МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

О. М. ДУДЧЕНКО, С. О. КАРПОВА

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з курсу «ПРОЄКТНИЙ ПРАКТИКУМ»

Рекомендовано Методичною радою НУК

ВИДАВНИЦТВО
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КОРАБЛЕБУДУВАННЯ
ІМ. АДМІРАЛА МАКАРОВА

2023

УДК 004.4`2
Д81

Автори: О. М. Дудченко, канд. техн. наук, професор;
С. О. Карпова, старш. викладач

Рецензент П. Й. Гучек, д-р техн. наук, доцент

Рекомендовано Методичною радою НУК

Дудченко О. М.

Д81 Методичні вказівки до виконання лабораторних робіт з курсу «Проєктний практикум» / О. М Дудченко, С. О. Карпова. – Миколаїв : НУК, 2023. – 84 с.

Вміщено методичні вказівки з визначення основних завдань до лабораторних робіт з курсу «Проєктний практикум». Методичні вказівки є організаційно-методичним документом.

Призначено для студентів спеціальності 121 «Інженерія програмного забезпечення» Херсонського навчально-наукового інституту НУК. Можуть бути використані студентами споріднених спеціальностей.

УДК 004.4`2

ВСТУП

Проєктний практикум – це предмет, під час вивчення якого, студенти можуть реалізувати свої знання, отримані протягом першого та другого курсів навчання, до того ж як ті, що стосуються програмування, так і ті, що відповідають за аналітику та проєктування програмного забезпечення.

Оскільки курс практичний, то важливою частиною є готовий фінальний продукт. Щоб створити його, скоріше за все не вистачить багажу програмістських знань, набутих за два роки навчання. Техносвіт постійно розвивається, тому під час створення нового проєкту треба використовувати новітні технології.

Основними вимогами до фінального продукту є його практичність, відповідність заданим вимогам, використання актуальних технологій. Завданням лабораторних робіт є цілісне розуміння всіх етапів розробки програмного забезпечення.

Проєктний практикум – це зменшена модель реальної практики, надана в рамках навчального процесу.

Програмне забезпечення (ПЗ) – це програма або група програм, яка є кінцевим продуктом проєкту програмної розробки. Програмну розробку називають *програмним інжинірингом*. Інститут програмного інжинірингу (*Software Engineering Institute, SEI*) надає таке визначення програмного забезпечення: ПЗ – програми, процедури та символічні мови, які управляють функціонуванням апаратних засобів[12].

Інжиніринг програмного забезпечення – це практичне застосування наукових знань під час розробки та створення комп'ютерних програм і пов'язаної з ними документації, необхідної для їх розробки, використання та підтримки.

Проєкт (визначення, запропоноване Джеймсом Льюїсом (*James Lewis*) [8] – одноразова робота, яка має певні дати початку і закінчення, чітко визначені цілі, можливості і здебільшого бюджет. За Гарольдом Керцнером (*Harold Kerzner*) [7]

проєкт – це довільний ряд дій або завдань, що має певну мету, яка буде досягнута в рамках виконання деяких завдань, що характеризуються певними датами початку і закінчення, межами фінансування (у разі прикладного проєкту) і ресурсами (гроші, трудовитрати, обладнання).

Інститут управління проєктами (PMI) дає таке визначення проєкту: **Проєкт** – це тимчасове зусилля, почате для того, щоб створити унікальний продукт або послугу з певною датою початку і закінчення дії, що відрізняється від триваючих, повторних дій, що потребує прогресивного вдосконалення характеристик.

Отже, у термінах розробки програмного забезпечення **проєкт** – це унікальна, тимчасова дія з визначеними датами початку і кінця, спрямована на те, щоб досягти однієї або кількох цілей при обмеженнях за вартістю, графіком і якості виконання.

Проєктування програмного забезпечення – це процес визначення архітектури, компонентів, інтерфейсів та інших атрибутів (структур даних, алгоритмів і т. п.) системи або компонента програмного забезпечення. Результатом цього процесу є проєкт програмного забезпечення (англ. *Software design*).

Проєктуванню зазвичай підлягають:

- архітектура програмного забезпечення;
- компоненти програмного забезпечення;
- користувацькі інтерфейси.

Розробка програмного забезпечення (англ. *software engineering, software development*) – це низка діяльності (професія) та процес, спрямований на створення та підтримку працездатності, якості та надійності програмного забезпечення, використовуючи технології, методологію та практики з інформатики, керування проєктами, математики, інженерії та інших галузей знання [18].

Розробка програмного забезпечення може бути поділена на кілька розділів:

• Вимоги до програмного забезпечення: витяг, аналіз, специфікація та ратифікація вимог для програмного забезпечення.

• Проєктування програмного забезпечення: проєктування програмного забезпечення засобами автоматизованої розробки програмного забезпечення (*CASE – Computer-Aided Software Engineering*) стандарти формату описів, такі як уніфікована мова моделювання (*UML – Unified Modeling Language*), використовуючи різні підходи: проблемно-орієнтоване проєктування і т. п.

• Інженерія програмного забезпечення: створення програмного забезпечення за допомогою мов програмування.

• Тестування програмного забезпечення: пошук та виправлення помилок у програмі.

• Обслуговування програмного забезпечення: програмні системи часто мають проблеми сумісності та перенесення, а також потребують подальших модифікацій протягом довгого часу після того, як створена їх перша версія. Підобласть має справу з цими проблемами.

• Керування конфігурацією програмного забезпечення: оскільки системи програмного забезпечення дуже складні та модифікуються у процесі експлуатації, їх конфігурації повинні управлятися стандартизованим та структурованим методами.

• Керування розробкою програмного забезпечення: керування системами програмного забезпечення має запозичення з керування проєктами.

• Процес розробки програмного забезпечення: процес побудови програмного забезпечення гаряче обговорюється серед практиків, основними парадигмами вважаються *agile* або *waterfall.*

• Інструменти розробки програмного забезпечення (такі як *Computer-Aided Software Engineering* (*CASE*) – набір інструментів і методів програмної інженерії для проєктування програмного забезпечення, що допомагає забезпечити високу

якість програм, відсутність помилок і простоту в обслуговуванні програмних продуктів): методика оцінки складності системи, вибору засобів розробки та застосування програмної системи.

• Якість програмного забезпечення: методика оцінки критеріїв якості програмного продукту та вимог до надійності.

• Локалізація програмного забезпечення.

Лабораторна робота № 1

РОЗРОБКА ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета роботи: засвоїти основні поняття про вимоги, що висуваються до програмного забезпечення. Розробити специфікацію вимог.

Короткі теоретичні відомості

Вимоги – це один з основних елементів для розробки будь-якого програмного продукту.

Вимоги до програмної системи – це набір вимог щодо властивостей, якості та функцій програмного забезпечення, що буде розроблено, або знаходиться в розробці. Вимоги визначаються у процесі аналізу вимог та фіксуються у специфікації вимог, діаграмах варіантів використання та інших артефактах процесу аналізу та розробки вимог [19].

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.

Основні типи вимог:

Вимоги до продукту – охоплюють умови користувачів щодо зовнішнього поводження системи і погляди розробників на деякі параметри системи.

Вимоги до програмного забезпечення є такі: системні, функціональні і нефункціональні вимоги.

Вимоги користувачів (*user requirements*) задаються умовами досягнення цілей і задач, віддзеркалюють вимоги споживачів до спектра розв'язуваних майбутньою системою задач.

Системні вимоги (*system requirements*) визначають зовнішні умови виконання системних функцій і обмежень на створення продукту, а також вимоги до опису програмно-апаратних підсистем. Системні вимоги накладають обмеження на архітектуру системи, засоби її візуального подання і функціонування. Для опису системних вимог використовують спеціальні шаблони і форми, що допомагають уявити вхідні і вихідні дані й автоматизувати ці вимоги.

Вимоги до атрибутів якості (*quality attributes*) – це деякі обмеження на властивості функцій або системи, важливі для користувачів або розробників.

Функціональні вимоги – це перелік функцій або сервісів, які повинна надавати система, а також обмежень на дані й поводження системи під час їхнього виконання.

Специфікація функціональних вимог (*software requirements specification*) – опис функцій та їхніх властивостей, які не містять у собі протиріч і виключень.

Нефункціональні вимоги визначають умови виконання функцій у середовищі, що безпосередньо не пов'язані з функціями, а відбивають потреби користувачів щодо їх виконання. Ці вимоги характеризують принципи взаємодії із середовищами або іншими системами, а також визначають показники часу роботи, захисту даних і досягнення якості з урахуванням рекомендацій використовуваного стандарту. Вони можуть установлюватися як числові значення в різних одиницях виміру, включаючи, наприклад, імовірність (значення ймовірності безвідмовної роботи системи – показника її надійності).

Для більшості сучасних програмних систем вимоги складаються з умов і обмежень типу:

– конфіденційність, безпека і захист даних;

– відмовостійкість;

– одночасність доступу користувачів до системи;

– час очікування відповіді під час звертання до системи (продуктивність);

– склад виконуваних функцій системи (запуск, швидкість реакції та ін.);

– положення стандартів з виконання сформульованих вимог.

Дані вимоги визначаються та формалізуються аналітиками на процесі аналізу і проєктування структури системи.

До вихідного продукту висуваються нефункціональні вимоги до:

– застосування (якість інтерфейсу, продукту та ін.);

– продуктивності (пропускна здатність, час реакції тощо);

– надійності виконання (без помилок і відмов);

– зовнішніх інтерфейсів, за якими виконується взаємодія з іншими компонентами або підсистемами.

Опис усіх видів вимог проводиться з урахуванням стандартів, наприклад, стандарту з якості ISO/IEC ДСТУ 9126 і стандартизованого понятійного і термінологічного довідника.

Специфікація вимог відображає принципи взаємодії проєктованої системи з іншими середовищами, платформами і загальносистемним забезпеченням. Формування документа зі специфікаціями вимог завершується на процесі проєктування архітектури, після чого він узгоджується із замовником системи і використовується як керування дій під час виконання задач розробки програмного продукту у процесах життєвого циклу й отриманні готового продукту. У разі виявлення на них неузгоджених вимог, проводиться їхнє уточнення і відповідно вносяться зміни в деякі задачі процесу розроблення системи або характеристики продукту.

Загалом існує чотири основні етапи роботи з вимогами [4] (рис.1.1):

1. **Попередні дослідження.** Під цим пунктом мається на увазі аналіз конкурентів / замінників, UX (*user experience*) дослідження: аналіз аудиторії, персон користувачів, їхніх основних проблем і завдань, а також розуміння бізнес-моделі продукту і рентабельної суми на його розробку.

Рис. 1.1. Створення специфікації вимог

2. **Формування та аналіз вимог.** На цьому етапі розробляється UX-прототип системи або 0-прототип (мінімально працює програма для ілюстрації алгоритму і/або основної ідеї). На підставі цього описуються функціональні вимоги.

3. **Специфікування вимог.** Тут з'являється власне специфікація. В один документ збираються опис прототипу, функціональні, призначені для користувача і системні вимоги. Найчастіше сформувати специфіковані вимоги не можна без проєктування архітектури ПЗ. На цьому етапі визначається структура ПЗ, дані, що є частиною системи, інтерфейси взаємодії компонентів системи й алгоритми. Саме собою проєктування – ітераційний процес, за якого кожен наступний етап впливає на попередній, а специфікації стають усе більш деталізованими ближче до завершення проєктування.

Кінцевими результатами процесу проєктування є точні специфікації на алгоритми і структури даних, що будуть реалізовані на етапі створення ПЗ.

4. **Затвердження вимог.** На цьому етапі перевіряємо здійсненність, узгодженість і достатність вимог (повнота вимог).

Завдання

Розробити специфікацію вимог у відповідності до отриманого індивідуального завдання та оформити її згідно з по-

даним нижче зразком та викладеними теоретичними відомостями.

Специфікація вимог до програмного продукту
(*Software Requirements Specification, SRS*)

1. Вступ (*Introduction*)

Вступ до документа «Специфікація програмних вимог» (*SRS*) має надати повний огляд. У нього необхідно включити призначення, галузь застосування, визначення, акроніми, скорочення, посилання та огляд специфікації.

Компоновку специфікації *SRS* можна виконувати різними способами. Ці питання разом з іншими можливостями організації специфікації *SRS* детальніше розкриваються в документі [IEEE830-1998].

1.1. Призначення (*Purpose*)

Укажіть мету створення цього документа. «Специфікація програмних вимог» повинна повністю описувати зовнішню поведінку визначеного додатка або підсистеми, дисфункції, проєктні обмеження та інші фактори, необхідні для повного та всебічного опису вимог до програмного забезпечення.

1.2. Галузь застосування (*Scope*)

Короткий опис програми, функції або групи підсистем, до яких відноситься дана специфікація. Указуються моделі сценаріїв використання з якими вона пов'язана та решта відомостей, що мають до неї відношення.

1.3. Визначення, акроніми та скорочення (*Definitions, Acronyms, and Abbreviations*)

У цьому розділі наведено визначення всіх термінів, акронімів та скорочень, які необхідні для правильної інтерпретації

Примітка. Специфікація *SRS* фіксує всі програмні вимоги до системи або її частини. Нижче наведено типову схему проєктної специфікації *SRS*, вимоги, які виражені природною мовою без застосування моделювання сценаріїв використання. Ця специфікація фіксує всі вимоги в одному документі і включає в себе відповідні розділи з додаткових технічних вимог (які стають непотрібними).

документа «Специфікація програмних вимог». Дана інформація може бути подана у вигляді посилання на словник проєкту.

1.4. Посилання (*References*)

У цьому розділі надається повний список усіх документів, на які посилається «Специфікація програмних вимог». Кожне посилання має містити заголовок документа, номер звіту (за наявності), дату та назву організації, що опублікувала документ. Слід також зазначити, з яких джерел можуть бути отримані ці відомості. Ця інформація може бути представлена як посилання на додаток або на інший документ.

1.5. Огляд (*Overview*)

Дається опис іншої інформації із «Специфікації програмних вимог» та структури цього документа.

2. Загальний опис (*Overall Description*)

Тут слід описати загальні фактори, що впливають на продукт та вимоги до нього. Конкретні вимоги докладно визначаються в розділі 3, а у цьому розділі необхідно дати лише обґрунтування, яке допоможе їх зрозуміти. Зокрема, розглядаються такі питання:

- перспектива товару;
- функції товару;
- характеристики користувача;
- обмеження;
- припущення та залежності;
- підмножини вимог.

3. Конкретні вимоги (*Specific Requirements*)

Цей розділ специфікації *SRS* має містити всі програмні вимоги. Рівень деталізації має бути достатнім для проєктування системи та її тестування на відповідність цим вимогам. У разі застосування моделювання сценаріїв використання, ці вимоги фіксуються в цих сценаріях та у відповідному документі додаткових технічних вимог.

3.1. Функціональність (*Functionality*)

У цьому розділі описуються функціональні вимоги до

системи, які виражаються природною мовою. Для багатьох програм ці відомості складають пакет специфікацій вимог до системи (*SRS*), тому слід добре продумати структуру цього розділу. Загалом цей розділ організується за функціями, що реалізуються, однак можна застосувати й альтернативні методи – наприклад, організацію по користувачам або підсистемам. Функціональні вимоги можуть включати в себе списки реалізованих функцій, можливостей і засобів забезпечення безпеки даних.

Якщо для з'ясування функціональних можливостей використовуються засоби розробки додатків, наприклад, інструменти керування вимогами або моделювання, цей розділ повинен містити посилання на доступні джерела цих даних із зазначенням місцезнаходження та назви використовуваного інструменту.

3.1.1. <Перша функціональна вимога> (*Functional Requirement One*)

Опис вимог.

3.2. Практичність (*Usability*)

У цей розділ слід включити всі вимоги, що стосуються практичності. Наприклад:

• укажіть необхідний час навчання звичайних та кваліфікованих користувачів, який дозволить їм досягти високої продуктивності під час виконання певних операцій;

• укажіть показники часу виконання для характерних завдань або обґрунтуйте вимоги до практичності нової системи на прикладі інших систем, добре знайомих користувачам;

• укажіть вимоги загальних стандартів практичності, наприклад, стандартів загального доступу користувача (*CUA*) корпорації IBM або стандартів графічного користувальницького інтерфейсу корпорації Microsoft.

3.2.1. <Перша вимога до практичності> (*Usability Requirement One*)

Опис вимог.

3.3. Надійність (*Reliability*)

Тут має бути описано вимоги до надійності системи. Передбачається наступний вид розділу:

• доступність – указується відсоток часу, протягом якого система буде доступна (хх. хх %), очікуваний годинник використання, доступ для технічного обслуговування, режим роботи у разі зниженої продуктивності тощо;

• середній час безвідмовної роботи (*MTBF*) – зазвичай указується в годинах, але може бути виражений у днях, місяцях чи роках;

• середній час усунення несправностей (*MTTR*) – указується допустимий час простою системи після збою;

• точність – наводяться показники чіткості (дозвіл) та точності (за яким-небудь відомим стандартом), які потрібні на виході системи;

• максимально допустима кількість помилок – зазвичай виявляється в кількості помилок на тисячу рядків коду (*KLOC*), чи кількість реалізованих функцій;

• рівень помилок – визначення категорій незначних, суттєвих та грубих помилок. Вимоги повинні визначати, що мається на увазі під грубою помилкою (наприклад, повна втрата даних або повна нездатність використовувати будь-які функціональні можливості системи).

3.3.1. <Перша вимога до надійності> (*Reliability Requirement One*)

Опис вимог.

3.4. Продуктивність (*Performance*)

У цьому розділі мають бути стисло викладені основні характеристики продуктивності системи. Укажіть характерні показники часу реакції. Там, де це можливо, посилайтеся на імена відповідних сценаріїв використання.

• час реакції для операції (середній, максимальний);

• продуктивність (наприклад, кількість транзакцій за секунду);

• повна потужність (наприклад, кількість замовників чи операцій, які може обслуговувати система);

• режими зниження продуктивності (опишіть прийнятні режими роботи, за якими продуктивність системи будь-яким чином знижується);

• використання ресурсів: пам'ять, дисковий простір, зв'язок тощо.

3.4.1. <Перша вимога до продуктивності> (*Performance Requirement One*)

Опис вимог.

3.5. Можливості підтримки (*Supportability*)

У цьому розділі слід описати всі вимоги, які можуть посилити можливості технічної підтримки створюваної системи, включаючи стандарти програмування, угоди про присвоєння імен, бібліотеки класів, доступ та засоби технічного обслуговування.

3.5.1. <Перша вимога до можливостей підтримки> (*Supportability Requirement One*)

Опис вимог.

3.6. Проєктні обмеження (*Design Constraints*)

У цьому розділі описуються всі проєктні обмеження, що застосовуються до системи, яка створюється. Ці обмеження є проєктними рішеннями, яких необхідно твердо дотримуватися. Прикладами можуть бути мови програмування, вимоги до програмного процесу, використання певних засобів розробки, архітектурні та проєктні обмеження, компоненти, бібліотеки класів і т. п.

3.6.1. <Перше проєктне обмеження> (*Design Constraint One*)

Опис вимог.

3.7. Вимоги до інтерактивної користувальницької документації та довідкової системи (*Online User Documentation and Help System Requirements*).

Опишіть існуючі вимоги до інтерактивної документації користувача, довідкових систем, відомостей про продукт тощо.

3.8. Придбані компоненти (*Purchased Components*)

У цьому розділі слід описати всі компоненти системи, усі ліцензійні норми або обмеження використання, а також усі пов'язані з цими компонентами стандарти сумісності, взаємодії або інтерфейсів.

3.9. Інтерфейси (*Interfaces*)

У цьому розділі визначаються інтерфейси, які мають підтримуватися цією програмою. Опис повинен містити їх відповідні характерні особливості, протоколи, порти, логічні адреси та інші відомості, що дозволяють контролювати відповідність програмного забезпечення, що розробляється, вимогам до його інтерфейсів.

3.9.1. Інтерфейси користувача (*User Interfaces*)

Опишіть інтерфейси користувача, що реалізуються в даному програмному забезпеченні.

3.9.2. Апаратні інтерфейси (*Hardware Interfaces*)

У цьому розділі слід визначити всі апаратні інтерфейси, що підтримуються даним програмним забезпеченням, включаючи їхню логічну структуру, фізичні адреси, очікувану поведінку і т. п.

3.9.3. Програмні інтерфейси (*Software Interfaces*)

У цьому розділі слід описати програмні інтерфейси між цією програмою та іншими компонентами програмної системи. До цих компонентів можуть входити модулі, повторно використовувані компоненти інших додатків або ті компоненти, які розробляються для підсистем поза функціональними межами даної специфікації програмних вимог, але з якими ця програма повинна взаємодіяти.

3.9.4. Комунікаційні інтерфейси (*Communications Interfaces*)

Опишіть усі комунікаційні інтерфейси, пов'язані з іншими системами або пристроями, такими як локальні мережі, віддалені пристрої тощо.

3.10. Ліцензійні вимоги (*Licensing Requirements*)

Опишіть усі обов'язкові ліцензійні вимоги або інші обмеження використання, які накладаються на програмне забезпечення.

3.11. Попередження щодо законодавства, авторських прав та інші зауваження (*Legal, Copyright and Other Notices*)

У цьому розділі слід описати всі необхідні юридичні попередження, гарантії, авторські та патентні права, текстові торгові марки, знаки або логотипи, що стосуються цього програмного забезпечення.

3.12. Стандарти, що застосовуються (*Applicable Standards*)

У цьому розділі повинні бути наведені посилання на всі стандарти та їх конкретні розділи, що застосовуються до описаної системи. До них можуть входити, наприклад, юридичні стандарти, стандарти якості та стабільності, галузеві стандарти практичності, взаємодії, локалізації, сумісності з операційними системами тощо.

4. Супроводжувальна інформація (*Supporting Information*)

Супроводжувальна інформація спрощує використання специфікації програмних вимог.

До неї входять:

• зміст;

• алфавітний покажчик;

• програми.

Сюди можуть входити деталізовані описи сценаріїв використання або прототипи інтерфейсів користувача. У разі використання програм у специфікації *SRS* необхідно чітко встановити, чи є ці програми частиною вимог.

Лабораторна робота № 2

РОЗРОБКА МОДЕЛІ ВАРІАНТІВ ВИКОРИСТАННЯ ТА ВИДИ СПЕЦИФІКАЦІЙ

Мета роботи: оволодіти навичками побудови діаграм варіантів використання та розробленням специфікацій.

Короткі теоретичні відомості

Стандартною нотацією для моделювання великих інформаційних систем (*ІС*) на основі об'єктно-орієнтованої методології слугує уніфікована мова моделювання *Unified Modeling Language* (*UML*). Мова *UML* є загальноцільовою мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова *UML* одночасно є простим і потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних і графічних моделей складних систем різноманітного цільового призначення [2].

Конструктивне використання мови *UML* ґрунтується на розумінні загальних принципів моделювання складних систем та особливостей процесу об'єктно-орієнтованого проєктування (ООП) зокрема. Вибір виразних засобів для побудови моделей складних систем зумовлює ті завдання, які можуть бути вирішені з використанням даних моделей.

Візуальне моделювання в *UML* можна представити як певний процес порівневого спуску від найбільш загальної й абстрактної концептуальної моделі вихідної системи до логічної, а потім і до фізичної моделі відповідної програмної системи. Для досягнення цих задач, спочатку будується модель у формі так званої діаграми варіантів використання (*use case diagram*), яка описує функціональне призначення систе-

ми або, іншими словами, те, що система буде виконувати в процесі свого функціонування.

Суть даної діаграми полягає в наступному: проєктована система представляється 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 варіантів використання – граф спеціального виду, який є графічною нотацією для представлення конкретних варіантів використання, акторів, можливо, деяких інтерфейсів, і відносин між цими елементами [3].

Елементами діаграми *Use Case* є:

– Варіант використання (*use case*). Позначається на діаграмі еліпсом, усередині якого міститься його коротка назва або ім'я у формі дієслова з пояснювальними словами. Мета варіанта використання полягає в тому, щоб визначити закінчений аспект або фрагмент поведінки деякої сутності без розкриття її внутрішньої структури. Як така сутність може виступати вихідна система або будь-який інший елемент моделі, який володіє власною поведінкою, подібно підсистемі або класу в моделі системи.

– Актор (*actor*) являє собою будь-яку зовнішню по відношенню до системи, що моделюється, сутність, яка взаємодіє із системою та використовує її функціональні можливості для досягнення певної мети або вирішення приватних завдань. До того ж актори слугують для позначення узгодженої безлічі ролей, які можуть грати користувачі у процесі взаємодії з проєктованою системою. Кожен актор може розглядатися як якась

окрема роль щодо конкретного варіанта використання. Стандартним графічним позначенням актора на діаграмах є фігурка «чоловічка», під якою записується конкретне ім'я актора.

– Інтерфейс (*interface*) слугує для специфікації параметрів моделі, які видимі ззовні без зазначення їх внутрішньої структури. На діаграмі варіантів використання інтерфейс зображується у вигляді маленького кола, поряд з яким записується його ім'я. Якщо ім'я записується англійською, то воно повинно починатися з великої літери *I*.

– Примітки (*notes*) у мові *UML* призначені для включення в модель довільної текстової інформації, що має безпосереднє відношення до контексту розроблюваного проєкту. Графічно примітки позначаються прямокутником із «загнутим» верхнім правим куточком. Усередині прямокутника міститься текст примітки. Примітка може відноситися до будь-якого елементу діаграми, у цьому випадку їх з'єднує пунктирна лінія. Якщо примітка відноситься до кількох елементів, то від неї проводяться, відповідно, декілька ліній. Зрозуміло, примітки можуть бути присутні не тільки на діаграмі варіантів використання, а й на інших канонічних діаграмах.

– Відносини (*association*). В *UML* є декілька стандартних видів відносин між акторами та варіантами використання:

Відношення асоціації (*association relationship*) слугують для позначення специфічної ролі актора в окремому варіанті використання. Це відношення встановлює, яку конкретну роль грає актор під час взаємодії з екземпляром варіанта використання. На діаграмі варіантів використання, відношення асоціації позначається суцільною лінією між актором і варіантом використання. Ця лінія може мати додаткові умовні позначення, такі, наприклад, як ім'я та кратність.

Відношення розширення (*extend relationship*) визначає взаємозв'язок примірників окремого варіанта використання з більш загальним варіантом, властивості якого визначаються на основі способу спільного об'єднання даних екземплярів.

Відношення розширення між варіантами використання позначається пунктирною лінією зі стрілкою, спрямованої від того варіанта використання, який є розширенням для початкового варіанта використання. Дана лінія зі стрілкою позначається ключовим словом «*extend*» («розширює»).

Відношення узагальнення (*generalization relationship*) слугує для вказівки того факту, що деякий варіант використання *A* може бути узагальнений до варіанта використання *B*. У цьому випадку варіант *A* буде спеціалізацією варіанта *B*. До того ж *B* називається предком або батьком по відношенню *A*, а варіант *A* – нащадком по відношенню до варіанта використання *B*. Слід підкреслити, що нащадок успадковує всі властивості та поведінку свого батька, а також може бути доповнений новими властивостями й особливостями поведінки. Графічно дане відношення позначається суцільною лінією зі стрілкою у формі незафарбованого трикутника, яка вказує на батьківський варіант використання. Ця лінія зі стрілкою має спеціальну назву – стрілка «узагальнення».

Відношення включення (*include relationship*) між двома варіантами використання вказує, що деяка задана поведінка для одного варіанта використання включається як складовий компонент в послідовність поведінки іншого варіанта використання. Графічно дане відношення позначається пунктирною лінією зі стрілкою (варіант відносини залежності), спрямованої від базового варіанта використання до такого, що включається. Дана лінія зі стрілкою позначається ключовим словом «*include*» («включає»).

Специфікація варіанта використання

Існують різні шаблони опису варіантів використання. Розглядаються наступні основні стилі опису:

- Вільний формат.
- Повний формат (запропонований А. Коберном).
- Таблиця у дві колонки.
- Таблиця у три колонки.

• Стиль *RUP*.

Крім того, іноді доцільно використовувати:

• Псевдокод.

• Діаграму активності *UML*.

• Інші графічні моделі.

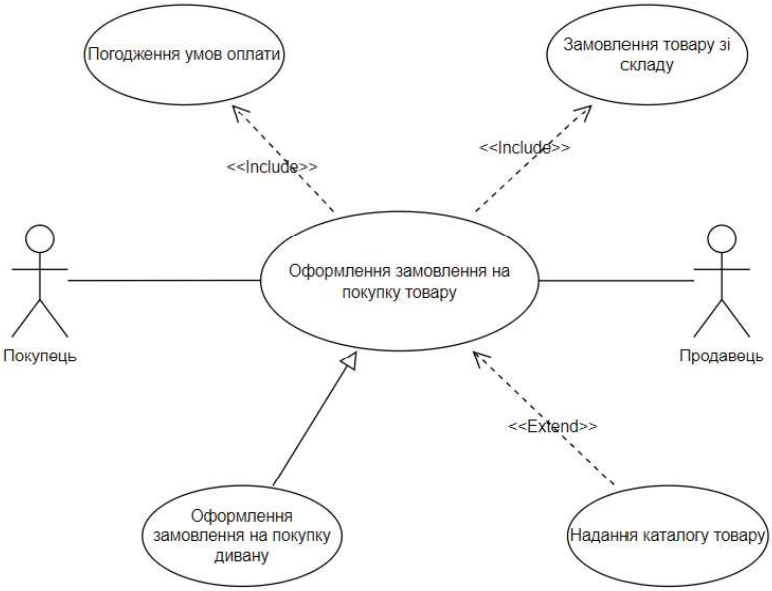


Рис. 2.1. Приклад графічного зображення відношень між варіантами використання

Вільний формат припускає опис дій користувача і системи в описовому стилі, наприклад: «Менеджер запрошує в пошуковій системі список замовлень за період. Система відображає на екрані знайдені замовлення даного Менеджера із вказівкою їх основних атрибутів». Вільний стиль допускає використання конструкцій «*Якщо..., то*». «*Якщо* Менеджер має повноваження Начальника Відділу, *то* Система надає можливість перегляду замовлень усіх менеджерів цього відділу».

Шаблон повного опису варіанта використання по А. Коберну [10]

Назва <коротка фраза у вигляді дієслова в невизначеній формі досконалого виду, що відображає мету>.

Контекст використання <уточнення мети, у разі потреби – умови її нормального завершення>.

Область дії <посилання на рамки проєкту>. Наприклад – підсистема бухгалтерського обліку.

Рівень <один із трьох: узагальнений, цілі користувача, підфункції>. Автор задає певну трирівневу класифікацію вимог, що загалом відповідає класифікації вимог на бізнес-вимоги, вимоги користувачів та функціональні вимоги.

Основна дійова особа <ім'я ролі основного актора або його опис>.

Учасники та інтереси <список інших акторів-учасників прецеденту із зазначенням їх інтересів>.

Передумова <те, що очікується, уже має місце>.

Мінімальні гарантії <що гарантується акторам-учасникам>. Наприклад – у разі невдалої транзакції всі дані, що були в системі до її початку, зберігаються постійними.

Гарантії успіху <що отримають актори-учасники в разі успішного досягнення мети>.

Тригер <те, що «запускає» варіант використання, зазвичай – подія в часі>.

Основний сценарій <тут перераховуються кроки основного сценарію, починаючи від тригера і до досягнення гарантії успіху>.

Формат опису:

<Номер кроку> <Опис дії>.

Розширення <тут послідовно описуються всі альтернативні сценарії>. Кожна з альтернатив прив'язана до основного сценарію. Формат опису:

<Номер кроку. Номер розширення> <Умова>:<Дія або посилання на підлеглий варіант використання>.

Будь-який крок основного сценарію може мати 1 або більше розгалужень. Кожне розгалуження оформляється як розширення. У блоці «Розширення» усі розширення описуються послідовно.

Якщо альтернативний сценарій не вдається описати одним рядком – застосовується наступний формат. Починаючи з рядка, наступного після опису розширення, іде опис його дій у форматі основного сценарію:

<Номер кроку. Номер розширення. Номер кроку розширення> <Дія>.

Опис розширення закінчується описом виходу із розширення. Основні варіанти виходу із розширення: повернення до чергового за номером кроку основного сценарію, закінчення варіантів використання, перехід до іншого кроку основного сценарію.

Список змін у технології та даних <що гарантується акторам-учасникам>. Наприклад – у разі невдалої транзакції всі дані, що були в системі до її початку, зберігаються постійними.

Додаткова інформація <додаткова інформація, корисна для опису варіанта використання>.

Табличні представлення варіанта використання

Іноді зручно поміщати сценарії варіантів використання в таблицю, як це показано нижче. Інформація до того ж набирає більш структурованого вигляду.

Таблиця у 2 колонки:

Актор	Дія
Користувач	Формує запит на пошук замовлень
Система	Відображає список замовлень
Користувач	Вибирає необхідне замовлення
Система	Показує докладну інформацію за замовленням

Таблиця у 3 колонки:

№ кроку	Користувач	Система
1	Робить запит на пошук замовлень	Відображає список замовлень
2	Вибирає необхідне замовлення	Показує докладну інформацію за замовленням

Шаблон варіанта використання *RUP* (*Rational Unified Process*)

RUP (*Rational Unified Process* – методологія розробки програмного забезпечення створена компанією *Rational Software*) – це ітеративна модель розробки. У кінці кожної ітерації (від 2 до 6 тижнів) розробники повинні досягти запланованих на дану ітерацію цілей, створити або доробити проєктні артефакти й отримати проміжну функціональну версію програмного забезпечення Така розробка дозволяє швидко реагувати на нові вимоги, виявити й усунути ризик на ранніх стадіях проєкту, а також високоефективно контролювати якість створюваного продукту. Процес розробки поділяється на 4 фази: початок (*inception*), проєктування (*elaboration*), побудова (*construction*), упровадження (*transition*) [11].

З шаблоном опису варіанта використання *RUP* і прикладами можна ознайомитися в інтерактивній версії *RUP*. Нижче наведений короткий огляд його розділів.

1. *Найменування і короткий опис*.

У цьому розділі вказується: найменування варіанта використання, актори варіанта використання, короткий (у один абзац) опис варіанта використання.

2. *Потік подій*.

Основний потік подій – перераховуються кроки основного сценарію, починаючи від тригера й аж до досягнення гарантії успіху.

Альтернативні потоки подій – кожен з альтернативних сценаріїв описується в окремому параграфі, у тому ж стилі, що і основний потік подій. Альтернативні сценарії описують поведінку системи за будь-яких відхилень від основного сценарію, а також поведінку у виняткових ситуаціях.

3. *Спеціальні вимоги* – перераховуються нефункціональні вимоги, що мають безпосереднє відношення саме до цього варіанта використання.

4. *Передумови* – події, що описуються передумовами або умовами поста, повинні бути станами, які користувач може

спостерігати. Передумову описує стан, у якому система повинна знаходитися до початку виконання варіанта використання.

5. *Постумови* – описують те, що коректно сформульована постумова повинна бути істинною за будь-якого можливого сценарію варіанта використання.

6. *Точки розширення* – визначають положення точок, що розширюють потік подій.

Вибір форми опису варіанта використання

Під час вибору форми і ступеня подробиці варіанта використання слід ураховувати такі чинники, як:

– розміри проєкту;

– важливість проєкту і варіанту використання;

– традиції, що склалися в колективі «Замовник-Розробник».

Для невеликого проєкту навряд чи буде доцільним застосовувати описи варіантів використання в розгорненому форматі, досить використовувати коротку форму вільного стилю. Для проєкту, де задіяно більше десяти учасників, виникають проблеми розбиття на мікроколективи, координації учасників, слід вибрати більш формалізований і докладніший варіант. Ступінь деталізації залежить також від критичності проєкту в цілому і критичності варіанта використання в даному проєкті.

А. Коберн [10] ділить усі програмні проєкти за ступенем критичності на 4 категорії, виходячи з ціни помилок, проєкти, помилки в яких можуть призвести до:

– небезпеки для життя людей;

– непоправним фінансовим витратам;

– фінансовим витратам в обмеженому об'ємі;

– зниженню комфортності кінцевого користувача.

Специфікація нефункціональних вимог

Опис нефункціональних вимог зазвичай здійснюється у формі, близькій до вільного формату опису варіанта використання. Методологія *RUP* рекомендує концентрувати нефункціональні вимоги в документі, що описує варіант вико-

ристання в усіх випадках, коли це можливо. У випадку, якщо нефункціональні вимоги носять загальний характер і не можуть бути прив'язані до конкретного прецеденту – вони виносяться в документ «Додаткова специфікація».

Атрибути вимог. Описи вимог повинні бути операбельні. Для цього всі вимоги повинні враховуватися в тій або іншій обліковій системі, чи то електронна таблиця *MS Excel*, спеціалізована база даних або інтегроване середовище управління змінами. Під час реєстрації вимоги вона проходить класифікацію відповідно до певної системи ознак. Для оперативного управління вимогами буває корисно призначити їм такі властивості, як проєкт, відповідальну особу, статус, ризик, ступінь закінченості і т. п. У *RUP* для управління атрибутами вимог передбачений артефакт «Атрибути вимог».

Артефакт «Атрибути вимог», пропонований *RUP*, є репозиторій (місце, де зберігаються і підтримуються будь-які дані) текстів вимог, їх атрибутів і трасованості. Атрибути вимог представлені матрицею атрибутів вимог, де для кожного типу вимог перераховуються вимоги по одній осі і атрибути вимог цього типу по іншій. Для кожної вимоги вказуються значення її відповідних атрибутів.

Приклади атрибутів: статус у часі, пріоритет, важливість, ризик № ітерації (етапу) у плані. Трасованість описується у вигляді дерева, що показує у графічному вигляді вхідні і/або витікаючі зв'язки трасованості.

Завдання

Побудувати діаграму варіантів використання та розробити специфікацією у відповідності до отриманого індивідуального завдання, а також до всіх необхідних вимог та викладених теоретичних відомостей.

Лабораторна робота № 3

РОЗРОБКА ТЕХНІЧНОГО ЗАВДАННЯ

Мета роботи: вивчити основні принципи й отримати базові навички підготовки технічного завдання на розробку програмного забезпечення.

Короткі теоретичні відомості

Технічне завдання (ТЗ) (англ. *Product Requirements Document*; *PRD*) –документ, що встановлює основне призначення, показники якості, техніко-економічні та спеціальні вимоги до виробу, обсягу, стадії розроблення та складу конструкторської документації.

Технічне завдання є вихідним документом для проєктування споруди (архітектурного комплексу), конструювання технічного пристрою (приладу, машини тощо), розробки автоматизованої системи чи інформаційної системи, створення програмного продукту, проведення науково-дослідних робіт (НДР) і дослідно-конструкторських робіт (ДКР).

Технічне завдання на надання послуг встановлює основні вимоги до робіт у певній сфері діяльності, обов'язки замовника й виконавця, показники якості, терміни проведення і звітності та економічні показники послуг. Цей документ використовують окремо або у складі договору (контракту) на виконання робіт.

Наявність ТЗ зумовлена розподілом праці між/на стадіях (етапах) життєвого циклу виробу. Функції ТЗ може виконувати інший документ (договір, угода, контракт, протокол тощо), який містить необхідні та достатні вимоги для виконання роботи з відповідним об'єктом і визнаний сторонами такого документа [21].

Структура технічного завдання

Згідно з ГОСТом 19.201–78 технічне завдання повинно містити такі розділи:

1. Вступ.

У розділі «Вступ» указують найменування, коротку характеристику області застосування програми чи програмного виробу та об'єкта, у якому використовують програму чи програмний виріб.

2. Підстави для розробки.

У розділі «Підстави для розробки» повинно бути зазначено:

– документ (документи), на підставі якого ведеться розроблення;

– організація, що затвердила цей документ, і дата його затвердження;

– найменування і/або умовне позначення теми розробки.

3. Призначення розробки.

У розділі «Призначення розробки» повинно бути зазначене функціональне та експлуатаційне призначення програми чи програмного виробу.

4. Вимоги до програми чи програмного виробу.

Розділ «Вимоги до програми чи програмного виробу» повинен містити наступні підрозділи:

– *Вимоги до функціональних характеристик*. У підрозділі «Вимоги до функціональних характеристик» повинно бути зазначено вимоги до складу виконуваних функцій, організації вхідних і вихідних даних (перелічені всі вхідні й вихідні дані та зазначено всі відомі вимоги до їх організації), тимчасових характеристик тощо. У даному підрозділі обов'язково має бути зазначено, якими групами користувачів та яким чином (із зазначенням відповідних функцій програми) планується використання програмного забезпечення.

– *Вимоги до надійності*. У підрозділі «Вимоги до надійності» повинно бути зазначено вимоги до забезпечення надійного функціонування (забезпечення стійкого функціонування,

контролю початкової та вихідної інформації, часу відновлення після відмовлення тощо). У даному підрозділі визначається, які саме збійні ситуації, ситуації неправильного введення даних користувачем тощо, мають оброблятися програмою.

– *Умови експлуатації*. У підрозділі «Умови експлуатації» повинно бути зазначено умови експлуатації (температура навколишнього повітря, відносна вологість тощо для обраних типів носіїв даних), за яких повинні забезпечуватися задані характеристики, вид обслуговування, необхідна кількість і кваліфікація персоналу (із зазначенням задач, які такий персонал повинен виконувати).

– *Вимоги до складу та параметрів технічних засобів*. У підрозділі «Вимоги до складу і параметрів технічних засобів» указують необхідний склад технічних засобів із зазначенням їхніх основних технічних характеристик.

– *Вимоги до інформаційної та програмної сумісності*. У підрозділі «Вимоги до інформаційної і програмної сумісності» повинно бути зазначено вимоги до інформаційних структур на вході і виході та методів розв'язання, вихідних кодів, мов програмування та програмних засобів, що використовуються програмою. За потреби повинен забезпечуватися захист інформації та програм.

– *Вимоги до маркування та упакування*. У підрозділі «Вимоги до маркування та упакування» у загальному випадку вказують вимоги до маркування програмного виробу, варіанти та засоби упакування.

– *Вимоги до транспортування та збереження*. У підрозділі «Вимоги до транспортування та збереження» повинні бути зазначені для програмного виробу умови транспортування, місця збереження, умови збереження, умови складування, терміни збереження в різних умовах.

– *Спеціальні вимоги*. У підрозділі «Спеціальні вимоги» у даній лабораторній роботі потрібно навести вимоги до графічного інтерфейсу користувача. Для цього потрібно виконати попереднє проєктування інтерфейсу. Для таких цілей

існує багато доволі простих і зручних програмних засобів, частина з яких є безкоштовними.

Вимоги до програмної документації.

У розділі «Вимоги до програмної документації» повинен бути зазначений попередній склад програмної документації і за потреби спеціальні вимоги до неї.

4. Техніко-економічні показники.

У розділі «Техніко-економічні показники» повинні бути зазначені: орієнтована економічна ефективність, передбачувана річна потреба, економічні переваги розробки порівняно з кращими вітчизняними та закордонними аналогами.

5. Стадії та етапи розробки.

У розділі «Стадії та етапи розробки» установлюють необхідні стадії розроблення, етапи та зміст робіт (перелік програмних документів, що повинні бути розроблені, погоджені та затверджені), а також зазвичай терміни розроблення, та визначають виконавців.

6. Порядок контролю та приймання.

У розділі «Порядок контролю та приймання» повинні бути зазначені види іспитів і загальні вимоги до приймання роботи. У даному розділі можуть також зазначатися конкретні випробування, які мають проводитися для контролю та приймання програмного продукту [22].

У ТЗ допускається включати додатки.

У залежності від особливостей програми чи програмного виробу допускається уточнювати зміст розділів, вводити нові розділи чи поєднувати окремі з них.

Технічне завдання на розробку програмного забезпечення відіграє величезну роль під час проєктування продукту, оскільки містить список ключових вимог і описує технічні засоби для реалізації ідеї замовника.

Під час написання ТЗ важлива гранична ясність – чим більше конкретики в документі, тим краще. Після його вивчення ні у клієнта, ні у команди розробників не повинно залишатися питань, що стосуються проєкту.

Якщо технічне завдання складено коректно, це дозволяє мінімізувати трудовитрати і уникнути багатьох проблем, обумовлених нерозумінням між сторонами: розбіжностей у критеріях оцінки, тривалого узгодження, зміни технічних параметрів і т. п.

Проєктування програмного забезпечення вимагає чітко структурованого підходу, оскільки готовий продукт повинен виконувати всі покладені на нього функції точно й надійно. Саме тому не слід нехтувати ранніми етапами розробки, що включають у себе аналіз вихідних даних, прототипування і написання ТЗ.

Під час роботи над проєктом нерідко виникає множина нових ідей, і головне в такій ситуації – уміти фокусуватися на початковій парадигмі, закладеної в технічному завданні. Заздалегідь складений документ дозволяє легко виміряти всі творчі розробки та проаналізувати їх відповідно до затверджених параметрів.

Залежно від форми подачі й розгорнення, можна виділити два типи технічних завдань:

1) Ескіз – документ, у якому фіксується загальний опис ПЗ, без урахування технологічного аспекту реалізації цифрового рішення. Він слугує для кращого розуміння завдання.

2) Повний технічний проєкт – різновид ТЗ, який включає в себе всі інструменти та технічні засоби. Його пряма реалізація приводить до створення програмного продукту.

Уникнути більшості суперечливих ситуацій допомагає документ, складений у вигляді повноцінного технічного проєкту. Такий варіант буде вигідним як для замовника, так і для розробника.

Під час опису призначення ПЗ слід навести основні сценарії роботи користувачів з ним, докладно охарактеризувати набір ролей для користувачів, їх права та доступи, указати математичні методи, моделі, типові алгоритми та алгоритми, що знаходяться у стадії розробки. Якщо система повинна інте-

груватися з іншими програмами або задіяти інші ресурси, то необхідно вказати їх у вигляді списку.

Технічне завдання може включати в себе різні зведені таблиці, схеми, діаграми, розрахунки та інші додаткові матеріали, які використовуються у проєктуванні.

Завдання

Розробити ТЗ у відповідності до отриманого індивідуального завдання, а також до всіх необхідних вимог та викладених теоретичних відомостей. Необхідно враховувати, що отримана тема та відповідні завдання мають ґрунтуватися на наступних принципах: розроблювальна система має працювати з певним сховищем даних, має виконувати обробку та збереження даних, повинна забезпечуватися можливість користування різних груп користувачів, а також мати візуальний користувацький інтерфейс.

Лабораторна робота № 4

РОЗРОБКА ЕСКІЗНОГО ПРОЄКТУ

Мета роботи: навчитися створювати формальні моделі та на їх основі визначати специфікації програмного забезпечення, що розробляються.

Короткі теоретичні відомості

Ескізний проєкт (англ. *Preliminary design*) – проєктна конструкторська документація, яка містить принципові конструктивні рішення і дає загальне уявлення про будову та принцип дії виробу, а також дані, що визначають його відповідність призначенню [23].

Ескізний проєкт, як стадія проєктування, передбачається ЄСКД (Єдина система конструкторської документації – комплекс державних стандартів, що встановлюють взаємопов'язані правила, вимоги і норми з розробки, оформлення й обігу конструкторської документації, що розробляється і застосовується на всіх стадіях життєвого циклу виробу) під час розроблення конструкторської документації і ЄСПД (Єдина система програмної документації – комплекс міждержавних стандартів, що встановлюють взаємопов'язані правила розробки, оформлення та обігу програм і програмної документації) під час розроблення програмної документації (сукупність документів, що містять відомості, необхідні для розробки, виготовлення, супроводу та експлуатації програм).

Ескізний проєкт розробляють з метою встановлення принципових (конструктивних, схемних та ін.) рішень виробу, що дають загальне уявлення про принцип роботи і/або будову виробу, коли це доцільно зробити до розробки технічного проєкту чи робочої документації. На стадії розробки ескізного проєкту розглядають варіанти виробу і/або його складових

частин. Ескізний проєкт може розроблятися і без розгляду на цій стадії різних варіантів.

У комплект документів ескізного проєкту включають конструкторські документи, згідно з ГОСТ 2.102-2013 передбачені технічним завданням і протоколом розгляду технічної пропозиції.

Форма подання документів ескізного проєкту (паперова чи електронна), якщо вона не вказана в технічному завданні та/або протоколі розгляду технічної пропозиції, визначається розробником за погодженням із замовником.

Розроблення ескізного проєкту під час створення програмного забезпечення передбачає:

– попередню розробку структури вхідних і вихідних даних;

– уточнення методів розв'язання задачі;

– розроблення загального опису алгоритму розв'язання задачі;

– розроблення техніко-економічного обґрунтування;

– розроблення пояснювальної записки;

– погодження та затвердження ескізного проєкту.

Розробка ПЗ починається з аналізу вимог до нього. У результаті аналізу отримують специфікації програмного забезпечення, що розробляється, будують загальну модель його взаємодії з користувачем або іншими програмами і конкретизують його основні функції. Під час структурного підходу до програмування на етапі аналізу і визначення специфікацій розробляють три типи моделей: моделі функцій, моделі даних і моделі потоків даних. Оскільки різні моделі описують проєктоване програмне забезпечення з різних боків, рекомендується використовувати відразу кілька моделей, що розробляються у вигляді діаграм, і пояснювати їх текстовими описами, словниками і т. п.

Структурний аналіз передбачає використання наступних видів моделей:

– діаграм потоків даних (*DFD – Data Flow Diagrams*), які описують взаємодію джерел і споживачів інформації через процеси, що повинні бути реалізовані в системі. Вони дозволяють описати необхідну поведінку системи у вигляді сукупності процесів, що взаємодіють за допомогою потоків даних, що їх зв'язують. *DFD* показує, як кожен з процесів перетворює свої вхідні потоки даних у вихідні потоки даних і як процеси взаємодіють між собою;

– діаграм «сутність-зв'язок» (*ERD – Entity-Relationship Diagrams*), які описують бази даних системи, що розробляється. Діаграма сутність-зв'язок – інструмент розробки моделей даних, що забезпечує стандартний спосіб визначення даних і відношень між ними. Вона включає в себе сутності та взаємозв'язки, що відображають основні бізнес-правила предметної області. У діаграму включаються основні сутності та зв'язки між ними, які задовольняють вимогам, що ставляться до інформаційних систем;

– діаграм переходів станів (*STD – State Transition Diagrams*), які характеризують поведінку системи у часі. За допомогою діаграм переходів станів можна моделювати подальше функціонування системи на основі її попереднього і поточного функціонування. Модельована система в будь-який заданий момент часу знаходиться точно в одному з кінцевої множини станів. З плином часу вона може змінити свій стан, до того ж переходи між станами повинні бути точно визначені;

– функціональних діаграм. Функціональні діаграми відбивають взаємозв'язки функцій програмного забезпечення, що розробляється. Вони створюються на ранніх етапах проєктування систем, для того, щоб допомогти проєктувальнику виявити основні функції і складові частини проєктованої системи і, якщо змога, виявити й усунути істотні помилки. Для створення функціональних діаграм пропонується використовувати методологію SADT (*Structured Analysis and Design Technique*);

– специфікацій процесів (зазвичай представляють у вигляді короткого текстового опису, схем алгоритмів, псевдокодів, *Flow*-форм або діаграм Нассі-Шнейдермана);

– словника термінів (являє собою короткий опис основних понять, що використовуються під час складання специфікацій. Він повинен включати в себе визначення основних понять предметної області, опис структур елементів даних, їх типів і форматів, а також усіх скорочень і умовних позначень [11].

Завдання

На основі технічного завдання з лабораторної роботи № 3 виконати аналіз функціональних та експлуатаційних вимог до програмного продукту. Визначити основні технічні рішення (вибір мови програмування, структура програмного продукту, склад функцій програмного продукту, режими функціонування), розробити діаграму діяльності, концептуальну модель даних у вигляді діаграми сутність-зв’язок та проєкт користувацького інтерфейсу.

Лабораторна робота № 5

ЕТАПИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ)

Мета роботи: засвоїти основні поняття про етапи розробки програмного забезпечення та програмну документацію.

Короткі теоретичні відомості

Розробка будь-якої програми складається з декількох етапів. Чітке дотримання етапів розробки програмного забезпечення (ПЗ) стає основним критерієм у створенні проєкту.

Аналіз вимог

Найпершим етапом розробки ПЗ є проведення аналізу висунутих замовником вимог, щоб визначити ключові цілі та завдання кінцевого продукту. У рамках цієї стадії відбувається максимально ефективна взаємодія клієнта і розробника, що допомагає чітко сформулювати вимоги, що висуваються до ПЗ. Результатом проведеного аналізу стає формування основного регламенту, на який спиратиметься виконавець у своїй роботі. Технічне завдання (ТЗ) має повністю описувати поставлені перед розробником завдання та охарактеризувати кінцеву мету проєкту у розумінні замовника.

Проєктування

Наступний ключовий етап розробки ПЗ – стадія проєктування. Найсучасніші засоби програмування дозволяють частково поєднати етапи проєктування та кодування, тобто технічної реалізації проєкту, будучи заснованими на об’єктно-орієнтованому підході, але повноцінне планування потребує більш ретельного моделювання. Якісний аналіз можливостей створюваного продукту стане основою для його повноцінного функціонування та виконання всього комплексу завдань.

Кодування

Наступним кроком стає робота з кодом, спираючись на обрану у процесі підготовки мову програмування. Якщо йдеться про написання коду для виконання вузькоспеціалізованих завдань у рамках конкретного підприємства, то від грамотного підходу до етапу кодування залежить ефективність роботи компанії, яка замовила розробку. Кодування може відбуватися паралельно з наступним етапом розробки – тестуванням програмного забезпечення, що допомагає вносити зміни безпосередньо під час написання коду. Рівень та ефективність взаємодії всіх елементів на поточному етапі є найважливішим.

Тестування та налагодження

Після написання коду йдуть тестування продукту і подальше налагодження, що дозволяє виявити несправності та досягти кінцевої мети. Процес тестування дозволяє змоделювати ситуації, у яких програмний продукт перестає функціонувати. Відділ налагодження потім локалізує та виправляє виявлені помилки коду. Ці два етапи займають не менше 30 % часу.

Упровадження

Процедура впровадження програмного забезпечення в експлуатацію є завершальною стадією розробки і нерідко відбувається разом із налагодженням системи. Зазвичай введення в експлуатацію ПЗ здійснюється у три етапи:

1. Первісне завантаження даних.
2. Поступове накопичення інформації.
3. Виведення створеного на проєктну потужність.

Ключовою метою поетапного застосування розробленої програми стає поступове виявлення не виявлених раніше помилок і недоліків коду. У рамках цього етапу і замовник, і виконавець можуть зіткнутися з низкою помилок досить вузького спектра.

Важливим етапом розробки програмного продукту є *складання програмної документації*. На кожен програмний

продукт повинні складатися два типи документації – для розробників і для різних груп користувачів. Програмна документація користувачів повинна містити всі необхідні відомості з експлуатації ПЗ. Документація розробника повинна містити відомості, необхідні для розробки й супроводу програмного забезпечення.

Документування програмного забезпечення здійснюється відповідно до Єдиної системи програмної документації (ГОСТ 19.ХХХ). ГОСТ 19.101-77 містить *види програмних документів* для програмного забезпечення різних типів [5]. У даному ГОСТі перераховані документи наступних типів:

• *специфікація* повинна містити перелік і короткий опис призначення всіх файлів програмного забезпечення, у тому числі і файлів документації на нього, і є обов'язковою для програмних систем, а також їх компонентів, що мають самостійне застосування;

• *відомість утримувачів оригіналів* повинна містити список підприємств, на яких зберігаються оригінали програмних документів. Необхідність цього документа визначається на етапі розробки та затвердження технічного завдання тільки для програмного забезпечення зі складною архітектурою;

• *текст програми* повинен містити текст програми з необхідними коментарями. Необхідність цього документа визначається на етапі розробки та затвердження технічного завдання;

• *опис програми* має містити відомості про логічну структуру і функціонування програми. Необхідність даного документа також визначається на етапі розробки та затвердження технічного завдання;

• *відомість експлуатаційних документів* повинна містити перелік експлуатаційних документів на програму. Необхідність цього документа також визначається на етапі розробки та затвердження технічного завдання;

• *формуляр* повинен містити основні характеристики програмного забезпечення, комплектність і відомості про експлуатацію програми;

• *опис застосування* має містити відомості про призначення програмного забезпечення, галузі застосування, застосовувані методи, класи вирішуваних завдань, обмеження для застосування, мінімальну конфігурацію технічних засобів;

• *керівництво системного програміста* має містити відомості для перевірки, забезпечення функціонування та налаштування програми на умови конкретного застосування;

• *керівництво програміста* має містити відомості для експлуатації програмного забезпечення;

• *керівництво оператора* містить відомості для забезпечення процедури спілкування оператора з обчислювальною системою у процесі виконання програми;

• *опис мови* – опис синтаксису і семантики мови програми;

• *керівництво з технічного обслуговування* містить відомості для застосування програми під час обслуговування технічних засобів.

Завдання

Описати етапи та стадії розробки, які були виконані під час реалізації програмного продукту у відповідності до отриманого індивідуального завдання, а також до всіх необхідних вимог та викладених теоретичних відомостей. Оформити документацію до розробленого програмного забезпечення.

Стадії та етапи розробки програми та терміни їх виконання мають відповідати затвердженому графіку (табл. 5.1).

Таблиця 5.1. **Стадії та етапи розробки програми, терміни їх виконання**

Етапи робіт	Час виконання
1. Аналіз і формування вимог	
2. Проєктування програмного забезпечення	
3. Кодування	
4. Тестування та налагодження	
5. Упровадження, супровід системи	

Лабораторна робота № 6

ТЕСТУВАННЯ ПРОГРАМИ ЗА ПРИНЦИПОМ «БІЛОЇ СКРИНЬКИ» ТА «ЧОРНОЇ СКРИНЬКИ»

Мета роботи: ознайомитися з методами тестування програмного забезпечення, які використовуються для виявлення помилок і дефектів у програмах.

Короткі теоретичні відомості

Тестування – невід'ємна складова процесу програмної інженерії, один з методів подальшого поліпшення якості розробленого програмного забезпечення системи за допомогою виявлення дефектів, що залишилися, не виявлених раніше іншими видами перевірок.

Тестування програмного забезпечення (англ. *Software Testing*) – техніка контролю якості, що перевіряє відповідність між реальною й очікуваною поведінкою програми завдяки кінцевому набору тестів, які обираються певним чином [1].

Тестування програмного забезпечення – 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 із метою оцінки [20].

Процес тестування складається з декількох етапів:

– тестування компонентів;
– тестування модулів;
– тестування підсистем;
– тестування системи;
– приймальні випробування.

Головна мета тестування ПЗ – підтвердження якості програмного комплексу шляхом систематичного налагодження

додатків у ретельно контрольованих умовах, визначення їх повноти та коректності, а також виявлення прихованих помилок.

Статичне та динамічне тестування

Тестова діяльність, що пов'язана з аналізом результатів розробки програмного забезпечення, називається *статичним тестуванням*. Воно передбачає перевірку програмних кодів, контроль та перевірку програми без запуску на комп'ютері. Тестова діяльність, що передбачає експлуатацію програмного продукту, називається *динамічним тестуванням*. Динамічне та статичне тестування доповнюють одне одного.

На етапі статичного тестування перевіряється вся документація, отримана як результат життєвого циклу програми. Це технічне завдання, специфікація, вихідний текст програми на мові програмування. Уся документація аналізується на предмет дотримання стандартів програмування. У результаті статичної перевірки встановлюється, наскільки програма відповідає заданим критеріям та вимогам замовника. Усунення неточностей та помилок у документації – запорука того, що створюваний програмний засіб має високу якість.

Динамічні методи застосовуються у процесі безпосереднього виконання програми. Коректність програмного засобу перевіряється на безлічі тестів або наборів підготовлених вхідних даних. Під час прогону кожного тесту збираються та аналізуються дані про відмови та збої в роботі програми.

Структурне тестування

Метод структурного тестування припускає створення тестів на основі структури системи й її реалізації. Такий підхід іноді називають тестуванням методом «білої скриньки», «скляної скриньки» або «прозорої скриньки», щоб відрізняти його від тестування методом «чорної скриньки».

Загалом структурне тестування застосовується до відносно невеликих програмних елементів, наприклад, до підпрограм або методів, що асоціюються з об'єктами. Під час такого

підходу випробувач аналізує програмний код і для отримання тестових даних використовує знання про структурну компоненту. Наприклад, з аналізу коду можна визначити, скільки контрольних тестів потрібно виконати для того, щоб у процесі тестування всі оператори виконалися принаймні один раз.

Метод «білої скриньки» (*White Box*) [14]

Відома: внутрішня структура програми.

Досліджуються: внутрішні елементи програми і зв'язки між ними.

Об'єктом тестування тут є не зовнішня, а внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним. Зазвичай аналізуються керуючі звязки елементів, рідше – інформаційні зв'язки. Тестування за принципом «білої скриньки» характеризується ступенем, у якому тести виконують або покривають логіку (вихідний текст) програми.

Особливості тестування «білої скриньки»

Зазвичай тестування «білої скриньки» засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, у яких:

• гарантується перевірка всіх незалежних маршрутів програми;

• знаходяться гілки *True, False* для всіх логічних рішень;

• виконуються всі цикли (у межах їхніх кордонів та діапазонів);

• аналізується правильність внутрішніх структур даних.

Недоліки тестування «*білої скриньки*»:

• кількість незалежних маршрутів може бути дуже велика;

• повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї;

• у програмі можуть бути пропущені деякі маршрути;

• не можна виявити помилки, поява яких залежить від даних.

Переваги тестування «білої скриньки» пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

Під час запису алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

Деякі результати у програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Кожна з цих причин є аргументом для проведення тестування за принципом «білої скриньки». Тести «чорної скриньки» не зможуть реагувати на помилки таких типів.

Тестування програми з посиланням на внутрішню структуру програмного компонента називається *тестуванням «білої скриньки»*.

Техніка тестування «білої скриньки».

Процедура виведення та/або відбору тестових випадків на основі аналізу внутрішньої структури компонента або системи.

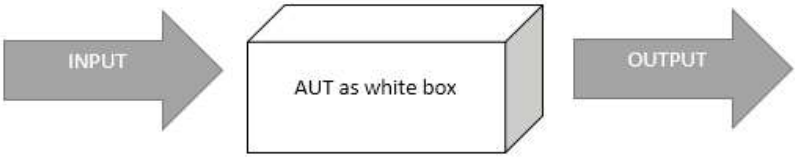


Рис. 6.1. Тестування *White Box*

Тестування «білої скриньки» – це тестовий підхід, який використовується для тестування частини реалізації програми, що тестувалась. Для проведення цього тестування тестувальник,

можливо розробник, повинен знати внутрішню структуру програми та її роботу.

Приклад. Автомеханік повинен знати внутрішню будову двигуна автомобіля для його ремонту.

У цьому прикладі автомобіль є *AUT* (тестується заявка). Користувач є тестер «чорної скриньки». Механік є тестер «білої скриньки».

Це основні визначення тестування «білої» та «чорної скриньок», і кожен метод тестування має різні методи, яких слід дотримуватися.

Тестування методом «чорної скрині» (*Black Box*)

Функціональне тестування, або тестування методом «чорної скриньки» базується на тому, що всі тести ґрунтуються на специфікації системи або її компонентів. Система представляється як «чорна скринька», поведінку якої можна визначити тільки за допомогою вивчення її вхідних і відповідних вихідних даних. Інша назва цього методу – «функціональне тестування» пов'язана з тим, що випробувач перевіряє не реалізацію ПЗ, а тільки його виконувані функції.

Відомі: функції програми.

Досліджується: робота кожної функції на всій області визначення.

Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ.

Ці тести демонструють:

- як виконуються функції програми;
- як приймаються вихідні дані;
- як виробляються результати;
- як зберігається цілісність зовнішньої інформації.

Під час тестування «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування загалом неможливе. Наприклад, якщо у програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів ста-

новитиме 1010. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми. Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів [15]. Під час такого підходу бажано мати:

набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ІТ);

набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його ОТ).

Будь-який спосіб тестування «чорної скриньки» повинен:

• виявити такі вхідні дані, які з високою ймовірністю належать набору ІТ;

• сформулювати такі очікувані результати, які з високою ймовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше він доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

• некоректних чи відсутніх функцій;

• помилок інтерфейсу;

• помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;

• помилок характеристик (необхідна ємність пам’яті і т. д.);

• помилок ініціалізації та завершення.

Подібні категорії помилок способами «білої скриньки» не виявляються.

Тестування «чорної скриньки» – це підхід тестування, який використовується для перевірки функціональності *AUT* (тестування тестової програми) на основі специфікацій / *SRS*

(*Software Requirements Specification*) без будь-якого знання технології, що використовується для реалізації програми, що тестована.

Рис. 6.2. Тестування *Black Box*

Під час тестування в «чорній скриньці» основне тестування стосуватиметься можливих входів та очікуваних результатів. Тестер повинен мати можливість ретельно вибирати дійсні дані тесту. Говорячи простими словами, тестувальник може бачити лише дії *AUT*. Тестувальник не повинен знати, як виконуються ці дії.

Приклад. Простим прикладом тестування «чорної скриньки» є телевізор. Користувач дивиться телевізор, але йому не потрібні знання про те, як побудований телевізор, як він працює і таке інше. Лише потрібно знати, як керувати пультом дистанційного керування, щоб увімкнути, вимкнути, змінити канали, збільшення / зменшення гучності тощо.

У цьому прикладі, телевізор є *AUT* (тестується заявка). Пульт – це користувальницький інтерфейс (*UI – user interface*), який використовуєте для тестування. Лише потрібно знати, як користуватися додатком.

Таблиця 6.1. **Різниця між тестуванням *Black Box* та *White Box***

№ з/п	Тестування *Black Box*	Тестування *White Box*
1	Основна мета цього тестування – перевірити функціональність / поведінку програми	Основна мета – перевірити інфраструктуру програми
2	Це може виконати тестувальник без будь-яких знань щодо кодування *AUT* (*Application Test Test*)	Тестер повинен мати знання про внутрішню структуру, та як це працює

Продовж. табл. 6.1

№ з/п	Тестування *Black Box*	Тестування *White Box*
3	Тестування можна проводити лише за допомогою графічного інтерфейсу користувача	Тестування можна провести на ранній стадії до того, як графічний інтерфейс готується
4	Це тестування не може охопити всі можливі вхідні дані	Це тестування є більш ретельним, оскільки воно може перевірити кожен шлях
5	Деякі методи тестування включають у себе аналіз граничних значень, розподіл еквівалентності, вгадування помилок тощо	Деякі методи тестування включають у себе умовне тестування, тестування потоку даних, тестування циклу тощо
6	Тестові кейси слід писати на основі специфікації вимог	Тестові кейси слід писати на основі детального проєктного документа
7	Тестові кейси матимуть більше подробиць про умови введення, етапи тестування, очікувані результати та дані тесту	Тестові кейси будуть простими з деталями технічних концепцій, таких як твердження, охоплення коду тощо
8	Це виконують професійні тестувальники програмного забезпечення	Це відповідальність розробників програмного забезпечення
9	Знання програмування та впровадження не потрібні	Потрібні знання з програмування та впровадження
10	В основному використовується для тестування вищого рівня, такого як перевірка прийнятності, системне тестування тощо	Загалом використовується на нижчих рівнях тестування, таких як модульне тестування та інтеграційне тестування
11	Це менш трудомістке і вичерпне	Це більш трудомістке та вичерпне
12	Дані тестів матимуть широкі можливості, тому буде важко визначити правильні дані	Легко визначити дані тесту, оскільки одночасно фокусується лише конкретна частина функціональності
13	Основна увага тестера зосереджена на тому, як працює програма	Основна увага буде зосереджена на тому, як будується додаток
14	Покриття тестуванням менше, оскільки воно не може створити дані тесту для всіх сценаріїв	Майже всі шляхи / потік програми охоплені, оскільки це легко перевірити частинами
15	Помилки, пов’язані з кодом неможливо виявити, або визначити технічні помилки	Допомагає виявити приховані помилки та допомагає оптимізувати код

Продовж. табл. 6.1

№ з/п	Тестування *Black Box*	Тестування *White Box*
16	Дефекти виявляються після розробки базового коду	Можливо раннє виявлення дефектів
17	Користувач повинен мати можливість виявити будь-які відсутні функції, оскільки обсяг цього тестування широкий	Тестер не може визначити відсутні функції, оскільки область застосування обмежена лише реалізованою функцією
18	Доступ до коду не потрібен	Потрібен доступ до коду
19	Покриття тесту буде меншим, оскільки тестувальник має обмежені знання про технічні аспекти	Охоплення тестів буде більшим, оскільки тестувальник матиме більше знань про технічні концепції
20	Професійний тестер фокусується на тому, як працює вся програма	Основна увага тестувальника / розробника полягає в тому, щоб перевірити, чи працює конкретний шлях чи ні

Тестування методами «білої» та «чорної скриньок» необхідне для успішної доставки програмного забезпечення, проте 100 % тестування неможливе в жодному з випадків.

Основна відповідальність тестувальника полягає у визначенні відповідних видів та методів випробувань для конкретного застосування, що приведе до виявлення максимальних дефектів і, таким чином, підвищення ефективності застосування.

Тестер повинен мати можливість визначити, скільки тестування можна провести як у «чорній скриньці», так і в «білій скриньці», щоб підтвердити, що програма працює належним чином.

Області еквівалентності

Вхідні дані програм часто можна розбити на декілька класів. Вхідні дані, що належать одному класу, мають загальні властивості, наприклад, це додатні числа, від'ємні числа, рядки без пробілів і тому подібне. Зазвичай для всіх даних з яко-

го-небудь класу поведінка програми однакова (еквівалентна). Через це такі класи даних іноді називають *областями еквівалентності*. Один із системних методів виявлення дефектів полягає у визначенні всіх областей еквівалентності, що обробляються програмою. Контрольні тести розробляються так, щоб вхідні та вихідні дані лежали в межах цих областей.

Тестування гілок

Це метод структурного тестування, під час якого перевіряються всі незалежно виконувані гілки компонента або програми. Якщо виконуються всі незалежні гілки, то й усі оператори повинні виконуватися принаймні один раз. Більш того, усі умовні оператори тестуються як з дійсними, так і з помилковими значеннями умов. У об'єктно-орієнтованих системах тестування гілок використовується для тестування методів, що асоціюються з об'єктами.

Після того, як протестовані всі окремі програмні компоненти, виконується збірка системи, унаслідок чого створюється часткова або повна система. Процес інтеграції системи включає в себе збірку й тестування отриманої системи, у ході якого виявляються проблеми, що виникають під час взаємодії компонентів. Тести, перевіряючи збірку системи, повинні розроблятися на основі системної специфікації, причому тестування збірки слід починати відразу після створення працездатних версій компонентів системи.

Завдання

Ознайомитися з методами тестування програмного забезпечення, які використовуються для виявлення помилок і дефектів у програмах. Протестувати розроблений додаток чи програму методом «чорної» та «білої» скриньки.

СПИСОК ЛІТЕРАТУРИ

Рекомендована література

1. Авраменко А. С., Авраменко В. С., Косенюк Г. В. Тестування програмного забезпечення : навч. посіб. Черкаси : ЧНУ імені Богдана Хмельницького, 2017. 284 с.

2. Коцюба И. Ю., Чунаев А. В., Шиков А. Н. Основы проектирования информационных систем : учеб. пособ. СПб: Университет ИТМО, 2015. 206 с.

3. Петрик М. Р., Петрик О. Ю. Моделювання програмного забезпечення : науково-методичний посібник. Тернопіль : вид-во ТНТУ імені Івана Пулюя, 2015. 200 с.

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

5. Погромська Г. С., Махровська Н. А. Програмні проєкти: управління та розробка : навч.-метод. посіб. Миколаїв, 2017. 153 с.

6. Carola Lilienthal. Sustainable Software Architecture : Analyze and Reduce Technical Debt Dpunkt.Verlag, 2019. 310 p.

7. Kerzner H. Project Management: A System Approach to Planning, Schedulling and Controlling / H. Kerzner. – NY: John Wiley & Sons, 1998. p. 2.

8. Lewis James P. Project Planning, Schedulling and Control: A HandsOn Guide to Bringing Projects in on Time and on Budget / James P. Lewis. – rev ed. Chicago, IL: Irwin, 1995. pp. 2–3.

9. https://courses.prometheus.org.ua/courses/course-v1:LITS+115+2017_T4/course/ – ОСНОВИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

10. http://www.dstu.dp.ua/Portal/Data/3/19/3-19-l21.pdf.

11. http://dspace.wunu.edu.ua/bitstream/316497/24190/1/%D0%BA%D0%BE%D0%BD%D1%81%D0%BF%D0%B5%D0%BA%D1%82%20%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D0%B9.pdf .

12. Institute of Electrical and Electronics Engineers. IEEE Std 610.12.1990 Standard Glossary of Software Engineering Terminology // Software Engineering Collection. – NY: Institute of Electrical and Electronics Engineers, 1983.

13. https://dou.ua/forums/topic/13389/.

14. https://uk.ellas-cookies.com/kompyutery/58028-metody-testirovaniya-programmnogo-obespecheniya-i-ih-sravnenie-testirovanie-metodom-chernogo-yaschika-i-testirovanie-metodom-belogo-yaschika.html.

15. https://uk.myservername.com/key-differences-between-black-box-testing#What_Is_Black_Box_Testing.

16. https://uk.myservername.com/white-box-testing-complete-guide-with-techniques.

17. https://www.utcluj.ro/media/page_document/78/Foundations%20of%20software%20testing%20-%20ISTQB%20Certification.pdf .

18. https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%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 – Розробка програмного забезпечення.

19. https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D0%BC%D0%BE%D0%B3%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 – Вимоги до програмного забезпечення.

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

21. https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D1%96%D1%87%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B2%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F – Технічне завдання.

22. http://eir.zntu.edu.ua/bitstream/123456789/1256/1/Software_engineering_practice.pdf.

23. https://uk.wikipedia.org/wiki/%D0%95%D1%81%D0%BA%D1%96%D0%B7%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%94%D0%BA%D1%82 – Ескізний проєкт.

24. http://www.cad.dp.ua/gost/files/GOST19.101-77.pdf–Единая система программной документации. Виды программ и программных документов.

Допоміжна література

1. Дворянкин А. М., Ерофеев А. А., Аникин А. В. Основные методы тестирования программного обеспечения : учеб. пособ. Волгоград : ВолгГТУ, 2015. 120 с.

2. Котляров В. П. Основы тестирования программного обеспечения. 2-е изд., Москва : Интуит, 2016. 348 с.

3. Куликов С. С. Тестирование программного обеспечения. Базовый курс. Минск : Четыре четверти, 2017. 312 с.

4. Titus WintersTom ManshreckHyrum Wright Software Engineering at Google: Lessons Learned from Programming Over Time. - O'Reilly Media, 2020.

5. https://uk.myservername.com/best-software-testing-tools-2021.

ДОДАТОК
Приклад оформлення лабораторної роботи

					ЛР 01-06.121.3157.01			
Змін.		№ докум.	Підпис	Дата				
Розробив		Петров К.Н.			Лабораторні роботи з дисципліни «Проєктний практикум»	Літ.	Арк.	Аркушів
Перевірив		Карпова С.О.					1	
						ХННІ НУК		

Лабораторна робота № 4

Тема: Розробка ескізного проєкту

Мета роботи: навчитися створювати формальні моделі та на їх основі визначати специфікації програмного забезпечення, що розробляються.

Короткі теоретичні відомості

Ескізний проєкт (англ. *Preliminary design*) – проєктна конструкторська документація, яка містить принципові конструктивні рішення і дає загальне уявлення про будову та принцип дії виробу, а також дані, що визначають його відповідність призначенню.

Розроблення ескізного проєкту під час створення програмного забезпечення передбачає:

– попередню розробку структури вхідних і вихідних даних;

– уточнення методів розв'язання задачі;

– розроблення загального опису алгоритму розв'язання задачі;

– розроблення техніко-економічного обґрунтування;

– розроблення пояснювальної записки;

– погодження та затвердження ескізного проєкту.

Розробка ПЗ починається з аналізу вимог до нього. У результаті аналізу отримують специфікації програмного забезпечення, що розробляється, будують загальну модель його взаємодії з користувачем або іншими програмами і конкретизують його основні функції. Під час структурного підходу до програмування на етапі аналі-

зу і визначення специфікацій розробляють три типи моделей: моделі функцій, моделі даних і моделі потоків даних. Оскільки різні моделі описують проєктоване програмне забезпечення з різних боків, рекомендується використовувати відразу кілька моделей, що розробляються у вигляді діаграм, і пояснювати їх текстовими описами, словниками і т. п.

Структурний аналіз передбачає використання наступних видів моделей:

– діаграм потоків даних (*DFD – Data Flow Diagrams*), які описують взаємодію джерел і споживачів інформації через процеси, що повинні бути реалізовані в системі. Вони дозволяють описати необхідну поведінку системи у вигляді сукупності процесів, що взаємодіють за допомогою потоків даних, що їх зв'язують. *DFD* показує, як кожен з процесів перетворює свої вхідні потоки даних у вихідні потоки даних і як процеси взаємодіють між собою;

– діаграм «сутність-зв'язок» (*ERD – Entity-Relationship Diagrams*), які описують бази даних системи, що розробляється. Діаграма сутність зв'язок – інструмент розробки моделей даних, що забезпечує стандартний спосіб визначення даних і відношень між ними. Вона включає в себе сутності та взаємозв'язки, що відображають основні бізнес-правила предметної області. У діаграму включаються основні сутності та зв'язки між ними, які задовольняють вимогам, що ставляться до інформаційних систем;

– діаграм переходів станів (*STD – State Transition Diagrams*), які характеризують поведінку системи в часі. За допомогою діаграм переходів станів можна моделювати подальше функціонування системи на основі її

попереднього і поточного функціонування. Модельована система в будь-який заданий момент часу знаходиться точно в одному з кінцевої множини станів. З плином часу вона може змінити свій стан, до того ж переходи між станами повинні бути точно визначені;

– функціональних діаграм. Функціональні діаграми відбивають взаємозв'язки функцій програмного забезпечення, що розробляється. Вони створюються на ранніх етапах проєктування систем, для того, щоб допомогти проєктувальнику виявити основні функції і складові частини проєктованої системи і по змозі виявити та усунути істотні помилки. Для створення функціональних діаграм пропонується використовувати методологію *SADT* (*Structured Analysis and Design Technique*);

– специфікацій процесів (зазвичай представляють у вигляді короткого текстового опису, схем алгоритмів, псевдокодів, *Flow*-форм або діаграм Нассі-Шнейдермана).

– словника термінів (являє собою короткий опис основних понять, що використовуються під час складання специфікацій. Він повинен включати в себе визначення основних понять предметної області, опис структур елементів даних, їх типів і форматів, а також усіх скорочень і умовних позначень.

Завдання

На основі технічного завдання з лабораторної роботи № 3 виконати аналіз функціональних та експлуатаційних вимог до програмного продукту «Температурний логер-термостат». Визначити основні технічні рішення (вибір мови програмування, структура про-

грамного продукту, склад функцій програмного продукту, режими функціонування), розробити діаграму діяльності, концептуальну модель даних у вигляді діаграми сутність-зв'язок та проєкт користувацького інтерфейсу.

4.1. Основні вимоги до створюваного програмного продукту

Головна форма програмного забезпечення повинна мати параметри підключення до мікроконтролера, температурний графік з фільтрацією даних за часом та датчиком, налаштування термостата, список датчиків та друк звіту.

Підключення повинно відбуватися з використанням серійного порту комп'ютера, тому повинна бути можливість вибрати потрібний порт. Також програма повинна мати змогу змінювати період зчитування датчика. Зчитування та зупинка зчитування повинні бути виконані кнопками. Бажано вивести панель прогресу у разі великих періодів зчитування.

Під час запуску програми графік повинен показувати дані за останню добу. Повинна бути можливість змінювати розмір графіка. Діяльність термостата теж може бути виведена на графік за бажанням. Щоб показати дані за певний період часу повинно бути вікно запиту даних з полями для введення часу. Для більш зрозумілого подання даних потрібна функція відключення графіків певних датчиків.

Налаштування термостата можуть бути виконані в іншій формі та повинні мати умови увімкнення та вимкнення термостата, за якими датчиками ці умови

спостерігають та поле споживання електроенергії. Має бути можливість як зберегти, так і скинути введені параметри.

Список підключених датчиків має показувати код датчика, виміряну температуру та його опис. Має бути можливість змінювати опис датчиків та зберігати ці дані. Редагування опису всіх датчиків можна зробити в іншій формі.

Звіт повинен створюватися на основі часових меж указаних у полях запиту. Запит повинен містити часовий проміжок, список датчиків з максимальною, мінімальною та середньою температурою, список дій термостату та температурний графік на всю сторінку.

4.2. Постановка задачі

З викладених вимог до програмного продукту була розроблена постановка задачі.

Головною метою даної роботи є розробка програмного забезпечення, яке відповідає наступним вимогам:

1. Вимоги до складу виконуваних функцій.

– Збереження даних датчиків.

Вхідна інформація: номер датчика, дата та час, температура.

Вихідна інформація: запис у базі даних.

– Згладжування даних для графіка.

Вхідна інформація: температурні дані, розмір вікна.

Вихідна інформація: температурні дані.

– Виведення графіків температури.

Вхідна інформація: часовий проміжок, ідентифікатор датчика.

Вихідна інформація: температурний графік.

– Керування термостатом.

Вхідна інформація: номер датчика, температура.

Вихідна інформація: зміна стану порту мікроконтролера, запис у базі даних.

– Формування звіту.

Вхідна інформація: часовий проміжок.

Вихідна інформація: файл або друкована форма звіту.

– Зміна опису датчиків.

Вхідна інформація: номер датчика, опис датчика.

Вихідна інформація: оновлення опису датчика в базі даних.

2. Вимоги до контролю вхідної і вихідної інформації.

Програма повинна забезпечувати правильне введення інформації за рахунок використання, там, де це доцільно, шаблонів введення, процедурного блокування введення некоректної інформації, списків та автопідстановки. Обробка виняткових ситуацій, пов'язаних з доступом до дисків, пристроїв введення-виведення інформації, повинна оброблятися програмно з виведенням відповідних інформаційних повідомлень, не призводити до блокування роботи програми.

3. Вимоги до інформаційної і програмної сумісності.

Для роботи з програмним забезпеченням необхідно встановити на комп'ютер такі програмні продукти:

– операційну систему *Windows XP* і вище;

– пакет *Microsoft Office*;

– *Microsoft .NET Framework*.

4. Вимоги до складу й параметрів технічних засобів.

Для роботи програмного забезпечення необхідна наявність у користувача персонального комп'ютера, що має такі системні характеристики:

– комп'ютер з процесором x86 або x64, та тактовою частотою не менш ніж 1,5 ГГц;

– обсяг оперативної пам'яті не нижче 1ГБ;

– послідовний порт COM з роз'ємом DB-9;

– не менше 350 МБ вільного дискового простору жорсткого диска;

– монітор;

– клавіатура;

– миша.

В ескізному проєкті представлені діаграма варіантів використання, специфікації до варіантів використання, діаграми діяльності, концептуальна модель та представлений проєкт користувацького інтерфейсу.

4.3. Побудова моделі варіантів використання ПЗ «Температурний логер-термостат»

На рисунку 4.1 зображена модель варіантів використання програмного забезпечення «Температурний логер-термостат». Актор: оператор ПК. Прецеденти: редагування налаштувань, збереження налаштувань, скидання налаштувань, збір температурних даних, побудова графіка, побудова звіту, введення часового проміжку, управління термостатом.

Дана діаграма демонструє основні відносини між актором та прецедентами, дозволяючи описати та продемонструвати систему на концептуальному рівні.

Рис. 4.1. Діаграма варіантів використання ПЗ «Температурний логер-термостат»

4.4. Специфікації варіантів використання ПЗ «Температурний логер-термостат»

До діаграми варіантів використання, наведеної у попередньому розділі, були розроблені специфікації варіантів використання.

1. Редагування налаштувань.

Основна дійова особа: Оператор ПК.

Інші учасники прецеденту: Відсутні.

Зв’язок з іншими варіантами використання: скидання налаштувань, збереження налаштувань.

Короткий опис: редагування та введення налаштувань, таких як:

– період зчитування температурних даних;

– номер підключеного послідовного порту;

– температурні межі термостата;

– датчики, за якими слідкує термостат;

– споживання енергії обігрівачем;

– опис датчиків.

Потоки подій:

Базовий потік:

– Оператор змінює будь-яке із зазначених вище значень.

– У разі потреби оператор зберігає введені дані або скидає їх натисканням відповідних кнопок.

Альтернативний потік:

Відмова від введених даних та повернення головної форми без скидання або збереження даних.

Спеціальні умови: відсутні.

Додаткові вимоги:

Передумова: відсутня.

Постумова: відсутня.

2. Збереження налаштувань.

Основна дійова особа: оператор ПК.

Інші учасники прецеденту: відсутні.

Зв’язок з іншими варіантами використання: редагування налаштувань.

Короткий опис: збереження змінених значень налаштувань у спеціальний файл налаштувань.

Потоки подій:

Базовий потік:

– Оператор натискає кнопку збереження.

– Програма перевіряє правильність введених даних.

– Програма зберігає дані у файл.

Альтернативний потік:

– Оператор натискає кнопку збереження.

– Програма перевіряє правильність введених даних.

– Програма показує повідомлення про помилку.

Спеціальні умови: відсутні.

Додаткові вимоги:

Передумова: редагування налаштувань.

Постумова: відсутня.

3. Скидання налаштувань.

Основна дійова особа: оператор ПК.

Інші учасники прецеденту: відсутні.

Зв’язок з іншими варіантами використання: редагування налаштувань.

Короткий опис: скидання введених налаштувань до збережених.

Потоки подій:

Базовий потік:

– Оператор натискає кнопку скидання.

– Програма завантажує параметри налаштувань з файлу.

– Програма відновлює параметри налаштувань.

Альтернативний потік: відсутній.

Спеціальні умови: відсутні.

Додаткові вимоги:

Передумова: редагування налаштувань.

Постумова: відсутня.

4. Збір температурних даних.

Основна дійова особа: оператор ПК.

Інші учасники прецеденту: відсутні.

Зв’язок з іншими варіантами використання: відсутні.

Короткий опис: запуск таймера збору температурних даних та їх запис у базу даних або відключення таймера.

Потоки подій:

Базовий потік:

– Оператор натискає кнопку старту та починається відлік до зчитування даних.

– По закінченню таймера програма виконує зчитування даних з датчиків, записує їх у базу даних та відображає їх на екрані.

Альтернативний потік: припинення оператором збору даних проводиться натисканням кнопки зупинки вимірювання.

Спеціальні умови: у поле періоду повинне бути введене значення більше одної секунди.

Додаткові вимоги:

Передумова: повинний бути підключений мікроконтролер з датчиками до вибраного послідовного порту.

Постумова: відсутня.

5. Введення часового проміжку.

Основна дійова особа: оператор ПК.

Інші учасники прецеденту: відсутні.

Зв’язок з іншими варіантами використання: побудова графіка, управління термостатом.

Короткий опис: введення часового проміжку потрібно для запиту до бази даних, побудови графіка та звіту.

Потоки подій:

Базовий потік: оператор вводить часовий проміжок у відповідні поля у групі запиту.

Альтернативний потік: під час запуску програми в полях часового проміжку записується проміжок в одну добу.

Спеціальні умови: відсутні.

Додаткові вимоги:

Передумова: відсутні.

Постумова: відсутня.

6. Побудова графіка.

Основна дійова особа: оператор ПК.

Інші учасники прецеденту: відсутні.

Зв'язок з іншими варіантами використання: введення часового проміжку.

Короткий опис: відображення температурних даних та показ роботи термостата за певний період часу.

Потоки подій:

Базовий потік:

– Оператор вводить часовий проміжок.

– Оператор вводить розмір вікна згладжування.

– Оператор натискає кнопку побудови графіка.

– На графіку відображаються наявні температурні дані та дані термостата за введений проміжок часу.

– Відображення даних певних датчиків може бути відключене натисканням на чекбокс з *id*-датчика.

Альтернативний потік: у разі відсутності даних відобразиться пустий температурний графік.

Спеціальні умови: відсутні.

Додаткові вимоги:

Передумова: введення часового проміжку.

Постумова: відсутня.

7. Побудова звіту.

Основна дійова особа: оператор ПК.

Інші учасники прецеденту: відсутні.

Зв’язок з іншими варіантами використання: введення часового проміжку.

Короткий опис: звіт будується на основі введеного часового проміжку та містить у собі дії термостата та статистику датчиків.

Потоки подій:

Базовий потік:

– Оператор вводить часовий проміжок.

– Оператор натискає кнопку побудови звіту.

– У вікні перегляду звіту відображається введений часовий проміжок, статистичні дані датчиків, дії термостата та температурний графік.

– Оператор за бажанням натискає кнопку друку або збереження файлу.

– Відкриття діалогового вікна друку або збереження файлу.

– Друк звіту або збереження його у файл.

Альтернативний потік:

– Ініціація запиту на побудову графіка.

– Оператор вводить часовий проміжок.

– У вікні перегляду відображається введений часовий проміжок, статистичні дані датчиків, дії термостата та температурний графік.

– Оператор виходить з вікна перегляду звіту без друку або збереження у файл.

Спеціальні умови: відсутні.

Додаткові вимоги:

Введення часового проміжку: відсутня.

Постумова: відсутня.

8. Управління термостатом.

Основна дійова особа: оператор ПК.

Інші учасники прецеденту: відсутні.

Зв’язок з іншими варіантами використання: відсутній.

Короткий опис: активація або дезактивація термостата в залежності від потреби регулювання температури.

Потоки подій:

Базовий потік: оператор натискає кнопку активації.

Альтернативний потік: оператор натискає кнопку дезактивації.

Спеціальні умови: для роботи обігріву до мікроконтролера потрібно підключити силову управлінську частину пристрою.

Додаткові вимоги:

Передумова: відсутня.

Постумова: відсутня.

9. Описування датчиків.

Основна дійова особа: оператор ПК.

Інші учасники прецеденту: відсутні.

Зв’язок з іншими варіантами використання: відсутній.

Короткий опис: зміна опису температурних датчиків.

Потоки подій:

Базовий потік:

– Оператор натискає кнопку відкривання форми опису датчиків.

– У вікні відображається таблиця з датчиками та їх описами.

– Оператор змінює опис датчиків.

– Оператор натискає кнопку збереження, та програма зберігає зміни опису датчиків.

Альтернативний потік:

– Оператор натискає кнопку відкривання форми опису датчиків.

– У вікні відображається таблиця з датчиками та їх описами.

– Оператор змінює опис датчиків.

– Оператор відмовляється від зміни опису та натискає кнопку скидання або закриває форму.

Спеціальні умови: відсутні.

Додаткові вимоги:

Передумова: відсутня.

Постумова: відсутня.

4.5. Сценарії основних варіантів використання ПЗ «Температурний логер-термостат»

Діаграми діяльності варіантів використання деталізують та показують потоки й перетворення даних у варіанті використання. Діаграми діяльності до програмного забезпечення «Температурний логер-термостат» наведені на рис. 4.2–4.7.

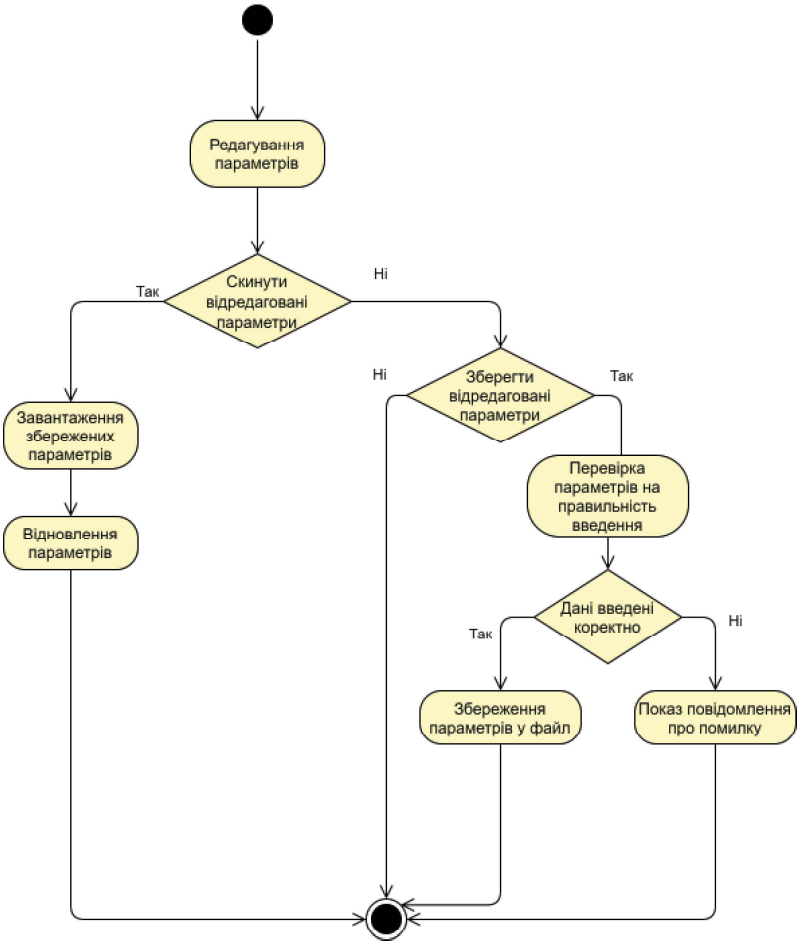


Рис. 4.2. Діаграма діяльності варіанта використання для редагування, скидання та збереження параметрів налаштувань

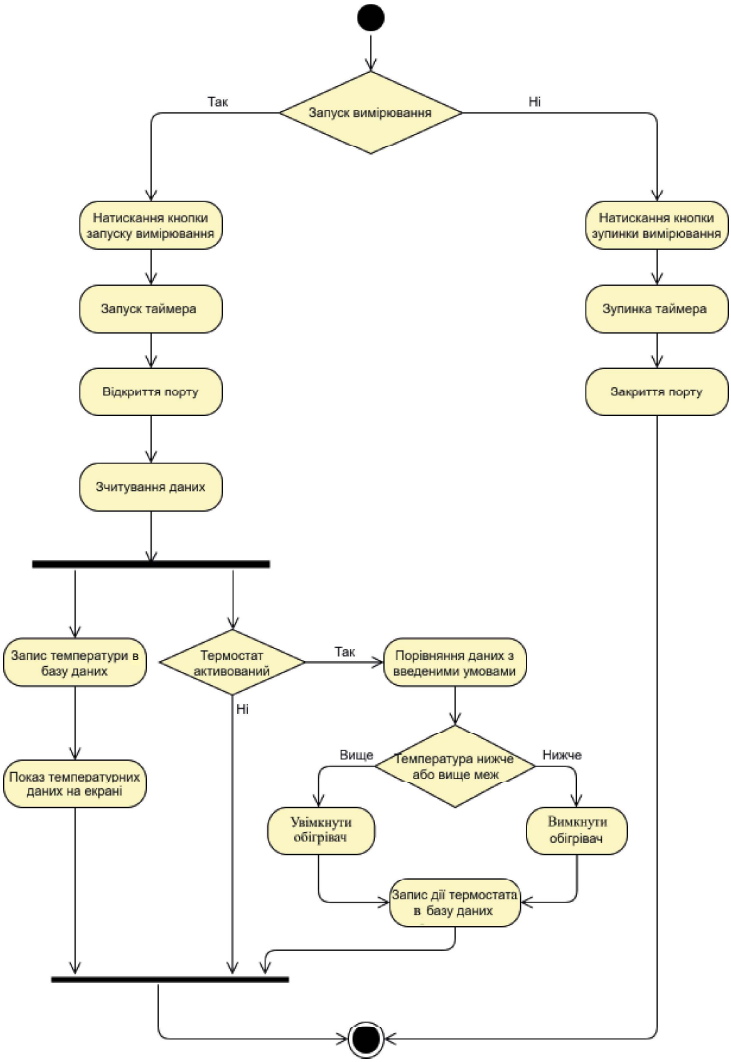


Рис. 4.3. Діаграма діяльності варіанта використання для збору температурних даних

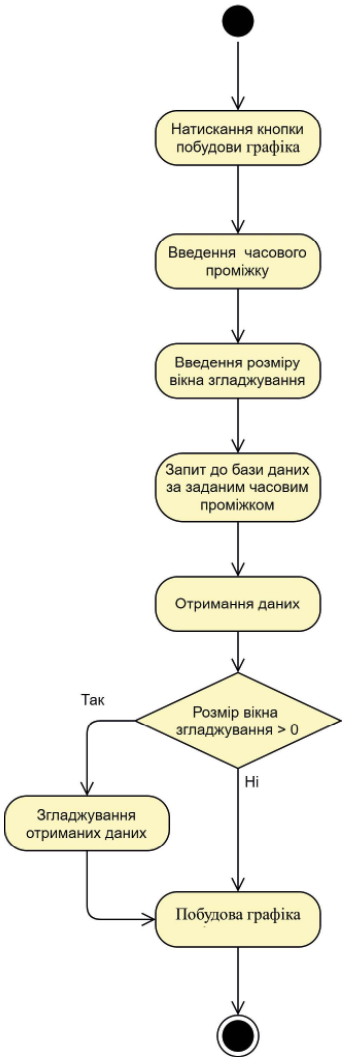


Рис. 4.4. Діаграма діяльності варіанта використання для побудови температурного графіка

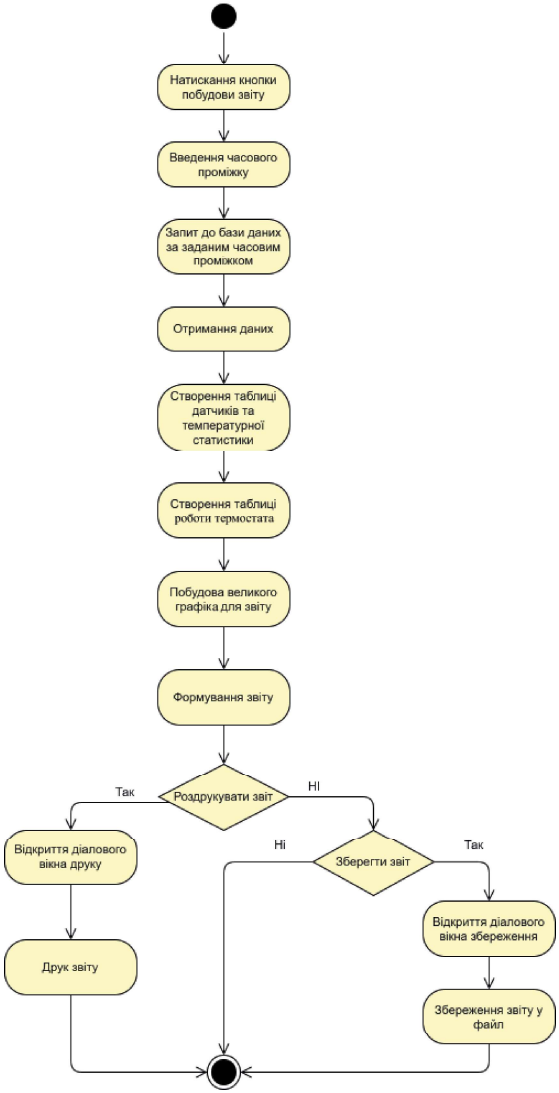


Рис. 4.5. Діаграма діяльності варіанта використання для побудови звіту

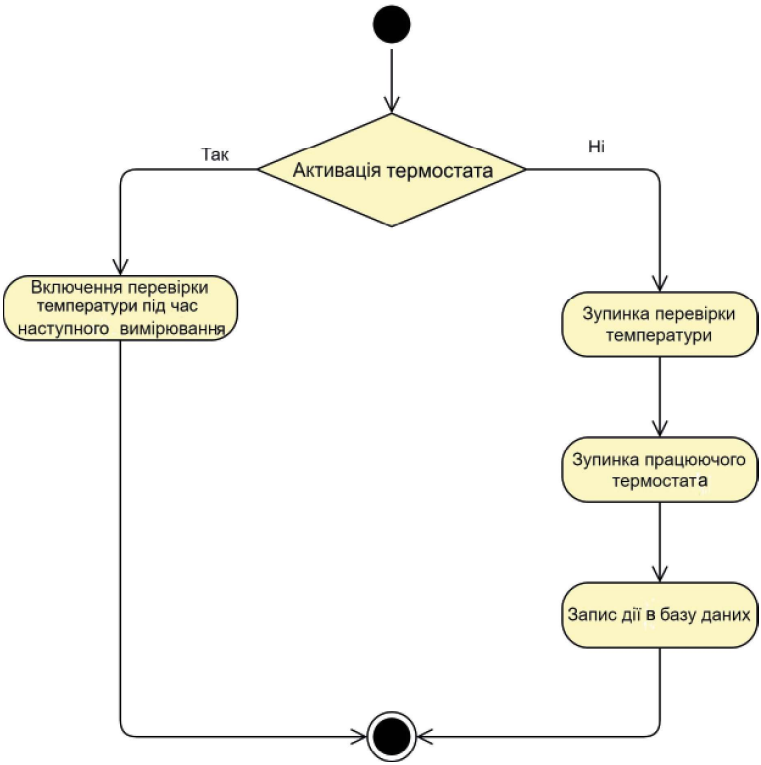


Рис. 4.6. Діаграма діяльності варіанта використання для управління термостатом

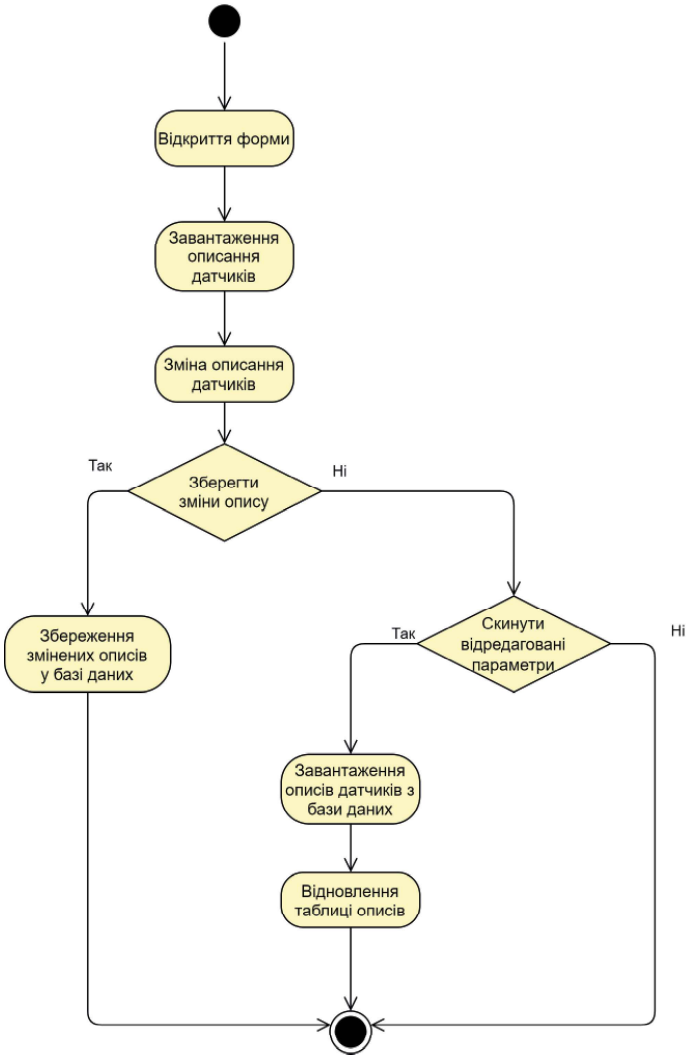


Рис. 4.7. Діаграма діяльності варіанта використання для описування датчиків

4.6. Інформаційна модель ПЗ «Температурний логер-термостат»

Концептуальна (змістовна) модель – це абстрактна модель, що визначає структуру модельованої системи, властивості її елементів і причинно-наслідкові зв'язки, властиві системі і суттєві для досягнення мети моделювання.

Нижче наведена концептуальна модель, яка представлена у вигляді діаграми сутність-зв'язок для розроблюваного програмного продукту «Температурний логер-термостат».

Рис. 4.8. Концептуальна модель даних у вигляді діаграми сутність-зв'язок для розроблюваного програмного продукту «Температурний логер-термостат»

4.7. Проєкт користувальницького інтерфейсу ПЗ «Температурний логер-термостат»

Перед початком програмування був розроблений макет призначеного для користувача інтерфейсу. Нижче представлений макет призначеного для користувача інтерфейсу.

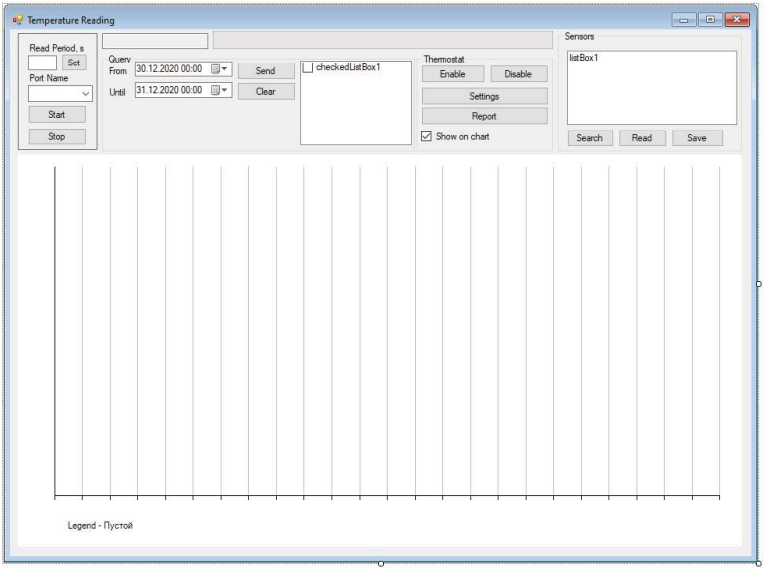


Рис. 4.9. Головна форма програми

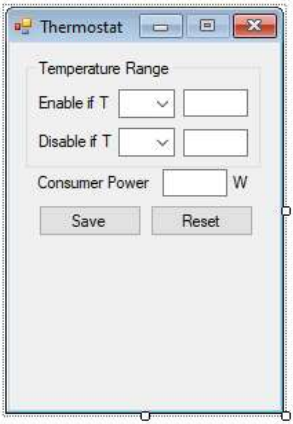


Рис. 4.10. Форма налаштування термостата

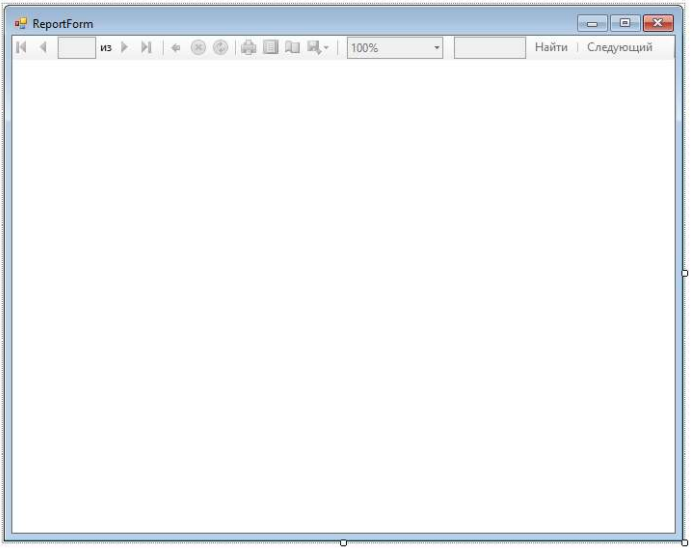


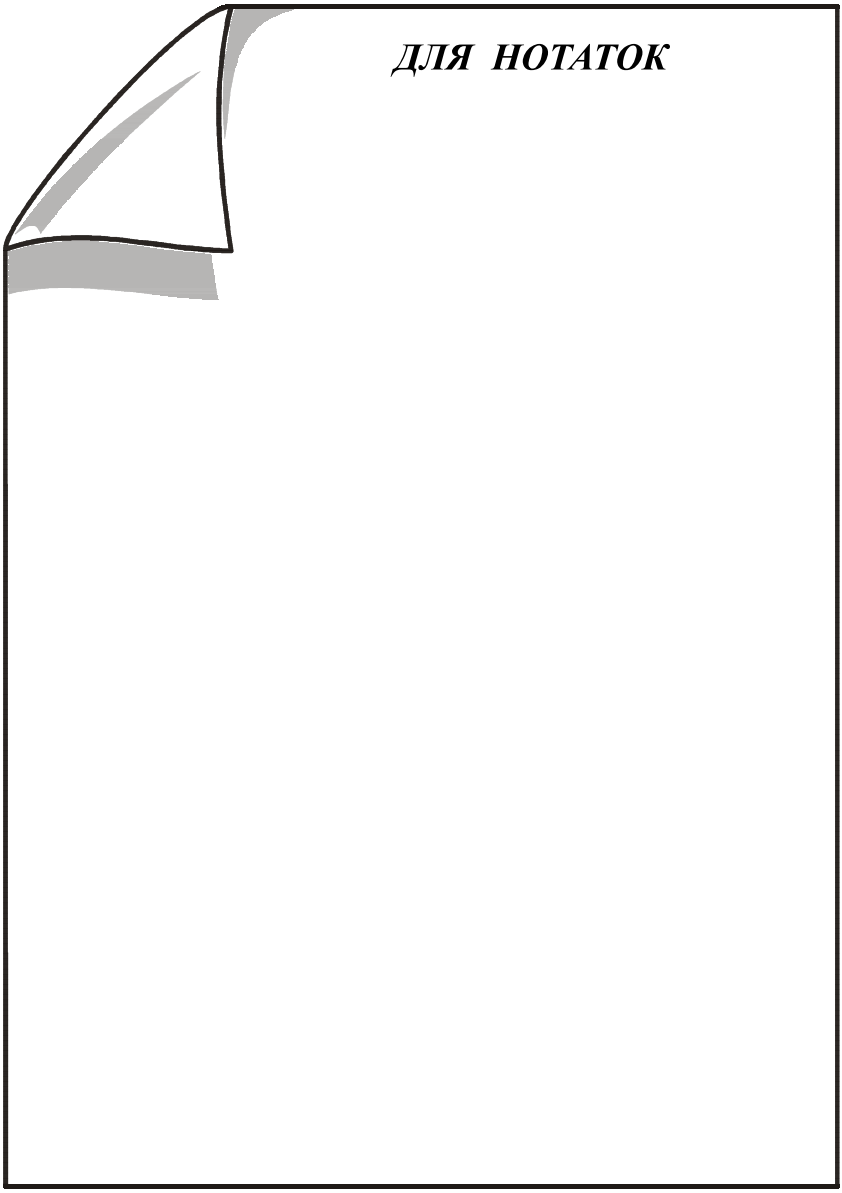
Рис. 4.11. Форма відображення звіту

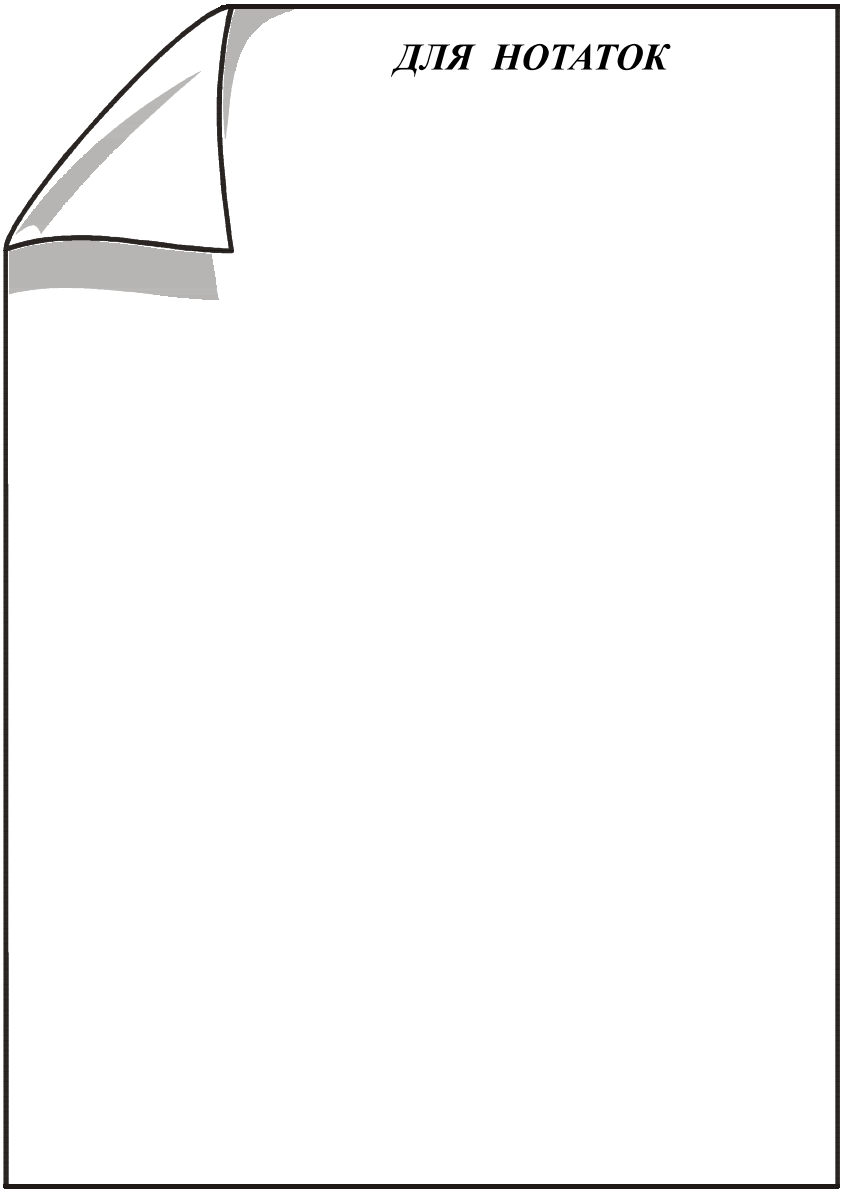
Висновок

Ескізний проєкт – проєктна конструкторська документація, яка містить принципові конструктивні рішення і дає загальне уявлення про будову та принцип дії виробу, а також дані, що визначають його відповідність призначенню.

На основі технічного завдання було виконано аналіз функціональних та експлуатаційних вимог до програмного продукту «Температурний логер-термостат». Визначено основні технічні рішення (вибір мови програмування, структура програмного продукту, склад функцій програмного продукту, режими функціонування), розроблена діаграма діяльності, концептуальна модель даних у вигляді діаграми сутність-зв’язок та проєкт користувацького інтерфейсу.

ЗМІСТ


ДЛЯ НОТАТОК


ДЛЯ НОТАТОК

Навчальне видання

ДУДЧЕНКО Олег Миколайович
КАРПОВА Світлана Олегівна

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
з курсу «ПРОЄКТНИЙ ПРАКТИКУМ»

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

Формат 60x84/16. Ум. друк. арк. 4,9. Тираж 100 прим. Вид. № 11. Зам. № 1206-23.
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.