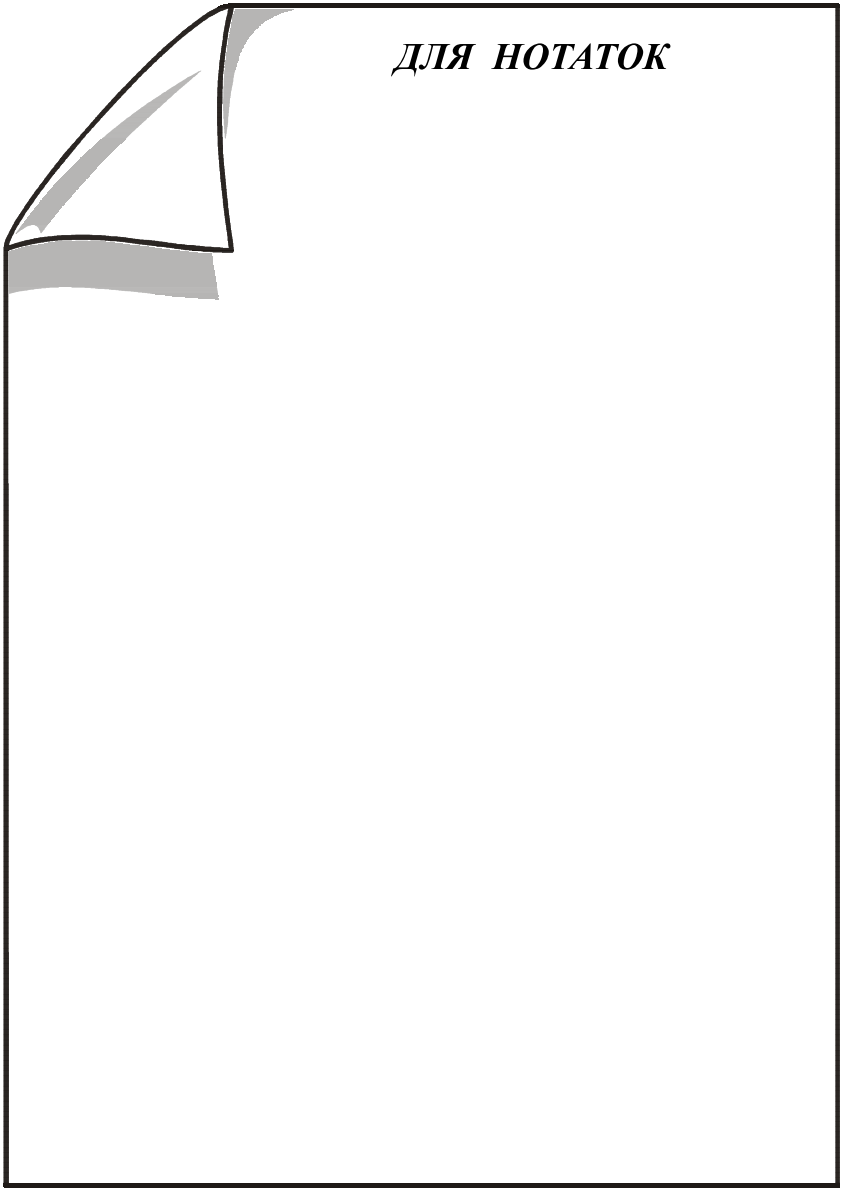
32. «ISO/IEC 19501:2005 – Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.3». Iso.org. 1 April 2005. Retrieved 7 May 2015.

33. https://www.geeksforgeeks.org/test-case/.

ЗМІСТ


ДЛЯ НОТАТОК


ДЛЯ НОТАТОК



ДЛЯ НОТАТОК

Навчальне видання

КАРПОВА Світлана Олегівна

ПРОФЕСІЙНА ПРАКТИКА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Методичні вказівки до виконання лабораторних робіт

Комп'ютерна правка й коректура *О. Г. Костенко*
Комп'ютерне верстання *А. Д. Сорочинська*

Формат 60x84/16. Ум. друк. арк. 3,0. Тираж 100 прим. Вид. № 14. Зам. № 1510-43.
Видавець і виготівник Національний університет кораблебудування
імені адмірала Макарова
просп. Героїв України, 9, м. Миколаїв, 54025
E-mail : publishing@nuos.edu.ua
Свідоцтво суб'єкта видавничої справи ДК № 6402 від 19.09.2018 р.