Міністерство освіти і науки України

Національний технічний університет

«Дніпровська політехніка»

Факультет інформаційних технологій

(факультет)

Кафедра системного аналізу та управління

(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеню бакалавра

Студентки *Костюк Дар'ї*

академічної групи *124-19-2*

спеціальності *124 Системний аналіз*

на тему: *«Порівняльний аналіз сучасних стратегій розробки вимог та контролю якості програмного забезпечення»*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*к.ф.-м.н., доц. Коряшкіна Л.С.*			
розділів:				
Інформаційно-аналітичний розділ	*к.ф.-м.н., доц. Коряшкіна Л.С.*			
Спеціальний розділ	*к.ф.-м.н., доц. Коряшкіна Л.С.*			
Рецензент				
Нормоконтролер	*к.ф.-м.н., доц. Хом'як Т.В.*			

Дніпро

2023

ЗАТВЕРДЖЕНО:

завідувач кафедри

Системного аналізу та управління
(повна назва)

_____ *к.т.н., доц. Желдак Т.А.*
(підпис) (прізвище, ініціали)

«____» _____ 20 ____ року

ЗАВДАННЯ

на кваліфікаційну роботу ступеню *бакалавра*

студентці *Костюк Д.* **академічної групи** *124-19-2*

спеціальності: *124 Системний аналіз*

на тему *«Порівняльний аналіз сучасних стратегій розробки вимог та контролю якості програмного забезпечення»*

затверджену наказом ректора НТУ «Дніпровська політехніка»

від *16.05.2023 № 350-с*

Розділ	Зміст виконання	Терміни виконання
1. Інформаційно-аналітичний	*Проаналізувати проблеми, пов'язані з проєктами, обгрунтувати вибір стратегій розробки вимог та тестування, виконати огляд стратегій та їх складових*	20.03.23-14.04.23
2. Спеціальний	*Застосувати стратегії до розробки фінансового застосунку, описати деталі та розрахувати показники проєкту, зробити порівняльний аналіз стратегій*	14.04.23-02.06.23

Завдання видано _____ *доц. Коряшкіна Л.С.*
(підпис керівника) (прізвище, ініціали)

Дата видачі: 13.03.2023

Дата подання до екзаменаційної комісії: 13.06.23

Прийнято до виконання _____ Костюк Д.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 67 с., 22 рис., 5 табл., 3 додатки, 25 джерел.

Об'єктом дослідження в роботі є сфера розробки програмного забезпечення.

Предметом дослідження є стратегії розробки вимог та контролю якості програмного забезпечення.

Завдання дослідження: з'ясування та опис сучасних стратегій розробки вимог та контролю якості програмного забезпечення та їх складових, застосування даних стратегій до розробки фінансового застосунку, порівняльний аналіз застосованих стратегій.

Метою даної кваліфікаційної роботи є вибір і застосування ефективних стратегій розробки вимог та артефактів забезпечення та контролю якості програмного забезпечення для підвищення результативності проєктів, що розробляються.

Методи дослідження: системний підхід, методи порівняння та аналогій, відображення метрик, аналіз причинно-наслідкових зв'язків.

В *інформаційно-аналітичному розділі* розглянуто методології розробки програмних продуктів, виконано огляд сучасних стратегій інженерії вимог та забезпечення якості програмного забезпечення та їх складових.

У *спеціальному розділі* застосовано стратегії до розробки вимог та контролю якості фінансового застосунку, на основі чого проведено їх порівняльний аналіз.

Практична цінність роботи полягає в тому, що отримані результати та наведені стратегії можуть бути використані для покращення процесів розробки вимог та для забезпечення якості застосунків для управління особистими фінансами та подібних проєктів.

Ключові слова: ВИМОГИ, МЕТОДОЛОГІЇ РОЗРОБКИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СТРАТЕГІЯ, ТЕСТОВІ АРТЕФАКТИ, ФІНАНСОВИЙ ЗАСТОСУНОК, WATERFALL, SCRUM.

ABSTRACT

Explanatory note: 67 pages, 22 figures, 5 tables, 3 appendices, 25sources.

The object of the research is the field of software development.

The subject of the research is the strategy of requirements development and software quality control.

Research task: clarification and description of modern strategies for development of requirements and quality control of software and their components, application of these strategies to the development of a financial application, comparative analysis of applied strategies.

The purpose of this qualification work is to apply effective strategies for the development of requirements and artifacts of software quality assurance and quality control to improve the effectiveness of the projects being developed.

Research methods: systematic approach, methods of comparison and analogies, display metrics, analysis of causal relationships.

In the *information-analytical section* the methodologies of developing software products are considered, an overview of modern strategies of requirements engineering and quality assurance of software and their components is performed.

In a *special section*, strategies for the development of requirements and quality control of financial applications are applied, on the basis of which their comparative analysis is carried out.

The practical value of the work lies in the fact that the obtained results and the given strategies can be used to improve the requirements development processes and to ensure the quality of personal finance management applications and similar projects.

Keywords: REQUIREMENTS, DEVELOPMENT METHODOLOGY, SOFTWARE, STRATEGY, TEST ARTIFACTS, FINANCIAL APPLICATION, WATERFALL, SCRUM.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

БД – база даних

API – прикладний програмний інтерфейс

QA – контроль якості

SDLC – життєвий цикл розробки програмного забезпечення

JSON – запис об'єктів JavaScript

HTML – мова гіпертекстової розмітки

CSS – каскадні таблиці стилів

ВСТУП

Індустрія інформаційних технологій постійно розвивається, що призводить до того, що програмне забезпечення (ПЗ) стає необхідною складовою як для людей, так і для бізнесу. Зростаючий попит на ПЗ створює необхідність правильної постановки процесів розробки системи, щоб вони могли задовольняти бізнес-потреби та створювати якісний, підтримуваний та функціональний продукт. Щодня з’являються нові проєкти, розробляються програмні продукти та програми, які потребують ефективного керування. На сьогоднішній день для цього застосовуються методології управління, що є комбінаціями логічно пов'язаних процесів, практик та методів.

Аналіз останніх досліджень і публікацій підтверджує, що питання вибору методології розробки ІТ-проєкту є актуальним як для українських, так і зарубіжних вчених. Відомо, що третина всіх проєктів не завершуються взагалі. Кожен десятий проєкт не відповідає вимогам, значна частина продуктів не реалізують поставленої задачі повністю, а їх недоліки інколи є настільки критичними для користувачів, що роблять неякісним або навіть неможливим використання розробленої програми.

Це свідчить про важливість виокремлення з методології управління проєктами таких етапів, як розробка вимог та контроль якості програми. Вимоги відображають очікування та потреби зацікавлених осіб, від чого напряму залежить успіх випуску програмного продукту та його подальше існування. При цьому постановка задачі забезпечення якості продукту зводиться до задачі визначення того, що будь-який стан системи відповідає

вимогам власника продукту або користувачів. Це дозволяє зробити процес розробки ПЗ прозорим і керованим для всіх учасників проєкту.

Ці процеси є частиною моделей життєвого циклу розробки ПЗ, які є перевіреними та стандартизованими, тому у даній роботі розглянуті сучасні стратегії інженерії вимог та забезпечення якості програмного продукту на їх основі, а саме каскадної методології та гнучкої методології Scrum. Вибір саме цих стратегій обґрунтований тим, що каскадна модель є традиційною, поширеною, логічною, та є першим встановленим сучасним підходом до побудови системи. Scrum є однією з найпопулярніших моделей гнучкої розробки, що стали поширеними впродовж останніх років та широко застосовуються у різних бізнес-галузях для успішної розробки різноманітних проєктів.

Питання, яка зі стратегій результативніша, є відкритим: одна стратегія може бути кращим варіантом для однієї системи, а інша стратегія може навпаки загальмувати або взагалі призвести до краху проєкту. Так як задачею є порівняння стратегій саме для ПЗ, визначення ефективнішої з них, для цієї цілі у роботі використовується розроблений застосунок для управління особистими фінансами.

Незважаючи на те, що існуючі стратегії інженерії вимог та забезпечення якості систем були випробувані часом, сфера їх застосування визначена, і кожна зацікавлена особа, чи то замовник, чи команда, можуть обрати для себе ефективний набір методів, нікуди не зникають можливості для покращення та розробки нових стратегій, що також розглянуто у даній роботі.

Об’єкт дослідження в роботі є сфера розробки програмного забезпечення.

Предметом дослідження є стратегії розробки вимог та контролю якості програмного забезпечення.

Завдання дослідження: з’ясування та опис сучасних стратегій розробки вимог та контролю якості програмного забезпечення та їх складових, застосування даних стратегій до розробки фінансового застосунку, порівняльний аналіз застосованих стратегій.

Метою даної кваліфікаційної роботи є застосування ефективних стратегій розробки вимог та артефактів забезпечення та контролю якості програмного забезпечення для підвищення результативності проєктів, що розробляються.

ІНФОРМАЦІЙНО-АНАЛІТИЧНИЙ РОЗДІЛ

1.1. Процес та методи розробки програмного забезпечення

1.1.1. Життєвий цикл програмного забезпечення

Стратегії розробки вимог та процеси, що забезпечують контроль якості програмного забезпечення (ПЗ), не можуть існувати окремо від інших етапів роботи над проєктом. Процес його виконання передбачає розтягнутість у часі і має назву життєвого циклу розробки програмного забезпечення (Software Development Life Cycle, SDLC), що є послідовністю кроків розробки, тестування та внесення змін, необхідних для реалізації проєкту, від його первинної ідеї до виходу продукту на ринок і подальшої його підтримки [1]. Увесь цикл SDLC можна поділити на 7 етапів (рис. 1.1).

Рис. 1.1. Етапи життєвого циклу розробки ПЗ

Основні фази життєвого циклу розробки ПЗ [2]:

- планування – узгодження та з’ясування деталей, мети, проблем та бачення продукту;
- збір та аналіз вимог – збір деталей розробки проєкту, підсумовування та створення документації, на основі чого визначаються пріоритети завдань;
- проєктування та дизайн – проєктування архітектури та зв’язку публічної частини програми з сервером, створення інтерфейсу користувача;
- розробка – фактичне написання коду на основі документації, прототипів, дизайну та архітектури;
- тестування – порівняння результату після етапу розробки з визначеними вимогами та етапі збору та аналізу вимог, знаходження дефектів та помилок;
- впровадження – надання готового ПЗ кінцевим користувачам;
- підтримка та розвиток – безперервна перевірка працездатності та продуктивності системи, впровадження нових функцій та виправлення помилок.

Використання поняття життєвого циклу дозволяє обрати найбільш ефективні підходи для задач певного етапу розробки ПЗ.

1.1.2. Методології розробки програмного забезпечення

Результатом проєкту є ПЗ, що характеризується унікальністю. Це передбачає, що процес його створення також має бути в тій чи іншій мірі унікальним. Для нових проєктів не може створюватись кожного разу життєвий цикл «з нуля», замість цього обирається перевірена та узагальнена практика, що адаптується під реалізацію певного продукту. Залежно від особливостей процесів розробки і супроводу програм існують різні моделі життєвого циклу. Знання про закономірності розвитку програмного продукту, які відображаються

в обраній моделі життєвого циклу, дозволяють отримати надійні орієнтири для планування стратегії розробки вимог та забезпечення якості системи, економно витрачати ресурси і підвищувати якість управління іншими процесами.

Методологія розробки програмного забезпечення (Software Development Methodology) – це систематичний підхід, який описує процеси, правила, методи та інструменти, які використовуються в межах проєкту, та дозволяє забезпечити найкращу ефективність усіх етапів розробки.

Протягом багатьох років були представлені різні методології, щоб отримати вигоду від наявних технологій і ресурсів. Сьогодні велика кількість IT-компаній погоджується з тим, що використання методології розробки ПЗ має вирішальне значення для їх команди. Проте питання про те, який метод найкращий, залишається під питанням. Найпопулярнішими є (рис. 1.2):

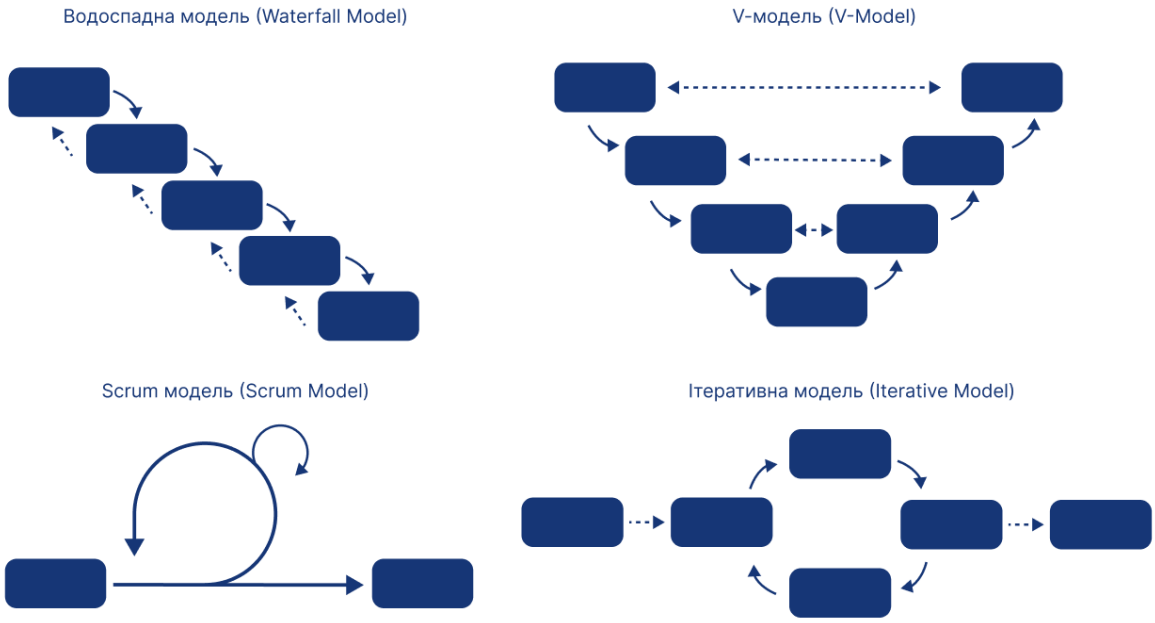


Рис. 1.2. Популярні моделі розробки ПЗ

– водоспадна (каскадна) модель (Waterfall Model) – лінійний процес розробки, кожен етап наступає після завершення попереднього;
– ітеративна модель розробки (Iterative Model) – передбачає розбиття розробки на ряд окремих міні-циклів, кожен з яких складається із все тих же базових стадій життєвого циклу;

- Scrum модель (Scrum Model) – використовується для розробки ПЗ, яке вимагає регулярних змін та необхідної гнучкості у вимогах користувачів; основна мета Scrum – задовольнити потреби клієнта через середовище прозорості в спілкуванні, колективної відповідальності та постійного прогресу;
- V-модель (V-Model) – на кожному етапі відбувається контроль поточного процесу, для того, щоб переконатися в можливості переходу на наступний рівень [2].

Доцільним є звернути увагу на водоспадну методологію, оскільки вона існує понад п’ятдесят років та є найстарішою. Цей факт не робить її несучасною, оскільки впродовж свого існування ця модель стала основою для формування більшості інших та зазнала змін, які забезпечили її використання і сьогодні.

Дослідження також варта Scrum модель, що є однією з моделей гнучкої розробки (Agile Software Development). В останні роки спостерігається значне зростання впровадження Agile. При цьому Scrum залишається найпопулярнішим гнучким підходом, опитування світової спільноти розробників показало, що 66% респондентів використовують дану методологію постійно, а 15% застосовують похідні Scrum [3].

1.1.3. Значення вимог та тестування у загальному циклі розробки

Від обраної методології, розміру, складності та типу проєкту залежить тривалість кожного з його етапів, яку точно можна визначити тільки під час детального планування. На рисунку 1.3 зображена орієнтовна тривалість кожної з основних фаз життєвого циклу ПЗ.

Рис. 1.3. Тривалість етапів розробки ПЗ

Незважаючи на загальноприйняту думку, центральне місце в процесі реалізації проєкту не займає безпосереднє написання коду. Що зазвичай залишається поза увагою, але в той же час є чи не найважливішою частиною життєвого циклу ПЗ, так це стадії попереднього збору та аналізу вимог, проєктування системи, а також забезпечення та контроль якості впродовж усього процесу розробки та після виходу продукту на ринок.

На основі статистичних даних, зібраних групою The Standish Group International в період з 1994 р. по 2000 р. у різних країнах [4], можна визначити основні причини успіхів і провалів проєкту:

- причини провалів: недостатньо вихідної інформації від клієнта, неповні вимоги і специфікації, зміни вимог і специфікацій;
- причини успіхів: підключення до розробки користувача, тестування на усіх етапах, чітка постановка вимог.

Аналізуючи дані причини, можна зробити висновок про важливість збору вимог при розробці, тому їхня інженерія потребує додаткового розгляду.

Забезпечення та контроль якості ПЗ є не менш важливою складовою на кожному етапі розробки, допомагає виявити та виправити помилки швидко та ефективно, оскільки для замовника робочий та якісний продукт стоїть на першому місці.

1.2. Технологія розробки вимог до програмного забезпечення

1.2.1. Рівні та етапи розробки вимог

Вимоги до ПЗ (Software Requirements) – це формально виражені умови або специфікації, які описують функціональні та нефункціональні характеристики системи. Вони визначають, для чого саме призначене ПЗ, функції, які будуть реалізовані, деталі взаємодії користувачів та ПЗ, які дії дозволені та обмежені, а також його технічні характеристики. Вимоги можна розділити на 3 рівні: бізнес-вимоги, вимоги користувачів та функціональні вимоги [5, 6].

При цьому їх можна класифікувати за характером на функціональні та нефункціональні, що схематично представлено на рисунку 1.4. Синій прямокутник – тип інформації, білий прямокутник – спосіб зберігання інформації.

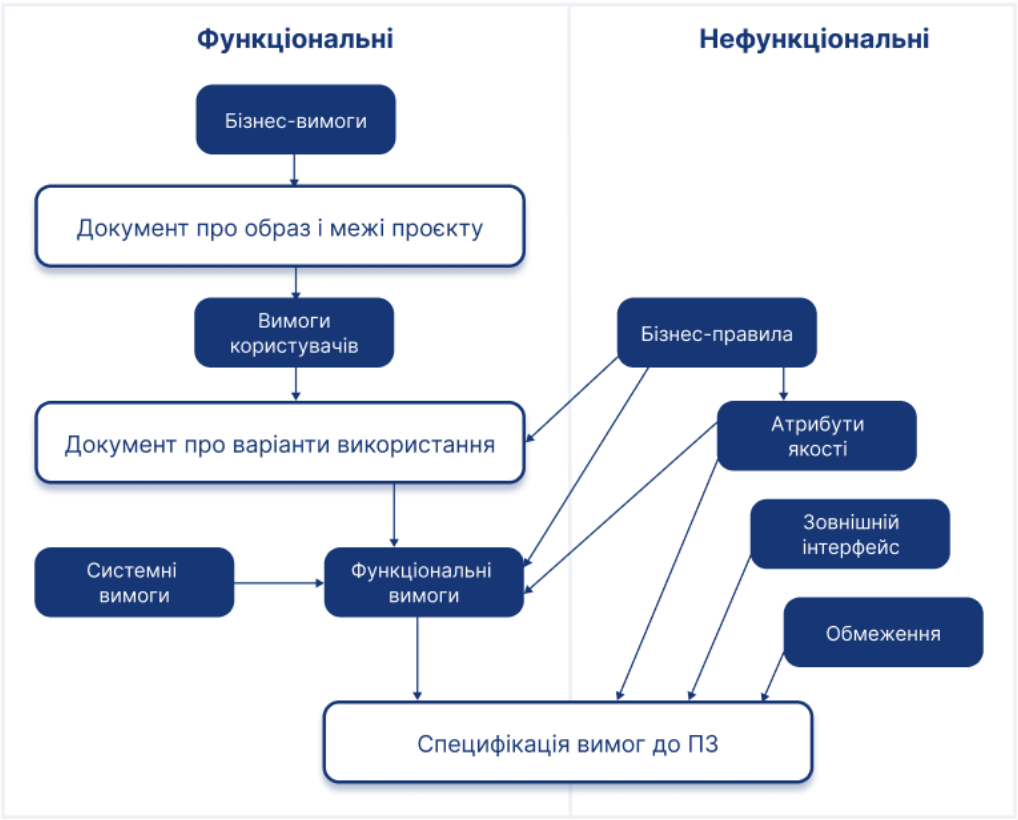


Рис. 1.4. Зв’язок типів інформації та способів їхнього зберігання для вимог

Бізнес-вимоги (Business Requirements) стосуються таких характеристик ПЗ, які необхідні для досягнення бізнес-цілей організації. Вони визначають, яким чином ПЗ може допомогти вирішити конкретні бізнес-проблеми, такі як збільшення продуктивності, зменшення витрат, поліпшення якості продукту або підвищення задоволеності клієнтів. Їх описують в формі документа про образ і межі проєкту, який ще називають уставом проєкту (Project Charter).

Вимоги користувачів (User Requirements) встановлюються з урахуванням потреб та очікувань користувачів щодо функціональності та інтерфейсу ПЗ. Визначають, як користувачі будуть взаємодіяти з ПЗ, які функції і можливості повинні бути доступні.

Функціональні вимоги (Functional Requirements) визначають функції або операції, які повинні бути виконані ПЗ, щоб користувачі могли задовольнити свої потреби в межах бізнес-вимог. Ці вимоги описують у документі про варіанти використання, завдяки чому розробник розуміє, що необхідно реалізувати.

Системні вимоги (System Requirements) – характеристики, які повинна мати комп'ютерна система для правильної роботи певного ПЗ.

Бізнес-правила (Business Rules) встановлюються в організації для забезпечення ефективної роботи бізнес-процесів, включають корпоративну політику, постанови управління, фактично не є вимогами. Вони можуть визначати обмеження для системи та є джерелами атрибутів якості.

Атрибути якості (Quality Attributes) – характеристики системи, що визначають її якість та відповідність вимогам. Можуть включати такі параметри, як надійність, ефективність, сумісність, безпека.

Зовнішній інтерфейс (External Interface) взаємодіє з користувачем та іншими системами. Це може бути графічний інтерфейс користувача, командний рядок або інші форми взаємодії.

Обмеження (Constraints) можуть накладатись на зовнішній вигляд, структуру та архітектуру продукту.

Зазначені вимоги описуються у специфікації вимог до ПЗ. Також слід очікувати циклів та додаткових зв’язків між вимогами.

Робота над вимогами є складовою області розробки технічних вимог, яка включає у себе такі етапи, як розробка та управління [5]. У свою чергу розробка складається з виявлення, аналізу, документування та перевірки вимог (рис. 1.5).

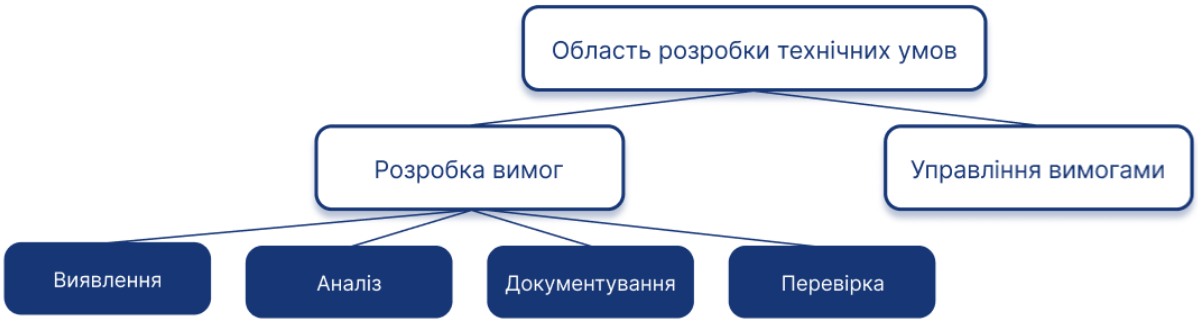


Рис. 1.5. Складові інженерії вимог

Можна визначити основні процеси, які відбуваються під час етапу розробки вимог:

- визначення цільової аудиторії продукту;
- з’ясування потреб, цілей та завдань кожного класу користувачів;
- встановлення бізнес-цілей та бізнес-правил;
- аналіз інформації, відокремлення функціональних вимог від інших;
- з’ясування атрибутів якості;
- документування інформації, побудова моделей та встановлення пріоритетів;
- опрацювання та деталізація специфікації вимог.

Не менш важливий етап управління вимогами включає у себе такі процеси:

- визначення основної версії вимог;
- розгляд пропонованих змін;
- включення схвалених змін в проєкт;
- узгодження плану проєкту з вимогами;
- спостереження за статусом вимог впродовж усього проєкту.

1.2.2. Методи вилучення та обробка вимог

Розробкою вимог є процес дослідження, розуміння та документування вимог, які відповідають характеристикам правильних вимог, які потребує проєкт від початку до кінця. Збирання вимог визначає якість усіх наступних етапів їх розробки та проєкту загалом [7].

Основним джерелами для вилучення вимог різних рівнів є:

- структуроване інтерв'ю зацікавлених осіб;
- пасивне та активне спостереження;
- перегляд та аналіз документів, що надає зацікавлена сторона;
- фокус-групи або групові інтерв'ю з представниками класів користувачі;
- опитування та анкетування;
- прототипування.

Після того як вимоги виявлені, вони узгоджуються між собою та перевіряються на інші визначені характеристики.

Узгодження і перевірка вимог не можуть бути відокремлені від підготовки специфікації, тобто процесу документування. Погодження розбіжностей між вимогами засноване на чорновому варіанті документа. Вимоги, перераховані в чорновому варіанті до документу, обговорюються і при необхідності модифікуються, тобто побічні вимоги видаляються, нововиявлені вимоги додаються.

Опрацьовані вимоги описуються у специфікації, на основі чого складаються завдання для розробників часто у вигляді історій користувачів та критерії приймання (User Stories and Acceptance Criteria). Історії користувачів відображають функціональні вимоги до ПЗ у форматі коротких історій, які описують поведінку користувача та його потреби. Критерії приймання описують умови, за яких вважається, що історію користувача виконано і функціональність працює належним чином.

Опрацьована специфікації є джерелом для створення дизайну, прототипів, технічного завдання, а також таких документів, як план тестування, тест-кейси та інші артефакти тестування.

1.2.3. Показники якості вимог

Вимоги до ПЗ мають дуже високу важливість з погляду забезпечення його якості. Але оскільки вимоги не завжди можуть бути задокументовані, то основним критерієм їх якості є сам факт наявності цих вимог. Окремо взяті вимоги або ж їх набір мають відповідати критеріям якості. Правильно сформульовані та описані вимоги мають деякі спільні характеристики, які допомагають забезпечити якість та ефективність розробки проєкту [6]. Деякі з них:

- коректність: вимоги точні та правильні, немає місця для двозначності;
- зрозумілість: вимоги легкі для розуміння та інтерпретації;
- повнота: вимоги описують усі необхідні функції та властивості ПЗ;
- атомарність; вимога не може бути розбита на ряд більш детальних вимог без втрати завершеності;
- керованість: вимоги змінюються відповідно до потреб зацікавлених осіб;
- перевірюваність: вимоги перевірені на відповідність можливості їх виконання.

Від того, чи були дотримані характеристики якісних вимог, напряму залежить виникнення наступних проблем:

- неповнота – описані не всі функції та властивості ПЗ, які необхідні для задоволення потреб користувачів;
- неоднозначність – вимоги сформульовані неоднозначно, що призводить до різних інтерпретацій та неправильного розуміння функціональності ПЗ;
- суперечливість – вимоги суперечать одна одній та не узгоджені;

- неактуальність – вимоги написані для певної версії ПЗ або платформи, але з часом умови змінюються, що призводить до невиконання вимог;
- низька якість документації – вимоги погано документовані або несистематизовані, внаслідок чого виникають проблеми з їхньою інтерпретацією.

Встановлюючи стандарти та характеристики якісних вимог, варто оцінювати прогрес у досягненні цих цілей. Використовувані метрики відрізняються залежно від деталей проєкту, але загальні показники для встановлення якості вимог включають:

- ступінь коректності/зрозумілості/повноти вимог у специфікації;
- кількість детально описаних вимог до компонентів системи;
- легкість визначення критерії прийняття з документації;
- кількість запитів на зміни вимог, що було подано з моменту написання специфікацій.

1.3. Забезпечення та контроль якості програмного забезпечення

1.3.1. Етапи забезпечення якості

Контроль якості ПЗ (Quality Control, QC) – це встановлення відповідності продукції та процесів вимогам нормативно-технічної документації. Поряд з цим існує поняття забезпечення якості (Quality Assurance, QA) – процеси, які застосовуються для забезпечення високої якості ПЗ в процесі його розробки та випуску. Включає в себе створення тестових документів та артефактів, контроль процесів розробки, перевірку вимог та визначення якості продукту. Оскільки ці два поняття нерозривно пов’язані, у даній роботі розглядаються не тільки дії направлені на контроль системи, а також і на забезпечення її якості.

Використання певної стратегії забезпечення якості та відповідних методів тестування дозволяє виявляти та виправляти вади ПЗ до його виходу на ринок, що зменшує ризик появи проблем в процесі експлуатації, час розробки, забезпечує задоволення користувачів та позитивний імідж компанії.

Етапи забезпечення якості ПЗ (рис. 1.6) відрізняються в залежності від методології розробки, обраної стратегії тестування, інструментів та процесів, які використовуються в компанії [8].

Рис. 1.6. Етапи забезпечення якості ПЗ

- планування включає в себе створення тест плану, тестової стратегії, визначення процесів, необхідні для забезпечення якості продукту та критеріїв успішності;
- тестування вимог – перевірка вимог на відповідність характеристикам якісних вимог та їх аналіз;
- тестування прототипів – процес перевірки наскільки коректно та зручно розроблено дизайн або прототипи;
- модульне тестування зосереджене на окремих компонентах, функціях або модулях системи; кожен модуль тестується окремо, з метою перевірки його функціональності та відповідності вимогам;
- системне тестування – процес тестування повного ПЗ, який виявляє помилки у системі в цілому, а не в окремих компонентах;

- аналізом результатів є оцінка тестування, вимірювання та аналіз якості ПЗ, кількості вад, їхній час виправлення; це дозволяє оцінити ефективність стратегії контролю якості ПЗ;
- післярелізний контроль здійснюється після випуску ПЗ та виявляє помилки у ньому.

1.3.2. Типи тестування та зв’язок між ними

Однією зі складових стратегії тестування є рівні та типи забезпечення якості ПЗ. Усі типи тестування тісно пов’язані між собою, тому не існує чіткою та універсальної їхньої класифікації. Різні джерела та методики використовують відмінні терміни, що ускладнює порівняння та систематизування. Поряд з цим, з розвитком технологій та вимог ринку, постійно з'являються нові методики, що можуть не вписуватись в існуючі класифікації [9].

Загалом усі типи тестування можна розділити на дві групи: функціональні та нефункціональні. Функціональне тестування спрямоване на визначення того, чи працює ПЗ згідно зі специфікацією. Перевіряється коректність введення та обробки даних, робота основних функцій та операцій, оборка помилок та некоректних даних. Нефункціональне тестування – перевірка якості виконання ПЗ, яка не пов’язана з його функціональністю. В першу чергу це тестування атрибутів якості. Ці типи тестування ПЗ мають велику кількість різноманітних розбиттів за особливостями, які тією чи іншою мірою переплітаються між собою.

У залежності від ступеню автоматизації, типи тестування можна умовно розділити на ручне, автоматизоване та напівавтоматизоване. За знанням системи на тестування чорної, білої та сірої скриньки. За ознакою позитивності сценаріїв поділяються на позитивне та негативне тестування. За хронологією розрізняють регресійне, приймальне та димове тестування, а також ретестинг.

Регресійне тестування проводиться з метою перевірки того, чи не впливають нові зміни на вже наявний функціонал. Також спрямоване на виявлення дефектів у вже протестованих модулях. У процесі використовуються тестові сценарії, які покривають весь функціонал ПЗ.

Приймальне тестування проводиться для підтвердження відповідності продукту вимогам та очікуванням користувачів або замовника.

Димове тестування перевіряє базовий функціонал ПЗ та виявляє критичні помилки або проблеми, які можуть заблокувати подальше тестування або розгортання ПЗ.

Окрім типів, існують рівні тестування, що включають сукупність тестових процедур [9, 12]:

- компонентне тестування (Component or Unit Testing) – процес перевірки окремих компонентів ПЗ, таких як функції, класи, модулі або підсистеми;
- інтеграційне тестування (Integration Testing) дозволяє перевірити, що окремі компоненти працюють правильно разом і взаємодіють один з одним так, як очікувалось;
- системне тестування (System or End-to-end Testing) полягає в перевірці функціональності та продуктивності системи в цілому, в тому числі її компонентів та взаємодії з зовнішніми системами.

Важливою процедурою забезпечення якості ПЗ є тестування прикладного програмного інтерфейсу (Application Programming Interface, API), оскільки він є основою для взаємодії програми з різними додатками, сервісами та пристроями. Тестування API зосереджено на аналізі бізнес-логіки і безпеці системи та полягає в відправці запитів на сервер і перевірці даних. Тести включають перевірки коректності кодів стану HTTP, заголовків відповідей, правильності формату та типів даних, швидкості та базової працездатності системи.

Не менш важливим є тестування інтерфейсу користувача. Це включає перевірку того, що всі візуальні елементи ПЗ коректно відображаються, що користувач може здійснювати необхідні дії та система правильно на це реагує.

Планування того, які типи тестування будуть застосовуватись, потребує розуміння потреб і вимог проєкту. Залежно від конкретної ситуації, певні типи можуть бути важливішими за інші. Але найчастіше для тестування ПЗ вони застосовуються у комплексі та доповнюються один одного.

1.3.3. Тестові документи та артефакти

У відповідність з методологією розробки ПЗ обраною стратегією, під час проведення тестування створюється і використовується певна кількість тестових документів, артефактів і моделей [11].

Тестова стратегія – високорівневий документ, що описує набір основних принципів, методів, підходів та інструментів, які є складовими обраної стратегії тестування та визначають розробку тестів та регулюють те, як буде здійснюватися контроль якості ПЗ. Це план дій процесу тестування на довгостроковій основі, що використовується на організаційному рівні.

Тест план – документ, що містить опис стратегії тестування та визначає, що, як і коли тестувати в процесі розробки ПЗ. Включає в себе опис тестових завдань, ресурсів, критеріїв прийняття та метрик для вимірювання ефективності тестування.

Тест-кейс – детально описаний сценарій або набір кроків, які потрібно виконати для перевірки конкретного функціоналу ПЗ. Тест-кейс містить інформацію про передумови, дії користувача та очікувані результати, виконується відповідно до специфікацій і вимог до ПЗ. Вони допомагають зменшити час тестування та зробити процес більш структурованим та систематичним.

Тестовий сценарій – послідовність дій над ПЗ, пов'язаних єдиним обмеженим бізнес-процесом використання, і відповідних перевірок коректності поведінки системи під час цих процесів.

Матриця простежуваності вимог – таблиця, яка дозволяє відстежувати зв'язки між вимогами, тест-кейсами та результатами тестування. Допомагає

забезпечити повноту тестування та визначити, які вимоги покриті тестами, а які потребують додаткового тестування.

Чеклист – це перелік тестів, які слід виконати за певною процедурою для тестування системи. Це деталізований список блоків, секцій, сторінок або інших елементів, які слід протестувати.

Одним з найважливіших документів контролю якості ПЗ є баг репорт, що описує проблему або неправильну поведінку системи. Він містить детальну інформацію про виявлений дефект, а саме опис проблеми, кроки для відтворення дефекту, оточення, інші деталі, які допомагають розробникам розібратися в проблемі та виправити її.

Документом, який містить опис результатів тестування, інформацію про виконані тести, виявлені помилки та пропозиції щодо подальшої роботи з продуктом є підсумковий звіт. Також, в звіті можуть бути наведені відомості про виконання розкладу, витрачені ресурси та зміни, які були внесені під час тестування.

1.3.4. Тестові метрики

Для аналізу застосованого життєвого циклу ПЗ, стратегії розробки вимог та тестування, необхідним є використання метрик, що визначають їх ефективність. Метрика – це кількісний показник, що дозволяє оцінити якості певного процесу або моделі. Основною метою їх використання на проєкті є розуміння ефективності застосованої стратегії тестування, зменшення кількості дефектів та пришвидшення строку релізу ПЗ [10-12].

Базові метрики являють собою комбінацію абсолютних чисел, які потім можуть бути використані для розрахунку похідних показників. Деякі з базових метрик:

- кількість знайдених дефектів;
- кількість відхилених дефектів;
- кількість прийнятих дефектів;

- кількість виправлених дефектів;
- кількість критичних дефектів;
- час виправлення дефекту;
- походження дефекту;
- кількість пройдених перевірок у чеклисті;
- кількість невдалих перевірок у чеклисті.
- кількість заблокованих перевірок у чеклисті;
- кількість пропущених перевірок у чеклисті.

Загальна ефективність процесу тестування може вимірюватись за наступною формулою:

$$Ефективність\ тестування = \frac{кількість\ вирішених\ дефектів}{загальна\ кількість\ дефектів} \cdot 100 \qquad (1.1)$$

Чим вищим є цей показник, тим якіснішим і ефективнішим був процес тестування. Якщо кількість вирішених дефектів є суттєво нижчою, ніж їх загальна кількість, це може свідчити про їх складність або брак ресурсів.

Також важливими похідними показниками для з'ясування ефективності застосованої стратегії є:

$$Коефіцієнт\ зміни\ вимог = \frac{кількість\ змін\ у\ вимогах}{загальна\ кількість\ вимог} \qquad (1.2)$$

Коефіцієнт зміни вимог призначений для відображення того, скільки реалізованих вимог потрібно переробляти при розробці нових функцій.

$$Відсоток\ перевідкритих\ дефектів = \frac{кількість\ повторно\ відкритих\ дефектів}{кількість\ знайдених\ дефектів} \cdot 100 \qquad (1.3)$$

Відсоток перевідкритих дефектів надає можливість оцінити професійність команди та якість тестування. Чим ближче значення до нуля, тим менше повторюються вже знайдені помилки.

$$\textit{Коефіцієнт «витоку» дефектів} = \frac{\textit{кількість невиправлених дефектів}}{\textit{кількість знайдених дефектів}} \cdot 100 \qquad (1.4)$$

Коефіцієнт витоку дефектів показує, скільки дефектів було пропущено та дійшло до замовника.

$$\textit{Коефіцієнт відхилення дефектів} = \frac{\textit{кількість неприйнятих дефектів}}{\textit{кількість знайдених дефектів}} \cdot 100 \qquad (1.5)$$

Призначення коефіцієнту відхилених дефектів показати скільки з них були заведені, але в подальшому видалені. Висока їх частка може свідчити про те, що вимоги змінюються надто часто.

$$\textit{Покриття вимог тестами} = \frac{\textit{кількість тестів/перевірок}}{\textit{кількість вимог}} \cdot 100 \qquad (1.6)$$

Покриття вимог тестуваннями вказує на те, наскільки повно та ефективно чеклист, тест-кейси або тестові сценарії охоплюють функціональні, системні та інші вимоги до ПЗ. Це показник, який відображає пропорцію вимог, які були протестовані в процесі розробки.

1.4. Сучасні стратегії розробки вимог та контролю якості

1.4.1. Поняття стратегії розробки вимог і тестування та її складові

Перед визначенням поняття стратегії, важливо нагадати, що вона не може існувати окремо від життєвого циклу розробки ПЗ. Оскільки етапи розробки

вимог і тестування є його складовими, від методології розробки програми напряму залежить сформована або обрана стратегія. Фактично, стратегія це довгостроковий план дій, спрямований на досягнення поставленої мети. Але у даному випадку стратегією є умовний план, що складається з набору принципів, методів, документів та артефактів розробки вимог та забезпечення якості ПЗ, базований на методології життєвого циклу ПЗ. Основною метою дотримання певної стратегії є реалізація проєкту високої якості, що відповідає вимогам зацікавлених осіб (Stakeholders), та зниження вад і помилок у його роботі.

Стратегія включає наступні складові:

- рівні вимог;
- методи розробки вимог;
- рівні та типи застосованих тестів;
- тестові документи та артефакти;
- порядок роботи з трекером завдань;
- критерії початку та завершення тестування;
- оцінку якості проєкту.

Кожен проєкт має свої унікальні потреби та обставини, тому вибір стратегії повинен бути обгрунтованим і базуватися на дослідженнях та зважати на потреби бізнесу. При виборі або розробці стратегії варто враховувати специфіку продукту та потреби зацікавлених осіб та користувачі. Для забезпечення ефективної стратегії необхідно використовувати ті методи та інструменти, які забезпечують найкращий результат для бізнесу. Немає універсальної стратегії, яка підійде для всіх продуктів і усіх організацій, але є традиційні та поширені, які найчастіше використовуються у сфері розробки програмних продуктів.

1.4.3. Стратегія на основі каскадної методології

Каскадна методологія бере свій початок у промисловості та будівництві, до яких можна приписати її суворий процес. Водоспадний підхід до системного

аналізу та проєктування був першим встановленим сучасним підходом до побудови системи. Вперше він був офіційно представлений як метод розробки ПЗ в статті, написаній Вінстоном Ройсом у 1970 році.

Дана стратегія розробки окреслює чітку послідовність фаз життєвого циклу ПЗ. Від якості розроблених вимог залежать усі наступні етапи, а від покриття системи тестами залежить кінцевий результат (рис. 1.7). Слід зазначити, що сучасна водоспадна модель все ж передбачає повернення до попередніх фах, якщо виникає необхідність внесення невеликих змін.

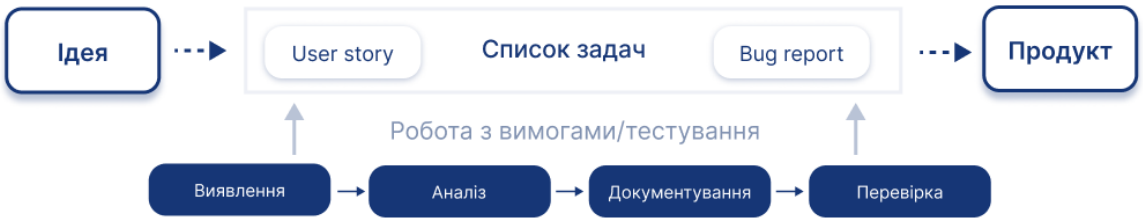


Рис. 1.7. Стратегія на основі каскадної методології

Згідно даної методології можна виділити наступні характерні особливості стратегії розробки вимог та контролю якості:

- усі вимоги збираються одразу, оцінюються та описуються у найдрібніших деталях у специфікації, яка перевіряється на правильність на повноту та затверджується замовником та є основою для подальшої розробки;
- зустрічі з замовником або представниками класів користувачів проводяться на початку, до подальших етапів вони не залучені;
- визначаються чіткі та конкретні цілі та задачі проєкту, строки та вартість робіт розраховуються на самому початку та найчастіше не можуть змінюватись у процесі;
- тестування проводиться чітко з урахуванням вимог клієнта;
- зміни вимог повинні бути мінімізовані після того, як фаза аналізу вимог завершена;
- для усіх процесів є описані правила та документація, такі як специфікація, тестова стратегія та тест план;

- тест кейси, тестові сценарії та чеклист складаються до початку етапу безпосереднього тестування системи;
- виявлення дефектів здійснюється одночасно, що забезпечує точну пріоритезації [13,14].

Таким чином, у проєктах на основі водоспадної стратегії детальне документування вимог відбувається на початковому етапі. Після збору інформації, пов'язаної з вимогами зацікавленої сторони, проводиться оцінка потенціальної складності реалізації кожної вимоги та їхня відповідність цілям проєкту та технічним вимогам. Останнім етапом є затвердження документації, в яку не вносяться правки впродовж наступних фаз життєвого циклу ПЗ.

Хоч контроль якості і відбувається на всіх етапах, безпосереднє тестування системи є окремою фазою і відбувається після завершення фази розробки. На цьому етапі застосовується певний набір видів тестування. Виявлені проблеми детально реєструються та повідомляються команді розробників.

1.4.2. Стратегія на основі гнучкої методології Scrum

Основу для структури Scrum було представлено в 1986 році в статті Harvard Business Review «The New New Product Development Game». Автори описали два підходи до управління розробкою продукту: деякі команди схожі на бігунів в естафеті, передаючи естафету, працюючи по прямій. Інші команди схожі на гравців у регбі, які беруть участь в одній грі та передають речі туди-сюди, якщо необхідно.

Вони дійшли висновку, що естафетний підхід, який використовується в системі поетапного планування програм НАСА, застарів. За їхніми словами, регбі-підхід дасть компаніям інструменти, необхідні для конкуренції в багатонаціональному діловому світі.

Ключовим в даній стратегії є короткі ітерації, можливість регулярного перегляду змісту проєкту та внесення змін до розробки.

Основні принципи даної методології:

– пряма комунікація між командою та замовником;
– розробка від загального до конкретних функцій;
– інкрементність та безперервна інтеграція;
– крос-функціональні команди користувачі;
– розуміння бізнес-контексту.

Оскільки детальний бізнес-аналіз є важливою складовою проєктів з гнучкою методологією, він проводиться не тільки на початкових фазах роботи над ПЗ, а й впродовж усього циклу розробки. Те саме можна сказати і про забезпечення якості, так як передбачається активна співпраця команди і зацікавленої сторони з метою створення високоякісного ПЗ, тестування за цією методологією починається з найперших етапів та продовжується на кожній ітерації (рис. 1.8).

Рис. 1.8. Стратегія на основі гнучкої методології Scrum

Характерні особливості стратегії розробки вимог та контролю якості за даною методологією:

– вимоги описуються у специфікації, яка постійно змінюється та доповнюється;
– зацікавлені особи своєчасно та на усіх етапах забезпечують команду інформацією про вимоги, приймають рішення та можуть бути активно залученими безпосередньо до процесу розробки;

- завдання пріоритезуються та найперше опрацьовуються та реалізуються ті, що є найбільш пріоритетними для зацікавлених осіб;
- на основі специфікації розробляються дизайн та прототипи, а дошка проєкту заповнюється завданнями, що описані у вигляді історій користувачів та критеріїв прийняття;
- робота над тестовою стратегією та тест планом відбувається впродовж усього проєкту, але їх дотримання не є більш важливим за робочий продукт та задоволення потреб клієнта;
- спочатку відбувається модульне, інтеграційне, а потім системне тестування;
- чеклист та тест кейси створюються паралельно з появою нової функціональності;
- регресійне тестування виконується на кожному етапі розробки та після кожної зміни в ПЗ;
- баг репорти створюються впродовж усього процесу тестування, відповідні помилки та вади ПЗ виправляються розробниками згідно пріоритету;
- підсумковий звіт про тестування складається після кожної ітерації, на основі чого складається список задач на наступний період [17,18].

Окрім зазначених практик, застосовується безліч інших, які залежать від призначення та деталей ПЗ.

Отже, у Scrum проєктах докладне документування всіх вимог відбувається не на самому початку. Спершу визначаються бізнес-потреби, потім виявляються високорівневі вимоги, які викладаються у вигляді історій користувача та критеріїв прийняття, якими наповнюється дошка завдань, після чого ці завдання пріоритезуються. Цей цикл повторюється на кожній ітерації проєкту. При цьому якомога раніше визначаються нефункціональні вимоги, щоб спроєктувати архітектуру системи. Поряд з цим, контроль якості ПЗ відбувається впродовж усього циклу розробки і забезпечує безперервний

прогрес, оскільки робочий продукт має бути готовий до випуску будь-якої миті життєвого циклу ПЗ.

Висновки

Одним з ключових факторів успіху ІТ-проєкту є правильне формування процесів розробки ПЗ. Дотримання певних підходів, злагоджена та ефективна робота команди є важливими чинниками у досягненні успішних результатів. Використання відповідних стратегії допомагає підтримувати структурованість та організованість у процесі розробки, але найголовніше, має на меті створення високоякісного продукту, що задовольняє бажання замовника та користувачів. Рівні та методи вимог, типи тестування, тестові документи та артефакти є невід'ємною частиною стратегій інженерії вимог та контролю якості програмного забезпечення, що спрямовані на досягнення поставленої мети проєкту. Врахування їх при виборі або створенні стратегії дозволяє підвищити ефективність розробки.

Огляд каскадної та стратегії на основі гнучкої методології у даному розділі підкреслює їх різні підходи. Водоспадна методологія передбачає послідовну роботу над проєктом, розбиту на фази. Ця стратегія є корисною, коли вимоги стабільні і чітко визначені. Вона дозволяє заздалегідь спланувати та контролювати кожен процес проєкту. Однак, жорсткий характер роботи та обмежена гнучкість можуть ускладнити внесення змін або адаптацію до нових вимог. Стратегія Scrum, натомість, базується на ітеративному підході. Вона покликана забезпечити гнучкість і спроможність швидко реагувати на зміни, сприяє зменшенню ризиків, покращенню комунікації та забезпечує можливість швидко внести зміни в проєкт. Однак, гнучка стратегія вимагає сильного залучення замовника та постійного нагляду за процесом розробки. Вибір між водоспадною та Scrum стратегіями залежить від контексту та характеру проєкту. У наступному розділі дані стратегії будуть застосовані до розробки фінансового застосунку та буде з'ясовано їхню ефективність та відмінності.

СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1. Застосування стратегій для розробки фінансового застосунку

2.1.1. Призначення та функціональні можливості системи

Стратегії розробки вимог та забезпечення якості не можуть розглядатись окремо від конкретних розроблених ПЗ. Тільки на основі результативності створеного проєкту можна зрозуміти ефективність застосованих стратегій, з’ясувати переваги та недоліки.

Останнім часом досить популярними стали застосунки для управління особистими фінансами. Оскільки трендом є планувати не тільки побут і роботу, а також ставити фінансові цілі та отримувати аналітику щодо своїх витрат і доходів, розроблено програмне забезпечення саме для управління фінансами.

Підставою для створення та дослідження даного проєкту є набуті навички з багатьох дисциплін курсу системного аналізу, використані сучасні технології, а саме мова програмування, фреймворки, а також популярні методології та стратегії. Для розробки проєкту використано дві з них: на основі каскадної методології та Scrum.

Проєкт не являє собою стартап або комерційний продукт, а розроблений з нуля виключно в навчальних цілях та використаний для дослідження сучасних стратегій розробки вимог та забезпечення якості програм.

Фінансовий застосунок SmartSpend – це ПЗ для відстеження доходів та витрат, головна мета якого забезпечити зручний та якісний інструментарій для керування особистими фінансами (рис. 2.1).

Основні можливості, що надає система:

- створення окремих облікових записів та обмеження доступу до фінансових даних;
- керування гаманцями та бюджетами;
- створення категорій витрат і доходів та транзакції до них;

– планування майбутніх транзакцій;
– перегляд та відстеження статистики фінансів;
– маніпулювання створеними компонентами;
– збереження усіх дій та даних.

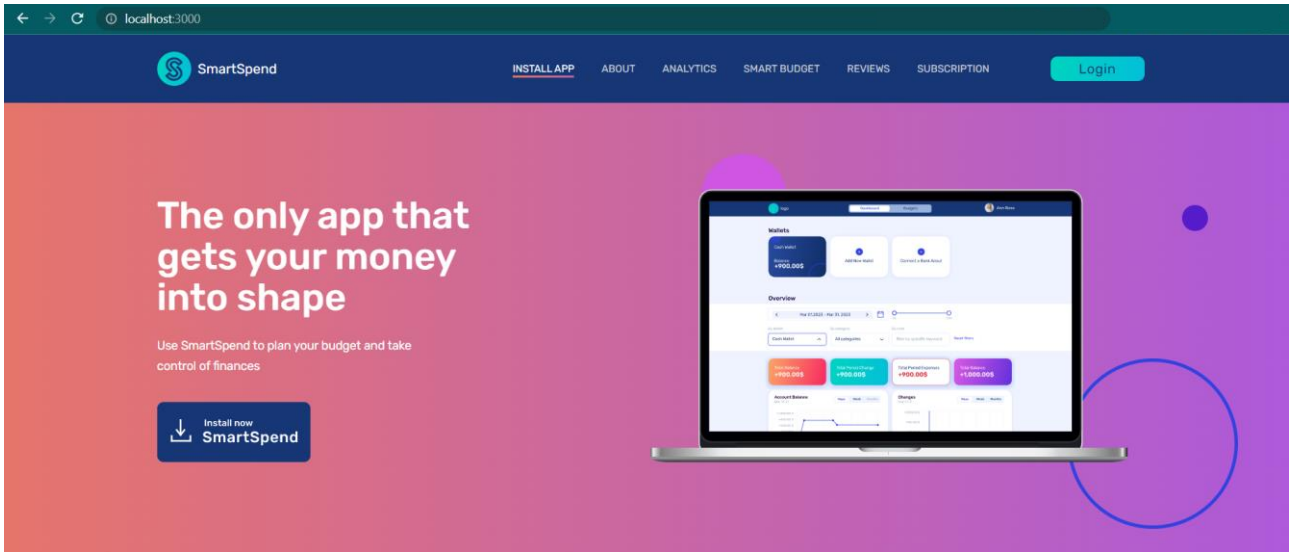


Рис. 2.1. Цільова сторінка застосунку

2.1.2. Використані технології та організація роботи

Застосунок написаний на мові програмування JavaScript: Backend частина реалізована на Node, а саме на основі фреймворку Fastify, ORM-бібліотеки Objection та конструктору запитів Knex. Frontend частина розроблена з використанням бібліотеки React та набору інструментів Redux + Redux Toolkit. База даних – PostgreSQL. Проєкт було розгорнуто за допомогою сервісу Netlify, що надає хостинг та безперверний backend.

Над створенням ПЗ працювала команда з 9 розробників та 2 QA, процес контролювали 3 досвідчених фахівців, зацікавленою особою був власник продукту (Product Owner). Життєвий цикл проєкту складав 5 тижнів: перші 2 тижні розробка відбувалась за водоспадною стратегією, наступні 3 тижні за Scrum стратегією.

Робота над проєктом відбувалась за допомогою GitHub. Репозиторій був центральним місцем для зберігання, керування та розповсюдження версій коду розробників. GitHub Actions використані для CI та CD. За допомогою GitHub Projects, організовано роботу над проєктом в рамках репозиторію, чим було забезпечено планування роботи, визначення завдань та відстеження прогресу.

2.2. Розробка вимог до фінансового застосунку

2.2.1. Збір та аналіз вимог

Основних джерелом бізнес-вимог для даного проєкту був власник продукту, який визначився з потребами та стратегічними цілями та надав відомості про те, яка система буде корисною для його бізнесу та які функції потрібні для підтримки діяльності. Оскільки продукт не є комерційним, бізнес-правила не накладались на функціональні та інші вимоги.

Саме функціональні вимоги є результатом аналізу конкурентів на ринку ПЗ для керування фінансами, визначено, що робить їх популярними та чим вони задовольняють потреби користувачів. З існуючих фінансових застосунків бізнес-вимогам замовника найбільше відповідав Spendee, який виступив аналогом для з’ясування усіх інших вимог до системи. Виявлення найбільш популярних функцій та інтерфейсів дозволило врахувати також і вимоги користувачів.

На основі детального аналізу аналогу описано функціональні вимоги до таких основних компонентів системи:

- головна сторінка;
- реєстрація, авторизація користувачів та створення профілю;
- управління гаманцями;
- управління бюджетами;
- управління транзакціями;
- управління майбутніми транзакціями;

- управління категоріями;
- імпорт даних у файлах формату CSV та з інших фінансових сервісів;
- генерація звітів та статистики фінансових операцій;
- декілька рівнів платної підписки.

Системні вимоги, що описують технічні характеристики та необхідні для ефективної роботи, згідно яким ПЗ повинно:

- мати веб-сервер для доступу до системи через Інтернет для та база даних для зберігання інформації;
- бути написаним на мові програмування JavaScript;
- мати клієнтську частину доступну для користувачів через браузери Google Chrome, Mozilla Firefox, Opera, Safari;
- мати мобільний та прогресивний веб-застосунок.

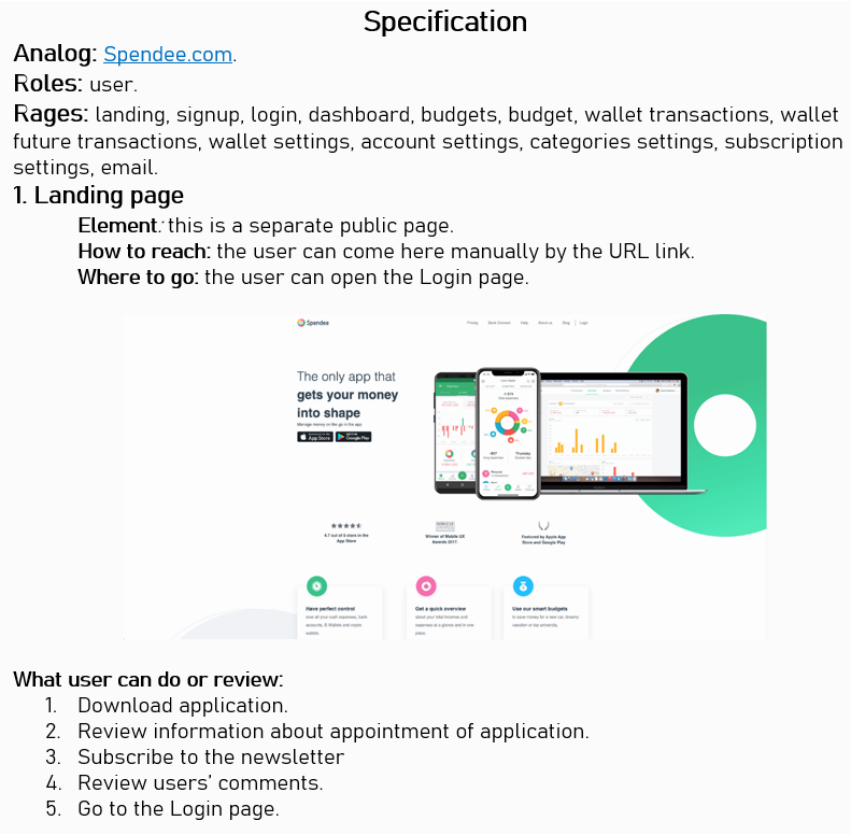
Вимоги до зовнішнього інтерфейсу:

- легко зрозумілий та чіткий для користувачів різного рівня технічної компетентності;
- зручний у використанні, користувачі можуть швидко знаходити потрібну інформацію та взаємодіяти з сайтом без зайвих зусиль.
- інтерфейс має привабливий та сучасний візуальний дизайн, який збільшує задоволення користувачів від використання сайту.
- застосунок має адаптивний дизайн, для зручного використання на різних пристроях та розмірах екранів.
- застосунок має інтерактивні елементи, такі як кнопки, меню, підказки.

Визначено атрибути, за якими можна вважати систему якісною: функціональність, швидкість, сумісність, безпека, простота використання, масштабованість.

2.2.2. Документування та перевірка вимог

Результатом виявлення вимог є їх реєстр. Вимоги зазвичай оформляються в простій письмовій формі, без особливої регламентації. Дані вимоги далеко не у всьому можуть задовольняти критеріям правильних вимог. Проте, специфікація, не дивлячись на невисокий рівень формалізації, грає важливу роль, тут зібрані думки всіх зацікавлених сторін і головна мета збору початкових вимог полягала в тому, щоб не пропустивши чогось важливого. Для того, щоб підвищити рівень інформативності вимог, усунути взаємні суперечності і добитися виконання їх інших основних характеристик, здійснюється перехід від повністю неформалізованих текстів до частково регламентованих. Таким чином, на основі проаналізованих функціональних, системних та бізнес-вимог була створена остаточна версія специфікації до ПЗ (рис. 2.2). Яка включила таку основну інформацію, як посилання на аналог майбутнього ПЗ, ролі та перелік сторінок. Опис того, як перейти на певну сторінку та на які сторінки вона веде, дає зрозуміти архітектуру сайту. Для кожної сторінки вказано публічний або приватний режим доступу, а також перелік дій, які повинен мати можливість здійснити користувач, та компонентів, які мають бути зображені. Зазначено також вимоги щодо їх організації на сторінках.



Specification

Analog: Spendee.com.

Roles: user.

Rages: landing, signup, login, dashboard, budgets, budget, wallet transactions, wallet future transactions, wallet settings, account settings, categories settings, subscription settings, email.

1. Landing page

Element: this is a separate public page.
How to reach: the user can come here manually by the URL link.
Where to go: the user can open the Login page.

What user can do or review:

1. Download application.
2. Review information about appointment of application.
3. Subscribe to the newsletter
4. Review users' comments.
5. Go to the Login page.

Рис. 2.2. Сторінка зі специфікації

Специфікація є основою для створення інших документів та артефактів, тому за допомогою її детального аналізу було розроблено ментальну карту (Mind Map), що детально зображує архітектуру ПЗ (рис. 2.3).

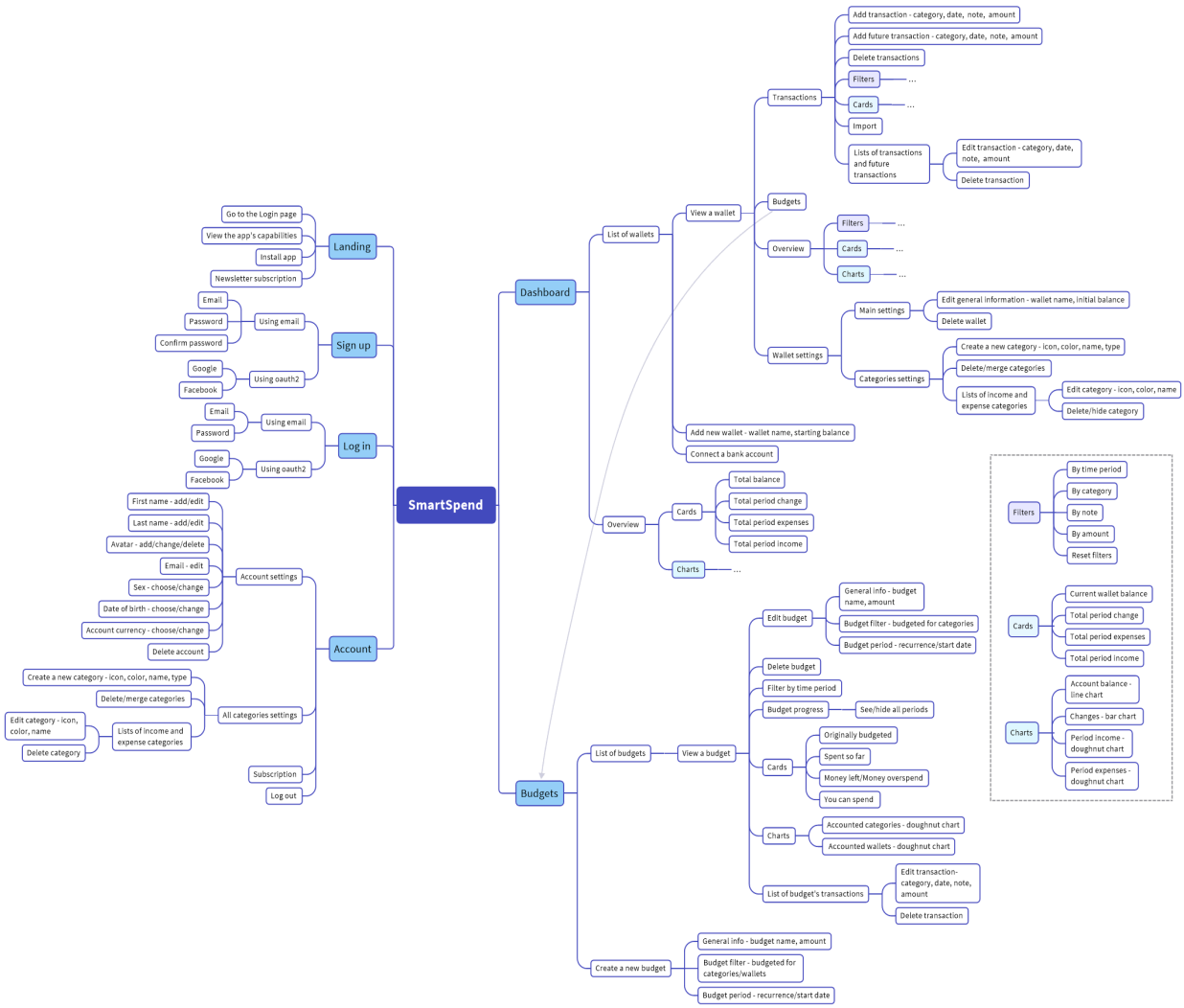


Рис. 2.3. Ментальна карта ПЗ

На схемі описано публічні та приватні сторінки, з яких складається ПЗ, дії, які можна виконати, компоненти, форми та їхні поля, графіки, картки та фільтри, за допомогою яких користувач може переглядати та фільтрувати статистику по своїм фінансам.

Мапа сайту є джерелом інформації для створення інтерфейсу, тому на її основі було розроблено дизайн ПЗ (рис. 2.4). Визначити загальний вигляд та структуру сторінок допомогли скетчі, створені на основі специфікації. Кольорова палітра та шрифти, які відповідають бренду та стилю сайту, були затверджені власником продукту. Дотримуючись загальної концепції, були створені елементу інтерфейсу, такі як кнопки, форми, меню, поля, графіки та інші.

Дизайн сформовано чітко з урахуванням проаналізованих вимог за допомогою інструменту Figma. Не менш важливим було дотримання однакового вигляду елементів кольорів та шрифтів на усіх сторінках, оскільки це забезпечить зрозумілість та зручність для користувачів та оптимізує внутрішню архітектуру ПЗ.

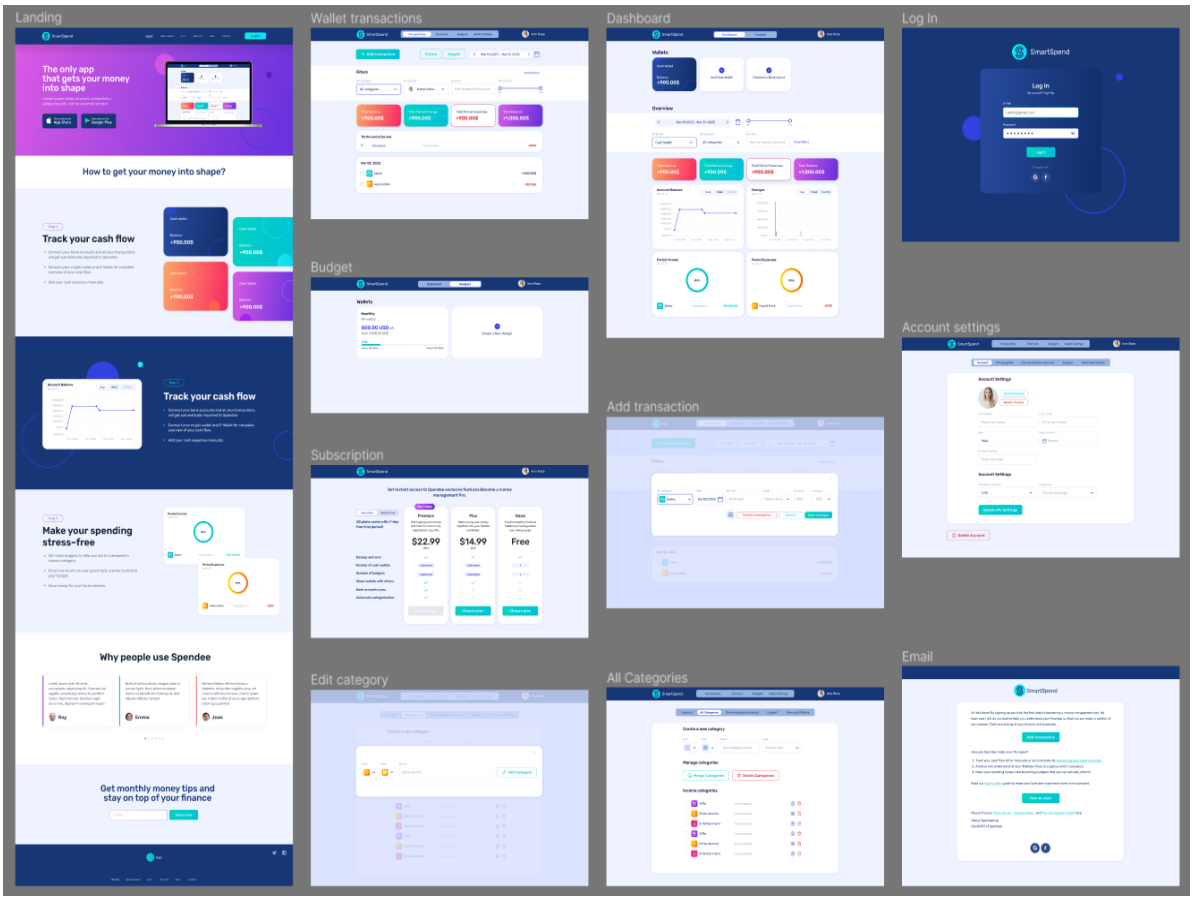


Рис. 2.4. Дизайн

Так як вимоги до ПЗ визначають не тільки те, які функції та можливості повинна мати система, а і як програма повинна працювати з даними, які будуть використовуватись, було розроблено базу даних (рис. 2.5). Перед безпосереднім створенням схеми бази даних, тобто визначення полів, типів даних, таблиць та зв’язків між ними, проведено детальний аналіз того, які дії зможе здійснити користувач, визначено сутності та зв’язки між ними. Визначено структуру даних, яка буде зберігатися в програмі, та обрано систему керування базами даних, а саме PostgreSQL.

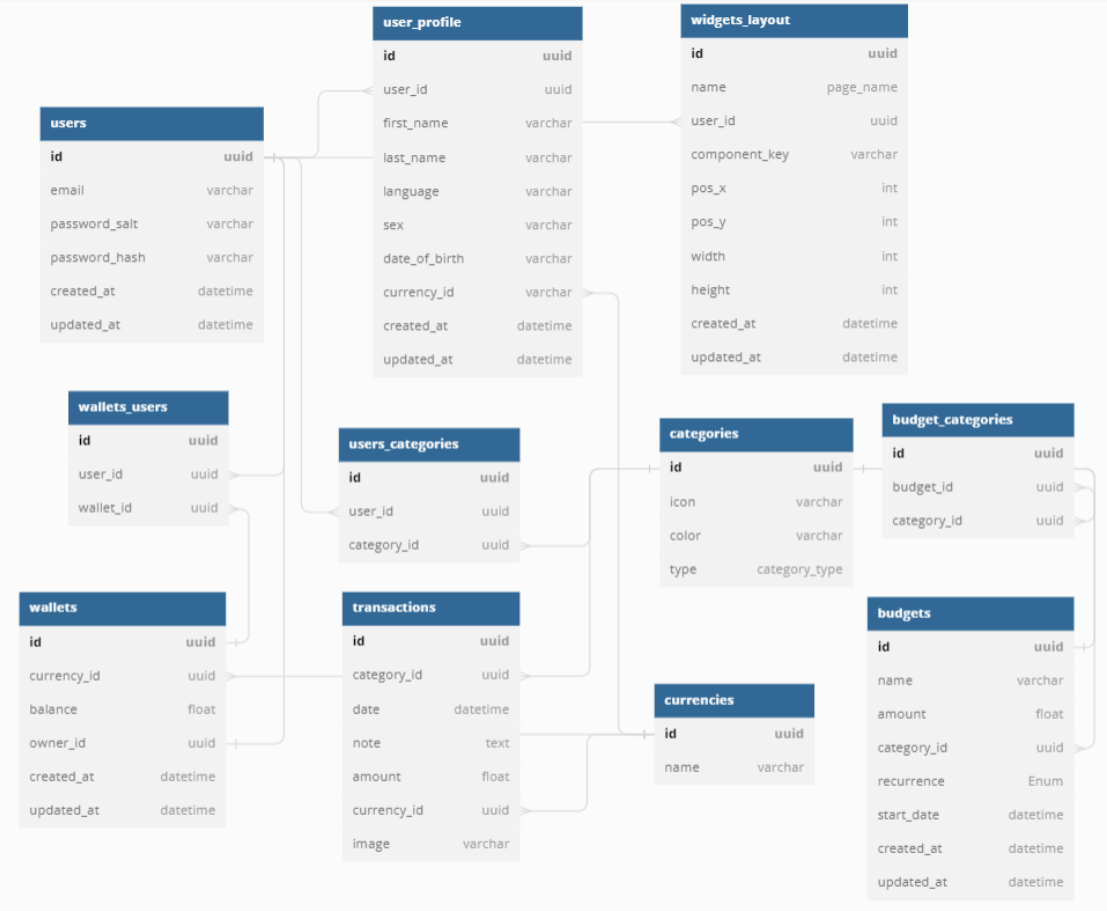


Рис. 2.5. База даних

Отже, розробка вимог до фінансового застосунку SmartSpend складалась з кількох етапів. За каскадною стратегією вилучення вимог відбувалось за допомогою інтерв'ю з власником продукту на початковому етапі проєкту. Він надав інформацію про бажану загальну концепцію продукту, бізнес-вимоги, основною з яких є надання користувачам безоплатного та платного доступу до ПЗ, що спрощує процес контролю фінансових операцій, зберігає інформації та допомагає збільшити свою ефективність у фінансовому плануванні та управлінні особистим бюджетом.

Джерелом функціональних, системних та інших вимог став аналог майбутнього сайту.

Аналіз вимог на першому етапі, розуміння бізнес-цілей, допомогли з'ясувати які з них повинні бути реалізовані обов'язково. Для перевірки вимог застосовано найпоширеніший метод тестування вимог: послідовний перегляд

на перевірка кожної з вимог. Впродовж перших двох тижнів ПЗ розроблялось чітко згідно вимог замовника, дотримуючи стратегії Scrum після кожного релізу у кінці другого тижня та впродовж наступних бажання та ідеї замовника фіксувались та переглядались, після оцінки можливих ризиків та з огляду на часові витрати вирішувалось, чи будуть ці функції включені в ПЗ.

Водоспадна стратегія передбачила документування у вигляді специфікації, а потім і у вигляді ментальної карти, що було проведено для того, щоб зберегти та організувати всі вимоги до ПЗ. Це дозволило розробникам, тестувальникам та іншим учасникам проєкту легко отримати доступ до всієї інформації щодо вимог та зрозуміти їх. Після фіксування вимог було проведено валідацію, тобто перевірку, щоб переконатися, що вони відповідають характеристикам правильних вимог та потребам бізнесу. Після успішної перевірки вимоги були затверджені бізнес-власником та стали основою для подальшої розробки архітектури сайту, дизайну, бази даних та заповнення дошки з завданнями.

Вимоги, зібрані за гнучкою стратегією, не описувались детально у специфікації, а одразу оброблялись та фіксувались у вигляді завдань, що могли включати історії користувачів та критерії прийняття, та одразу передавались у роботу розробникам.

2.3. Забезпечення та контроль якості фінансового застосунку

2.3.1. Використані документи та артефакти

Важливо зазначити та коротко описати усі документи та артефакти, що були використані для контролю якості ПЗ, а саме тестову стратегію, тест план, чеклист, документ з правилами валідації полів, баг репорти а також набір API тестів.

Високорівневим документом, що описував усі процеси тестування фінансового застосунку SmartSpend, була тестова стратегія. У ньому визначено мету тестування ПЗ, надано інформацію про аналог, використані технології,

посилання на розроблені специфікацію, дизайн і тд. Також тестова стратегія містить інформацію про склад команди, строки початку та завершення проєкту, визначає підхід до комунікації.

У документів формульовано тестовий підхід, визначальною характеристикою якого є проактивність. Це підхід, при якому процес тестування починається якомога раніше, щоб знайти та виправити дефекти до об’єднання окремих модулів програми в одну систему. Визначено ступені автоматизації: ручне тестування, яке буде застосовуватись переважно, та автоматизоване, але тільки для API. Основні типи тестування, які застосовувались, це функціональне, регресійне, приймальне, димове та тестування користувацького інтерфейсу. Відповідно визначено пріоритети тестової діяльності:

- аналіз специфікації та вимог до системи;
- оновлення ментальної карти, дизайну, бази даних;
- тестування виконаних завдань та створення звітів про помилки;
- регресійне тестування;
- написання та виконання автоматизованих тестів;
- створення історій користувачів та критеріїв прийняття;
- написання та оновлення чеклисту;
- оновлення тестової стратегії та тест плану;
- визначення правил перевірки полів.

Згідно цього, на кожен тиждень розписані види діяльності, які слід застосувати для контролю якості. Визначено тестові середовища, а саме браузери, розширення дисплеїв, орієнтація та локалізація. Вказано посилання на документи та артефакти, що будуть використовуватись на проєкті, та зазначено перелік розділів тест плану. Визначено життєвий цикл баг репорту та його шаблон. Оскільки завдання для команди розташовуються на дошці GitHub та потребується пояснення організації роботи, у тестовій стратегії описано, хто відповідальний за завдання у певній колонці та що саме необхідно робити.

Визначено основний критерій прийняття продукту зацікавленою стороною: реалізовано не менше 80% вимог та не менше 90% основних функцій , описаних у специфікації та інших документах, що містять вимоги.

Оскільки необхідно розуміти, що очікується від тестування ПЗ та які результати потрібно отримати, стратегія визначає вхідні та вихідні тестові критерії, які є наступними:

- критерії початку тестування: процес тестування починається на етапі розробки вимог, контроль якості відбувається на усіх етапах, безпосереднє тестування компоненту системи відбувається після завершення завдання розробником;
- критерії закінчення тестування: усі роботи по забезпеченню якості повинні бути завершені до дати релізу, виправлено 95% виявлених дефектів, немає блокуючих та критичних дефектів.

З’ясовано можливі ризики та управління ними: відсутність члена команди, неправильний результат тестування, недоступність тестового середовища, відсутність часу для тестування.

Наступним високорівневим документом є тест план, на початку його створення важливо зрозуміти ключові функції ПЗ та розставити їх за пріоритетністю реалізації на основі їх важливості. У цьому порядку слід віддати перевагу таким функціям:

- реєстрація та авторизація: метою реєстрації є створення облікового запису користувача та збір інформації про нього, яка буде використана для персоналізації його досвіду; метою авторизації є надання користувачу доступу до приватних сторінок;
- інформаційна панель для відображення фінансової інформації: це основна сторінка, яка забезпечує візуальний огляд фінансової інформації користувача, відображає ключові фінансові діаграми та показники, такі як загальний баланс, доходи та витрати;
- управління гаманцями та бюджетами: забезпечує створення, редагування та видалення гаманців та бюджетів;

- управління транзакціями: метою додавання транзакцій є відстеження доходів та витрат, тобто фінансової діяльності;
- категоризація транзакцій: призначення категорій транзакціям є важливою функцією в інструментах управління особистими фінансами;
- імпорт транзакцій: можливість імпортувати власні транзакції безпосередньо в застосунок SmartSpend;
- заплановані операції: функція запланованих транзакцій, яка дозволяє користувачам планувати майбутні витрати або доходи;
- трансфери: функція забезпечує можливість переказувати кошти між різними гаманцями всередні застосунку;

З огляду на це у тест плані зазначено перелік функцій, які планується тестувати та перелік реалізованих функцій.

Головним документом, що забезпечував контроль якості, був чеклист, який використовувався для системного підходу до тестування програми. У документі перелічено дії та пункти, які необхідно виконати для перевірки функціональних та нефункціональних вимог до продукту.

Кожен пункт містить додаткові пояснення, посилання на документацію та баг репорт, якщо перевірку не пройдено. Після проходження пункту, він відмічається як виконаний, невдалий, заблокований або пропущений, це дозволяє проводити регресійне тестування та відслідковувати прогрес всього проєкту.

Документ з правилами валідації полів – спеціальний документ, який містить перелік полів та вимоги до їх заповнення. Він використовувався для забезпечення якості введення даних користувачами, зменшення кількості помилок та полегшення користування програмою. У даному документі визначено наступні пункти:

- назва сторінки або форми;
- назва поля, до якого застосовуються вимоги;
- символи, які можуть бути введені;
- обов'язковість поля;

– мінімальна та максимальна довжина;
– повідомлення, яке виводиться користувачеві у разі помилкового введення даних.

Найважливішим документом, що допомагав забезпечувати та контролювати якість систему, був звіт про помилку, тому під час безпосереднього тестування застосунку та виявлення невідповідностей дизайну, функціональним вимогам, інформація про дефект фіксувалась у вигляді баг репорту (рис. 2.6).

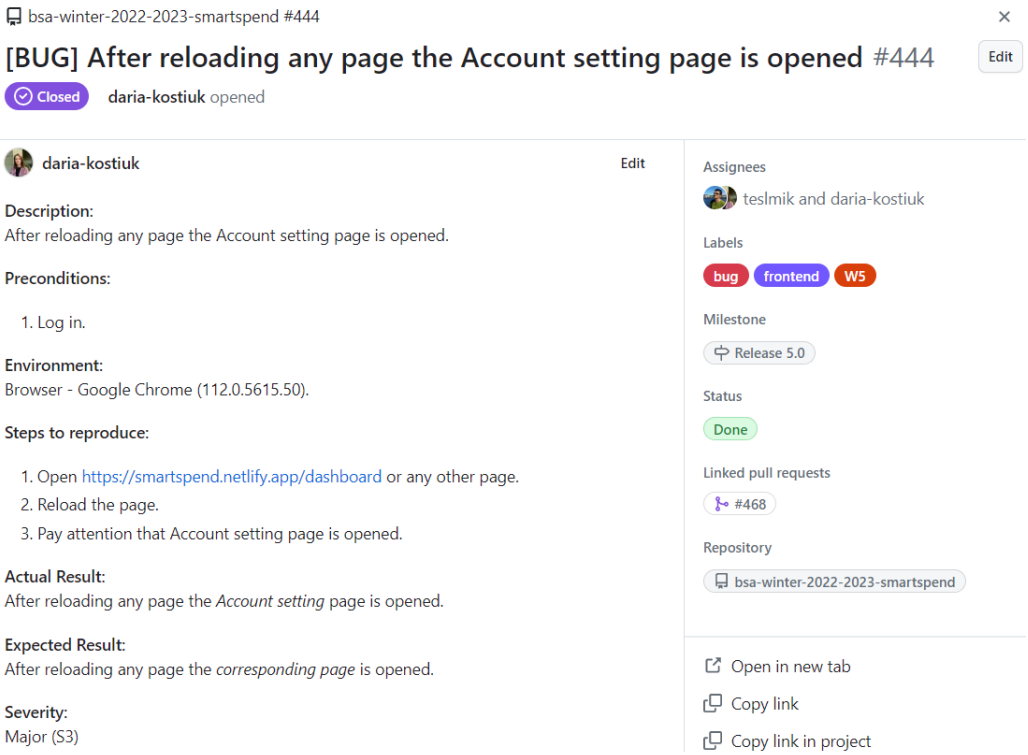


Рис. 2.6. Приклад баг репорту

Цей документ має наступний шаблон:

– summary – назва помилки;
– description – детальний опис помилки;
– preconditions – умови, що потрібно виконати до кроків відтворення;
– environment – назва та версія браузеру, де виявлено помилку;
– steps to reproduce – кроки для відтворення багу;
– actual result – помилкова поведінка системи;

– expected result – очікувана поведінка, вигляд або результат роботи програми;
– visual proof (screenshots, videos, text) – візуальне підтвердження помилки;
– severity – ступінь впливу на програму.

Баг репорт призначається для виправлення розробнику, після виконання перевіряється і закривається, якщо баг не відтворюється, або в іншому випадку повертається.

3.3.2. Процес тестування

Як вже було зазначено, тестова стратегія визначає, які типи тестування застосовувались для забезпечення якості ПЗ, тож варто вказати яким саме чином та за допомогою яких документів вони проводились. Оскільки основним критерієм початку безпосереднього тестування були зміни в коду, воно починалось після завершення відповідного завданням розробником.

Тестування користувацького інтерфейсу, а саме коректності відображення, відбувалось за допомогою порівняння деталей дизайну та написаного розробниками коду HTML та CSS завдяки інструментам розробника (Development Tools) (рис. 2.7). Також це включало перевірку взаємодії з елементами інтерфейсу, такими як кнопки, поля введення, прапорці та інше.

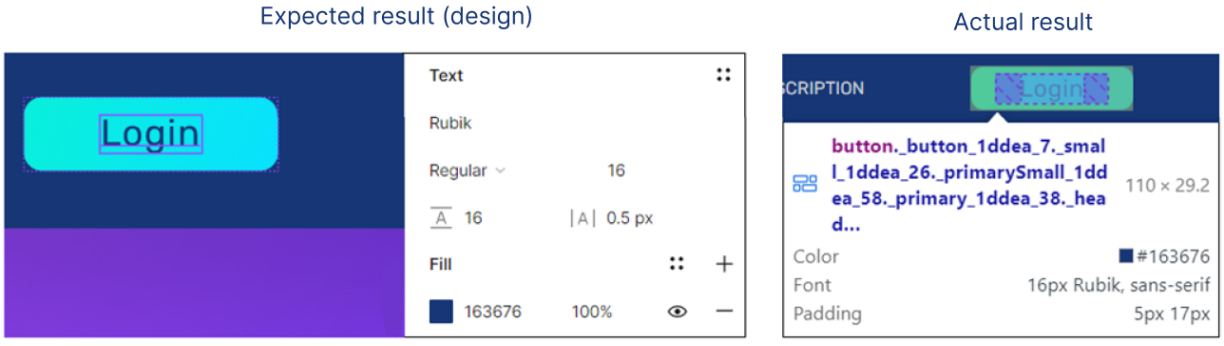


Рис. 2.7. Тестування користувацького інтерфейсу

Чеклист забезпечив функціональне тестування всієї системи, а також регресійне тестування. Димове тестування відбувалось без використання тестових артефактів та усіма членами команди, включало перевірку основного функціоналу ПЗ, щоб швидко виявити великі проблеми або помилки перед релізом. Приймальне тестування доповнювало функціональне та виконувалось з метою підтвердження того, що впродовж ітерації було реалізовано усі очікування клієнта.

Для забезпечення надійності системи чиненайважливішим було тестування API, яке відбувалось за допомогою інструменту Postman, що дозволяє відправляти HTTP-запити на сервера та отримувати відповіді на них. Детально були протестовані сторінки реєстрації та логіну, а саме: реєстрація нового користувача та авторизація з даними, що відповідають правилам валідації, неправильними та невалідними, а також тими, що вже зареєстровані. Проведено тестування основного сценарію поведінки користувача на сайті з правильними даними, а саме створення користувача, авторизація, створення/редагування нового гаманця/бюджету/ категорії/транзакції, а також у кінці видалення створених компонентів та користувача.

Приклад запиту для перевірки часу відповіді, статус коду та повідомлення, яке бачить користувач, який вводить неправильну електронну адресу, зображений на рисунку 2.8.

Кроки тестування API:

- створення нового запиту, у якому вказано метод, який дозволяє переглянути/додати/змінити/видалити дані та URL-адресу на яку будуть відправлятися запити;
- формування тіла запиту у форматі JSON;
- написання тестів до запиту, а саме на перевірку HTTP-коду, часу відповіді та валідаційного повідомлення, якщо перевіряється реакція системи на неправильні дані;
- отримання відповіді від серверу, аналіз її вмісту, часу та статус коду;

Невідповідність отриманого статус коду, валідаційного повідомлення або часу відповіді очікуваному результату є підставою для створення баг репорту.

При тестуванні застосунку SmartSpend за допомогою Postman було використано такі його можливості, як: об’єднання і збереження запитів в колекції для повторного використання, використання змінних для автоматизованого запуску тестів та тестування автентифікації і авторизації допомогою токенів.

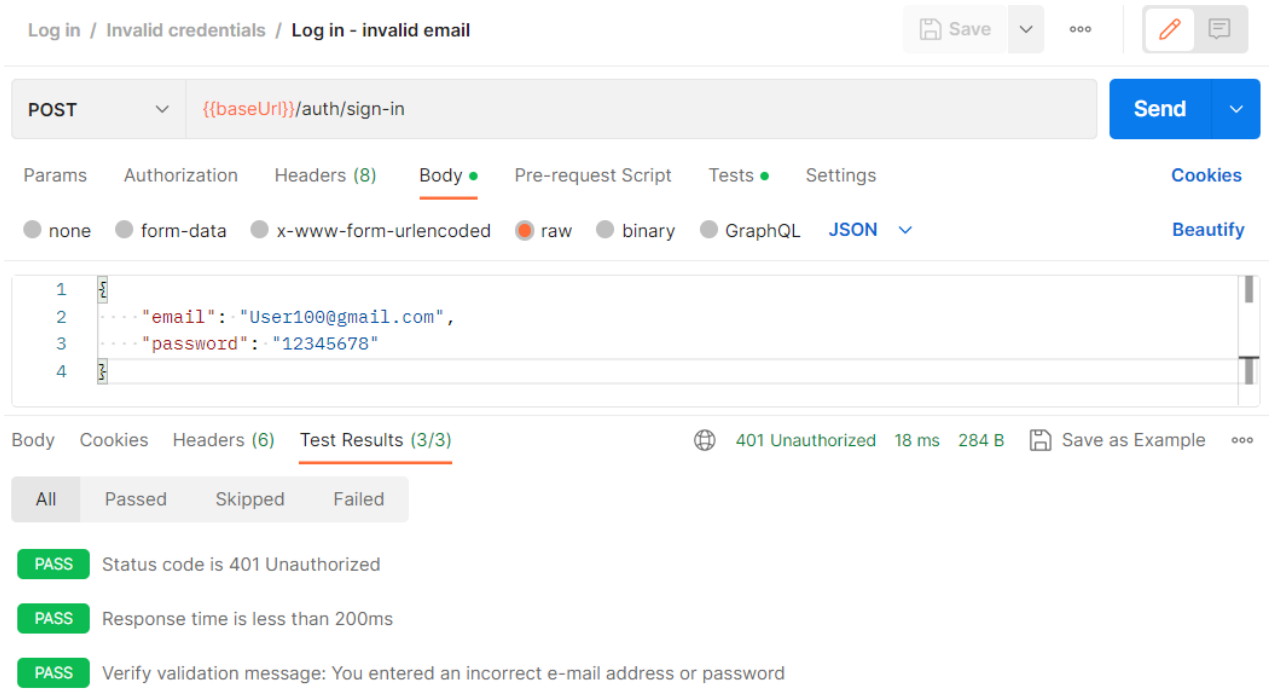


Рис. 2.8. Приклад API тестів за допомогою Postman

Таким чином, при використанні водоспадної стратегії тестування було зосереджене на контролі якості системи, тобто встановленні того, що виконане завдання задовольняє критеріям прийняття та вимогам, акцент був зроблений на роботі з високорівневими документами, складанні чеклисту та написанні API тестів. По факту знаходження дефектів у кінці першого, а потім другого тижня фіксувались баг репорти та виставлялись пріоритети. Поряд з цим, стратегія на основі Scrum методології прияла не тільки контролю, а у більшій мірі забезпеченню якості ПЗ та превентивних процесах. Перевірки по чеклисту,

тобто регресійне тестування, API тестування застосовувались у другій половині життєвого циклу ПЗ.

2.4. Аналіз отриманих результатів

2.4.1. Остаточна версія програмного забезпечення

Кінцевим продуктом можна вважати результат роботи команди протягом п’ятьох тижнів, представлений на останній зустрічі з власником продукту, одну з основних сторінок можна побачити на рисунку 2.9. З десяти основних функцій, перелічених у тестовій стратегії, було реалізовано усі, окрім функції імпорту даних з інших фінансових сервісів, тобто 90% замовлених можливостей, як і потребував замовник. У специфікації та інших документах описано 180 функціональних вимог, з який було реалізовано 150, що склало 83,33% від загального числа і ледве перевищило потрібні 80%, що були визначені зацікавленими особами. Зі 131 виявленого багу, 6 не були виправлені, але оскільки жоден з них не був блокуючим або критичним і їх кількість склала 4,58% від загальної, це задовольнило основні критерії прийняття власника продукту.

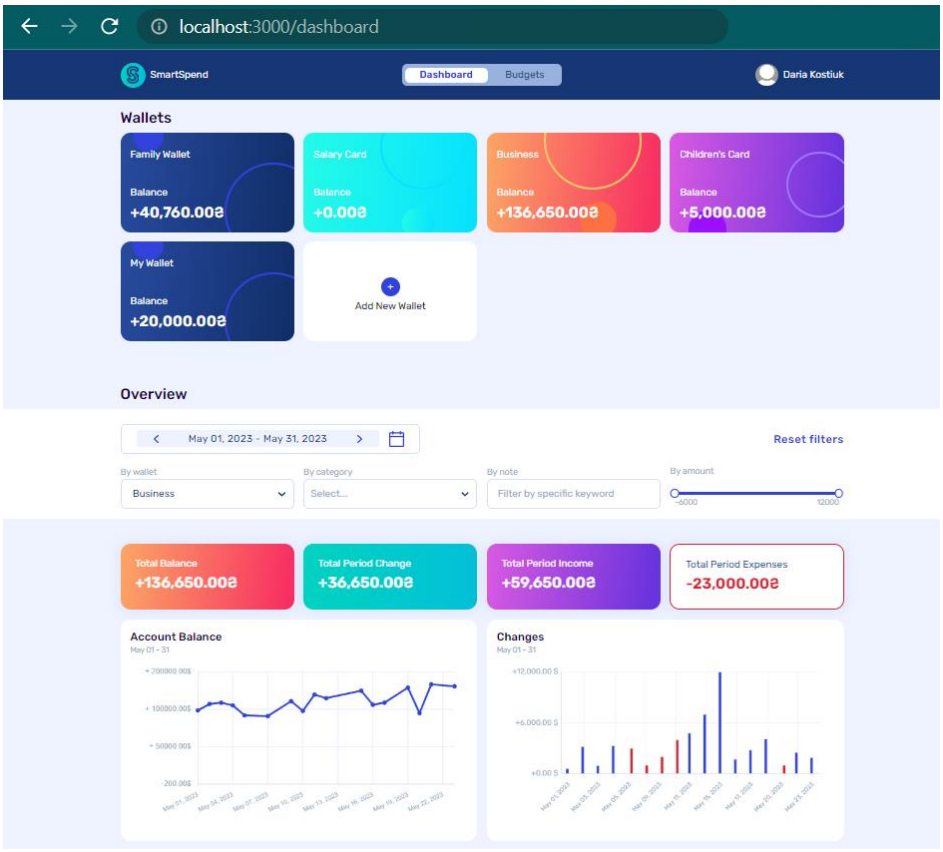


Рис. 2.9. Головна сторінка застосунку

Розроблену систему можна вважати якісною, оскільки вона відповідає визначеним атрибутам якості. ПЗ має широкий набір функцій, що дозволяє користувачам керувати своїми фінансами, працює ефективно, відповідає на запити користувачів без зайвої затримки, що підтверджує API тестування. Застосунок сумісний з різними веб-браузерами, а також з різними пристроями та їх операційними системами. Оскільки фінансовий застосунок містить конфіденційну інформацію, було забезпечено шифрування даних та захист від несанкціонованого доступу. Система має зрозумілий та інтуїтивний інтерфейс, що дозволяє користувачам легко використовувати різні функції. Мінімальна кількість кроків для виконання операцій і логічна організація елементів допомагають забезпечити простоту використання. Також передбачено, що застосунок може бути масштабований, оскільки якби це був комерційний проєкт, він міг би набути популярності та кількість користувачів, які його використовують, могла зрости, у такому випадку можна буде збільшувати обсяги ресурсів для обробки більшої кількості запитів без впливу на продуктивність та функціональність.

2.4.2. Оцінка та порівняння застосованих стратегій

Порівняння стратегій розробки вимог та контролю якості допомагає визначити найбільш ефективний підхід для конкретного проєкту. Дві застосовані стратегії мають свої переваги та недоліки, аналіз допоможе виявити їх та визначити ризики, пов’язані з кожною з них, а також обрати ту стратегію, яка найкраще відповідає потребам даного проєкту та подібних, та для запобігання проблемам у майбутніх проєктах. За результатами дослідження важливо також внести зміни у поточну стратегію або впровадити нові методики, що допоможуть поліпшити якість тестування та зменшити ризики.

Перед розрахунком метрик для визначення ефективнішої з двох стратегій, важливо проаналізувати темп роботи розробників протягом усього життєвого циклу проєкту.

Абсолютним показником є кількість внесених змін у код (комітів) за кожен тиждень (рис. 2.10). Слід звернути увагу, що впродовж першого тижня відправлено найбільше змін вихідного коду в репозиторій, це пов’язано з тим, що на першому етапі розробники постійно вносили невеликі зміни. Але важливо розуміти, що потім розмір коміту був визначений та залишався однаковим впродовж останніх трьох тижнів.

Також абсолютним показником є середня кількість внесених змін по дням тижня за однією та другою стратегіями (рис. 2.11).

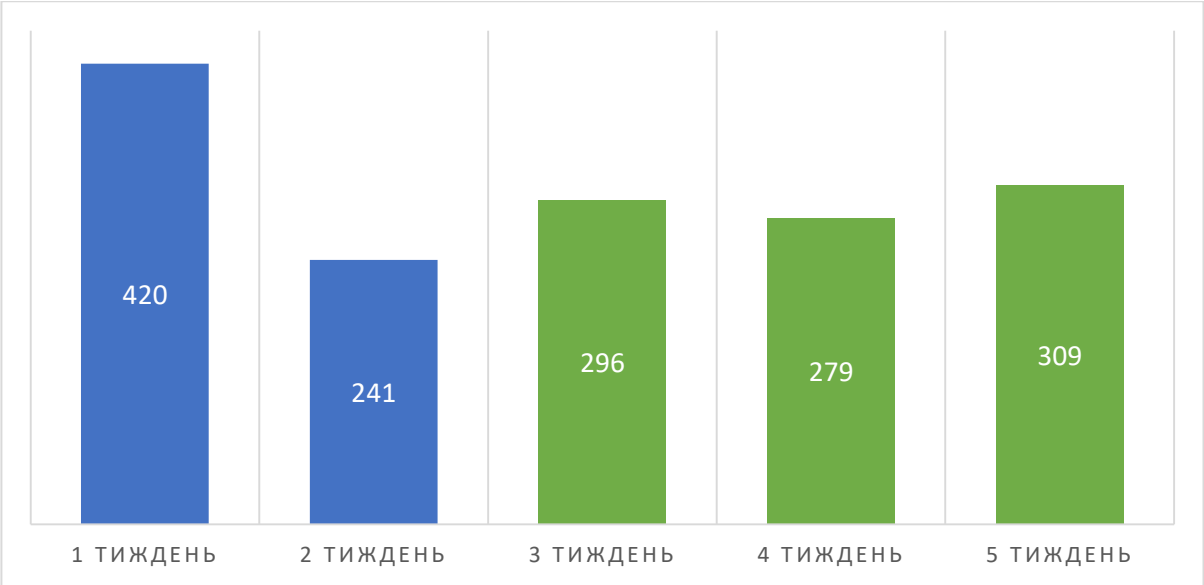


Рис. 2.10. Кількість внесених змін

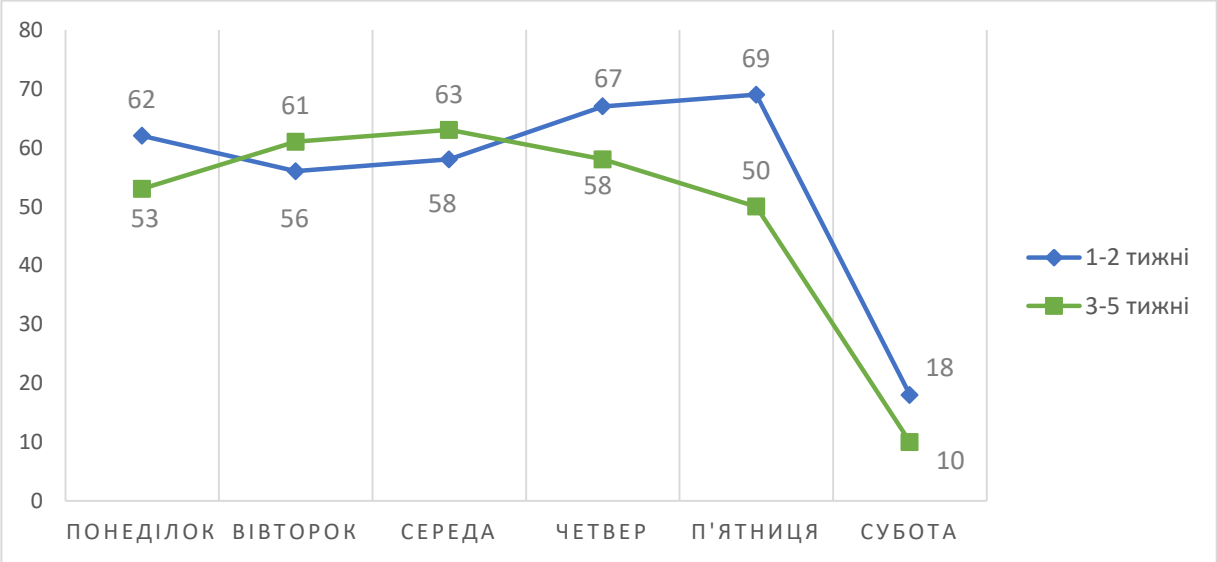


Рис. 2.11. Середня кількість внесених змін по дням тижня

Оскільки кожна ітерація закінчувалась щосуботи і проводились основні зустрічі з власником продукту для демонстрації ПЗ, можна помітити, що у ці дні вносилось найменше змін. При дотриманні каскадної стратегії розробники найбільше додавали нового коду під кінець тижня, що тягло за собою виявлення більшості помилок в останні його дні, а деяких на початку вже нової ітерації. Під час використання Scrum стратегії, розробка була зосереджена на першій половині тижня, що забезпечувало виявлення дефектів більш рівномірно, тобто кожного дня.

Певний коміт вирішує помилки або впроваджує зміни, пов'язані з певною задачею, тож більш показовою є статистика завершених завдань протягом кожного тижня (рис. 2.12).

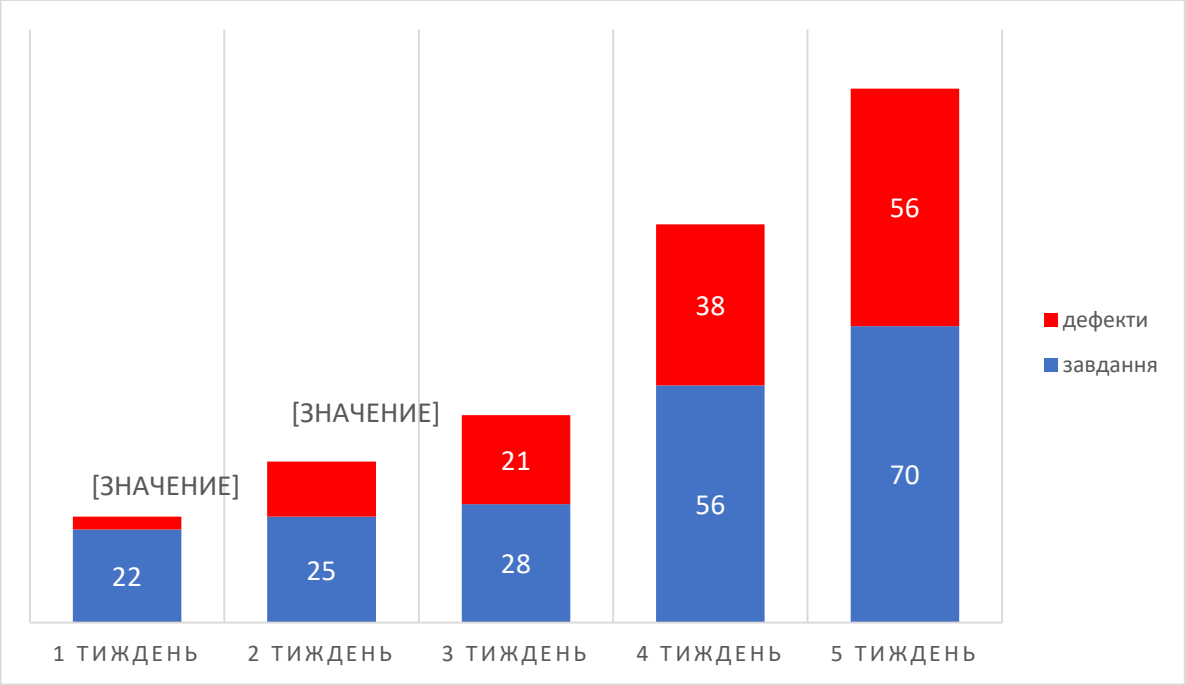


Рис. 2.12. Кількість завершених завдань та знайдених дефектів

За перший та другий тижні виконано найменше завдань, хоч і внесено найбільше змін у код, це пов’язано з тим, що основні функції та компоненти, які є найбільш об’ємними, реалізовувались на початку. Але основною причиною є строге дотримання вимог, описаних у специфікації, що визначає Waterfall стратегія, та критеріїв прийняття, без яких завдання не може бути прийняте, якщо виконано не в повній мірі. Можна побачити, що кількість закритих завдань за останню ітерацію у п’ять разів більша, ніж за першу, протягом третього та четвертого тижнів також виконано переважну більшість завдань. Окрім природнього збільшення продуктивності роботи команди перед наближенням релізу, на це вплинула зміна стратегії розробки вимог на гнучку.

Важливе значення має кількість виявлених дефектів, що за перший тиждень склала 3, за другий – 13, за третій – 21, за четвертий – 38, та за п’ятий –

56. Що склало 12%, 34,1%, 42,9%, 40,4% та 44,4% від загальної кількості завдань відповідно.

Низька кількість дефектів за перші два тижні підтверджує важливість встановлення зрозумілих та якісних вимог, що забезпечує Waterfall стратегія, згідно чого розробникам не довелось вносити зміни в завдання або трактувати бажання зацікавленої сторони неправильно. Наслідком переходу на Scrum стратегію стала велика кількість дефектів протягом останніх трьох тижнів, які становлять значну частину від виконаних завдань загалом.

Розподіл дефектів по пріоритетам (рис. 2.13-2.14) дає зрозуміти наскільки робота з вимогами впливає на їх виявлення та серйозність.

У цьому контексті потребує уваги власне походження дефектів. На першому тижні були пов’язані з неправильною організацією програмних файлів. На другому тижні незначні помилки були пов’язані з недотриманням деталей дизайну, оскільки вимоги до нього не були гнучкими та потребували повного наслідування. Неправильна робота компонентів (кнопок, полів) та реалізованих функцій потягнула за собою баги вищого пріоритету. Причиною критичного та блокуючого дефектів при дотриманні Waterfall стратегії стало неправильне створення користувацького токену. Тобто переважна більшість дефектів не була безпосередньо пов’язана з функціональними вимогами. Але при переході на Scrum стратегію значно зросла кількість помилок різної серйозності, а особливо критичних та блокуючих, це пов’язано з тим, що зміна таблиці у базі даних, компоненту або додавання нової функціональності можуть сильно впливати на всю систему. Варто звернути увагу, що більшість помилок при використанні даної стратегії була пов’язана не з помилками програмістів, а зі швидкою зміною вимог, які не документувались належним чином.

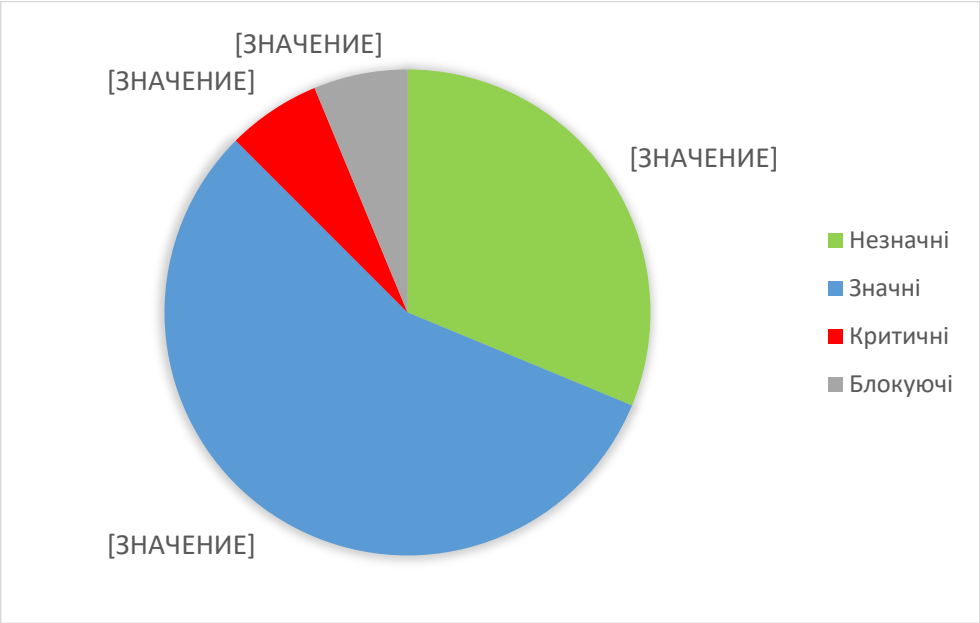


Рис. 2.13. Розподіл дефектів по серйозності при водоспадній стратегії

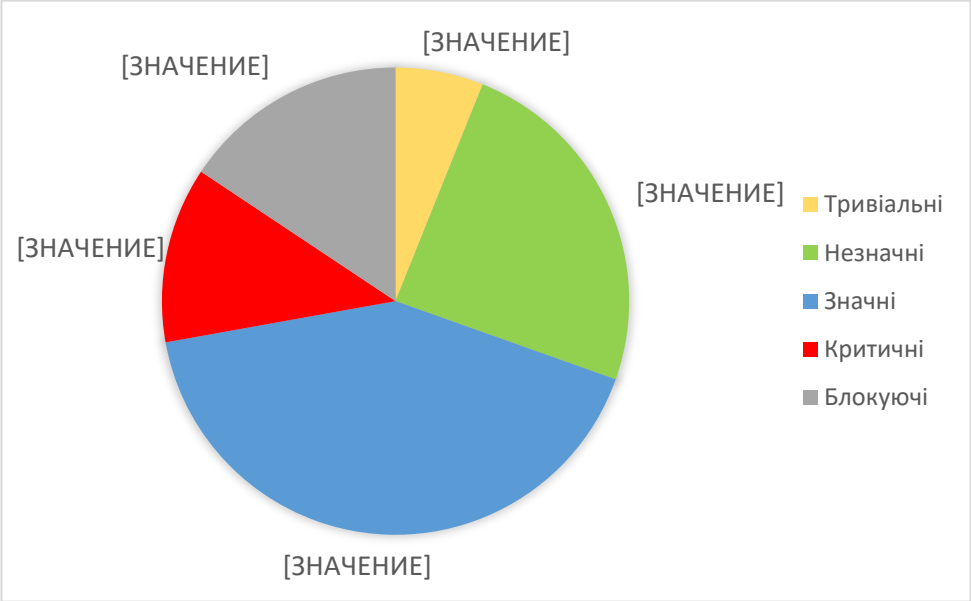


Рис. 2.14. Розподіл дефектів по серйозності при Scrum стратегії

Для обґрунтування подальших висновків, за допомогою базових та похідних метрик, у тому числі формул 1.1 – 1.6, розраховано основні показники та наведено у таблиці 2.1.

Таблиця 2.1 – Метрики ПЗ

Метрика	Стратегія	
	Waterfall	Scrum

Кількість знайдених дефектів	16	115
Відсоток дефектів від загальної кількості завдань	25,4%	42,8%
Кількість змінених/доданих/видалених вимог	0	45
Коефіцієнт стабільності вимог	0	0,25
Кількість відхилених дефектів	2	9
Коефіцієнт відхилених дефектів	12,5%	7,83%
Кількість прийнятих дефектів	14	106
Кількість виправлених дефектів	14	100
Коефіцієнт "витоку" багів	0%	4,58%
Кількість критичних дефектів	1	14
Відсоток критичних дефектів	6,25%	12,17%
Ефективність тестування	87,5%	95,42%
Кількість перевідкритих багів	1	22
Коефіцієнт перевідкриття багів	6,25%	10,44%
Кількість реалізованих основних функцій	3	6
Кількість реалізованих вимог	35	115
Відсоток реалізованих вимог від загальної кількості	23,33%	76,67%
Кількість перевірок у чеклисті	80	130
Кількість виявлених дефектів з чеклисту	–	10
Кількість API тестів	75	192
Кількість виявлених дефектів з тестування API	–	8
Покриття вимог тестами	86,11%	178,89%
Час виявлення дефектів	1-3 дні	до 1 години

Слід пам’ятати, що на виявлених дефектів впливає відповідність тестових зусиль вимогам ПЗ. При збільшенні відсотку покриття програми тестами збільшується і кількість знайдених дефектів. Це пояснюється тим, що більша кількість тестів охоплює більше функціональних аспектів програми і перевіряє їх на відповідність очікуваному поведінці та може виявити непередбачувані проблеми та неочікувані взаємодії між компонентами системи. З табл. 2.1 можна побачити, що цей відсоток значно зріс при переході на Scrum стратегію. Це забезпечило якісне опрацювання чеклисту та API тестів, тобто зросла кількість перевірок та тестів, і відповідно ефективність тестування.

Для з’ясування продуктивності кожної зі стратегій, розглянемо їх окремо. Порівняння каскадної та гнучкої стратегій наведено у таблиці 2.2.

Таблиця 2.2 – Порівняльна характеристика стратегій

Опис	Стратегія	
	Waterfall	Scrum
Процес розробки вимог та тестування	Лінійний	Ітеративний
Управління директивне	Так	Ні
Комунікація між аналітиками/тестувальниками	За допомогою документації	Регулярні зустрічі
Простота розуміння та використання	Висока	Середня
Детермінованість вимог	Усі	Основні
Детальність та підтримка тестової документація	Висока	Мінімальна
Стан вимог впродовж проєкту	Статичний, дозволені мінімальні зміни	Гнучкий, вимоги можуть змінюватись
Повернення до попередніх процесів	Небажане та малоймовірне	Постійне
Задоволеність клієнта	Висока	Вище середнього
Співпраця команди і замовника	Низька	Вище середнього
Чіткість та послідовність процесів	Висока	Нижче середнього
Ефективність тестування	Вище середнього	Висока
Стабільність задач	Висока	Нижче середнього

Стратегія на основі Waterfall методології забезпечила те, що усі вимоги, описані в документації були повними, зрозумілими, коректними та перевірюваними, тобто усі вони були детермінованими та взагалі не змінювались впродовж застосування даної стратегії, завдяки цьому критерії прийняття кожного завдання були чітко зрозумілими для команди. Стабільність вимог, відсутність внесення до них змін забезпечили низьку загальну кількість дефектів, і в тому числі критичних та перевідкритих. Простота розуміння та використання була високою, управління директивним, а загальний процес лінійним. Оскільки стабільність вимог і задач була високою, коефіцієнт

перевідкритих дефектів є мінімальним, таким чином розробники не виконували зайву роботу та не повертались до вже завершених завдань. При цьому ефективність тестування склала 87,5%, так як 2 з 16 дефектів були вирішені вже протягом третього тижня, а покриття вимог тестами становило 86,11%, що є нижчим показником у порівнянні зі стратегією Scrum, та свідчить про те, що стратегія тестування була менш продуктивною. Але варто зазначити, що докладна документація забезпечила ефективну комунікацію між командою та в подальшому облегшила роботу з вимогами, коли вони змінювались при використанні Scrum стратегії. Отже, дана стратегія була більш ефективною саме для розробки вимог, ніж для тестування.

Стратегія на основі Scrum методології показала себе краще у процесі забезпечення якості програми. Відсоток покриття вимог тестами становить 178,89%, що у два рази більше у порівнянні з попередньою стратегією, а загальна ефективність тестування – 95,42%. При цьому дефекти знаходили швидше за допомогою API тестування та чеклисту. Оскільки співпраця команди і замовника була тісною, швидка та іноді безконтрольна зміна вимог збільшили відсоток перевідкриття дефектів та коефіцієнт відхилених дефектів, так як вимоги змінювались швидше, ніж розробники встигали виконувати завдання.

Це свідчить про те, що для забезпечення якісного результату, задоволеності клієнта, може стати поєднання цих двох стратегій. Тобто використання стратегії Waterfall для розробки вимог і проєктування системи на їх основі, оцінки вартості всього проєкту і його планування. Після цього варто з обережністю та серйозністю ставитись до зміни вимог, оскільки це у багатьох випадках призводить до розповзання бюджету, визначених часових меж та постійного перероблення завдань. Зате гнучке, швидке та якісне тестування системи впродовж усього циклу існування проєкту найкраще може забезпечити стратегія Scrum.

2.5. Порівняльна характеристика стратегій

2.5.1. Переваги, недоліки та сфера застосування стратегій

Після глибшого вивчення стратегій Waterfall і Scrum за допомогою певного проєкту можна зробити висновки щодо продуктивності кожної з них. Керівники та команди проєктів зобов’язані прийняти обґрунтоване рішення під час впровадження певної стратегії для своєї організації, водночас пам’ятаючи про характер і тип проєкту та вимоги зацікавлених осіб. Але замість того, щоб зациклюватись на певній стратегії, важливо зосередитись на досягненні найкращого можливого результату для проєкту.

Фундаментальна відмінність між двома стратегіями розробки вимог та контролю якості на основі Scrum і Waterfall методологій, полягає в тому, що процеси гнучкої розробки орієнтовані на робочий продукт, що задовольняє усі побажання замовника, і мають на меті доставку продукту у найкоротші строки, тоді як процеси каскадної розробки орієнтовані на точне планування, оцінку ресурсів і дотримання графіка [15, 19]. У таблиці 2.3 наведені переваги кожної зі стратегій інженерії вимог та забезпечення якості ПЗ.

Таблиця 2.3 – Переваги стратегій

Waterfall	Scrum
Стратегія методична, етапи окреслені, що дозволяє учасникам команди чітко розуміти свої обов'язки та простіше вимірювати прогрес	Зосереджена на виправленні помилок, оскільки це сприяє прозорості та регулярному зворотному зв’язку з клієнтом
Вимоги чітко визначені, тому завдання залишаються максимально стабільними протягом усього процесу розробки	Команда і замовник співпрацюють, вимоги постійно переглядаються, це заохочує нові та інноваційні рішення
Детальний опис вимог та написання тестової документації забезпечують стійкість до змін кадрового потенціалу	Кожен етап процесу дає низку результатів, тому клієнтам не потрібно чекати до кінця, щоб побачити результат

Продовження табл. 2.3

Waterfall	Scrum
Тестування перед початком написання коду допомагає команді структурувати та використати свої найкращі практики	Спринти фіксовану тривалість і встановлені цілі, що дозволяє швидко отримувати працюючий функціонал
Забезпечує легкий контроль і прозорість для клієнта завдяки більш жорсткій системі звітності	Проводиться постійний огляд роботи і рефлексія, аналіз результатів, виявляються можливі покращення
Дата випуску готового продукту, а також його остаточна вартість можуть бути розраховані до розробки	Для випуску готового продукту зазвичай потрібно менше часу, порівняно з іншими стратегіями

Окрім переваг, важливо окреслити недоліки, які можуть завадити отриманню ефективного результату, при використанні тієї чи іншої стратегії. Негативні риси стратегій на основі Waterfall і Scrum методологій наведено у таблиці 2.4.

Таблиця 2.4 – Недоліки стратегій

Waterfall	Scrum
Раптова зміна вимог може зробити значну частину роботи виконаної роботи марною, що призведе до затримки кінцевого терміну	Постійні зустрічі між учасниками команди та з замовником можуть відволікати від виконання основної роботи
Участь замовника та користувачів у процесах розробки обмежена, тому інснує висока ймовірність виявлення критичних проблем на останніх етапах	Передбачає тривалі періоди інтенсивної роботи, усі члени команди повинні успішно виконувати власні завдання, але для цього потрібен достатній досвід
Програма не набуває свого робочого вигляду до останніх стадій існування проєкту	Відсутність певної кінцевої дати призводить до розповзання обсягу часу і фінансів
Для досягнення кінцевої мети може знадобитися більше часу порівняно з використанням гнучких методологій	Відсутність регулярного забезпеченя якості може серйозно вплинути на ефективність проєкту

Продовження табл. 2.4

Waterfall	Scrum
Необхідні кваліфіковані аналітики, здатні сформулювати прийнятні для продуктивної роботи специфікацію та технічне завдання	Оскільки не передбачається докладна документація, відсутність або зміна члена команди є великим ризиком

Якщо розглядати галузі розробки ПЗ загалом, а не конкретні проєкти, можна визначити сфери застосування обох стратегій (табл. 2.5).

Таблиця 2.5 – Сфера застосування стратегій

Назва стратегії	Доцільність та сфера застосування	Компанії, що використовують
Waterfall	Використовується для проєктів, де вимоги та процеси можна чітко визначити заздалегідь: ПЗ для медичної галузі, фінансового сектору, інженерних проєктів, урядових агенств, військового, космічного, авіаційного сектору.	Cisco, EPAM, IBM, Microsoft IT, SAP
Scrum	Доцільним є використання у тих проєках, де необхідна часте та швидке впровадження змін:мобільні, ігрові, веб-застосунки, стартапи та іноваційні проєкти	Unilever, Google, Facebook, Amazon, стартапи

Раніше розробка проєктів, з використанням каскадної моделі, наприклад системи CRM, системи керування ланцюгами поставок тощо, могла займала рік або довше. З розвитком технологій були випадки, коли великомасштабні корпоративні системи розроблялися протягом 2-3 років, але були надмірними, або навпаки неповними, на момент їх завершення. Навіть якщо програму було розроблено з використанням нової технології, такі фактори, як поява на ринку конкурентів, доступність дешевших альтернатив, краща функціональність із використанням новітніх технологій, зміна вимог клієнтів, збільшують ризик розробки програми протягом кількох років. Не зважаючи на це, у деяких галузях продовжують надавати перевагу саме Waterfall стратегії. Для розробки тих систем, де на кону стоїть людське життя, де збій програми може призвести

до однієї чи кількох смертей, де час і гроші є на другому плані, а безпека людини – на першому. У деяких країнах такі нещасні випадки можуть призвести до тюремного ув'язнення тих, хто несе відповідальність. Розвиток Міністерства оборони, військових і авіаційних програм використовує каскадну стратегію у багатьох організаціях. Це пояснюється суворими стандартами та вимогами, яких необхідно дотримуватися, у таких галузях вони відомі завчасно, а в контрактах чітко вказано результати проєкту. На даний момент стратегія Waterfall застосовується на проєктах, пов’язаних з банківською справою, охороною здоров'я, системами управління ядерними установками, космічними човниками [14, 23].

Стратегія Waterfall може бути ідеальним вибором для невеликих проєктів із чітко визначеними вимогами та мінімальними шансами на зміни, коли можна попередньо спланувати всі етапи розробки та побудувати лінійний процес. У тих випадках, коли проєкт має встановлені технологічні обмеження, бізнес-плавила та системні вимоги, що суттєво впливають на функціональні вимоги, а також для розробки ПЗ, коли час, бюджет та людські ресурси обмежені. Поряд з цим, для розробки складних та об’єктно-орієнтованих ПЗ ця стратегія не є хорошою, як і для довгострокових проєктів і тих, що мають помірний або високий ризик зміни вимог.

Стратегії на основі Scrum методології також охоплює велику кількість галузей: освіту, маркетинг, будівництво, страхування, виробництво. Важливо відзначити, що дана стратегія добре працює не тільки для розробки ПЗ у маленьких компаніях, а й у великих та багатонаціональних організаціях, таких як Google, Facebook, Amazon та багатьох інших. Наприклад Google забезпечує своєчасний аналіз вимог, оновлення своїх додатків у швидкоплинному світі, використовуючи гнучку методологію.

Стратегію на основі Scrum методології слід застосовувати у випадках, коли вимоги до ПЗ ще не повністю визначені або можуть змінюватися протягом розробки. Також більш доцільним є її використання в інноваційних проєктах, де важливо швидко прототипувати, тестувати та отримати зворотній зв'язок від

клієнтів, оскільки це дозволяє швидко експериментувати та вносити необхідні зміни у продукт. Ефективним буде використання з командами розробки ПЗ, які мають високий рівень самоорганізації та вміння приймати рішення щодо пріоритетів, розподілу завдань та управління власними процесами [20, 22].

2.5.2. Перспективні стратегії та подальші дослідження

У сфері стратегій розробки вимог вимог та контролю якості ПЗ існує кілька перспективних стратегій та напрямків досліджень, з них можна виділити User-centered Design та Lean Development.

Дизайн, орієнтований на користувача (User-centered Design) – стратегія, що дозволяє послідовно визначити потреби зацікавлених осіб та перетворити їх на продукт. Головною цінністю є концентрація на користувачі та тому, що відповідає його основним потребам. Згідно цієї стратегії цільова аудиторія продукту ділиться на групи, для кожної з яких визначається представник, що буде виступати посередником між зацікавленою стороною та компанією. Основним джерелом вимог є спостереження, інтерв'ю та масові опитування як користувачів, так і представників бізнесу. На основі з'ясованих ключових характеристик, потреб та поведінкових стереотипів, а також їх аналізу та зібраної статистики формуються функціональні та інші вимоги до системи. Створені на основі специфікації прототипи та дизайн тестуються реальними користувачами, а тест кейси та тестові сценарії складаються згідно потребам груп користувачів. Оскільки тестування сконцентроване на задоволенні вимог користувачів, докладна документація високорівневих документів може не передбачатись, але використовуються інші артефакти та застосовуються основні типи та рівні тестування, як і в класичних стратегіях [24].

Бережлива розробка (Lean Development, Lean Startup) базується на філософії Ощадливого виробництва (Lean Manufacturing) та Agile, спрямована на мінімізацію витрат та часу, необхідних для реалізації ПЗ. Основна ідея полягає в тому, щоб якомога швидше випустити продукт на ринок, дати

протестувати користувачам та внести необхідні зміни. На початку визначається тільки головна ідея, гіпотези про продукт і загальних вимоги, які перевіряються на підтверджуються реальними користувачами. На основі результатів тестування та інформації від користувачі проводиться аналіз даних та знаходяться шляхи покращення продукту. Функціональні вимоги до кожної нової версії ПЗ описуються в документації та редагуються впродовж розробки ПЗ, бізнес-вимоги, системні та інші можуть бути зафіксовані на початку та не змінюватись. Тестування за цією стратегією передбачає роботу в економічно обмеженому середовищі. Це означає, тестувальники працюють з обмеженими ресурсами, такими як бюджет та час, що вимагає максимальної ефективності та оптимізації [25].

У дані сфері продовжуються експерименти з метою поліпшення процесів, інструментів та методологій. Активні дослідження спрямовані на розробку інтелектуальних методів та алгоритмів для автоматизованого аналізу вимог. Це включає розпізнавання та розуміння натуральної мови, виявлення суперечностей та неповноти вимог, а також підтримку процесу прийняття рішень. Розробляються формальні методів, що допомагають перевіряти повноту та коректність вимог і забезпечують відповідність між вимогами та реалізацією ПЗ. Використання штучного інтелекту (ШІ) також може покращити ці процеси. ШІ може бути застосований для автоматичної генерації вимог на основі вхідних даних, таких як бізнес-правила, вимоги замовника або документація. Він може аналізувати код, вхідні дані та взаємодії системи для виявлення уразливостей або недоліків ПЗ.

Висновки

У даному розділі застосовано стратегії на основі Waterfall і Scrum методологій для розробки вимог та контролю якості фінансового застосунку. Використання Waterfall стратегії дозволило систематично визначити та задокументувати вимоги до програмного продукту перед початком розробки.

При дотриманні цього підходу було проведено детальний аналіз вимог, визначено послідовність етапів забезпечення якості та складено тестові документи. Перехід на Scrum стратегію дозволив швидко реагувати на зміни та вносити необхідні корективи під час розробки.

У результаті, застосування обох стратегій сприяло розробці програми для управління особистими фінансами з урахуванням всіх вимог і визначених показників якості. Але розраховані метрики проєкту та остаточна версія ПЗ дали можливість зробити висновок про те, що Waterfall стратегія краще підходить саме для фази роботи з вимогами, а Scrum стратегія для контролю якості розроблюваної системи.

Оскільки клієнти дедалі більше вимагають реалізації усіх своїх ідей при мінімальних часових та фінансових витратах, керівники проєктів та команди стикаються з проблемою підтримки високої якості ПЗ та доставки продукту. Використання сильних сторін кожної стратегії, одночасне пом’якшення їхніх слабких стороні, тобто використання гібридної стратегії, могло б стати більш ефективним рішенням при розробці не тільки проєкту SmartSpend, розглянутого у даному розділі, а й подібних проєктів. За цією стратегію передбачається проєктування, розробка вимог, дизайну з дотриманням традиційної методології водоспаду, що дає більше часу, для отримання схвалення зацікавлених сторін, перед повним зосередженням на розробці. Після цього можна визначати тривалість ітерацій, створювати резерви та виконувати всі завдання в рамках гнучкого підходу, щоб досягти остаточного вигляду ПЗ.

Перспективні стратегій свідчать про те, що подальші дослідження можуть знайти більш ефективні і інноваційні підходи для досягнення успіху.

ВИСНОВОК

Метою даної роботи було зробити порівняльний огляд ефективних стратегій інженерії вимог та контролю якості ПЗ та застосувати їх для розробки фінансового застосунку, щоб показати як вони можуть підвищити результативність проєктів, що розробляються, або бути впровадженими у майбутні проєкти.

Кожна зі стратегій, а саме Waterfall та Scrum, має певний набір характеристик та та підходить для реалізації проєктів різної спрямованості. Як і каскадна стратегія, так і гнучка, допоможуть у створенні практично будь-якого продукту. Проте, насамперед обирається та методологія, яка може максимально ефективно та якісно реалізувати проєкт. Якщо продукт універсальний, варто відштовхуватись від таких параметрів, як час, бюджет, кваліфікація команди та інших важливих критеріїв. Гнучка модель вимагає високого професіоналізму від виконавців, передбачає тривалі періоди інтенсивної роботи, зате забезпечує якісне ПЗ та ідеально підходить для стартапів. При водоспадній моделі для досягнення кінцевої мети може знадобитись значна кількість часу та кваліфіковані аналітики, здатні чітко зрозуміти побажання замовника, зате дана стратегія забезпечує легкий контроль і прозорість для клієнта та зрозумілість і послідовність для команди. Зацікавленим сторонам важливо провести детальний аналіз сильних та слабких сторін кожного підходу, врахувати поради експертів та визначити вимоги до конкретного проєкту. Тільки після цього можна прийняти обґрунтоване рішення щодо вибору оптимальної стратегії.

Обидва розглянутих підходи до організації та управління процесами розробки вимог та тестування ПЗ показали свою продуктивність. Тож на основі проведеного дослідження можна зробити висновок про ефективність Waterfall стратегії саме для стадії розробки вимог та Scrum стратегії для контролю якості продукту. Зроблено припущення про те, що в рамках одного проєкту можна оптимально поєднати ці дві стратегії, а також наведено огляд перпективних

стратегій, що може стати основою для подальших досліджень в цьому напрямку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Технології програмування та створення програмних продуктів: конспект лекцій /укладач О. В. Алексенко. – Суми : Сумський державний університет, 2013. – 133 с.
2. Despa M. Comparative study on software development methodologies. Database Systems Journal; 2014.
3. State of Agile Report. URL: https://stateofagile.com/
4. Extreme Chaos. The Standish Group International Inc. – The Report of the Standish Group International, 2005. – 12 p.
5. Соммервилл, Иан. -. Инженерия программного обеспечения, 6-е издание. : Пер. с англ. - М. : Издательский дом "Вильямс”, 2002. – 624 с. : ил.
6. K. Wiegers, J. Beatty. Software Requirements, 3rd edition, Microsoft Press, 2013, ISBN 978-0-7356-7966-5
7. Козак О.Л. Опорний конспект лекцій з курсу – Аналіз вимог до програмного забезпечення для студентів напрямку підготовки – Програмна інженерія / О.Л. Козак. – Тернопіль, 2011. – 56 с.
8. Сэм Канер. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бізнесприложений/ Сэм Канер, Джек Фолк, Енг Кек Нгуен; пер. с англ. – К.: Изд-во «ДиаСофт», 2001. – 544 с
9. Black, R., E. Van Veenendaal, Graham, D., (2012). Foundations of Software Testing: ISTQB Certification 3rd Edition. Hampshire, United Kingdom, Cengage Learning EMEA.
10. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. - М.: «Дело», 2007. - 312 с.
11. Myers G.J. The Art Of Software Testing / G.J. Myers - New York: John Wiley & Sons, Inc., 2004. - 254 p. - ISBN 0-471-46912-2.

12. Рекс Блэк. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование. - М.: «Лори», 2006. - 544 с.
13. Stioca M., Ghlic-Micu B., Mircea M., Uscatu C. Analyzing Agile Development –from Waterfall Style to Scrumban. Informatica Economică.2016. No4. C. 5-14
14. The Ultimate Guide to the Waterfall Model. URL: https://www.projectmanager.com/guides/waterfall-methodology
15. Waterfall Strategy: How to Release Singles Using the Waterfall Strategy. URL: https://blog.groover.co/en/tips/waterfall-strategy-release-singles/
16. What is the Waterfall Software Development Methodology?. URL: https://www.developer.com/project-management/sdlc-waterfall-model/
17. Cohn M. Agile Estimating and Planning / Mike Cohn. - Prentice Hall, 2005. - 368 p.
18. Schwaber, K. & Sutherland, J. (2017). The Scrum Guide. The definitive Guide to Scrum: The Rules of the Game
19. What is Scrum Strategy. URL: https://www.atlassian.com/agile/scrum
20. What Is Scrum in Project Management? URL: https://www.wrike.com/project-management-guide/faq/what-is-scrum-in-project-management/
21. 25 Scrum Process Best Practices that Set Your Agile Workflow for Efficiency. URL: https://www.altexsoft.com/blog/business/25-scrum-process-best-practices-that-set-your-agile-workflow-for-efficiency/
22. Scrum vs. Waterfall: Which Methodology Is Right for Your Project? URL: https://www.float.com/resources/scrum-vs-waterfall/
23. Crossing a Gantt Chart With a Kanban Board: Agile-Waterfall Hybrid in Project Management. URL: https://www.teamly.com/blog/agile-waterfall-hybrid-project-management/
24. User-Centered Design – The Design Process and Tools. URL: https://www.cuelogic.com/blog/user-centered-design

25. Everything You Need to Know about Lean Software Development. URL: https://railsware.com/blog/lean-software-development-guide/