Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __інфокомунікацій_____
(повна назва)
Кафедра _____інформаційно-вимірювальних технологій_____
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____

_____Забезпечення якості програмних засобів_____
_____протягом життєвого циклу_____
(тема)

Виконав:
студент _2_ курсу, групи ____ЗЯм-22-2_____
_____Яковенко А.В._____
(прізвище, ініціали)

Спеціальність __152 Метрологія та_____
___інформаційно-вимірювальна техніка_____
(код і повна назва спеціальності)
Тип програми ____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)
Освітня програма ___Забезпечення якості___
_____
( повна назва освітньої програми)

Керівник _____доц. Запорожець О.В._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ ____Захаров І.П._____
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет інфокомунікацій

Кафедра інформаційно-вимірювальних технологій

Рівень вищої освіти другий (магістерський)

Спеціальність 152 Метрологія та інформаційно-вимірювальна техніка
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Забезпечення якості
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

# ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Яковенко Андрію Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Забезпечення якості програмних засобів протягом життєвого циклу

затверджена наказом університету від 03 листопада 2023 р. № 1294 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 05 січня 2024 р.

3. Вихідні дані до роботи

1. Об'єкт дослідження: методи та принципи забезпечення якості ПЗ протягом усього життєвого циклу.

2. Мета дослідження: розробка рекомендацій щодо забезпечення та вимірювання показників якості на кожному етапі життєвого циклу ПЗ.

4. Перелік питань, що потрібно опрацювати в роботі

1. Проблеми забезпечення якості в індустрії програмного забезпечення.

2. Аналіз наявних теоретичних та практичних підходів щодо забезпечення якості ПЗ.

3. Рекомендації щодо забезпечення якості життєвого циклу ПЗ.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
___Слайди презентації кваліфікаційної роботи._____
_____
_____
_____
_____
_____
_____
_____
_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1 )

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз сучасного стану проблеми та | 06.11.2023 – 12.11.2023 | |
| | методів її вирішення | | |
| 2 | Формулювання проблем забезпечення якості ПЗ | 13.11.2023 – 20.11.2023 | |
| 3 | Аналіз теоретичних та практичних підходів | 21.11.2023 – 27.11.2023 | |
| | до забезпечення якості ПЗ | | |
| 4 | Розробка рекомендацій щодо забезпечення | 28.11.2023 – 10.12.2023 | |
| | якості ПЗ протягом життєвого циклу | | |
| 5 | Написання пояснювальної записки | 11.12.2023 – 23.12.2023 | |
| 6 | Виконання графічної частини | 24.12.2023 – 04.01.2024 | |
| 7 | Представлення закінченої кваліфікаційної | 05.01.2024 | |
| | роботи на кафедру | | |
| | | | |

Дата видачі завдання _6_ ___листопада___ 2023 р.

Студент _____
(підпис)

Керівник роботи _____ ___доц. Запорожець О.В._____
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка до магістерської кваліфікаційної роботи містить 118 сторінок, 17 рисунків, 19 формул, 1 додаток, перелік посилань з 48 назв.

ЯКІСТЬ, ЗАБЕЗПЕЧЕННЯ ЯКОСТІ, ПРОГРАМНИЙ ЗАСІБ, МОДЕЛЬ ЯКОСТІ, ХАРАКТЕРИСТИКА ЯКОСТІ, ЗАЦІКАВЛЕНА СТОРОНА

Об’єкт дослідження – методи та принципи забезпечення якості ПЗ протягом усього життєвого циклу.

Мета роботи – розробка рекомендацій щодо забезпечення та вимірювання показників якості на кожному етапі життєвого циклу ПЗ.

У кваліфікаційній роботі було проведено аналіз теоретичних та практичних тенденцій з забезпечення якості протягом життєвого циклу ПЗ.

Визначені, розроблені та запропоновані рекомендації щодо забезпечення, оцінки та вимірювання (якщо застосовно) якості для кожного етапу життєвого циклу ПЗ.

# ABSTRACT

The explanatory note to the master's qualification thesis contains 118 pages, 17 figures, 19 formulas, 1 appendix, a list of references from 48 titles.

QUALITY, QUALITY ASSURANCE, SOFTWARE, QUALITY MODEL, QUALITY CHARACTERISTICS, STAKEHOLDER

The object of the study is the methods and principles of software quality assurance during the entire life cycle.

The purpose of the document is to develop recommendations for ensuring and measuring quality indicators at each stage of the software life cycle.

In the qualification document an analysis of theoretical and practical trends in quality assurance during the software life cycle has been carried out.

Guidelines have been identified, developed and proposed for ensuring, evaluating and measuring (if applicable) quality for each phase of the software life cycle.

# ЗМІСТ

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ЗМІ – засоби масової інформації

КУП – кодекс усталеної практики

ПЗ – програмний засіб

ТМ – торгівельна марка

AI – штучний інтелект (artificial intelligence)

ConOps – концепція операцій (Concept of Operations)

FR – функціональні вимоги (functional requirements)

LOC – рядки коду (lines of code)

MOEs – міри ефективності (Measures of Effectiveness)

MVP – мінімально життєздатний продукт (minimum viable product)

NFR – нефункціональні вимоги (non-functional requirements)

NGOs – потреби, цілі та задачі (Needs, Goals, and Objectives)

PSR – продуктивність, масштабованість і надійність (Performance, Scalability and Reliability)

# ВСТУП

В теперішні часи для задоволення різноманітних потреб бізнесу, фізичних осіб, державних структур та некомерційних організацій здебільшого використовуються комп'ютерні системи, в яких переважну частку складають програмні засоби. Для реалізація цілей і задач усіх зацікавлених сторін потрібно використовувати високоякісні програмні засоби та системи, що є важливою ланкою у виробництві матеріальних цінностей, наданні послуг, а також у запобіганні можливим негативним наслідкам.

В останні десятиріччя чітко простежується тенденція підвищення вимог до нефункціональних аспектів програмних засобів (наприклад, usability, availability, мобільність, тощо). Причиною цього процесу з одного боку є те, що суспільство стає більш освіченим і більш вимогливим, а з іншого боку бурхливий розвиток індустрії розробки програмних засобів та виникнення транснаціональних корпорацій, основною діяльністю яких є розробка програмних засобів або надання електронних послуг (наприклад, Google, Microsoft, Meta), зробило ринок програмного забезпечення надто конкурентним. В результаті чого вимоги до усіх аспектів якості програмного забезпечення наразі високі як ніколи, наприклад, неможливість входу у Facebook на протязі декількох годин для широкого кола користувачів кілька років тому була основною новиною тижня та сильно вдарила по репутації компанії Meta.

В цій роботі буде розглянуто, яким чином можна забезпечити якість на кожному з етапів життєвого циклу ПЗ – проведено аналіз наявної ситуації з точки зору теорії та практики, та запропоновані заходи щодо забезпечення та, за можливості, оцінки якості для кожного конкретного етапу життєвого циклу ПЗ.

# 1 ПРОБЛЕМИ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ В ІНДУСТРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

На сьогоднішній день тяжко уявити будь-яку діяльність людини без використання інформаційних технологій та систем. Невід'ємною частиною яких є програмне забезпечення, що дозволяє на базі майже однакових (або дуже схожих) апаратних засобів отримати величезну кількість різноманітних програмних систем, сервісів, додатків тощо. Індустрія розробки програмного забезпечення є доволі молодою та динамічною галуззю, де нові проривні технології з'являються часто та регулярно, що призводить до кардинальних змін у механізмах розробки, тестування та моніторингу. Завдяки цьому розробка програмного забезпечення стає дедалі все більш швидшою та ефективнішою, але зворотною стороною медалі є недостатня визначеність щодо забезпечення якості.

Для сучасних програмних продуктів високого рівня, що обробляють великі обсяги інформації, є дуже складними та складаються з багатьох модулів (наприклад, Twitter передає ~500млн повідомлень щодня) підхід, коли якість програмного продукту забезпечується лише тестуванням є не лише застарілим, занадто повільним та дорогим, але й цілковито шкідливим.

Додатковою проблемою стає той факт, що розробка кожного окремого програмного продукту повинна сприйматися як новий проєкт в термінах індустріального виробництва. Але на відміну від останнього, розробка програмного забезпечення є надто гнучкою та динамічною, щоб до неї можна було застосовувати формалізовані методи забезпечення якості подібні до тих, що використовуються на промислових виробництвах.

Одна команда розробників може паралельно розробляти декілька проєктів, використовуючи різні методи організації виробництва на кожному з них чи навіть застосовувати різні моделі для різних етапів життєвого циклу одного й того самого ПЗ! Нормальною вважається ситуація, коли команда розробників поєднує два протилежні agile-методи (scram та kanban) при розробці проєкту.

Через велику різноманітність та варіативність проєктів, завдань та проблем дуже складно та майже неможливо використовувати статистичні методи, які дуже поширені у інших галузях промисловості. Наявні реалізації таких методів у розробці ПЗ оперують так званими «папугами» – ідеалізованими та дуже відносними одиницями виміру завдань на кшталт «маленька», «середня» та «велика» чи «не складна», «складна» та «дуже складна».

До того ж через надзвичайно вузькі часові рамки, зазвичай надані для реалізації проєкту, система забезпечення якості повинна бути з одного боку ефективною, а з іншого достатньо гнучкою, щоб її можна було з мінімальними змінами застосовувати до різноманітних проєктів, а не кожен раз розробляти нову. Та достатньо простою та легкою, щоб не обтяжувати собою процес розробки та не впроваджувати зайві формальності.

Очевидним рішенням є впровадження забезпечення якості в процеси на усіх етапах життєвого циклу програмного забезпечення. Поєднання процесного підходу з ризик-орієнованим мисленням задля своєчасного виявлення наявних проблем та ситуацій, що можуть призвести до них на наступних етапах розробки, тестування або використання, зарекомендувало себе з найкращої сторони. Проактивний підхід до виявлення проблем та якості в цілому дозволяє суттєво знизити як витрати фінансів, часу та інших ресурсів, так і репутаційні втрати, що є особливо важливим для бізнесу сьогодення.

Також в поточних практиках та тенденціях простежуються спроби винести перевірки у початкові етапи життєвого циклу ПЗ, щоб виявити проблеми якомога раніше. Це дозволяє значно знизити витрати на усунення таких проблем, чи взагалі попередити їх.

# 2 АНАЛІЗ НАЯВНИХ ТЕОРЕТИЧНИХ ТА ПРАКТИЧНИХ ПІДХОДІВ ЩОДО ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПЗ

## 2.1 Оцінка якості бізнес ідеї та/або проблеми

У світовій історії є багато прикладів, коли геніальні та інноваційні, на перший погляд, ідеї не були реалізовані через погану пропрацьованість. Тому дуже важливо ще на початковому етапі провести оцінку бізнес-ідеї.

Бізнес-ідея – це чітко виражена концепція комерційних способів та/або цілей, реалізація яких може дозволити отримати фінансову вигоду від надання цінності продукту чи послуги кінцевим споживачам [4].

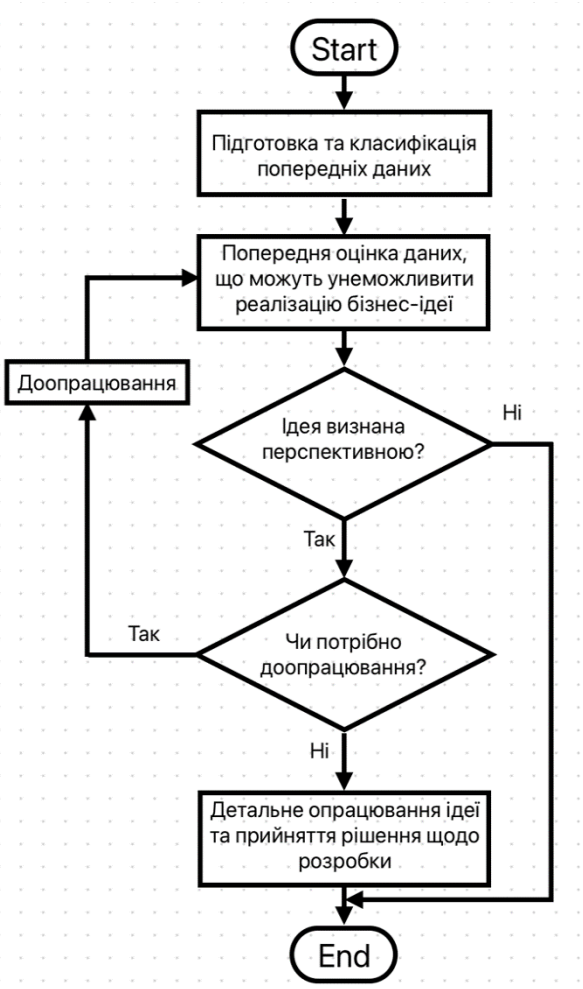


Рисунок 2.1 - Алгоритм проробки бізнес-ідеї

Загалом, кожна ідея вимагає індивідуальних досліджень та опрацювання через свою унікальність, але у цьому підрозділі будуть розглянуті лише спільні моменти з точки зору якості самої ідеї та рівня її опрацьованості та наведені можливі методи та заходи щодо оцінки цього рівня. В загальному випадку слід використовувати алгоритм, наведений на рис 2.1.

### 2.1.1 Актуальність та терміновість

На самому початку аналізу слід зрозуміти, наскільки взагалі новий продукт цікавий споживачам та ринку, а також наскільки терміново це потрібно.

Лише продукт чи послуга, що відповідає сучасним запитам (в ідеальному варіанті передбачає ці запити) та потребам користувачів або містить гарантований спосіб генерації таких потреб, є актуальним.

У найкращому становищі зазвичай є перший у своєму роді продукт або послуга для конкретної потреби.

### 2.1.2 Реалістичність та здійсненність

Реалізація бізнес-ідеї має бути економічно та практично виправданою, тому вона повинна спиратися на об'єктивні закони світобудови, соціальні та юридичні норми, а також існуючі або перспективні технології. З останніми потрібно бути дуже обережними, тому що вони можуть розроблятися да доводитися до ладу досить багато часу.

Задля розуміння об’єктивної можливості реалізації проєкту треба проаналізувати та скласти високорівневий список учасників проєкту, ресурсів та технологій, що потрібно залучити для успішної реалізації.

Також потрібно провести попередню оцінку витрат, пов'язаних із реалізацією ідеї. Щоб розуміти потенційний прибуток (чи його наявність взагалі) в доступних для огляду часових рамках.

### 2.1.3 Унікальність та інноваційність

Сама ідея чи концепція продукту, послуги чи бізнесу в ідеалі повинна бути унікальною у своєму роді, але це не є обов’язковим – у сучасному світі досить багато успішних проєктів, які не є цілком унікальними.

Щоб отримати суттєві переваги, потрібно виділятися на тлі конкурентів. Навіть якщо бізнес-ідея сама по собі не нова, вона має бути унікальною у своєму роді – пропонувати значні покращення та вигоди для споживача, мати свої особливі відмінні якості, що кардинально її виділятимуть.

### 2.1.4 Конкретність та стратегія

Добра ідея повинна бути зрозумілою, мати чіткі межі та сформульовані цілі з прорахованими характеристиками, а кінцевий продукт чи послуга виражатися у конкретних та об'єктивних даних.

Також повинна бути визначена адекватна стратегія просування бізнес-ідеї.

### 2.1.5 Прибутковість

У найширшому сенсі проєкт повинен надавати якусь цінність для сторони, що його розробляє та впроваджує, тому що в сучасному світі більшість ідей для реалізації та підтримки потребують досить багато грошей.

У найбільш поширеному варіанті проєкти повинні бути фінансово привабливими. Зазвичай отримують інвестиції ті бізнес-ідеї, які можуть принести більшу прибутковість. Тому потрібно звертати увагу на наступні моменти:

- аудиторія та ринки – зазвичай продукти не є універсальними, які можуть однаково добре працювати усюди, тому треба розуміти та чітко визначитися з цільовою аудиторією та для яких ринків призначено продукт;
- обсяг ринків, де буде реалізовано ідею, його потрібно заздалегідь обрахувати у обсязі (штуках для продуктів, операціях для послуг) та грошах щоб зрозуміти, чи буде рентабельним запуск проєкту для цих ринків;
- потенціал для масштабування – бізнес-ідея повинна мати потенціал для розширення після старту проєкту як якісний так і кількісний, які додаткові пропозиції можна буде впроваджувати. По-перше, це дозволяє залучити більше інвестицій, по-друге зрозуміти план розвитку та розширення і по-третє – визначити потенційні обмеження

(які закладені у продукті чи послузі), що можуть надалі перешкодити масштабуванню;

- орієнтовні собівартість та витрати на реалізацію проєкту, які потрібно приблизно прорахувати на цьому етапі:
    - стартові витрати на розробку і запуск проєкту;
    - поточні витрати для підтримки проєкту;
    - джерела стартового капіталу;
    - варіанти покриття фінансового розриву у період між впровадженням і виходом на стабільний прибуток;
- цінова стеля – скільки потенційний покупець спроможний та, особливо, готовий заплатити за подібний товар чи послугу;
- витрати на залучення клієнтів – середня оцінка витрат сил та ресурсів на залучення нових клієнтів та утримання вже існуючих;
- термін окупності.

Навіть у більш специфічних випадках для неприбуткових соціально-значущих проєктах, що фінансуються за рахунок держави чи якихось благодійних фондів, перевагу отримує проєкт з найменшими витратами та найбільшим позитивним ефектом.

### 2.1.6 Терміни реалізації

У сучасному світі надважливим є розуміння термінів реалізації та графіку виходу продукту, бо своєчасність появи продукту на ринку наразі є ключовим фактором. Дужа часто компанії роблять поступку щодо якості та/або обмежують функціонал заради швидшого виходу на ринок.

У разі довгої реалізації може статися так, що на момент релізу продукту чи послуги вони вже втратили свою актуальність взагалі чи очікування від них кардинально змінилися і потрібна додаткова розробка, а в найгіршому варіанті, на ринку вже з’явилися такі самі пропозиції.

### 2.1.7 Недоліки та негативні фактори ідеї

Необхідно окремо максимально неупереджено оцінити недоліки та негативні фактори самої ідеї, а також обставини, що можуть негативно позначитися на реалізації та впровадженні проєкту.

На цьому етапі слід на високому рівні оцінити потенційні ризики щодо усього життєвого циклу продукту, зрозуміти які з них є критичними, як можна зменшити їх негативний вплив, чи можна якісь із них просто прийняти або взагалі ігнорувати.

### 2.1.8 Дослідження та джерела інформації

На початковому етапі задля виявлення слабких сторін ідеї рекомендується провести попередні обговорення з експертами, потенційними клієнтами та іншими фахівцями, які можуть виявитися корисними в конкретній області та провести деякі з наступних досліджень:

- маркетингові дослідження;
- збір матеріалів по ринку;
- технологічні дослідження;
- аналіз патентів, ТМ та розробок;
- математичне моделювання;
- оцінка потенціалу ринку;
- демографічні дані;
- юридична експертиза;
- та інші, залежно від конкретної ідеї.

А використання різних джерел інформації дозволяє побачити нові можливості та вигоди бізнес-ідеї:

- відгуки потенційних споживачів продукту чи послуги;
- поради та аналітика експертів у цій та дотичних галузях;

- аналіз конкурентних продуктів чи послуг;
- рекомендації потенційних нішевих продавців та маркетологів;
- архіви та реєстри патентних бюро;
- технічні експерти у відповідній сфері;
- форуми та соціальні мережі;
- поглиблений тематичний пошук в інтернеті (бази знань, бізнес-клуби, тощо);
- поточний моніторинг ринку, цін, конкурентів, ЗМІ;
- юридичні спеціалісти (необхідність ліцензій, дозволів та ін.);
- будь-які інші джерела, які є актуальними для конкретної бізнес-ідеї.

Основною задачею таких оцінок та досліджень є перевірити спроможність ідеї в цілому. Через те, що кожне з вищеперерахованого потребує чимало зусиль і часу, потрібно заздалегідь пріоритезувати дослідження і перевірки та обмежити витрачений на них час.

### 2.2 Очікування та вимоги щодо ПЗ

Згідно ДСТУ ISO 9000:2015 основним результатом роботи організації є додавання цінності через задоволення потреб і очікувань замовників й інших відповідних зацікавлених сторін. А якість продукції та послуг визначають здатністю задовольняти замовників, а також передбаченим і непередбаченим впливом на відповідні зацікавлені сторони.

Тому дуже важливо визначити та пріоритезувати усі зацікавлені сторони та їхні очікування від ПЗ.

### 2.2.1 Зацікавлені сторони

Зацікавлена сторона; причетна сторона (interestedparty, stakeholder) – особа чи організація, яка може вплинути на рішення чи діяльність, піддана впливу, чи сприймає себе такою, що піддана впливу рішень або діяльності [1].

Важливо розуміти, що стейкхолдер – це носій певної ролі, завдяки якій він впливає на компанію або проєкт. Тому важливо вибирати зацікавленою стороною не людину чи організацію, а виконувану функцію [41].

Внутрішні зацікавлені сторони – це групи або люди, інтереси яких до проєкту чи компанії виникають через прямі стосунки – працевлаштування, власність чи інвестиції, тобто менеджери, співробітники, власники тощо.

Зовнішні зацікавлені сторони – це групи поза бізнесом або люди, які не працюють всередині бізнесу, але на них певним чином впливають рішення та дії бізнесу. Прикладами зовнішніх зацікавлених сторін є клієнти, постачальники, кредитори, місцева громада, суспільство та уряд [7].

Для кожної організації надважливим є визначити стейкхолдерів, задля розуміння середовища, ще на початку збирання бізнес-вимог щодо продукту чи послуги. Для цього визначаються критичні зацікавлені сторони, виявляються їх коротко- та довгострокові інтереси, потенційний вплив та прогнозується, як реалізація проєкту може вплинути на них – позитивно чи негативно. Потрібно усвідомлювати, що це поняття не зосереджується лише на замовнику – важливо враховувати всі відповідні зацікавлені сторони.

Відповідні зацікавлені сторони – це ті, з якими пов’язано значний ризик для сталості організації, якщо їхні потреби та очікування не буде виконано. Організації визначають, які результати необхідно подати цим відповідним зацікавленим сторонам, щоб знизити цей ризик [1].

ДСТУ ISO 9000:2015 наводить деякі можливі ключові вигоди при вірному визначенні відповідних стейкхолдерів:

- підвищена дієвість організації та її відповідних зацікавлених сторін завдяки врахуванню можливостей і обмежень, пов’язаних з кожною зацікавленою стороною;
- спільне розуміння цілей і цінностей стейкхолдерами;
- збільшена спроможність створювати цінність для зацікавлених сторін завдяки спільному користуванню ресурсами та знаннями, а також керуванню ризиками, пов’язаними з якістю;
- належно-керований ланцюг постачання, який забезпечує стабільний потік продукції та послуг;
- запроваджування спільної діяльності щодо розвинення та поліпшування з постачальниками, партнерами та іншими зацікавленими сторонами;
- заохочування та визнавання поліпшення та досягнення постачальників і партнерів.

Немає стандартного переліку стейкхолдерів, бо він буде змінюватися в залежності від сфери та цілей проєкту, специфіки роботи в ньому та інших факторів. Та для більшості випадків зацікавленими сторонами є:

- власники бізнесу, акціонери, інвестори;
- керівництво та команда;
- замовники, клієнти, покупці, споживачі;
- постачальники;
- органи влади та держструктури;
- громадські групи.

### 2.2.2 Користувачі

Користувачі є однією з зацікавлених сторін, що будуть безпосередньо контактувати з ПЗ, тому дуже важливо визначити усі види користувачів, їх ролі та базові сценарії використання продукту.

Користувач (user) – особа або група осіб, яка взаємодіє з системою або отримує користь від застосування системи [2].

Згідно ДСТУ ISO/IEC 25010:2016 однією з трьох моделей якості є модель якості під час застосування, що вимірює людино-машинну систему загалом, складовою частиною якої є людина, тобто користувач.

Цей документ наводить три типи користувачів:

- основний користувач — особа, яка взаємодіє з системою, щоб досягти основних цілей;
- побічні користувачі, що виконують підтримку, наприклад:
    - провайдер контенту, менеджер/адміністратор системи, менеджер з безпеки;
    - фахівець з обслуговування, аналітик, фахівець з перенесення (портування), установник;
- опосередкований користувач – особа, яка отримує результати, але не взаємодіє з системою.

Якість під час застосування (quality in use) – ступінь, в якому продукт або систему можуть застосовувати певні користувачі у певних умовах, щоб задовольнити потреби у досягненні певних цілей із результативністю, економічною ефективністю, свободою від ризику та задоволеністю [2].

### 2.2.3 Метод вимірювання цінності інновацій, запропонованих користувачем

Сьогодення онлайн технології надають широкі можливості спілкування, обміну ідеями та створення спільнот за інтересами, що включають в себе людей з різних куточків земної кулі. Наразі все більше і більше бізнес залучає спільноти користувачів до багатьох аспектів продукту – генерації ідей, розробки та тестування, позиціювання продукту та управління прибутками. Jingjing Yang у своїй статті «A Framework of User Classification Model of Online User Innovation Communities Based on User Innovation Value» посилаючись на de Jong, J. P., &

Flowers, S. (2018) вказує, що 23% ідей щодо вдосконалення продуктів було висунуто користувачами відповідно до їх власних потреб.

Очевидно, що не усі користувачі можуть однаково ефективно генерувати ідеї, робити твердження та впливати на спільноту. У цій статті наводиться метод вимірювання цінності інновацій, запропонованих користувачем (User Innovation Value Measurement Method), що базується на вимірюванні активності кожного окремого користувача у онлайн спільноті, реакції на його або її пости, коментарі та активність загалом.

Розроблена Jingjing Yang модель спирається на ідею, що інноваційна цінність користувача стосується не лише на його здатності генерувати інноваційні ідеї, але також має комплексно враховувати професійні здібності, здатність впливати та активність інших користувачів:

- професійні здібності відображають запас експертних знань користувача щодо продуктів/послуг (лише користувач із високим досвідом у певному або широкому діапазоні знань може висунути цінні інноваційні ідеї);
- можливість впливу відображає популярність та вплив користувача на інших користувачів у спільноті (користувачі з великим впливом, як правило, добре відомі більшості людей і мають високе лідерство та привабливість у спільноті);
- активність відображає загальний рівень активності користувача в спільноті (активні користувачі можуть безперервно вливати життєві сили та вносити знання в спільноту).

У цій статті також наводяться метрики, які дозволяють кількісно визначити наступні значення для кожного користувача спільноти.

#### 2.2.3.1 Інноваційний потенціал

Кількість інновацій (innovation quantity, $I_{nn}C_N$) базується на кількості інноваційних публікацій, опублікованих користувачем. Можна використовувати

такі ключові слова, як «пропозиція», «доповнення», «покращення» та інші слова, щоб визначити, чи є це інноваційна публікація, і узагальнити такі інноваційні публікації.

Якість інновацій (innovation quality, $I_{nn}C_Q$) – ступінь визнання офіційними представниками проєкту ($X_1$), та іншими користувачами спільноти ($X_2$):

$$I_{nn}C_Q = \alpha_1 X_1 + \alpha_2 X_2, \quad (2.1)$$

де $\alpha_1$ і $\alpha_2$ – вагові коефіцієнти визнання офіційними представниками та іншими користувачами спільноти проєкту відповідно.

Інноваційний потенціал (innovation capability, $I_{nn}C$) означає здатність користувача активно впроваджувати інновації в онлайн-спільноті та отримувати визнання в цьому процесі. Чим більша здатність до інновацій, тим більше інноваційних дій користувачі виконують у спільноті, і тим більша ймовірність, що вони будуть визнані. Він вимірюється з точки зору кількості та якості інновацій:

$$I_{nn}C = \beta_1 I_{nn}C_N + \beta_2 I_{nn}C_Q = \beta_1 I_{nn}C_N + \beta_2(\alpha_1 X_1 + \alpha_2 X_2), \quad (2.2)$$

де $\beta_1$ і $\beta_2$ – вагові коефіцієнти кількості та якості інновацій відповідно.

### 2.2.3.2 Професійні навички

Глибина знань (knowledge depth, $K_D$) характеризує ступінь досвіду та глибину знань користувача, що відображається в глибині знань користувача про кілька продуктів або тем. Його можна виміряти середнім показником професіоналізму кожного допису (post's professionalism, $P_P$)

$$K_D = AVG(P_p) = AVG(\frac{N_k}{N_n}), \quad (2.3)$$

де $N_k$ – кількість професійних слів;

$N_n$ – кількість слів-ознак після шумових слів або стоп-слів (тобто кількість іменників).

Широта знань (knowledge breadth, $K_B$) визначає багатство та диверсифікацію знань, накоплених користувачем, що відображається в широті знань про продукт і темах, які охоплює користувач. Чим у обговоренні більшої кількості областей продукту приймає участь користувач, тим ширше його знання про продукт, тим більший обсяг його знань. Цей показник вимірюється кількістю різних тем, у яких бере участь користувач.

Професійні навички (professional capability, $PC$) стосуються здатності користувача виражати свої накопичені професійні знання в громадській діяльності. Коли користувач бере участь у різних темах спільноти, він показує свої професійні знання. Цю здатність можна виміряти глибиною та широтою знань, якими володіє користувач

$$PC = \beta_3 K_D + \beta_4 K_B, \tag{2.4}$$

де $\beta_3$ і $\beta_4$ є ваговими коефіцієнтами глибини та широти знань відповідно.

#### 2.2.3.3 Здатність впливати

Ступінь посередництва (betweenness centrality) – це міра центральності у графі, заснована на найкоротших шляхах. Для будь-якої пари вершин у зв'язному графі існує щонайменше один (найкоротший) шлях між вершинами, для якого мінімальне або число ребер, якими шлях проходить (для незважених графів), або сума ваг цих ребер (для зважених графів). Ступінь посередництва кожної вершини дорівнює числу цих найкоротших шляхів через вершину [48].

Здатність впливати (influence capability, $I_{nfl}C$) – ступінь впливу користувача в мережах інтерактивних та соціальних відносин, що відображає позицію

користувача в мережах інтерактивних і соціальних відносин. Його можна виміряти ступенем посередництва між мережею інтерактивних відносин і соціальною мережею зв'язків.

Ступінь посередництва у мережі інтерактивних відносин (betweenness centrality of interactive relationship network, $I_{ib}$). Коли користувач бере участь у взаємодії спільноти, він формує мережу інтерактивних відносин з іншими користувачами. Ступінь посередництва у мережі інтерактивних відносин може пояснити ступінь впливу користувача в мережі інтерактивних відносин.

Ступінь посередництва у мережі соціальних відносин (betweenness centrality of social relationship network, $I_{sb}$). Коли користувач встановлює прямі соціальні стосунки з іншими користувачами в спільноті, він сформує мережу інтерактивних стосунків з іншими користувачами. Ступінь посередництва у мережі соціальних відносин пояснює рівень впливу користувачів серед друзів.

Здатність впливати вимірюється за допомогою наступної формули:

$$I_{nfl}C = \beta_5 I_{ib} + \beta_6 I_{sb}, \tag{2.5}$$

де $\beta_5$ і $\beta_6$ є ваговими коефіцієнтами ступені посередництва у мережі інтерактивних та соціальних відносин відповідно.

#### 2.2.3.4 Активність

Активність (active capability, AC) – це те, наскільки активний користувач у спільноті онлайн-інновацій, виміряний загальною кількістю публікацій, відповідей і тривалості онлайн:

$$AC = \beta_7 N_P + \beta_8 N_R + \beta_9 N_D, \tag{2.6}$$

де $N_P$, $N_R$ та $N_D$ – кількість дописів, відповідей та загальний час проведений он-лайн, $\beta_7$, $\beta_8$ та $\beta_9$ – їх вагові коефіцієнти.

### 2.2.3.5 Розрахунок інноваційної цінності користувача

Базуючись на методі вимірювання цінності інновацій запропонованих користувачем (User Innovation Value Measurement Method), розраховуємо значення інноваційної цінності для кожного користувача (innovation value, IV):

$$VI = \omega_1\left(\beta_1 I_{nn} C_N + \beta_2(\alpha_1 X_1 + \alpha_2 X_2)\right) + \omega_2(\beta_3 K_D + \beta_4 K_B) + \omega_3(\beta_5 I_{ib} + \beta_6 I_{sb}) + \omega_4(\beta_7 N_P + \beta_8 N_R + \beta_9 N_D), \quad (2.7)$$

де $\omega_1$, $\omega_2$, $\omega_3$ і $\omega_4$ – вагові коефіцієнти інноваційного потенціалу, професійних навичок, здатності впливати та активності відповідно.

### 2.2.4 Очікування та вимоги

Національний стандарт ДСТУ ISO 9000:2015 надає наступне визначення: вимога (requirement) – сформульовані потреба чи очікування, загальнозрозумілі чи обов’язкові.

Примітка 1. «Загальнозрозумілі» означає, що є звичайною чи загальноприйнятою практикою для організації та зацікавлених сторін уважати потребу чи очікування, про які йдеться, самі собою зрозумілими.

Примітка 2. Установлена вимога – це вимога, сформульована, наприклад, у задокументованій інформації.

Примітка 4. Вимоги можуть ставити різні зацікавлені сторони чи сама організація.

Примітка 5. Для досягнення високого рівня задоволеності замовника може бути потрібним задовольнити очікування замовника, навіть якщо воно не є сформульованим, загальнозрозумілим чи обов’язковим.

У ДСТУ ISO/IEC 25010:2016 наведено наступну діаграму щодо визначення та аналізу вимог зацікавлених сторін (рис. 2.2).

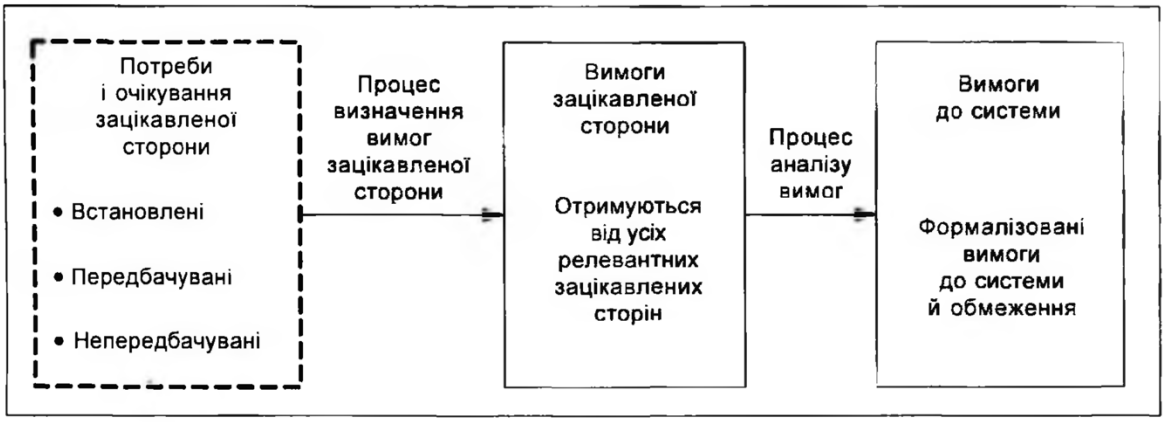


Рисунок 2.2 – Визначення і аналіз вимог зацікавлених сторін

### 2.2.4.1 Вхідні дані задля визначення очікувань та вимог

Зазвичай для процесу визначення очікувань зацікавлених сторін потрібні щонайменше наступні вхідні дані [9]:

- початкові очікування клієнта – це потреби, цілі, задачі, бажання, можливості та інші обмеження, отримані від клієнта щодо продукту в межах рівня продукту. Для продуктів найвищого рівня (остаточний кінцевий товар) це очікування вихідного клієнта, який запитував продукт. Для кінцевого продукту в межах рівня продукту це очікування одержувача кінцевого продукту під час переходу;
- очікування інших зацікавлених сторін – це очікування ключових зацікавлених сторін, крім клієнта. Наприклад, такими зацікавленими сторонами може бути команда випробувань, яка отримуватиме переведений продукт (кінцевий продукт і допоміжні продукти), або інструктори, які навчатимуть операторів або менеджерів, відповідальних за продукт на цьому рівні;
- вимоги клієнта до нижнього рівня – це будь-які вимоги, які передаються або розподіляються з вищого рівня (тобто батьківські вимоги). Вони допомагають визначити очікування клієнта на цьому рівні.

Ретельне розуміння очікувань зацікавлених сторін щодо проєкту/продукту є одним із найважливіших кроків у процесі проєктування систем, що допомагає гарантувати, що всі сторони мають однакове розуміння та що продукт, задовольнить клієнта. Взаємно погодження щодо функцій, характеристик, поведінки, зовнішнього вигляду та продуктивності продукту, встановлює більш реалістичні очікування з боку замовника та допомагає запобігти суттєвим розповзанням вимог пізніше в життєвому циклі.

### 2.2.4.2 Очікування, їх основи та обов'язковість виконання

Слід розуміти основи, на яких формуються очікування: цінності, інформація, досвід чи особистий інтерес [10]:

- очікування, засновані на цінностях, ґрунтуються на ідеалах або нормах;
- очікування, засновані на інформації, залежать від інформації, яка є (або недоступна);
- очікування, засновані на досвіді, випливають з прямого або опосередкованого попереднього досвіду;
- очікування, засновані на особистих інтересах, –це очікування, на які впливає те, що вважається заслуженим або бажаним.

Laura Olkkonen у «Stakeholder Expectations. Conceptual Foundations and Empirical Analysis» зазначає, що в емпіричному аналізі очікуванням надається чотири рівні залежно від уявлення про те, на що вони спрямовані:

- мінімальна прийнятність (обов'язкові очікування, must expectations) – мінімальний рівень, який повинен бути;
- ймовірність (ймовірнісні очікування, will expectations) – ймовірний рівень, який, ймовірно, буде;
- нормативні надії та побажання (бажані очікування, should expectations) – нормативний рівень того, що має бути;

- ідеальні можливості (можливі очікування, could expectations) – ідеальний рівень того, що може бути.

Цей рівень впливає на те, як оцінюється виконання очікування, оскільки порушення обов'язкового очікування навряд чи буде допустимим, тоді як виконання можливих очікувань, як правило, не очікується.

Також у цій роботі зроблено акцент на очікуваннях у контексті відносин між організацією та зацікавленими сторонами або, точніше, на оцінці, яка впливає на кінцеву форму очікування, якщо розглядати її з точки зору організації. Вона залежить від довіри до організації, тобто сприйнятого бажання та здатності організації забезпечувати результати, які цінуються зацікавленими сторонами.

Тут слід розуміти, що хоча цінності, інформація, досвід та особисті інтереси можуть впливати на формування очікувань, вони належать до різних процесів, які не є окремими, а вбудованими: цінності та інтереси відносно статичні й не залежать від жодної конкретної організації, тоді як інформація та досвід впливають на оцінку, коли вона застосовується до конкретної організації.

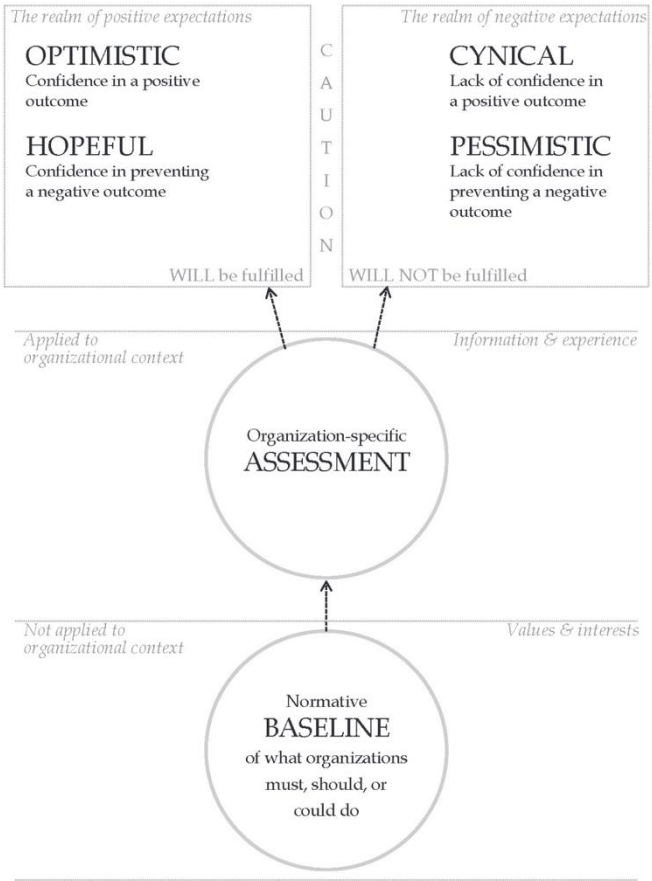


Рисунок 2.3 – Модель формування очікувань стейкхолдерів[10]

Відокремлення імовірнісного від обов'язкового, необхідного та можливого пов'язано з вкоріненістю різних фаз формування очікувань: «must», «should» і «could» стосуються нормативних (статичних) оцінок, пов'язаних із цінностями та інтересами, тоді як «will» є прогнозною, а отже, завжди пов'язана з оцінкою, специфічною для організації.

На рис 2.3 наведено модель формування очікувань у контексті відносин між організацією та зацікавленою стороною. Він пов’язує різні основи очікувань з різними фазами формування очікувань – цінності та інтереси з нормативною основою, а інформацію та досвід – з прогнозною оцінкою. У результаті нормативної та прогнозної фаз зацікавлені сторони можуть сформувати оптимістичні, повні надії, цинічні чи песимістичні очікування, які формують позитивні та негативні сфери очікувань зацікавлених сторін. Якщо зацікавлені сторони не впевнені у своїй довірі до організації, вони, швидше за все, матимуть обережні очікування, які не є ні позитивними, ні негативними, такі очікування існують між позитивною та негативною сферами.

Формування очікувань починається з того, що Laura Olkkonen називає базовою лінією очікувань (нижня частина рис. 2.3). Базовий рівень залежить від цінностей та інтересів, а в контексті відносин між організацією та зацікавленими сторонами він стосується сприйняття, ставлення та думки зацікавлених сторін щодо того, що організації повинні, мають та можуть робити. «Must» представляє мінімальні вимоги, «should» – це те, що вважається розумним і можливим, а «could» – ідеальний рівень. Базовий рівень очікувань є відносно статичним і не застосовується до жодної конкретної організації.

Очікування змінюється під впливом інформації та досвіду, коли воно застосовується до конкретної організації та відносин з цією організацією (центр рис. 2.3). Це коли прогностичний елемент «will» впливає на оцінку на основі наявної інформації, а також попереднього досвіду. Ця оцінка є фундаментальною для очікувань з точки зору організацій, оскільки вона визначає, чи буде очікування позитивним чи негативним.

Прогностична оцінка веде до фактичного очікування (верхня частина рис. 2.3). Позитивне очікування може бути оптимістичним, коли очікується, що організація досягне позитивного результату, або обнадійливим, коли очікується, що організація запобіжить негативному результату. Негативне очікування може бути або цинічним, коли очікується, що організація не зможе досягти позитивного результату, або песимістичним, коли очікується, що організація

принесе негативний результат. Ці різні форми очікувань містять позитивні та негативні сфери очікувань зацікавлених сторін. Між цими сферами є зона обережності, яка виникає, коли зацікавлені сторони не впевнені, чи можуть вони довіряти готовності та здатності організації або забезпечити позитивний результат, або запобігти негативному результату. Різниця між цинічними та песимістичними очікуваннями незначна, оскільки обидва вони вказують на брак впевненості. Те ж саме стосується оптимістичних і повних надії очікувань; різниця полягає в тому, наскільки активною сприймається роль організації, тобто чи організація активно сприяє чомусь позитивному, чи просто запобігає чомусь негативному.

#### 2.2.4.3 Механізм потреб, цілей та задач

NGOs забезпечують механізм, який гарантує, що всі (реалізатор, замовник та інші зацікавлені сторони) погоджуються на початку проєкту щодо визначення проблеми, яку потрібно вирішити, та її масштабу.

Добре написані NGOs забезпечують чітке відстеження від потреб, потім до цілей, а потім до завдань. Наприклад, якщо дана ціль не підтримує потреби або задача не підтримує ціль, вона не повинна бути частиною інтегрованого набору NGOs. Ця відстежуваність допомагає переконатися, що команда дійсно надає те, що потрібно.

Наведемо наступні визначення потреб, цілей та задач [9]:

- потреба – одне твердження, яке керує всім іншим. Воно повинно стосуватися проблеми, яку система повинна вирішити, але не бути рішенням. Висловлення потреби в однині. Спроба задовольнити більше ніж одну потребу вимагає обміну між ними, що легко може призвести до невиконання принаймні одного, а можливо, і кількох очікувань зацікавлених сторін;
- цілі – розробка потреби, яка становить певний набір очікувань для системи. Цілі стосуються критичних питань, визначених під час

оцінки проблеми. Цілі не обов'язково мають бути в кількісній чи вимірній формі, але вони повинні дозволити оцінити, чи досягла їх система;

- задачі – конкретні цільові рівні результатів, яких система повинна досягти. Кожна задача має бути пов'язана з певною ціллю. Загалом цілі мають відповідати чотирьом критеріям:
    - мають бути достатньо конкретними задля забезпечення чіткого керівництва, щоб розробники, клієнти та тестувальники їх зрозуміли. Вони повинні бути спрямовані на досягнення результатів і відображати те, що система повинна зробити, але не вказувати, як реалізувати рішення;
    - повинні бути вимірними, кількісно визначеними та такими що можна перевірити. Проєкт потребує моніторингу успіху системи в досягненні кожної мети;
    - мають бути агресивними, але досяжними, складними, але доступними, а цілі мають бути реалістичними. Цілі «Визначити пізніше» (To Be Determined, TBD) можуть бути включені, доки не будуть проведені комерційні дослідження, концепції операцій не зміцняться або технологія не зріє. Цілі повинні бути здійсненними до того, як будуть написані вимоги та розроблені системи;
    - мають бути орієнтованими на результати, зосереджуючись на бажаних кінцевому продукті та результатах, а не на методах, які використовуються для досягнення цілі (що, а не як).

#### 2.2.4.4 Рекомендації щодо бізнес-вимог

Бізнес-вимоги визначають бізнес-необхідність, потреби і критерії успіху проєкту, зважаючи на цільову аудиторію. Вони на високому рівні описують важливість проєкту, проблеми що вирішуються, деталізують його бенефіціарів,

користувачів, їх очікування, цілі та які переваги вони отримають, визначають час і місце та критерії оцінки прогресу та успіху проєкту. Бізнес-вимоги не пояснюють, як створювати проєкт – вони підкреслюють загальні результати проєкту і не обговорюють функціональні деталі.

Добре організовані та точно визначені вимоги допомагають зрозуміти, як вимоги можуть сприяти зростанню організації та зменшити ймовірність провалів проєкту через погано визначені вимоги. Точне уявлення про вимоги також може допомогти створити бізнес-кейс, документи бачення, статут проєкту, визначення обсягу проєкту та план проєкту.

Загальні рекомендації щодо вимог розглянуті у додатку А.

Бізнес-вимоги можуть включати [36]:

- ключові цілі та визначення проблеми;
- переваги запропонованого рішення;
- обсяг проєкту;
- правила, норми та політики;
- ключові особливості проєкту (наприклад, що проєкт запропонує користувачам);
- функції продуктивності та безпеки;
- метрики для вимірювання успіху проєкту.

Бізнес-вимоги не повинні включати наступне [36]:

- деталі вимог інших типів щодо системи;
- деталі впровадження вимог інших типів;
- деталі того, як впроваджувати політики та правила.

## 2.3 Функційні та нефункційні вимоги

Функціональні вимоги (functional requirements, FR) – це вимоги до рішень, тобто засоби надання ефективного рішення, яке відповідає бізнес-вимогам і

очікуванням для цього проєкту, вони є конкретними та вузькими й написані з точки зору системи (що повинна робити система) [29].

Функціональні вимоги:

• пояснюють, як досягти бажаних результатів і цілей;

• деталізують рішення проблеми, що потрібно вирішити.

Нефункціональні вимоги (non-functional requirements, NFR) – це інший тип вимог до рішення, у вигляді набору специфікацій, що описують робочі можливості, властивості, пояснюють обмеження та стримуючі фактори, а також намагаються покращити функціональність продукту, ще їх називають атрибутами якості.

Виділяють атрибути важливі для будь-яких зацікавлених сторін, тобто зовнішні нефункціональні вимоги [11, 12, 24, 29]:

- доступність – запланований час доступності (uptime), протягом якого система доступна для використання і повністю працездатна;
- ефективність – показник того, наскільки ефективно система використовує продуктивність процесора, місце на диску, пам'ять або смугу пропускання з'єднання;
- продуктивність – наскільки швидко програмна система або її окрема частина реагує на певні дії користувачів за певного навантаження;
- гнучкість – показує, з якою легкістю в продукт вдається додати нові можливості;
- цілісність, безпека та конфіденційність – включає безпеку, пов'язану з блокуванням неавторизованого доступу до системних функцій, запобіганням втраті інформації та захистом конфіденційності і безпеки даних, введених в систему;
- здатність до взаємодії або сумісність – показує, яким чином система обмінюється даними або сервісами з іншими системами;

- надійність – наскільки ймовірно, що система або її елемент працюватимуть без збоїв протягом заданого періоду часу за заздалегідь визначених умов;
- стійкість до збоїв – рівень, до якого система продовжує коректно виконувати свої функції, не дивлячись на не вірне введення даних, недоліки підключених програмних компонентів, компонентів устаткування або несподівані умови роботи;
- локалізація – наскільки добре система або її елемент відповідає контексту місцевого ринку. Контекст включає місцеві мови, закони, валюти, культури, правопис та ін;
- зручність використання – наскільки легко користуватися системою, зазвичай оцінюється за п’ятьма параметрами:
    - можливість навчання;
    - ефективність;
    - запам'ятовуваність;
    - як часто користувачі роблять помилки?
    - чи дизайн приємний у використанні?
- можливість одночасного доступу користувачів до системи – яку кількість одночасно працюючих користувачів з однаковими чи різними ролями може витримувати система;
- термін зберігання даних – період, протягом якого система буде зберігати введені дані.

Також атрибути важливі для команди проєкту, тобто внутрішні нефункціональні вимоги [11, 12, 24, 29]:

- легкість внесення змін та ремонтопридатність – наскільки зручно виправляти помилки або модифікувати ПЗ, він залежить від того, наскільки просто розібратися в роботі ПЗ, змінювати його і тестувати, і тісно пов'язано з гнучкістю і тестованістю;

- легкість переміщення або портативність – зусилля, необхідні для переміщення ПЗ з одного операційного середовища в інше;
- можливість повторного використання коду або внутрішніх модулів – показує зусилля, необхідні для перетворення програмних компонентів; з метою їх використання в інших застосуваннях;
- тестованість або перевірюваність – показує легкість, з якою програмні компоненти або інтегрований продукт можна перевірити на предмет дефектів;
- масштабованість – оцінює найвищі навантаження, за яких система все ще відповідатиме вимогам продуктивності;
- конфігурованість – вимоги щодо можливості зміни конфігурації системи;
- обслуговуванність – вимоги щодо можливості виконувати технічне обслуговування системи;
- технічні обмеження – внутрішні обмеження системи:
    - вимоги до мереж, серверів, мобільних засобів;
    - обсяг доступної пам'яті;
    - процесорного часу;
    - дискового простору;
    - пропускну здатність мережі.

В залежності від типу продукту різні атрибути мають різне значення, наприклад [24]:

- для вбудованих систем:
    - ефективність;
    - надійність;
    - безпека;
    - легкість і простота встановлення та обслуговування;
- для інтернет-застосувань і застосувань для мейнфреймів:
    - доступність;

- цілісність;
- легкість в експлуатації;
- масштабованість;

- для настільних систем:
  - здатність до взаємодії;
  - зручність;
  - простота використання.

Загальні рекомендації щодо вимог розглянуті у додатку А, але при формуванні NFR слід зважати на наступні принципи [29]:

- вимоги краще встановлювати до компонентів системи, а не до цілих продуктів, щоб уникнути марних витрат ресурсів на маловикористовувані компоненти;
- зв'язок NFR із бізнес-цілями – потрібно розуміти, чи є необхідним дотримання чи реалізація тієї чи іншої вимоги з точки зору бізнесу та прибутку;
- обмеження через залучення третіх сторін – іноді при використанні third-party неможливо досягти очікуваних характеристих через обмеження сторонніх систем;
- врахування архітектурних обмежень – кожна архітектура накладає певні обмеження на системи, реалізовані з її використанням – цих обмежень не можна уникнути.

Визначаються наступні переваги формулювання на використання нефункціональних вимог [29]:

- гарантія дотримання правових норм;
- визначення атрибуту якості програмного забезпечення;
- забезпечення належного рівня надійності, доступності, продуктивності і масштабованості програмного забезпечення;
- побудова політики безпеки системи;

- забезпечення хорошої взаємодії з користувачем, легкість роботи з програмним забезпеченням і мінімізація фактору витрат;
- покращення загальної якості системи;
- підвищення задоволення користувачів;
- краще узгодження з бізнес-цілями;
- зменшення кількості доопрацювань;
- покращення масштабованості і адаптивності системи;
- краще технічне обслуговування системи.

На жаль, окрім переваг NFR мають наступні недоліки[29]:

- може впливати на різні підсистеми програмного забезпечення високого рівня;
- збільшення вартості, оскільки вони вимагають особливої уваги на етапі архітектури програмного забезпечення/проєктування високого рівня;
- труднощі з визначенням усіх вимог;
- труднощі з прогнозуванням майбутніх вимог;
- проблеми вимірювання та тестування;
- великі часові витрати, які можуть уповільнити процес розробки;
- ризик зміни вимог – нефункціональні вимоги можуть змінюватися з часом, що може призвести до плутанини;
- конфліктні вимоги – нефункціональні вимоги можуть конфліктувати одна з одною, і може бути важко збалансувати їх та визначити пріоритети для виконання;
- непередбачені вимоги – нефункціональні вимоги можуть бути не повністю визначені на етапі збору вимог, а деякі вимоги можуть бути виявлені лише після розгортання системи.

Ключова відмінність між FR та NFR полягає в тому, що система не може функціонувати, не задовольняючи всім FR, але вона дасть бажаний результат, навіть якщо вона не задовольняє NFR.

## 2.4 Планування. Процесний підхід та врахування ризиків

Стандарти ISO серії 9000 визначають принципи процесного підходу та ризик-орієнтованого мислення ключовими для забезпечення якості. Розглянемо концепції цих принципів, що встановлені у зазначених вище документах у контексті планування.

### 2.4.1 Процесний підхід

Суть процесного підходу полягає в тому, що кожен співробітник забезпечує життєдіяльність конкретних бізнес-процесів, безпосередньо беручи участь в них. Обов’язки, область відповідальності, критерії успішної діяльності для кожного співробітника сформульовані і мають сенс лише в контексті конкретного завдання або процесу.

При побудові процесно-орієнтованої системи основний упор робиться на опрацювання механізмів взаємодії в рамках процесу як між структурними одиницями всередині компанії, так і з зовнішнім середовищем, тобто з клієнтами, постачальниками і партнерами. Саме процесний підхід дозволяє врахувати такі важливі аспекти бізнесу, як орієнтація на кінцевий продукт та зацікавленість кожного виконавця в підвищенні якості кінцевого продукту.

При процесному підході організація сприймається як мережа бізнес-процесів, яка є сукупністю взаємопов’язаних і взаємодіючих бізнес-процесів, що включають всі функції, виконувані в підрозділах організації.

Загалом виділяють наступні принципи [3]:

- діяльність компанії розглядається як сукупність бізнес-процесів;
- у кожного бізнес-процесу є внутрішній або зовнішній клієнт і власник (особа, яка відповідає за результат бізнес-процесу);
- кожен бізнес-процес характеризується ключовими показниками, що описують його виконання, результат або вплив на результат діяльності організації в цілому;

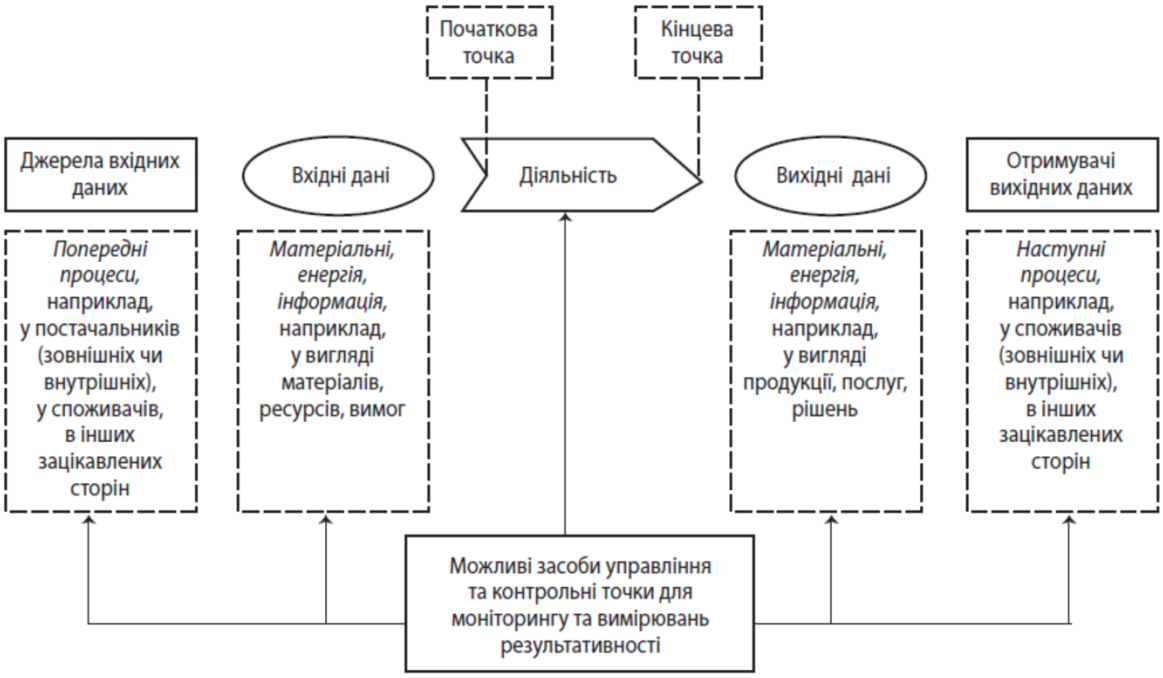


Рисунок 2.4 – Схема одиничного процесу [3]

Переваги процесного підходу [3]:

- чітка система взаємних зв'язків всередині процесів і в відповідних підрозділах;
- чітка система єдиноначальності – один керівник зосереджує в своїх руках керівництво всією сукупністю операцій і дій, спрямованих на досягнення поставленої мети і отримання заданого результату;
- швидка реакція виконавчих процесних підрозділів на зміну зовнішніх умов;
- в роботі керівників стратегічні проблеми домінують над оперативними;
- критерії ефективності і якості роботи підрозділів і організації в цілому узгоджені і спрямовані в одному напрямку.

Недоліки процесного підходу [3]:

- підвищена залежність результатів роботи організації від кваліфікації, особистих і ділових якостей рядових працівників і виконавців;

- управління змішаними в функціональному сенсі робочими командами – більш складне завдання, ніж управління функціональними підрозділами;
- наявність в команді декількох чоловік різної функціональної кваліфікації неминуче призводить до деяких затримок і помилок, що виникають при передачі роботи між членами команди, проте втрати тут значно менше, ніж при традиційній організації робіт, коли виконавці підпорядковуються різним підрозділам компанії.

### 2.4.2 Цикл PDCA (Демінга)

Цикл PDCA (Plan-Do-Check-Act) – концепція постійного циклічного покращення якості процесів завдяки зменшенню варіацій і виключенню причин, які порушують стабільність процесів при виконанні 4-х етапів робіт: планування – виконання – перевірка – дія. Даний цикл був запропонований В. Шухартом і розвинутий у окрему концепцію Е. Демінгом [3].


Рисунок 2.5 – Цикл PDCA [3]

Цикл PDCA може бути стисло описано так [3]:

1) плануй – установлюй цілі системи та її процеси, а також ресурси, потрібні для отримання результатів відповідно до вимог замовників і політик організації, а також передбачай ризики та можливості;

2) виконуй – упроваджуй те, що заплановано;

3) перевіряй – здійснюй моніторинг і там, де це можливо, вимірюй процеси та отримані в результаті продукцію та послуги, зважаючи на політики, цілі, вимоги та заплановані роботи, а також звітуй про результати;

4) дій – за потреби вживай заходів для поліпшування дієвості.

Плюси моделі PDCA [21]:

- простий у використанні;
- може допомогти зменшити втрати та підвищити ефективність;
- покращує продукти, людей і бізнес-процеси;
- зменшує повторювані помилки в процесі роботи;
- універсальний і може використовуватися для:
    - управління проєктами;
    - управління змінами;
    - розробки продукту;
    - управління якістю.

Мінуси циклу PDCA можуть включати [21]:

- модель PDCA займає багато часу і може не працювати належним чином у разі термінових проблем;
- вимагає залучення всіх, один керівник чи працівник не може правильно впровадити цю модель;
- повторюваний процес, він корисний лише при повторному використанні – якщо його застосувати лише один раз, це може не дати задовільних результатів.

### 2.4.3 Ризик-орієнтоване мислення

Успішна діяльність підприємств у невизначеному середовищі завжди пов’язана з виникненням ризиків та можливими втратами, що негативно впливають на діяльність. Дані умови актуалізують потребу в певному інструменті попереджувальної системи управління, який би дозволив найбільш раціональним способом врахувати ризик та мінімізувати можливі втрати, адекватно реагуючи на зміни. З цієї точки зору ризик-менеджмент має займати ключову позицію в структурі сучасного підприємства, як один з найбільш перспективних механізмів забезпечення стійкого та прибуткового

функціонування підприємств. Це є об’єктивно необхідною задачею, вирішення якої вимагає розробки практичних рекомендацій щодо створення підсистеми ризик-менеджменту з метою забезпечення стійкості підприємства в умовах невизначеності.

Стандарт ДСТУ ISO 9001:2015 установлює вимоги для організації щодо розуміння свого середовища та визначання ризиків як основи для планування. Це відображає застосування ризик-орієнтованого мислення для планування та запровадження процесів забезпечення якості і сприятиме у визначенні обсягу задокументованої інформації.

Хоча стандарт зазначає, що організація повинна планувати дії стосовно ризиків, немає вимоги щодо формалізованих методів керування ризиками або задокументованого процесу керування ризиками. Організації можуть самі вирішувати, чи потрібно розробляти ширшу методологію керування ризиками, ніж цього вимагає цей стандарт [3].

Завдяки використання комплексу з процесного підходу, циклу PDCA та зосередженості на ризик-орієнтованому мисленні дозволяє [3]:

- розуміти та постійно задовольняти вимоги;
- розглядати процеси з погляду створення додаткових цінностей;
- досягати результативного функціонування процесів;
- поліпшувати процеси на основі оцінювання даних та інформації.

## 2.5 Якість. Моделі якості. Якість потягом життєвого циклу ПЗ

Якість системи – це ступінь, у якому система задовольняє встановлені та передбачувані потреби різних зацікавлених у ній сторін [2].

Стандарти SQuaRE встановлюють три моделі якості, які покривають усі характеристики якості, що є важливими для широкого кола зацікавлених сторін, взаємозв’язок моделей показано на рисунку 2.6.

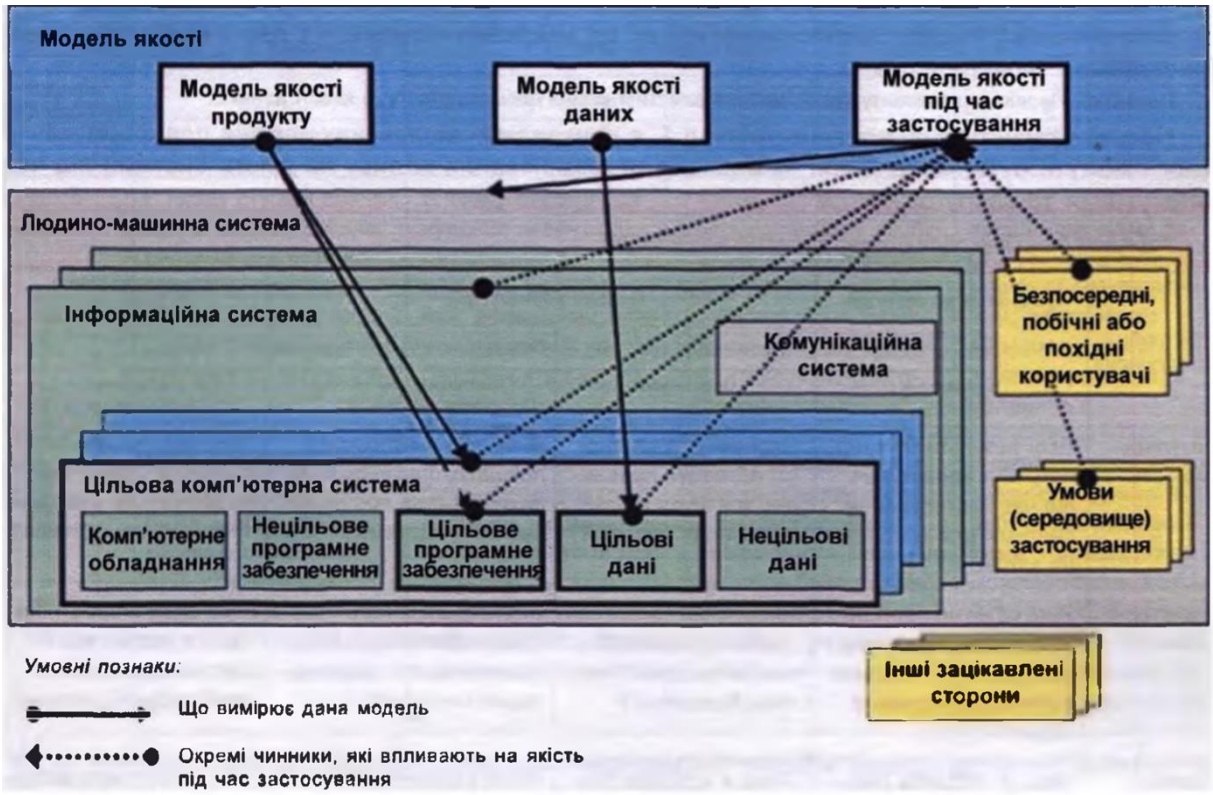


Рисунок 2.6 – Моделі якості та їх цільові об’єкти [2]

### 2.5.1 Модель якості даних

Міжнародний стандарт ISO/IEC 25012:2008 (гармонізований національний стандарт ДСТУ ISO/IEC 25012:2016 не є опублікованим, тому його неможливо використовувати) наводить наступні характеристики в моделі якості даних:

- Accuracy (точність, достовірність)
- Completeness (повнота, завершеність)
- Consistency (послідовність, системність)
- Credibility (достовірність, надійність)
- Currentness (актуальність)
- Accessibility (доступність, безбар’єрність)
- Compliance (відповідність)
- Confidentiality (конфіденційність)
- Efficiency (ефективність, оперативність)
- Precision (точність, чіткість)

- Traceability (простежуваність)
- Understandability (зрозумілість)
- Availability (доступність, працездатність)
- Portability (мобільність, переносність)
- Recoverability (відновлюваність)

### 2.5.2 Модель якості продукту

Модель якості продукту згідно ДСТУ ISO/IEC 25010:2016 містить наступні характеристики та підхарактеристики:

- Функційна придатність:
  - Функційна повнота;
  - Функційна коректність;
  - Функційна доцільність;
- Ефективність виконання:
  - Реактивність;
  - Застосовність ресурсів;
  - Потенційність можливостей;
- Сумісність;
  - Співісновність;
  - Взаємодійність;
- Зручність застосування:
  - Визнаність відповідності;
  - Опановність;
  - Керовність;
  - Захищеність від помилок користувача;
  - Естетичність інтерфейсу користувача;
  - Доступність;
- Надійність:

    - Завершеність;
    - Готовність;
    - Відмовостійкість;
    - Відновність;
- Захищеність:
    - Конфіденційність;
    - Цілісність;
    - Неспростовність;
    - Обліковність;
    - Автентичність;
- Супроводженість:
    - Модульність;
    - Повторна застосовність;
    - Аналізовність;
    - Модифікованість;
    - Тестовність;
- Мобільність:
    - Адаптовність;
    - Інстальовність;
    - Замісність.

### 2.5.3 Модель якості під час застосування

Національний стандарт ДСТУ ISO/IEC 25010:2016 містить такі характеристики та підхарактеристики для моделі якості під час застосування:

- Результативність;
- Ефективність;
- Задоволеність:
    - Корисність;

- Довірчість;
- Приємність;
- Комфортність;

- Свобода від ризику:
  - Зменшення економічного ризику;
  - Зменшення ризику для здоров’я та безпеки;
  - Зменшення екологічного ризику;
- Покриття контексту:
  - Повнота контексту;
  - Гнучкість.

### 2.5.4 Вимірювання якості програмного продукту

Характеристики якості за допомогою цих моделей розподіляються на категорії характеристик з підхарактеристиками, завдяки такій ієрархічній структурі досягається зручний розподіл якості продукту.

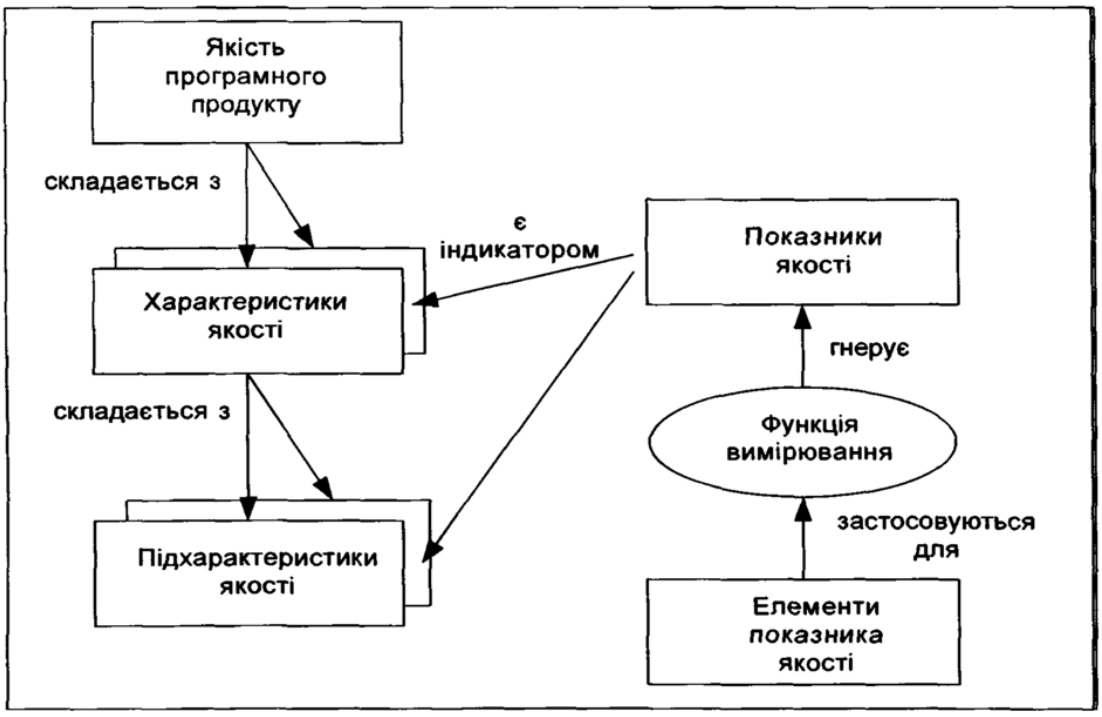


Рисунок 2.7 – Рамкова модель вимірювання якості програмного продукту [2]

У разі коли характеристику чи підхарактеристику неможливо виміряти безпосередньо, визначають властивості, що її цілком покривають (окрема властивість може визначати більше ніж одну характеристику чи підхарактеристику). Щоб визначити кількісно ту чи іншу властивість, застосовують відповідний метод вимірювання, що є логічною послідовністю операцій, а в результаті його застосування отримується елемент показника якості. Отримані показники якості комбінують за допомогою спеціального алгоритму, що називають функцією вимірювання, на виході якої отримують показники якості. Таким чином кожну характеристику чи підхарактеристику можна кількісно виразити за допомогою функцій вимірювання, показниками якості (може застосовуватися більше ніж один показник якості).

### 2.5.5 Взаємозв'язок властивостей та показників якості

Користувачі стикаються із ПЗ під час його використання, тобто їх різноманітні потреби потрібно враховувати у моделі якості під час застосування (різні групи користувачів очікувано можуть використовувати систему у різні способи або користуватися різними її функціями). Визначивши ці потреби, їх можна використовувати задля формулювання показників зовнішньої та внутрішньої якості через характеристики та підхарактеристики якості. На рисунку 2.8 наведений взаємозв'язок між різними рівнями властивостей та показників якості.

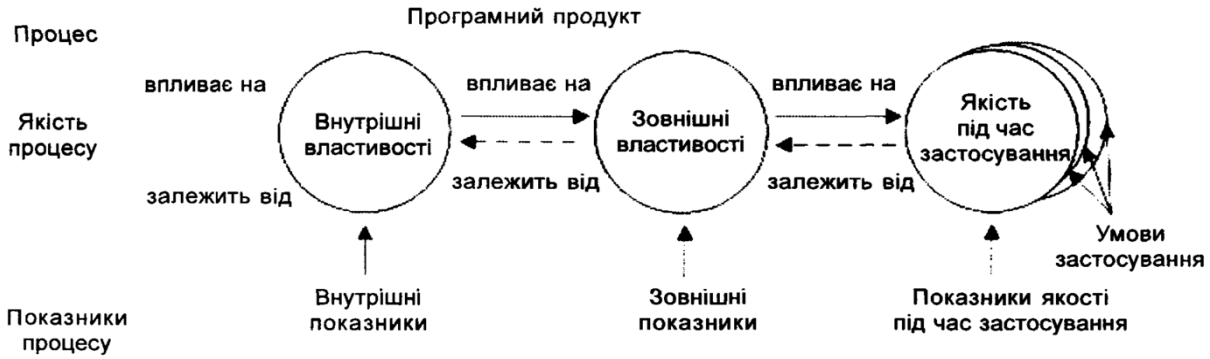


Рисунок 2.8 – Взаємозв'язок властивостей та показників якості [2]

### 2.5.6 Модель життєвого циклу якості системи/програмних засобів

Протягом життєвого циклу ПЗ з точки зору якості виділяють 3 принципові фази:

- розробка – продукт є об’єктом вимірів внутрішньої якості програмних засобів;
- тестування – продукт є об'єктом вимірів зовнішньої якості програмних засобів;
- застосування – продукт є об'єктом якості під час застосування.

Як було розглянуто раніше, показники якості наступної фази залежать від показників якості попередньої, тобто внутрішні властивості впливають на зовнішні та разом впливають на якість під час застосування. Комбінуючи взаємозв'язок властивостей та показників якості з моделлю життєвого циклу ПЗ, отримуємо модель життєвого циклу якості системи/програмних засобів (рисунок 2.9).

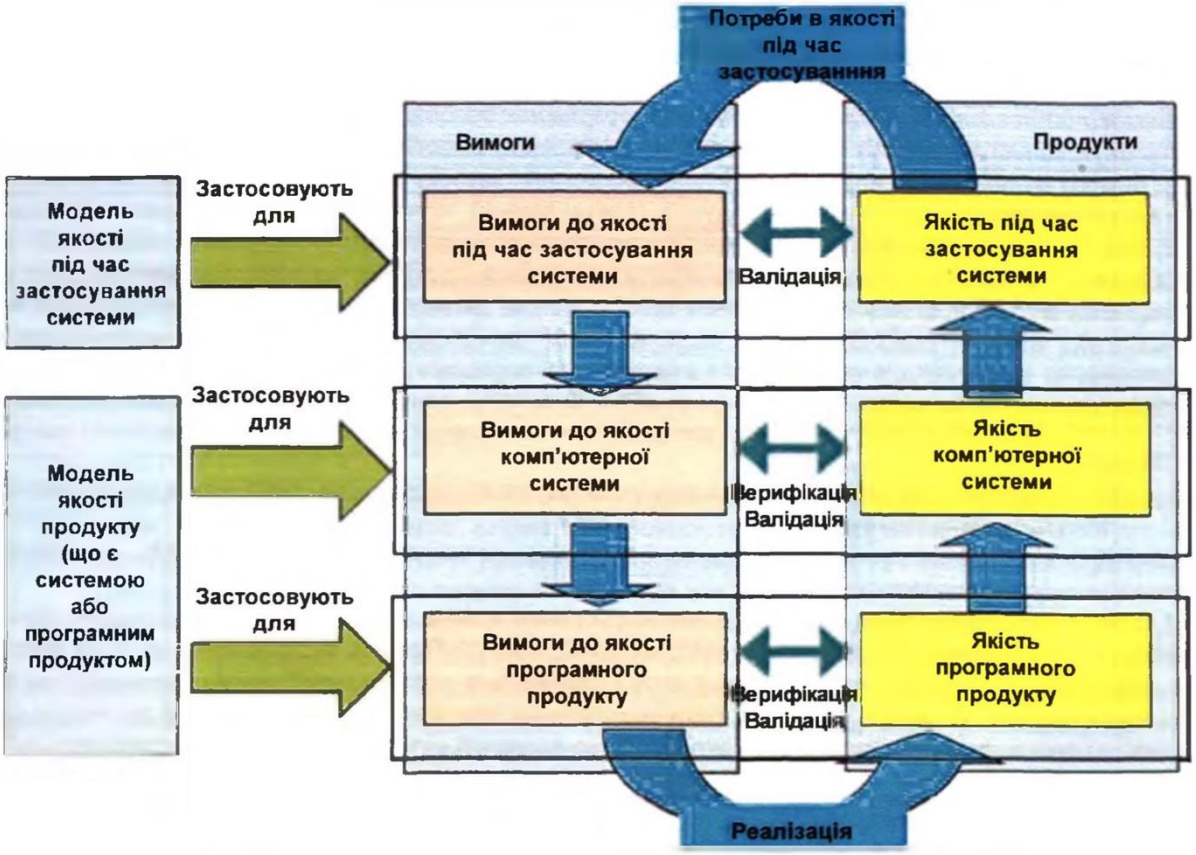


Рисунок 2.9 – Модель життєвого циклу якості системи/програмних засобів [2]

## 2.6 Практичні механізми підвищення якості програмного коду

Розглянемо практичні універсальні механізми, що дозволяють значно підвищити якість програмного коду.

### 2.6.1 Парне програмування (pair programming)

Парне програмування – це agile-практика об’єднання розробників у пари для роботи над завданнями з програмування. Вона є найефективнішою, коли обидві людини привносять щось унікальне, будь то знання, досвід чи різноманітні точки зору [22].

Щоб парне програмування було ефективним, обидва учасники повинні вербалізувати свої процеси мислення. Успішне парне програмування залежить

від успішного спілкування так само, як і від навичок програмування. Передумова полягає в тому, що під час роботи над складним завданням «дві голови краще, ніж одна» [22].

Під час огляду спостерігач також розглядає «стратегічний» напрямок роботи, висуваючи ідеї щодо вдосконалення та можливі майбутні проблеми, які потрібно вирішити. Це має на меті звільнити людину, що друкує, задля зосередження всієї увагу на «тактичних» аспектах виконання поточного завдання, використовуючи спостерігача як сітку безпеки та порадника [39].

Оскільки вимоги до парного програмування мінімальні: два розробники та спільне середовище для розробки, існує багато реалізацій цього підходу, що мають дві важливі спільні риси – чергування та відкрите спілкування [22, 33]:

- Пінг Понг (Ping Pong) – виконується в поєднанні з розробкою, керованою тестуванням. Одна особа пише тест, а інша робить його успішним, кожен учасник по черзі пише тести та здає тести. Дуже важливо уникати ситуації коли одна особа завжди пише тести, а інша – програмний код;
- Водій-Навігатор (Driver-Navigator) – це більш вільна форма шаблону пінг-понгу: водій (driver) – це особа за клавіатурою, яка активно пише код, навігатор (navigator) спостерігає, перевіряє код на точність і стежить за більшою картиною. Цей стиль є ефективною протиотрутою шаблону, коли людина, яка друкує, часто контролює те, що вводиться, переводячи іншу особу в пасивний режим, де вона просто спостерігає за тим, що відбувається. Водій і навігатор регулярно міняються ролями приблизно кожні 15 хвилин;
- Неструктуроване поєднання (Unstructured Pairing) – це вільний процес (немає жодного конкретного підходу), коли водій і навігатор змінюють один одного, коли це має сенс. Ця свобода від структури може допомогти природно підібраним парам рухатися ще швидше. Однак пари, які мають різні стилі, можуть мати проблеми з відсутністю структури.

Переваги парного програмування [22, 33]:

- виробляються кращі рішення – пари програмістів часто створюють кращі рішення, ніж кожен розробник міг би створити самостійно. Проблеми виявляють раніше, а потенційні помилки виявляють дві людини, а не одна. Перш ніж пара вибере підхід до конкретної проблеми, вони її аналізують, проводять мозкові штурми, оцінюють і обговорюють будь-які можливі компроміси. Рішення оцінюються заздалегідь, а не після їх впровадження;
- обмін знаннями та контекстом на льоту – забезпечується процес обміну знаннями та контекстом, який вбудовано у щоденний робочий процес. декілька розробників одночасно працюють на тим самим кодом, що забезпечує вбудоване резервування, на випадок звільнення, відпустки чи переходу до іншої команди або проєкту однієї людини. Без парного програмування розробникам доводиться докладати додаткових зусиль, щоб обмінюватися знаннями, як правило, через додаткові зустрічі та сеанси перегляду коду;
- запобігання професійному вигоранню – розподіл контексту та знань щодо проєкту між декількома особами значно знижує професійне та емоційне навантаження на ключових працівників;
- взаємне навчання та розвиток навичок – чудовий механізм для обміну знаннями та навичками, що дозволяє швидше обучити молодшого розробника (junior-developer) чи залучити нову людину до існуючого проєкту.

Недоліки парного програмування [23, 33]:

- більша складність – завдання, що доручаються парам зазвичай складніші за ті, що виконує лише одна людина. Для цих складних завдань, слід створити та узгодити підхід, що додає додатковий рівень складності та вимагає багато часу та роздумів. В ідеалі програмісти мусять мати достатньо часу, щоб попрацювати над завданнями з програмування самостійно та в парі;

- втома – коли ви працюєте самостійно, ви можете робити перерви та працювати у своєму власному темпі. Це складніше, коли ви працюєте в парі, тому важливо обговорити графіки перерв зі своїм партнером з програмування в парі;
- клімат у команді – високоінтенсивне спілкування парного програмування не підходить для кожного розробника. Деякі люди можуть не погодитися з ідеєю сидіти, буквально пліч-о-пліч, з колегою по вісім годин на день. До того ж досвідчені розробники можуть бути більш продуктивні в програмуванні соло, ніж у парі;
- керування рівнями навичок – якщо між двома партнерами є суттєва різниця в рівнях навичок, один може зрештою занадто покладатися на іншого. Важливо бути відкритими та дати іншій людині можливість написати код, зробити помилки та виправити себе (наприклад, вказувати на помилку через деякий час даючи можливість самостійно виправити її);
- пара новачок–новачок – такий підхід може дати кращі результати, ніж робота двох новачків незалежно, хоча зазвичай цієї практики уникають, оскільки новачкові важче виробити хороші звички без відповідного зразка для наслідування;
- немає часу для індивідуальних активностей – деякі завдання потрібно виконувати самостійно, як-от електронна пошта чи відповіді на телефонні дзвінки. Очікувати, що пара весь час працюватиме разом, нереально і може призвести до вигорання. Замість цього зробіть це невід’ємною частиною розкладу, який включає час для роботи наодинці.

Рекомендації щодо парного програмування [22]:

- продовжуйте говорити – програмування при цьому підході ніколи не повинно виконуватися мовчки. Коли пара мовчить, швидше за все вони не діляться своїми процесами мислення і дуже рідко – настільки ідеально синхронізовані, що нічого не потрібно говорити. Гарне парне

програмування передбачає багато розмов і роздумів вголос. Хороший трюк – завжди розповідати про те, що ви робите та думаєте, коли друкуєте;

- дотримуйтесь рівного часу набору тексту – ролі водія та навігатора повинні бути розподілені належним чином, що обидва учасники друкували по черзі приблизно однакову кількість часу (не більше за півгодини). Якщо одна особа має тенденцію домінувати на клавіатурі, може бути корисно встановити будильник на 20-хвилинні інтервали, щоб стимулювати обмін ролями;
- середовище розробки повинно бути однаково зручно обом людям – рекомендується використовувати IDE з якою обидва приблизно однаково знайомі щоб не порушувати баланс парного програмування через те що одна людина буде менш охоче «керувати» у незручній для неї середі розробки.

### 2.6.2 Рецензування програмного коду (code review)

Успішна стратегія експертної перевірки програмного коду вимагає балансу між суворо задокументованими процесами та незагрозливим середовищем для співпраці. Суворо впорядковані експертні оцінки можуть пригнічувати продуктивність, але неформалізовані процеси часто неефективні.

Розглянемо рекомендації щодо рецензування програмного коду [13]:

- слід переглядати менше ніж 400 рядків коду за один підхід – мозок може ефективно обробляти лише 200-400 LOC за раз, при при перевищені цієї кількості здатність знаходити дефекти зменшується. На практиці перевірка 200-400 LOC протягом 60-90 хвилин повинна дати 70-90% виявлення дефектів;

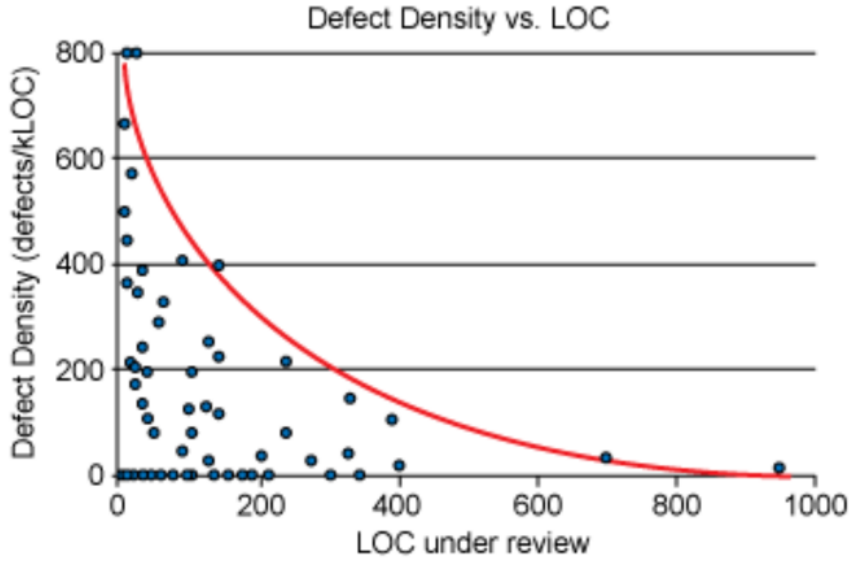


Рисунок 2.9 – Кількість знайдених дефектів в залежності від кількості перевірених LOC [13]

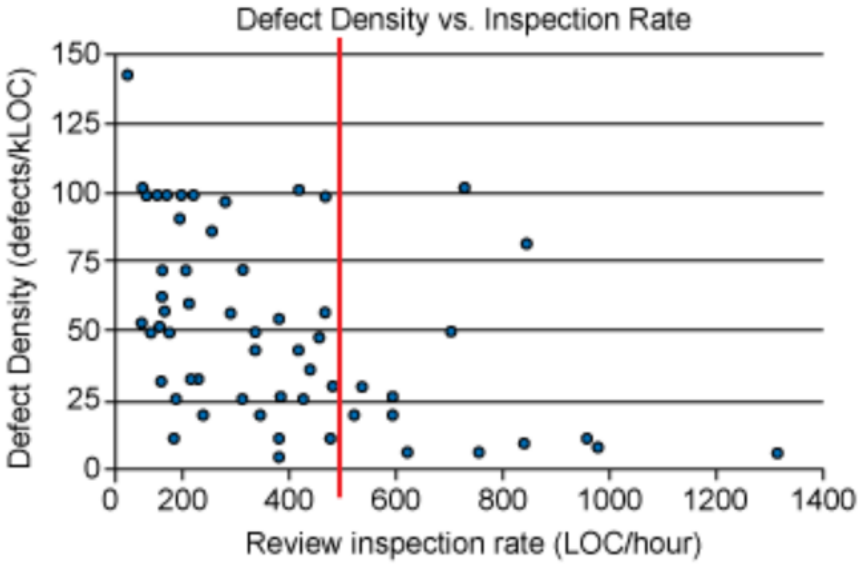


Рисунок 2.10 – Кількість знайдених дефектів в залежності від кількості перевірених LOC за годину [13]

- швидкість перевірки не повинна перевищувати 500 LOC на годину – дослідження SmartBear показують значне зниження щільності дефектів зі швидкістю, вищою за 500 LOC на годину. Перевірка коду в розумній кількості, повільніше протягом обмеженого проміжку часу призводить до найефективнішого перегляду коду;

- використовувати сесії не більше 60 хвилин за раз – коли люди займаються будь-якою діяльністю, яка вимагає зосереджених зусиль протягом певного періоду часу, продуктивність починає падати приблизно через 60 хвилин. Дослідження показують, що певні перерви в роботі можуть значно підвищити якість роботи. Проведення більш частих перевірок має зменшити потребу в проведенні перевірки такої тривалості;
- ставити цілі та фіксувати показники – потрібно вирішити, як буде вимірюватись ефективність експертної перевірки. Для цього можуть використовуватись зовнішні показники (наприклад, «зменшити на 15% кількість дефектів, утворених розробкою»), або показники внутрішнього процесу:
    - швидкість, з якою виконується перевірка;
    - кількість знайдених помилок за годину перевірки;
    - середня кількість знайдених помилок на рядок коду;
- автори повинні анотувати вихідний код перед рецензуванням – анотації направляють рецензента через зміни, показуючи, які файли слід переглянути в першу чергу, полегшуючи процес і забезпечуючи більшу глибину контексту. Як додаткова перевага, автор часто знаходить додаткові помилки ще до початку рецензування;
- використовувати контрольні списки (check lists) – контрольні списки є найефективнішим способом усунення частих помилок і боротьби з проблемами пошуку пропусків. Контрольні списки перевірки коду також надають членам команди чіткі очікування щодо кожного типу перевірки та можуть бути корисними для відстеження для звітування та покращення процесу;
- використовувати процес контролю усунення виявлених дефектів – найкращий спосіб переконатися, що дефекти виправлені. Для цього рекомендується використовувати інструмент спільного перегляду

коду, який дозволяє рецензентам реєструвати помилки, обговорювати їх з автором і затверджувати зміни в коді;

- виховувати позитивну культуру перегляду коду – експертна оцінка може погіршити міжособистісні стосунки в команді. Важко, коли кожна частина роботи була піддана критиці колегами, а керівництво оцінювало та вимірювало щільність дефектів у вашому коді. Звіти, отримані в результаті експертної перевірки коду, ніколи не слід використовувати у звітах про продуктивність. Якщо особисті показники стануть основою для винагороди чи підвищення, розробники почнуть вороже ставитися до процесу й, природно, зосередяться на покращенні особистих показників, а не на написанні кращого загального коду;
- підсвідомі наслідки експертної оцінки – факт того, що інші перевірятимуть їхню роботу, природним чином спонукає людей створювати кращий продукт – «вибіркова перевірка» від 20% до 33% коду призвела до зниження щільності дефектів з мінімальними витратами часу;
- надавайте перевагу швидкому огляду коду – для оптимізації часу вашої команди та ефективного вимірювання її результатів рекомендується легкий процес із підтримкою інструментів.Спрощена перевірка коду займає менше ніж 20% часу, від часу формальної перевірки, і знаходить стільки ж помилок.

### 2.6.3 Методи тестування білої скриньки

Тестування за принципом білої скриньки характеризується ступенем, у який тести виконують або покривають логіку (вихідний текст) програми, використовуючи граф всіх можливих маршрутів потоку передач керування – алгоритм програми.

### 2.6.3.1 Покриття рішень або покриття переходів

Цей метод передбачає використання достатнього числа тестів, щоб кожне рішення у цих тестах набуде значення істина (TRUE) або хибність (FALSE) принаймні один раз, тобто кожний напрямок переходу повинен бути реалізований принаймні один раз.

Покриття рішень зазвичай задовольняє покриттю операторів через те що кожний оператор лежить на деякому шляху, що виходить або з оператора переходу, або із точки входу програми, тому при виконанні кожного напрямку переходу кожний оператор повинен бути виконаний. Однак існують деякі виключення. Перше – патологічна ситуація, коли програма не має рішень. Друге зустрічається в програмах або підпрограмах з декількома точками входу; даний оператор може бути виконаний тільки в тому випадку, якщо виконання програми починається з відповідної точки входу [3].

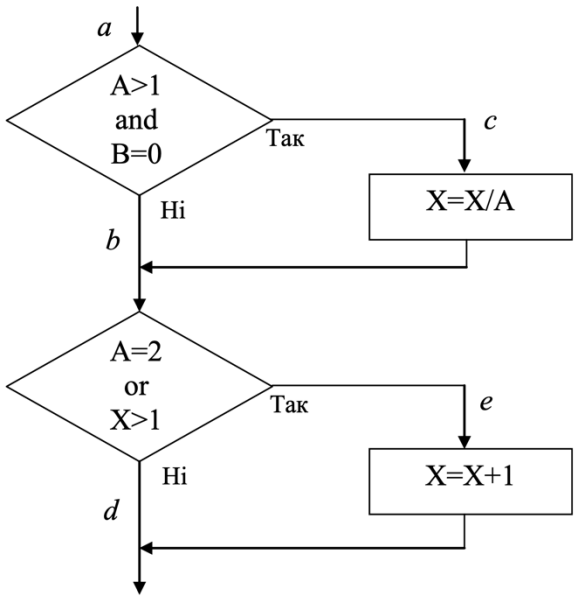


Рисунок 2.11 – Схема алгоритму програми [3]

### 2.6.3.2 Покриття умов

Метод покриття умов передбачає використання кількості тестів, достатнього для того, щоб всі можливі результати кожної умови в рішенні виконувалися принаймні один раз. Цей критерій є кращим у порівнянні з попереднім бо може (але не завжди) викликати виконання рішень в умовах, не реалізованих при покритті рішень. Але через те що, як і при покритті рішень, використання цього методу не завжди приводить до виконання кожного оператора, цьому критерію потрібне доповнення, яке полягає в тому, що кожній

точці входу в програму або підпрограму повинне бути передане керування при виклику принаймні один раз [3].

#### 2.6.3.3 Покриття рішень/умов

Через те що критерій покриття умов не завжди задовольняє критерію покриття рішень використовується критерій покриття рішень/умов, що вимагає такого достатнього набору тестів, щоб всі можливі результати кожної умови в рішенні виконувалися принаймні один раз, та всі результати кожного рішення виконувалися принаймні один раз і кожній точці входу передавалося керування принаймні один раз.

Недоліком критерію покриття рішень/умов є неможливість його застосування для виконання всіх результатів всіх умов; часто подібне виконання має місце внаслідок того, що результати умов у виразах І та АБО можуть приховувати і блокувати дію інших умов. Наприклад, якщо умова І є хибність, то ніяка з наступних умов у виразі не буде виконана. Аналогічно якщо умова АБО є істина, то ніяка з наступних умов не буде виконана. Отже, критерії покриття умов і покриття рішень/умов недостатньо чутливі до помилок у логічних виразах. [3]

#### 2.6.3.4 Комбінаторне покриття умов

Критерієм, що вирішує проблеми попередніх методів та деякі інші є комбінаторне покриття умов, що вимагає створення такого числа тестів, щоб всі можливі комбінації результатів умови в кожному рішенні і всі точки входу виконувалися принаймні один раз. Набір тестів, отриманий за допомогою цього методу, задовольняє також і критеріям покриття рішень, покриття умов і покриття рішень/умов, але такі тести не гарантують покриття всіх шляхів алгоритму [3].

## 2.7 Тестування ПЗ

Тестування програмного забезпечення – техніка контролю якості, що перевіряє відповідність між реальною і очікуваною поведінкою програми завдяки кінцевому набору тестів, які обираються певним чином [14].

Існує наступна класифікація тестування за певними ознаками[14, 15, 30]:

- за об’єктом тестування:
    - функціональне тестування (functional testing);
    - тестування продуктивності (performance testing):
        - навантажувальне тестування (load testing);
        - стрес-тестування (stress testing);
        - тестування стабільності (stability / endurance / soak testing);
    - тестування зручності використання (usability testing);
    - тестування інтерфейсу користувача (user interface (UI) testing);
    - тестування безпеки (security testing);
    - тестування локалізації (localization testing);
    - тестування сумісності (compatibility testing);
- за залежністю від цілей тестування:
    - функціональне тестування (functional);
    - нефункціональне тестування (non-functional);
    - тестування пов’язане зі змінами:
        - тестування нової функціональності (new feature testing);
        - регресивне тестування (regression testing);
        - повторне тестування/тестування підтвердження (retesting)
- за ступенем підготовки:
    - тестування за документацією (formal testing);
    - інтуїтивне тестування (ad-hoc testing);
    - дослідницьке тестування (exploratory testing).
- За знанням системи:
    - тестування чорної скриньки (black box);

- тестування білої скриньки (white box);
- тестування сірої скриньки (gray box);

- за часом проведення тестування:
  - альфа-тестування (alpha testing):
    - димне тестування (smoke testing);
    - тестування нової функціональності (new feature testing);
    - повторне тестування/ тестування підтвердження (retesting);
    - регресійне тестування (regression testing);
    - тестування при здачі (acceptance testing);
  - бета-тестування (beta testing);
- за ступенем автоматизації:
  - ручне тестування (manual testing);
  - автоматизоване тестування (automated testing);
  - напівавтоматизоване тестування (semiautomated testing);
- за ступенем ізольованості компонентів:
  - компонентне (модульне) тестування (component / unit testing);
  - інтеграційне тестування (integration testing);
  - системне тестування (system / end-to-end testing);
- за позитивністю сценаріїв:
  - позитивне тестування (positive testing);
  - негативне тестування (negative testing).

Розглянемо деякі найбільш поширені види тестування.

Вид тестування – це група тестових дій, спрямованих на перевірку специфічних характеристик програмної системи або частини системи на основі конкретних цілей тесту [3].

Функціональне тестування перевіряє, чи реалізовані функціональні вимоги, тобто можливості ПЗ в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить продукт, які завдання вирішує [15].

Нефункціональне тестування відповідає на питання: «Як добре працює система», воно направлено на перевірку тих аспектів ПЗ, які можуть бути описані в документації, але не відносяться до функцій програмних продуктів [30].

Тестування підтвердження перевіряє чи успішно було усунено вихідний дефект. Для цього програмне забезпечення може бути перевірено з усіма тестовими прикладами, які не пройшли через дефект, а у разі потреби – за допомогою нових тестів, щоб охопити зміни, необхідні для усунення дефекту [3].

Регресійне тестування визначається як тестування програмного забезпечення для підтвердження того, що нещодавня зміна програми чи коду не вплинула негативно на наявні функції. Для гарантування, що старий код продовжує працювати після внесення останніх змін виконується повний або частковий вибір уже виконаних тестів [14].

## 2.8 Розгортання та моніторинг

Розгортання програмного забезпечення (software deployment) – це усі дії, що роблять програмну систему готовою до використання. Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника, так і з боку споживача. Оскільки кожна програмна система є унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, «розгортання» можна трактувати як загальний процес відповідно до певних вимог та характеристик [16].

Моніторинг наразі здійснюється за допомогою лог-файлів. і системних метрик, що генеруються ПЗ, операційною системою та апаратними засобами.

### 2.8.1 Логування

Лог-файли – це свого роду електронні журнали, в яких систематично фіксуються події, помилки та інша важлива інформація про роботу програми.

Вони є невід’ємною частиною процесу розробки, оскільки надають розробникам цінні дані для відстеження та аналізу роботи своїх додатків [17].

В реаліях теперішнього часу з появою розподілених систем та хмарних технологій, лог-файли стали ключовим інструментом для моніторингу та налагодження програм у реальному часі. Сьогодні складні системи логування надають цілий арсенал інструментів, як-от централізовані панелі управління, рівні логування та засоби фільтрації, спрощуючи стеження за роботою застосунків у різноманітних середовищах [17].

Лог-записи зазвичай зберігаються у спеціальних файлах, які можуть бути текстовими, бінарними або в інших форматах залежно від обраного розробником підходу. Файли можуть бути стиснуті або ротовані з плином часу для оптимізації використання дискового простору та забезпечення зручності аналізу. Розглянемо приклади основних форматів [17]:

- текстовий – найпростіший формат, де кожен запис представлений у вигляді текстового рядка. Це зручно для читання, але може бути складним для аналізу автоматизованими системами;
- XML (eXtensible Markup Language) – формат із розміткою, що представляє дані в структурованій формі. Це полегшує читабельність людиною та машинний аналіз;
- JSON (JavaScript Object Notation) – легкочитаний формат, що представляє дані у вигляді пар “ключ-значення”. Ефективний для обміну даними та підтримує складні структури даних.

Для ефективного ведення логів необхідно дотримуватися наступних практик (best practices):

- використання різних рівнів логування, це дає змогу гнучко налаштовувати деталізацію записів залежно від поточного контексту та потреб:
    - DEBUG – для докладної налагоджувальної інформації;
    - INFO – для операційних подій;
    - WARNING – для попереджень;

    - ERROR – для повідомлень про помилки;
- форматування – читабельність логів відіграє важливу роль. Наприклад додавання часових міток, ідентифікаторів події та контексту робить лог-файли більш зрозумілими, а використання структурованих форматів (JSON або XML), полегшує автоматичний аналіз даних;
- ротація логів – періодичне оновлення або ротація лог-файлів важливе для запобігання переповнення дискового простору. Це також полегшує аналіз подій, оскільки логи можуть бути розділені за часом або розміром.

### 2.8.2 Метрики

Показник – це результат аналізу одного чи більше конкретних вимірів для оцінки потреби, цінності, результату, діяльності чи вхідної інформації [37].

Метрика – це вимірна у конкретні моменти часу величина показника, що використовується для виміру прогресу. Метрика може бути конкретним значенням, порогом чи діапазоном у межах різних періодів часу (рік, місяць, тиждень, день) [37].

Існують три основні підходи до збору метрик: USE (Tom Wilkie), RED (Brendan Gregg), LTES (Google SRE). Вони відрізняються за метою моніторингу і, природно, включають різний набір метрик (але "E" скрізь відповідає за помилки) [31].

У методології USE для кожного ресурсу (CPU, дискова підсистема, пам'ять тощо) рекомендується знімати такі метрики [31]:

- Utilization – час або процент використання ресурсу, зайнятого «корисною роботою»;
- Saturation – насиченість, тобто кількість відкладеної чи поставленої у чергу «роботи»;

- Errors – кількість помилок у роботі компонента.

RED пропонує моніторити [31]:

- Rate – кількість запитів за одиницю часу (наприклад, rps на мікросервіс чи сервер);
- Errors – кількість помилок;
- Duration (воно ж latency) – час обробки одного запиту.

У LTES за аналогією з двома попередніми методологіями відстежують [31]:

- Latency – час на обробку одного запиту (з поділом на успішні та помилкові запити);
- Traffic – кількість запитів до компонента (для вебсервера це можуть бути http-запити, для бази даних – транзакції тощо);
- Errors – кількість помилок;
- Saturation – тут це кількісна метрика, що відображає, наскільки компонент використовує свої ресурси та скільки у нього «роботи у черзі».

### 2.8.3 Система сповіщень (alerts)

Побудова аллертингу – системи автоматичного оповіщення про те, що метрика досягла порогового значення, – ґрунтується на концепціях SLI, SLA та SLO [31]:

- SLI (Service level indicators) – набір ключових метрик, якими можна визначити життєвий статус сервісу, його продуктивність, «задоволеність» кінцевих користувачів роботою сервісу;
- SLA (Service level agreement) – так звана «угода про рівень доступності сервісу», яка визначається як зовнішнє зобов'язання перед кінцевим користувачем або клієнтом;
- SLO (Service level objectives) – набір цільових, «бажаних» значень SLI, вихід межі яких може призвести до порушення SLA конкретного

сервісу чи компонента. Максимально допустиме відхилення від «ідеальних» показників у цій концепції називається Error Budget (право помилку).

# 3 РЕКОМЕНДАЦІЇ ЩОДО ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОТЯГОМ ЖИТТЄВОГО ЦИКЛУ ПЗ

## 3.1 Оцінка бізнес-ідеї

Перевірка бізнес-ідеї – це визначення того, що запропонована ідея відповідає мінімальним критеріям, необхідним для розробки, розвитку та підтримки процвітаючого проєкту [47].

Спочатку усі наявні дані слід поділити на ті, що можуть унеможливити, ті що впливають та ті що не впливають на реалізацію чи впровадження продукту.

### 3.1.1 Попередня оцінка

На першому етапі невелика кількість експертів - менеджерів та спеціалістів аналізують характеристики з першої категорії, до якої належать:

- актуальність
    - які існуючі запити споживачів буде задоволено?
    - які передбачувані запити споживачів буде задоволено?
    - які нові запити споживачів буде згенеровано?
- реалістичність
    - чи можливо використовуючи існуючі технології реалізувати задум?
    - чи потрібно чекати на будь-які перспективні технології?
    - чи потрібно розробляти нові технології?
- здійсненність
    - чи можливо використовуючи наявні ресурси реалізувати проєкт?
    - які додаткові ресурси потрібні задля реалізації задуму?
    - які терміни потрібні для реалізації та впровадження проєкту?

- унікальність
    - чи продукт абсолютно новий на ринку?
    - чим продукт буде відрізнятися від конкурентів (існуючих та можливих)?
    - які переваги цей продукт буде мати перед конкурентами?
    - чи цього достатньо задля успішної конкуренції?
- юридичні обмеження
    - чи дозволено з юридичної точки зору розробка та впровадження таких продуктів чи послуг у цій галузі?
    - чи можливо використовувати наявні патенти, технології та іншу інтелектуальну власність?
    - які інші юридичні обмеження наявні (наприклад, в багатьох державах висуваються особливі вимоги щодо обробки персональних даних, зберігання та обробки даних (сервери повинні фізично знаходитися на території країни) або логування)?

Дані для цього аналізу можуть бути надані в будь-якому вигляді: відповідні звіти, доповіді, обговорення чи мозкові штурми (brain storm). В результаті цього етапу кожна з характеристик повинна бути оцінена якісно. Якщо ідея задовольняє всім характеристикам, вона визнається перспективною та переходять до другого етапу. В протилежному випадку ідея визнається або неперспективною і вся робота припиняється, або перспективною але недопрацьованою та спрямовується на додаткові дослідження та аналіз чи відкладається.

### 3.1.2 Детальна проробка ідеї

На наступному етапі усі характеристики, окрім тих, що не впливають на реалізацію чи впровадження продукту, беруться у розробку відповідними робочими групами, що можуть складатися з декількох чи однієї особи.

Тут треба провести необхідні дослідження (варіанти досліджень та можливі джерела інформації наведені у пункті 2.1.8), результатом яких повинні бути розгорнуті звіти з обґрунтованими висновками щодо кожної характеристики чи підхарактеристики.

1. Актуальність
    a. групи клієнтів та користувачів використовуючи географічні, демографічні та психографічні характеристики тощо;
    b. наявні потреби та вимоги до ПЗ у цих груп
        i. використовувані апаратні засоби (смартфон, лептоп, десктоп чи якесь спеціальне обладнання);
        ii. наявність доступу до інтернету та його швидкість;
        iii. використовувані мови;
        iv. додаткові потреби (кольоросприйняття, контрастність, голосові сповіщення тощо);
    c. очікувані потреби та вимоги;
    d. проблеми цих груп які може вирішити ПЗ;
    e. як ці люди справляються з цими проблемами зараз.
2. Терміновість – як швидко потрібно вирішувати наявні проблеми та/або запобігати прогнозованим.
3. Унікальність
    a. основні конкуренти, їх продукти та послуги;
    b. які типи конкуренції мають місце:
        i. пряма (ті самі продукти тим самим клієнтам);
        ii. непряма (дещо різні продукти різним клієнтам на одному ринку);
        iii. альтернативна (пропонуються різні продукти та послуги тим самим клієнтам на одному ринку);
    c. визначення своєї конкурентної переваги;
    d. аналіз сильних та слабких сторін ваших та конкурентів.

4. Інноваційність – ключові відмінності розробляємого ПЗ від вже наявних з технічної точки зору та пропонуємих можливостей.
5. Конкретність
    a. чітко сформульовані цілі розробки та впровадження ПЗ;
    b. ПЗ описаний у конкретних та об’єктивних високорівневих показниках;
    c. обмеження щодо ПЗ визначені;
    d. сформоване розуміння бажаного функціоналу ПЗ та MVP.
6. Стратегія просування бізнес-ідеї
    a. оцінка існуючої ситуації;
    b. перелік конкурентів, їх маркетингова і комунікаційна активність;
    c. сегментація цільової аудиторії;
    d. асортиментна і цінова політика;
    e. модель продажів;
    f. опис компаній з продажу та маркетингу та підвищення лояльності споживачів;
    g. напрямки можливого розвитку як то ринки, потреби та позиціонування;
    h. план робіт на один рік;
    i. необхідні ресурси, бюджет;
    j. прогноз розвитку;
    k. система оцінки ефективності;
    l. альтернативні сценарії в разі зміни ринкових умов.
7. Прибутковість (детальний опис підхарактеристик, наведений у пункті 2.1.5) – в результати оцінки цієї характеристики потрібно відповісти на наступні питання:
    a. чи можна залучити усі необхідні ресурси?
    b. який потенціал прибутку має проєкт?
    c. чи має сенс втілювати в життя цей проєкт.
8. Терміни реалізації

a. які реальні терміни реалізації та релізу проєкту?
b. чи не втратить проєкт за цей час своєї актуальності?

9. Недоліки та негативні фактори – які недоліки, обмеження чи фактори ризику є у проєкту та наскільки вони критичні.

Наприкінці цього етапу потрібно провести перехресне рецензування (cross-review) кожного звіту суміжними робочими групами, це робиться щоб уникнути очевидних помилок, занадто сильного фокусування на окремих деталях або пропуску якоїсь частини дослідження.

### 3.1.3 Оцінка загальної якості ідеї

В результаті попереднього етапу було отримано матеріали щодо кожної характеристики якості бізнес-ідеї що розглядається. Для остаточної оцінки якості ідеї загалом, потрібно провести сесію чи кілька послідовних зборів відповідних експертів (менеджерів та спеціалістів) щодо досліджуваних характеристик та підхарактеристик.

Кожен звіт повинно бути критично розглянуто та прийнято рішення щодо його висновку та обґрунтувань – рекомендованим є формат захисту.

Після цього проводиться оцінка самої бізнес-ідеї. Для доволі простих варіантів вердикт може бути очевидним після розгляду усіх звітів, але у випадку комплексних потрібен більш формалізований підхід.

В такому випадку пропонується використовувати кваліметричний метод комплексування за трьохрівневою шкалою. Для кожної характеристики чи підхарактеристіки експертним методом встановлюється один показників якості:

- задовольняє;
- прийнятний;
- не задовольняє.

Також для показників якості найважливіших характеристик, щонайменше прибутковості, термінів реалізації та недоліків з негативними факторами,

вводиться коефіцієнт вето (формула 3.1), через який комплексний показник якості падає до нуля, якщо хоча б один з них виявляється неприйнятним.

Коефіцієнт вето – це функція, яка при виході будь-якого з найважливіших одиничних показників за допустимі межі звертається в нуль, в усіх інших випадках коефіцієнт вето залишається рівним одиниці [18]:

$$\varphi(Q) = \begin{cases} 1, \text{ якщо усі важливі показники якості} \\ \text{задовольняють або прійнятні,} \\ 0, \text{ якщо хоча б один важливий} \\ \text{показник якості неприйнятний.} \end{cases} \quad (3.1)$$

Якщо після застосування коефіцієнта вето функція якості не дорівнює нулю, для оцінки загальної якості використовуємо формулу 3.2.

$$Q = 1 - 0{,}5\frac{n_{\text{п}}}{n} - \frac{n_{\text{н}}}{n}, \quad (3.2)$$

де $n_{\text{н}}$ – число показників якості, що не задовольняють,

$n_{\text{п}}$ – число прийнятних показників якості,

$n$ – загальна кількість показників якості.

В результаті отриманого значення приймаємо остаточне рішення, базуючись на наступній шкалі:

- $0{,}75 \leq Q < 1$ – перспективна ідею, яку слід розробляти;
- $0{,}5 \leq Q < 0{,}75$ – ризикована ідея, яку можна розробляти;
- $0 \leq Q < 0{,}5$ – занадто ризикована ідея, яку не рекомендується розробляти.

## 3.2 Аналіз бізнес-вимог. Зацікавлені сторони. Користувачі ПЗ

Визначення стейкхолдерів – ключовий момент життєвого циклу будь-якого проєкту, тому відразу після створення його плану потрібно визначити зацікавлені сторони.

### 3.2.1 Ідентифікація зацікавлених сторін

Задля правильного визначення стейкхолдерів потрібно використовувати наступну інформацію:

- дані щодо минулих та наявних проєктів – створюється повний перелік усіх, з ким взаємодіяла команда, тут треба зазначити кожного, хто здатний вплинути на процес чи результат, а не лише співробітників, бізнес-партнерів та інвесторів;
- документацію замовника – при аналізі бюджету, обмежень фінансування, вищого керівництва та об’єктів, стане зрозумілою більша частина зацікавлених сторін;
- дослідження ринку – визначити потенційних споживачів, конкурентів та партнерів. Також треба звернути увагу як реагують ЗМІ на подібні продукти чи послуги, та чи є потенційні тригери або соціальні фактори, що мають відношення до проєкту;
- внутрішня структура компанії – важливо оцінити пряму чи неочевидну залученість в проєкт усіх співробітників компанії (особливо суміжних відділів), а не лише членів команди.

### 3.2.2 Аналіз зацікавлених сторін

Після того як стейкхолдерів визначено, їх потрібно проаналізувати та визначити їх важливість для проєкту, для цього враховуються інтереси, ступінь

впливу та ставлення до проєкту кожної окремої зацікавленої сторони або цілих груп.

В процесі треба з’ясувати:

- рівень зацікавленості у досягненні поставлених цілей;
- інтереси стейкхолдера;
- як можуть зацікавлені сторони перешкодити досягненню цілей?

Важливо розуміти, що не всі стейкхолдери однаково важливі, насправді вони мають різні рівні впливу, зацікавленості та наслідків від реалізації проєкту. Основною задачею аналізу є виявлення і розуміння потреб та очікувань всередині та поза організаційним середовищем. Завдяки такому аналізу визначаються сторони, які зацікавлені та/або впливають на проєкт.

Типовою проблемою, яка виникає у разі наявності численних стейкхолдерів – їх інтереси можуть не узгоджуватися, а іноді й мати прямий конфлікт. Тому виникає потреба в пріоритизації зацікавлених сторін, часто найважливішими визначаються ті, що можуть завдати значних збитків (або недоотриману вигоду) та ті, що найчастіше контактують з командою проєкту, з ким потрібно консультуватися, та хто буде відігравати суттєву роль під час реалізації проєкту.

В результаті потрібно створити список або таблицю що має містити всіх стейкхолдерів, їх рід діяльності, посаду чи роль у проєкті та задачі. Для кожного вказується ступінь підтримки/протидії проєкту, виражений у цифрах від –5 (найбільш активна протидія) до +5 (максимальна підтримка), та рівень впливу на проєкт чи роботу компанії за шкалою від 0 до 5 (сторони з нульовим значенням не слід відсікати автоматично, оскільки мінімум впливу може сигналізувати ризики для проєкту). Також додають іншу наявну інформацію, яка потім може стати у нагоді.

Слід розуміти що занадто великий перелік зацікавлених сторін суттєво знижує ефективність аналізу через кількість інформації, що треба дослідити. Для спрощення потрібно групувати подібних стейкхолдерів та розглядати такі групи як окремий елемент.

Також потрібно пам’ятати, що не лише зацікавлені сторони можуть впливати на проєкт, так само стратегія реалізації проєкту може впливати на стейкхолдерів. Тому потрібно визначити, які з груп мають прямі або непрямі суттєві претензії щодо проєкту або які потенційно можуть негативно впливати на нього.

У випадку, якщо цього недостатньо для визначення важливості та значущості зацікавлених сторін, використовуємо інструменти, описані у підпунктах. Вони розташовані за рівнем комплексності та складності реалізації, тому слід використовувати наступний лише у випадку, якщо поточний є недостатнім.

### 3.2.2.1 Матриця зацікавлених сторін

Матриця – це інструмент відображення рівня впливу, підтримки та інтересу стейкхолдерів до проєкту, який полегшує побудову тактики взаємодії із зацікавленими сторонами [6].

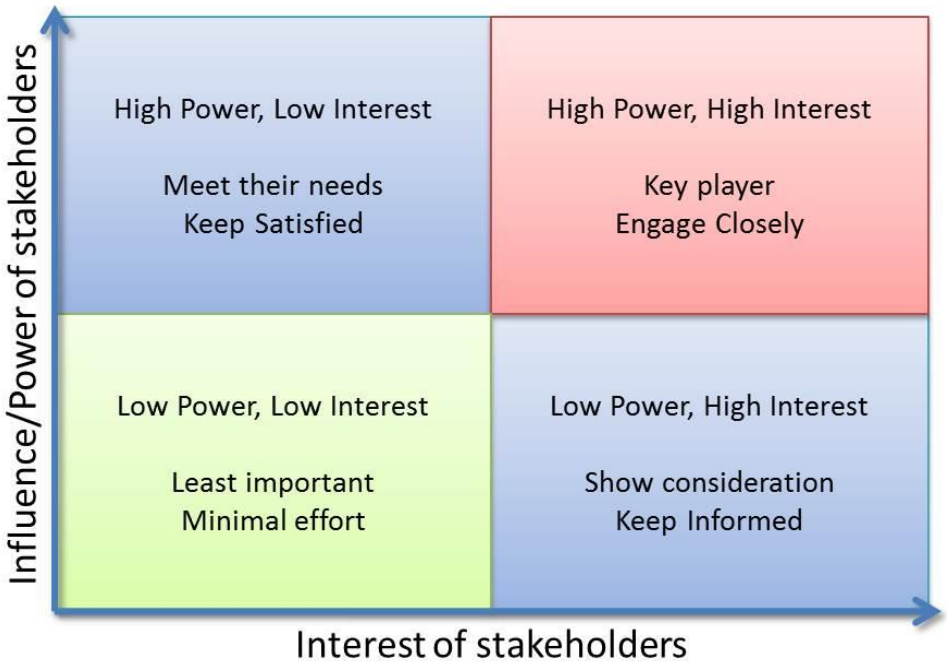


Рисунок 3.1 – Матриця зацікавлених сторін [19]

Для її створення використовують систему координат, де на осі абсцис відмічають зацікавленість або ставлення стейкхолдерів від поганого (знизу) до хорошого (вгорі), а на осі ординат – силу впливу від слабкої (зліва) до сильної (справа). На даному етапі використовуються лише значення «високий» або «низький», позиції кожної зацікавленої сторони всередині такого значення можуть відрізнятися, але наразі це не є принциповим. В результаті отримуємо матрицю з чотирьох квадрантів (рис. 3.1), розглянемо кожен з них.

Високий вплив та інтерес (вгорі справа) – тут вказуються сильні та впливові люди/організації з найвищим рівнем зацікавленості в успішних результатах роботи.

Високий інтерес, але низькій вплив (внизу справа) – це союзники проєкту, які в нього вірять, однак не мають значного впливу.

Низький інтерес, але високий вплив (вгорі зліва) – це впливові «вороги» проєкту, цю групу слід ретельно пропрацьовувати, бо учасники третього квадранту є основним джерелом потенційних ризиків, що можуть негативно вплинути на проєкт.

Низькій вплив та інтерес (внизу зліва) – квадрант, призначений для недоброзичливців, що не можуть значно зашкодити проєкту, однак бажано слідкувати за активністю цих стейкхолдерів.

Пріоритет стейкхолдерів за зменшенням визначається в залежності до квадранту, куди він потрапив:

$$\text{вгорі справа} \rightarrow \text{вгорі зліва} \rightarrow \text{внизу справа} \rightarrow \text{внизу зліва}$$

### 3.2.2.2 Таблиця інтересів зацікавлених сторін

У разі якщо матриці стейкхолдерів недостатньо, наприклад, у кожному квадранті забагато елементів або важко визначити позицію зацікавлених сторін, використовується таблиця інтересів, що дає розширену версію списку. Для цього він доповнюється додатковими полями:

- потреби/вимоги – що потрібно стейкхолдеру та яке це має для нього значення;
- очікування – яких результатів чекає зацікавлена сторона, на які дії та комунікації розраховує в межах проєкту, а також що очікується безпосередньо від стейкхолдера;
- рівень зацікавленості (союзники, підтримка, нейтралітет, опоненти, супротивники);
- вплив компанії на зацікавлену сторону (високий, середній, низький);
- можливі проблеми – чи є стейкхолдер потенційним джерелом ризиків для проєкту, яких саме, як їх уникнути та що робити в разі виникнення;
- комунікації – варіанти спілкування з зацікавленою стороною;
- стратегія взаємодії – способи підвищення залученості, інтересу та лояльності стейкхолдера до проєкту.

Ця таблиця допомагає детальніше вивчити зацікавлені сторони та систематизувати інформацію про них, що дозволить ранжувати їх за пріоритетом.

### 3.2.2.3 Карта зацікавлених сторін

Карта зацікавлених сторін –- інструмент візуалізації стейкхолдерів та їх взаємозв’язків, за допомогою якого визначається взаємний вплив проєкту, у лиці його лідера, та зацікавлених сторін (рис. 3.2).

На карті виділяють три зони:

- внутрішні зацікавлені сторони – знаходяться у зоні повноважень/відповідальності проєкту, тобто безпосередньо підпорядковуються лідеру команди та діють згідно з його рішеннями;
- пов’язані стейкхолдери – не є підлеглими лідера команди, але можуть тісно взаємодіяти з ним;

- зовнішні зацікавлені сторони – тут розміщуються ті, на кого лідер не має прямого впливу.

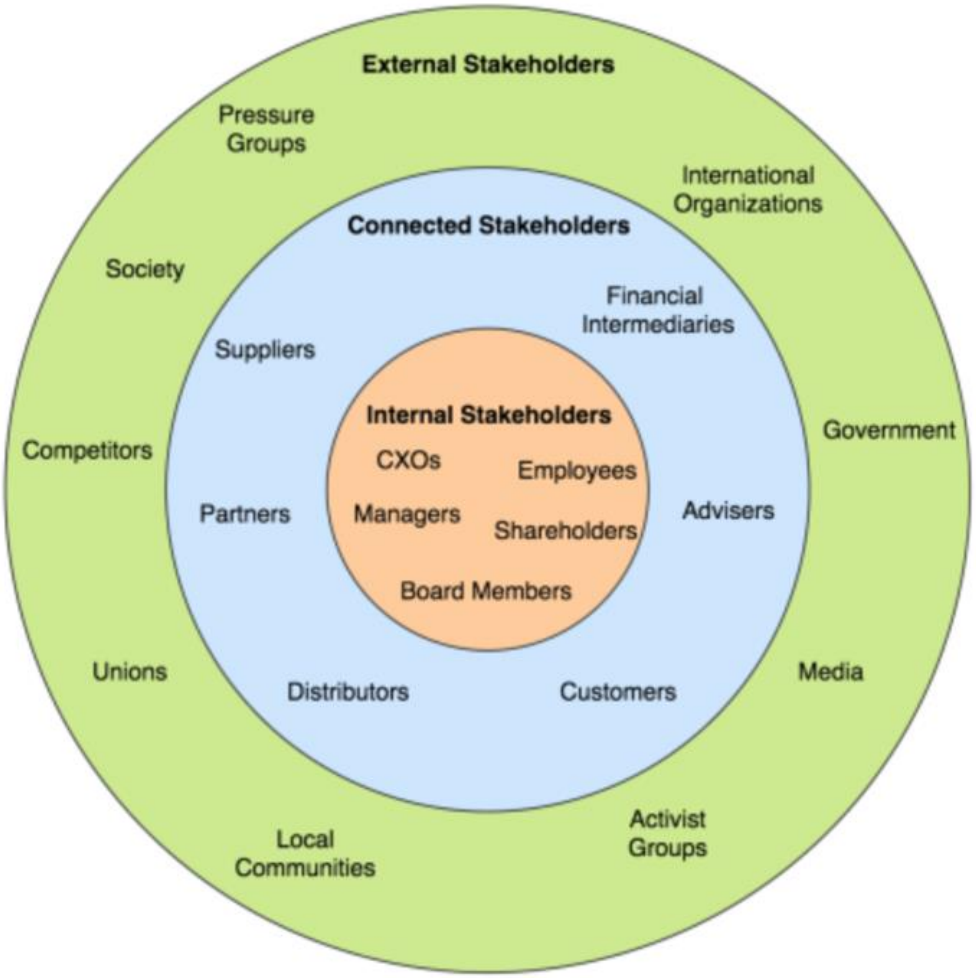


Рисунок 3.2 – Карта зацікавлених сторін [20]

Таким чином, чим ближче до центру, тим вищий вплив на нього має лідер проєкту.

Важливість оцінюється за факторами «ступінь підтримки/протидії» (від –5 до +5), «ступінь впливу» (від 0 до 5), цифри можуть фіксуватися безпосередньо на карті.

У разі виявлення стейкхолдерів з однаковими кількісними показниками але у різних зонах слід розташувати їх за зменшенням пріоритету наступним чином:

зовнішні → пов’язані → внутрішні

### 3.2.2.4 Модель значущості зацікавлених сторін

Концепція значущості зацікавлених сторін була запропонована Рональдом К. Мітчеллом, Бредлі Р. Еглом та Донною Вуд у статті для журналу "The Academy of Management Review" у 1997 році [38].

За цією концепцією кожен стейкхолдер характеризується трьома властивостями [38]:

- влада (вплив на проєкт) – ступінь, до якого сторона має або може отримати доступ до примусових (фізичні засоби), утилітарних (матеріальні засоби) або нормативних (престиж, повага та соціальні) засобів для нав'язування своєї волі;
- легітимність – узагальнене сприйняття або припущення, що дії суб'єкта є бажаними, належними або доречними в рамках певної соціально сконструйованої системи норм, цінностей, переконань та визначень;
- терміновість – ступінь, до якого вимоги зацікавлених сторін вимагають негайної уваги, що залежить не лише від чутливості до часу, але й від того, наскільки "критичними" є відносини із зацікавленими сторонами або важливість їхніх вимог.

Зацікавлені сторони пріоритезуються згідно кількості наявних властивостей, комбінації перетинів цих характеристик, що візуалізуються за допомогою діаграми Венна (рис. 3.3).

В залежності від кількості наявних властивостей виділяють 4 групи:

- латентні (1, 2, 3) – одна ознака;
- очікувані (4, 5, 6) – дві властивості;
- визначені (7) – усі три характеристики;
- не стейкхолдери (8) – не мають жодного атрибуту.

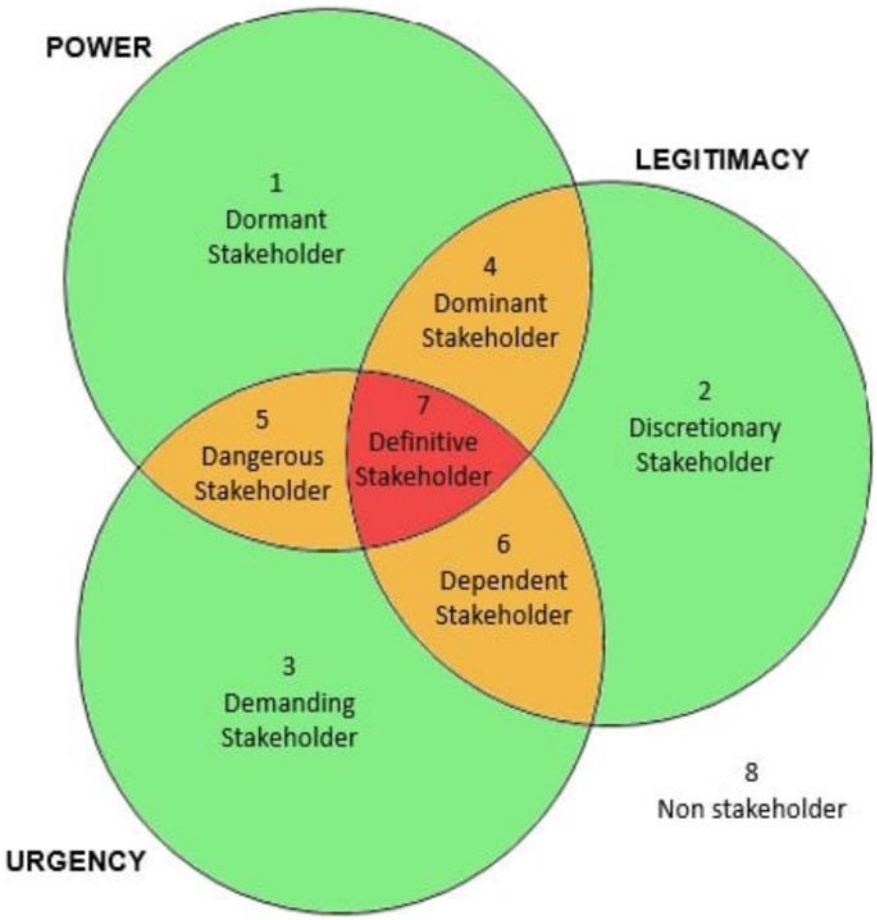


Рисунок 3.3 – Діаграма значущості зацікавлених сторін [38]

Розглянемо детальніше кожний варіант отриманих підгруп:

- неактивні/dormant (1) – є влада, але недостатньо легітимності та нагальності, тому не беруть участі в проєкті, або мають незначну взаємодію чи залучення;
- дискреційні/discretionary (2) – можуть отримувати користь від реалізації проєкту на кшталт благодійності бо не мають ані влади, ані терміновості;
- вимогливі/demanding (3) – мають нагальні вимоги, але відсутні легітимність та влада, тому частіше за все ігноруються;
- домінуючі/dominant (4) – мають легітимність та владу, але не нагальність, їх вимоги дуже важливі, але задоволення цих вимог може бути відкладене;

- небезпечні/dangerous (5) – мають сильні та нагальні вимоги, але їхні вимоги та очікування можуть не визнаватися через брак легітимності;
- залежні/dependent (6) – зацікавлені сторони чиї вимоги мають легітимність да терміновість, але не можуть впливати на рішення щодо їх виконання через брак влади (ігнорування цих стейкходлерів може значно зашкодити іміджу проєкту або, навіть, компанії вцілому);
- визначальні/definitive (7) – мають усі три атрибути, тому є найпріоритетнішими зацікавленими сторонами.

В результаті отримуємо наступну градацію за зменшенням пріоритету:

визначальні → залежні → небезпечні → домінуючі →
дискреційні → неактивні → вимогливі

### 3.2.3 Визначення користувачів ПЗ

Щоб продукт приносив користь, він повинен використовуватися. Стандарт ДСТУ ISO/IEC 25010:2016 визначає три типи користувачів:

- основні;
- побічні;
- опосередковані.

Зазвичай для кожного типу буде декілька ролей, які можна розглядати як окремі групи користувачів. Потрібно визначити так званий «портрет» користувача з кожної групи: його вимоги, очікування, потреби та сценарії використання ПЗ.

Після цього потрібно визначити важливість кожної групи для проєкту та компанії. Це питання не є очевидним, бо основні кошти можуть надходити від однієї групи, наприклад, опосередкованих користувачів – менеджменту організації, що є замовником, а використовувати та приносити користь буде зовсім інша група – основні користувачі. В деяких випадках без врахування

інтересів зовсім невеличкої групи опосередкованих користувачів, наприклад, інтеграторів або менеджерів контенту використання, ПЗ взагалі стає неможливим або втрачає сенс.

Задля розуміння значущості груп використовуємо шкалу «1–3–9» (на даному етапі декілька груп можуть отримати однакову вагу – це є прийнятним, тому що доволі часто занадто складно точно визначити пріоритетність груп базуючись на їх різних характеристиках):

- 9 – найбільш значущі;
- 3 – значущі;
- 1 – малозначущі.

Хоча в результаті такого аналізу визначені групи користувачів можуть доволі сильно відрізнятися, згідно ДСТУ ISO/IEC 25010:2016 вони мають загальний перелік потреб:

- результативність;
- ефективність;
- задоволеність;
- свобода від ризику;
- надійність;
- захищеність;
- покриття контексту;
- опановність;
- доступність.

Далі необхідно скласти таблицю де для кожної групи користувачів буде визначено, чого потребує користувач під час застосування ПЗ для виконання задач задля задоволення кожної з перелічених потреб. Цю інформацію можна визначити експертним методом, аналізуючи наявні продукти, статистичні дані, використовуючи дослідження, а також залучаючи спільноту користувачів. Найбільш цінних користувачів, яких необхідно залучати та ідеї яких потрібно відслідковувати, визначаємо за допомогою методу вимірювання цінності

інновацій, запропонованих користувачем (User Innovation Value Measurement Method), який розглянуто у пункті 2.2.3.

Для кожної групи визначається значущість кожної з потреб використовуючи цілочисельну шкалу від 9 (найбільш значуща) до 1 (найменш значуща).

Далі будуємо матрицю пріоритетів, вказуючи здобуток значущості групи користувачів та потреби (3.3) у відповідних клітинах – збільшення значень визначає більш пріоритетні напрями,

$$S_{ij} = U_i \cdot N_j \tag{3.3}$$

де $U_i$ – група користувачів, $N_j$ – потреба.

Сума цих здобутків для кожної потреби визначає найбільш пріоритетні потреби для користувачів загалом.

$$\sum_{i=1}^{m} S_i \tag{3.4}$$

### 3.2.4 Визначення вимог та очікувань зацікавлених сторін щодо ПЗ

Базуючись на визначеному вище пріоритеті стейкхолдерів (включно з користувачами), потрібно використовуючи усі доступні канали зв’язку та джерела інформації зібрати якомога більше даних задля їх подальшої класифікації та групування. Чим більш повний набір очікувань, ідей, сподівань да сценаріїв було акумульовано, тим менша вірогідність появи «несподіваних» функцій пізніше в життєвому циклі, додавання яких до вже розробленої системи може бути дорогим.

### 3.2.4.1 Приклади джерел інформації

Слід використовувати якомога більше каналів зв’язку та джерел інформації, але потрібно розуміти що не всі вони однаково корисні, тому треба обирати лише авторитетні та ті що користуються довірою у технічному середовищі та суспільстві. У разі занадто великого їх обсягу можна зробити репрезентативну вибірку базуючись на наявності представників зацікавлених сторін (з урахуванням їх пріоритетів).

Наведемо неповний перелік каналів зв’язку та джерел інформації що рекомендовано використовувати:

- прямі запити;
- початковий набір вимог замовника;
- наявні технічні завдання (для цього і аналогічних проєктів/продуктів);
- прототипування;
- інтерв’ю та опитування;
- семінари щодо вимог;
- обговорення в спеціалізованих групах у соціальних сітях, форумах та спільнотах;
- мозкові штурми;
- маркетингові дослідження;
- аналіз аналогічних проєктів/продуктів:
    - документації;
    - конкурентних переваг;
    - скарг користувачів/клієнтів;
    - відомих інцидентів/відмов;
- аналіз суміжних технологічних областей;
- опитування стейкхолдерів вже існуючих аналогічних продуктів;
- тенденції запитів та очікувань в суспільстві;
- висвітлення подібних проєкті/продуктів у ЗМІ.

#### 3.2.4.1 Формування початкового переліку очікувань зацікавлених сторін

Визначення очікувань зацікавлених сторін починається з визначення рівня впровадження (кінцевий користувач буде сам обирати ПЗ чи він буде вимушений ним користуватися через законодавчі чи інші вимоги) та стратегічних цілей, яких проєкт має досягти. Оскільки це є основою для розробки проєкту, вони мають бути чітко визначені та сформульовані. Також завдяки цьому виникає впевненість, що команда проєкту має спільне бачення із стейкхолдерами.

Щоб визначити цілі та задачі, на базі інформації, отриманої від зацікавлених сторін, формуються потреби, прагнення, бажання, можливості, зовнішні інтерфейси, припущення та обмеження [9].

Потреби визначаються у відповіді на запитання «Яку проблему ми намагаємося вирішити?» Цілі стосуються того, що необхідно зробити для задоволення потреб; тобто те, що клієнт хоче, щоб система робила. Задачі розширюють цілі та забезпечують засоби документування конкретних очікувань - можна надати обґрунтування, щоб пояснити, чому існує потреба, ціль чи задача, будь-які зроблені припущення та будь-яка інша інформація, корисна для розуміння чи управління NGOs [9]. Більш детально цей механізм розглянуто у пункті 2.2.4.3.

Команда проєкту також повинна визначити обмеження, які можуть застосовуватися. «Обмеження» – це умова, яка має бути виконана. Ці обмеження можуть включати витрати (ресурси), час на виконання, очікування щодо підтримки життєвого циклу, цілі продуктивності, операційні обмеження, цілі навчання або інші менш очевидні величини, такі як організаційні потреби чи соціально-політичні цілі. Іноді обмеження продиктовано зовнішніми факторами, такими як фізичні обмеження, існуюча система, яку необхідно використовувати, зовнішній інтерфейс, нормативне обмеження або стан технології; іноді обмеження є результатом загального бюджетного середовища [9].

Ці очікування зацікавлених сторін фіксуються та вважаються початковими, доки вони не можуть бути уточнені шляхом розробки концепції діяльності та остаточної згоди зацікавлених сторін [9].

### 3.2.4.2 Зовнішні та внутрішні вимоги

Додатковим етапом визначення вимог щодо продукту/проєкту є визначення зовнішніх вимог, що не пов’язані напряму з стейкхолдерами, а саме:

- законодавчі вимоги та обмеження – тут може знадобитися допомога відповідних фахівців юридичного профілю задля аналізу не лише наявних законів та норм, але й юридичної практики в цілому, включно з тенденціями як в середині країни так і міжнародних;
- обмеження щодо використання патентів та/або авторського права – тут також слід скористатися допомогою відповідних фахівців;
- стандарти/КУП/тощо – у разі, коли потрібно доводити відповідність самого ПЗ або процесу розробки;
- вимоги щодо інтеграції з зовнішніми системами/сервісами.

Також у кожної компанії є перелік внутрішніх вимог щодо проєктів, розробки ПЗ та використовуваних процесів:

- внутрішні стандарти/практики/політики;
- вимоги щодо інтеграції із внутрішніми системами/сервісами;
- вимоги щодо впровадження систем, сервісів або процесів;
- вимоги щодо використовуваних систем моніторингу;
- правила щодо підтримки систем.

Зовнішні та внутрішні вимоги та обмеження можуть суттєво впливати на можливість реалізації тих чи інших очікувань щодо проєкту.

### 3.2.4.3 Концепція операцій

У разі розробки комплексного проєкту чи системи для уточнення початкових очікувань зацікавлених сторін використовується концепція операцій (Concept of Operations, ConOps) яка додатково гарантує, що технічна команда повністю розуміє очікування та те, як вони можуть бути задоволені продуктом.

ConOps є важливою рушійною силою системних вимог, тому її слід враховувати на ранніх етапах процесів проєктування системи. Продумуючи цю концепцію і сценарії використання, часто виявляються вимоги та функції дизайну, які інакше можна було б проігнорувати. ConOps має включати сценарії та враховувати всі аспекти для всіх важливих операційних ситуацій, включаючи відомі номінальні та неномінальні ситуації/операції під час інтеграції, тестування та запуску через утилізацію. ConOps також є важливим помічником у визначенні цілей життєвого циклу, персоналу та розподілу функцій між людьми та системами [9].

Задля розробки корисного та повного набору сценаріїв, слід розглянути важливі несправності та робочі ситуації в погіршеному режимі.

Типова інформація, що міститься в ConOps, включає опис основних етапів; терміни операції; робочі сценарії та стратегії управління несправностями, опис людської взаємодії та необхідного навчання, стратегія наскрізної комунікації; архітектура команд і даних; експлуатаційні засоби; рівень персоналу та необхідні навички; критичні події. Експлуатаційні сценарії описують динамічне уявлення про роботу систем і включають те, як система сприймається, як функціонує в різних режимах і переходах між режимами, включаючи взаємодію із зовнішніми інтерфейсами, відповідь на очікувану небезпеку та несправності, а також під час пом’якшення відмов [9].

Це може призвести до подальшого уточнення початкового набору очікувань зацікавлених сторін, якщо будуть виявлені прогалини або неоднозначні твердження. Ці сценарії та концепції поведінки системи забезпечують вільне від реалізації розуміння очікувань зацікавлених сторін шляхом визначення того, що

очікується, без розгляду того, як задовольнити потребу. Він фіксує необхідні поведінкові характеристики та те, як люди взаємодіятимуть із системою. Стратегії підтримки включають положення щодо виготовлення, тестування, розгортання, експлуатації, підтримки та утилізації.

#### 3.2.4.4 Фіналізація бізнес-вимог

Після з'ясування зовнішніх та внутрішніх вимог глобальні обмеження щодо майбутнього проєкту стають зрозумілими – законодавчі вимоги й обмеження так само, як обмеження щодо використання патентів та/або авторського права та відповідність національним чи міжнародним стандартам та КУП не можуть бути порушені. Внутрішні правила, політики чи вимоги компанія також навряд чи буди змінювати, можливо лише заради дуже важливого клієнта або проєкту. Питання щодо інтеграції з зовнішніми чи внутрішніми системами та/або сервісами більш гнучке, але все-одно обмежено щодо вибору проміж існуючими варіантами.

Початкові очікування зацікавлених сторін (доповнені у разі використання ConOps) потрібно додатково перевірити на відповідність глобальним обмеженням та вимогам. Також потрібно оцінити з боку обов'язковості їх виконання (детально розглянуто у пункті 2.2.4.2) тому що не всі очікування однаково значущі з точки зору задоволеності стейкхолдерів. Різниця між ними полягає в тому, що передбачувані очікування описують те, що вважається ймовірним (прогноз того, що станеться), тоді як нормативні очікування представляють те, що має чи може статися (надія на те, чого можна очікувати). Базою для будь-яких очікувань є цінності, оскільки навіть коли очікування пов'язане з негативним результатом, воно порівнюється з цінностями та інтересами, пов'язаними з бажаним станом. Однак цінності можуть мати багато різних форм, наприклад, від економічних цінностей до суспільних цінностей. Інтереси також можуть бути різними, наприклад, від дуже обмеженого егоїзму до утилітаризму. Тобто для реальної оцінки очікувань потрібне глибоке

розуміння їх формування, бо воно може відкрити, як зацікавлені сторони транслюють інституційні контексти та реагують на роботу організацій: погоджуються вони (мають оптимізм чи надію) чи бачать прогалини (цинічно або песимістичний).

Команда проєкту повинна усвідомлювати, що з огляду задоволеності клієнтів існують адекватні та бажані рівні очікувань, де адекватний рівень очікувань – це найнижчий рівень, на якому зберігається задоволеність клієнтів, а рівень бажаних очікувань вказує на ідеальний рівень сподівань і побажань клієнта. Також між адекватним і бажаним рівнем існує зона толерантності, виконання очікувань з якого буде підвищувати задоволеність зацікавлених сторін.

Тож задля успішності проєкту потрібно визначити нормативні або базові очікування стейкхолдерів. Але слід усвідомлювати, що базова лінія для очікувань походить від цінностей та інтересів зацікавлених сторін, які не залежать від дій будь-якої конкретної організації, через це реалізація лише цих очікувань є абсолютно недостатньою. Саме тому також треба визначити ненормативні очікування, які є надважливими задля отримання конкурентних переваг та унікальності проєкту.

Коли ці визначені очікування будуть оцінюватися в контексті конкретної команди проєкту з урахуванням наявної або відсутньої інформації та досвіду, очікування виявляться позитивними або негативними, залежно від впевненості в готовності та здатності команди забезпечити бажані результати або запобігти небажаним результатам. Тому команда може визначити деякі очікування як надважливі бо їх реалізація дозволить суттєво підвищити рівень довіри поміж зацікавлених сторін.

У разі, коли перелік очікувань є занадто великим, задля більш ефективної та очікуваної роботи над проєктом слід визначити вимоги, які життєво необхідні для успішності продукту. Вони сформують так званий MVP, який можна надати користувачам. Решта вимог розділяються на «потрібні» та «бажані» з відповідними пріоритетами.

### 3.2.4.4 Визначення критеріїв ефективності досягнення очікувань зацікавлених сторін

Щоб зрозуміти чи проєкт є успішним в цілому, необхідно визначити його критерії успіху які визначають:

- чого він повинен досягти;
- як повинні бути виконані визначені вимоги, враховуючи:
  - очікування зацікавлених сторін;
  - програмні вимоги;
  - обмеженнями, що використовуються в рамках вимог високого рівня.

Зазвичай критерії успіху формалізуються мірами ефективності (Measures of Effectiveness, MOEs) – це міри успіху, які розроблені таким чином, щоб відповідати досягненню цілей системи, визначених очікуваннями зацікавлених сторін. Вони викладені з точки зору зацікавленої сторони та представляють критерії, які повинні бути виконані для того, щоб зацікавлена сторона вважала проєкт успішним [9].

Зрозуміло що в реальному світі кожен проєкт має бути не лише обмеженим у часі але й мати якісь контрольні часові точки з переліком вимог які повинні бути виконані для кожної з них. Це робиться задля більш прозорого розуміння прогресу та контролю за виконанням проєкту.

На цьому етапі повинна бути сформована чітко пов’язана структура високого рівня очікувань, вимог та обмежень проєкту яка у загальному варіанті може бути проілюстрована діаграмою потоку інформації, що базується на очікуваннях зацікавлених сторін (рис. 3.4), де пунктирними стрілками позначені шляхи перевірки елементів.

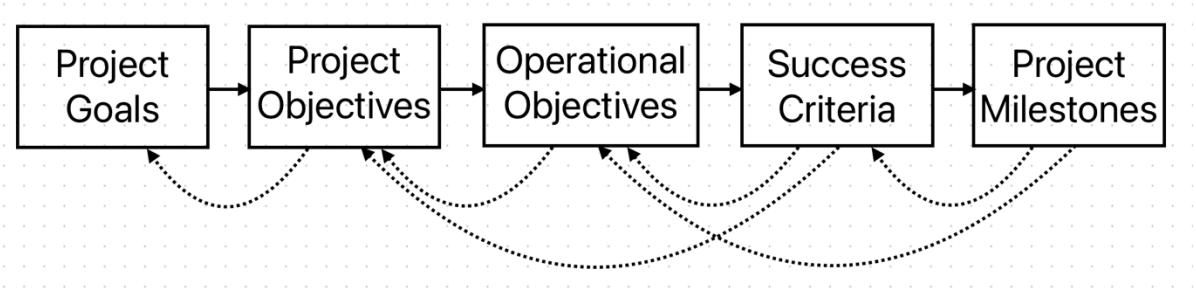


Рисунок 3.4 – Діаграма потоку інформації щодо очікувань стейкхолдерів

### 3.2.4.5 Фіксація бізнес-вимог

Після фіналізації та пріоритизації бізнес-вимог, визначення критеріїв їх досягнення, контрольних точок та успіху проєкту загалом потрібно виконати наступні дії маючи повний об’єм інформації [45]:

- проаналізувати вплив змін, що будуть реалізовані у проєкті;
- переконатися, що наслідки, які матиме ваш проєкт для існуючих процесів, продуктів і людей, повністю зрозумілі та прийнятні;
- усі конфліктні питання вирішені та узгоджені з усіма причетними зацікавленими сторонами.

Після усунення будь-яких прогалин або неоднозначностей і досягнення розуміння між технічною групою та зацікавленими сторонами щодо того, що саме очікується/призначається для системи/продукту, очікування бажано офіційно задокументувати. Це може відбувається у формі безпосередньо вимог, NGOs, критеріїв успіху проєкту та ключових точок (milestones) – бажано додавати усю інформацію, яка допоможе у реалізації проєкту. Вони можуть бути зафіксовані в документі, електронній таблиці, моделі чи іншій формі, яка відповідає продукту. Документація повинна фіксувати джерело очікувань, що залежно від розташування на рівні продукту можуть бути пов’язані з NGOs або вимогою продукту вищого рівня, стратегічними планами організації чи іншими джерелами [9].

Залежно від об’єму проєкту узгодження та фіналізація може відбуватися на одних зборах, коли весь об’єм інформації зібраний та оброблений (для невеликих

проєктів, де ця робота може бути виконана однією або декількома особами), або можуть бути потрібні мітинги після кожного етапу задля узгодження зібраної інформації та отриманих висновків з усіма робочими групами. У будь якому разі за деякий час до таких зустрічей тема обговорення та наявна інформація щодо неї повинна бути доведена до відома усіх учасників, та погоджена в цілому (наприклад, за допомогою агенди чи e-mail розсилання) – це дозволить обговоренню бути максимально конструктивним та ефективним. Саме узгодження може бути у будь якому форматі (залежно від рівня експертності та довіри щодо робочих груп) – презентація, дискусія або захист.

Типові результати для фіксації очікувань зацікавлених сторін включають наступне [9]:

- перевірені очікування зацікавлених сторін – це узгоджений набір очікувань для цього рівня продукту. Зазвичай вони фіксуються у формі потреб, цілей і задач із визначеними обмеженнями та припущеннями. Вони також можуть бути у формі моделей або інших графічних форм;
- концепція операцій – ConOps описує, як система буде працювати протягом фаз життєвого циклу, що відповідатиме очікуванням зацікавлених сторін. Він описує характеристики системи з операційної точки зору та допомагає полегшити розуміння цілей і завдань системи та очікувань інших зацікавлених сторін;
- стратегії впровадження та підтримки продукту – вони включають будь-які спеціальні положення, які можуть знадобитися для виготовлення, тестування, розгортання, підтримки операцій та утилізації кінцевого продукту. Вони визначають, яка підтримка буде потрібна, і будь-які сприятливі продукти, які потрібно буде розробити для створення кінцевого продукту;
- показники ефективності – на основі очікувань зацікавлених сторін розробляється набір MOEs. Це заходи, які представляють очікування, які мають вирішальне значення для успіху системи, і невиконання цих

заходів призведе до того, що зацікавлена сторона вважатиме систему неприйнятною;

Також може бути згенерований розподіл функцій людини/системи, що описує взаємодію апаратних і програмних систем з усім персоналом та їх допоміжною інфраструктурою. У багатьох проєктах людина-оператор є критично важливим компонентом загальної системи, і ролі та обов'язки людей у системі мають бути чітко усвідомлені. Це має включати всі взаємодії між людиною та системою, необхідні для проєкту [9].

## 3.3 Аналіз функційних та нефункційних вимог щодо ПЗ

Зафіксовані бізнес-вимоги транслюються у перелік функційних, що описують функції системи, та нефункційних вимог, які пояснюють обмеження та стримуючі фактори системи.

### 3.3.1 Специфікації функційних та нефункційних вимог

Під час процесу створення переліку функційних та нефункційних вимог окрім рекомендацій щодо кожної конкретної вимоги ,визначених у пункті 2.3 та додатку А, слід дотримуватися загальновизнаних кращих практик (best practices) щодо переліків вимог та специфікацій в цілому, які полягають в наступному [24, 25]:

- повнота – усі бізнес-вимоги повинні бути відображені функційними та/або нефункційними вимогами, жодні дані не повинні бути пропущені, тобто містити достатньою інформації для реалізації проєкту;
- узгодженість – жодна вимога не конфліктує з іншими вимогами такого ж типу або з високорівневими;
- трасованість – для кожної вимоги можна простежити як

першоджерела, так і реалізацію з метриками що дозволяють перевірити коректність реалізації;

- унітарність – кожна вимога визначає лише конкретні та унікальні потреби, тобто немає «злипання» декількох вимог в одну;
- здійсненність – вимоги мають відображати фактичний стан справ, включаючи вартість, терміни та технологію. Вони не повинні залежати від майбутніх технологічних досягнень;
- гнучкість – документ має бути достатньо адаптивним, щоб піддаватися змінам, бажано не включати зайвий матеріал який доведеться оновлювати щоразу, коли буде зміна;
- відсутність обмежень щодо впровадження – специфікація повинна бути достатньо детальною, щоб можна було виконати роботу, але не настільки конкретною, щоб зупинити розробку.

### 3.3.1 Аналіз специфікацій функційних та нефункційних вимог

Аналіз отриманих специфікацій повинен виконуватися у два етапи. На першому виконується рецензування (review) вимог на предмет виконання формальних рекомендацій наведених у пунктах 2.3, 3.3.1 та додатку А. Після узгодження усіх питань щодо форми та суті вимог на другому етапі потрібно детально розглянути, обговорити та визначити необхідні ресурси, що можуть бути визначені за допомогою експертного, статистичного методів та прототипування: кваліфікацію, час, інструменти та технічні засоби, потрібні для реалізації зазначених вимог. Особливу увагу на цьому етапі слід приділити можливим ризикам щодо цих ресурсів.

До аналізу специфікацій слід залучати відповідних бізнес-аналітиків, керівників та технічних спеціалістів з кожної причетної галузі, напрямку або технології. Однак слід уникати ситуацій, коли великий стек технологій представлений одним фахівцем, тому що людина може фокусуватися лише на

обмеженій кількості тем, що призводить до «випадання» окремих частин та технологій з обговорення.

Найприйнятнішим є формат обговорення та дискусії, бо дозволяє врахувати думку усіх учасників, отримати узгоджені та прийнятні для усіх сторін вимоги.

### 3.4 Планування розробки

Використовуючи дані, отримані в результаті визначення і аналізу специфікацій функціональних та нефункціональних вимог, приступають до планування подальших етапів життєвого циклу ПЗ, а саме розробки, тестування та впровадження.

Детальний план дуже залежить від моделі життєвого циклу, але для процесів загалом потрібно використовувати процесний підхід, детально розглянутий у пункті 2.4. Для кожного процесу потрібно побудувати відповідну діаграму, що поєднує та візуалізує вхідні дані та їх джерела, можливі засоби управління, контрольні точки з вихідними даними та їх отримувачів у рамках процесу, чітко фіксуючи його початкову та кінцеві точки. Такі діаграми будуються починаючи з контексту процесу (A00 рівень) та деталізуються до необхідного рівня (A1 або A2) – рис. 3.5.

Ці діаграми дозволяють визначити додаткові ризики пов’язані з послідовністю процесів, використанням різними процесами одних й тих самих ресурсів, наявності достатньої кількості ресурсів та взаємодією процесів.

При плануванні повинно бути визначено:

- послідовність початку й закінчення процесів, які процеси можуть виконуватися одночасно;
- розподіл ресурсів та керування між процесами;
- пріоритети процесів та розподілу ресурсів між ними;
- ризики, коли вони виникають, їх оцінка, дії та реакції щодо виникнення цих ризиків (попереджувальні, коригувальні дії чи ігнорування).

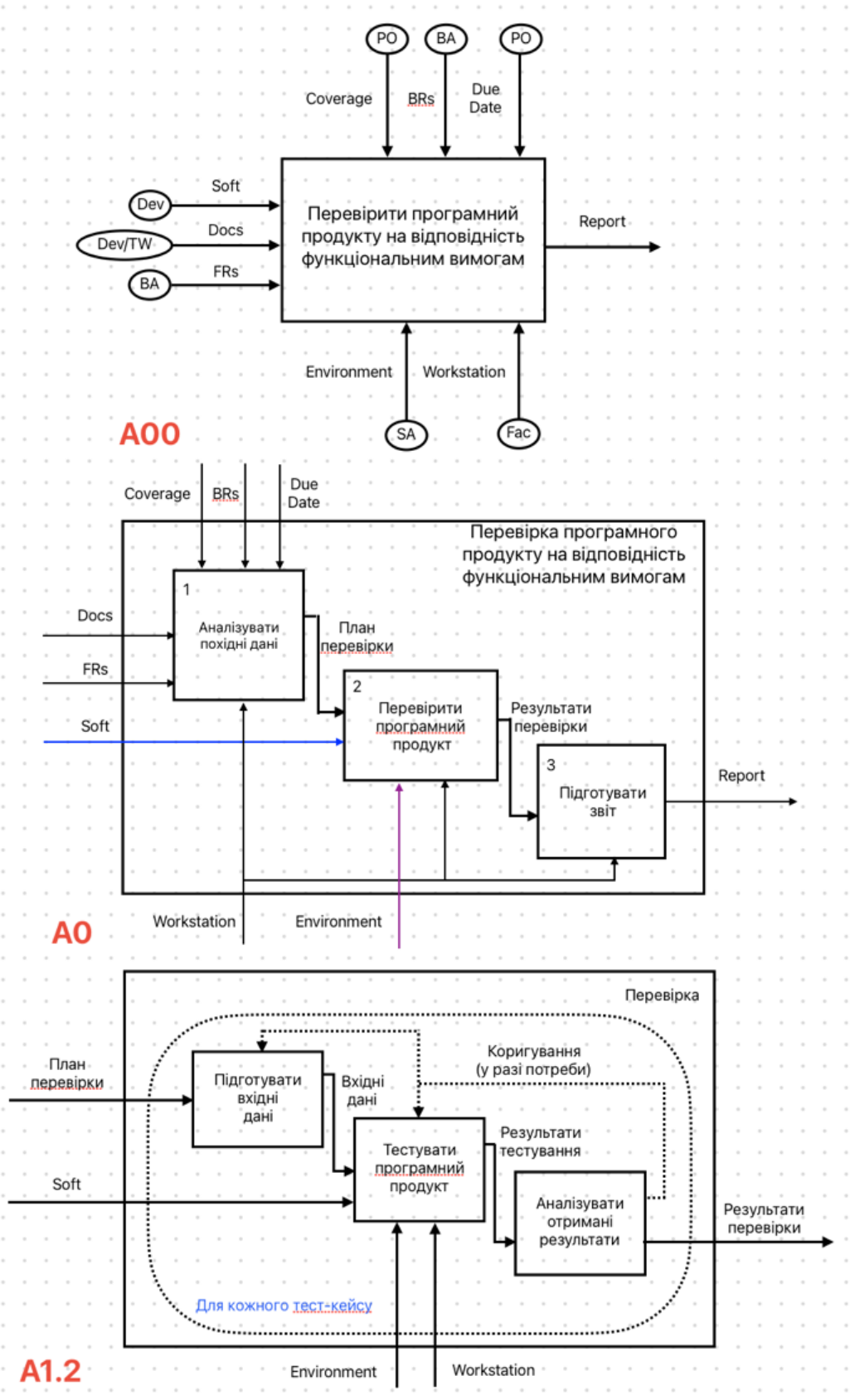


Рисунок 3.5 – Приклад деталізації процесу тестування за допомогою діаграм процесу

Задля оцінки якості планування потрібно пересвідчитися, що послідовність процесів на часовій шкалі не порушена та, в ідеальному варіанті, максимально оптимізована. Розподіл ресурсів виконано таким чином, щоб уникнути нестачі чи недоступності ресурсів, а кожен процес вчасно отримує необхідну кількість ресурсів. У разі нестачі ресурсів вони розподіляються згідно пріоритетів. В даному випадку термін «ресурси» слід розуміти максимально широко – він включає в себе співробітників з необхідною кваліфікацією, їх робочий час, технічні інструменти, доступ до необхідних систем та сервісів та робочі приміщення. Кожен ризик розглянуто та прийнято рішення та розроблено план на випадок його реалізації (тут треба пам’ятати що ризик це відхилення в обидва боки, тобто результат ризику може бути не лише негативним, а й позитивним).

## 3.5 Забезпечення якості на етапі безпосередньо розробки ПЗ

Під час розробки ПЗ є можливість уникнути багатьох можливих проблем, користуючись моделями якості даних та продукту, детально розглянутих у пункті 2.5.

### 3.5.1 Дотримання вимог щодо якості даних

Оскільки програмний код оперує, тобто виконує операції з даними, надважливим є дотримуватися вимог щодо якості даних. На жаль, сьогодні існує велике розмаїття усіляких систем зберігання даних, що накладають відповідні обмеження на формат, структуру даних та зв’язки між ними. Також для деяких таких систем можуть існувати автоматизовані механізми перевірки відповідності даних згідно усіх або деяких параметрів, наведених у моделі якості даних.

У разі відсутності чи неможливості застосування таких механізмів виконувати перевірку щодо дотримання вимог наведених у моделі якості даних слід довіряти одному чи декільком фахівцям з відповідною кваліфікацією та досвідом роботи з системами, що використовуються для зберігання даних.

На початку з’ясовується, які характеристики з моделі якості даних актуальні та мають бути дотримані у даному проєкті, зважаючи на системи зберігання, механізми передачі та обробки даних (можуть бути окремі набори для кожного з них). Кожна з характеристик оцінюється за бінарною шкалою «виконується» чи «1» та «не виконується» чи «0». Якість кожного набору оцінюється як кон’юнкція усіх характеристик, тобто дорівнює «0», якщо хоча б одна з них не виконується

$$Q_{ds} = \Lambda_{i=1}^{m} q_i. \tag{3.5}$$

Загальна якість даних оцінюється як комплексний показник якості усіх наборів з урахуванням вагових коефіцієнтів

$$Q = \sum_{j=1}^{n} P_j Q_{ds\,j}, \tag{3.6}$$

де $P_j$ –ваговий коефіцієнт j-го набору.

У разі однаковості, відсутності або невизначеності вагових коефіцієнтів – як середнє арифметичне якості усіх наборів

$$Q = \frac{\sum_{j=1}^{n} Q_{ds\,j}}{n}. \tag{3.7}$$

### 3.5.1 Дотримання вимог щодо якості продукту

В залежності від обраної моделі життєвого циклу розробки можуть використовуватися різноманітні методи забезпечення якості вихідного програмного коду.

### 3.5.1 Методи підвищення якості програмного коду

На додачу до спеціальних методів слід використовувати й універсальні методи, що не залежать від обраної моделі життєвого циклу розробки, а по суті є методами організації виробничого процесу.

Під час розробки рекомендується використовувати хоча б деякі елементи парного програмування (pair programming), яке детально розглянуто у пункті 2.6.1. Головне щоб у проєкті було задіяно щонайменше два програмісти – це знижує ризики на випадок звільнення, відпустки чи переходу до іншої команди або проєкту розробників.

Обов’язково повинна бути запроваджена культура рецензування програмного коду (code review), що не лише підвищує його якість, але й захищає від можливих програмних вкладок та вбудованих бекдорів (backdoor).

У випадку, коли проєкт не є одноразовим та надто простим, а є комплексним та має на увазі подальший розвиток та підтримку, впровадження модульного тестування (unit tests) на етапі розробки є життєвою необхідністю. Для кожного модуля (дуже часто під цим розуміють окремі функції у програмному коді) повинні бути створені відповідні тести (unit test) що мають бути успішно пройдені до того як цей модуль буде вважатися готовим. Тут потрібно знайти баланс між важливістю тестованих модулів та ресурсами, потрібними для кодування юніт-тестів.

Пропонується використовувати наступний підхід:

1) для кожного модуля (фунції) повинен бути написаний щонайменше один тест що його ініціалізує (викликає);
2) в залежності від частоти використання, важливості та/або критичності модуля розробляється набір юніт-тестів використовуючи методи тестування білої скриньки, розглянуті у пункті 2.6.3;
3) для найважливіших, найкритичніших та модулів що мають складну логіку потрібно розробити набір юніт-тестів, що базується на методі

комбінаторного покриття умов (детально розглянутий у пункті 2.6.3.4).

На додаток до перелічених вище заходів рекомендується використовувати статичний аналіз програмного коду (static code analysis), або взагалі його аудит, що дозволяє виявити наступні проблеми:

- використання невідповідних сторонніх бібліотек (thirdparties);
- використання блоків програмного коду, захищених авторським правом;
- використання неефективного програмного коду;
- використання поганих практик програмування;
- потенційні витоки пам’яті (memory leaks), переповнення стеку (stack overflow), тощо;
- порушення правил іменування (naming conventions);
- використання заборонених назв чи ненормативної лексики.

Також не слід забувати про обмін знаннями (knowledge sharing), де окрім очевидних презентацій, демонстрацій (demo), сесій з передачі знань та тренінгів особливу важливість мають артефакти у вигляді технічного дизайну, структури даних, діаграм потоку даних (data flow diagram) тощо. Бо вони будуть використовуватися й самою командою проєкту під час підключення нових членів команди, внесення змін та підтримки продукту.

### 3.5.2 Оцінювання якості програмного коду

Використовуючи специфікації функційних та нефункційних вимог та модель якості продукту, потрібно визначитися які характеристики та підхарактеристики якості має сенс вимірювати на цьому етапі.

Для кожної такої характеристики чи підхарактеристики потрібно розрахувати її кількісний показник якості. Для функційної повноти у найпростішому варіанті пропонується використовувати коефіцієнт вето:

$$\begin{cases} 0, \text{якщо } MVP \text{ не виконано} \\ 0.5, \text{у разі реалізовано більше ніж потребує } MVP, \\ \text{але менше ніж бажаний рівень} \\ 1, \text{якщо реалізовано бажаний рівень.} \end{cases} \quad (3.8)$$

Для характеристик та підхарактеристик, де визначені лише якісні показники, потрібно користуватися бінарною шкалою «виконується» чи «1» та «не виконується» чи «0».

Для характеристик та підхарактеристик, де визначені кількісні базові (необхідні) показники, використовуються розширені диференційні принципи визначення рівня якості

$$q = \begin{cases} 0, \text{якщо } P_{min} \text{ задано та } P < P_{min} \\ \frac{P}{P_b} \\ 1, \text{якщо } \frac{P}{P_b} > 1 \end{cases} \quad (3.9)$$

$$q = \begin{cases} 0, \text{якщо} P_{max} \text{ задано та } P > P_{max} \\ \frac{P_b}{P} \\ 1, \text{якщо } \frac{P_b}{P} > 1 \end{cases} \quad (3.10)$$

Формула (3.9) використовується у випадку коли краще, якщо значення показника більше, а (3.10) – в противному випадку.

Для критичних характеристик та підхарактеристик слід застосовувати коефіцієнти вето (принцип наведений у формулі (3.1)).

Якщо потрібно, для кожної характеристики та/або підхарактеристики вводяться вагові коефіцієнти $m_i$, дотримуючись принципу:

$$\sum_{i=1}^{n} m_i = 1. \quad (3.11)$$

Розрахунок комплексного показника якості даного етапу відбувається за формулою

$$Q = \sum_{i=1}^{n} m_i q_i, \tag{3.12}$$

(використовуючи рівні якості кожної з характеристик), але спочатку потрібно розрахувати рівні якості характеристик, що складаються з підхарактеристик (використовуючи ту саму формулу).

У разі, якщо вагові коефіцієнти не застосовуються, $m_i = 1$.

### 3.6 Тестування ПЗ

Коли програмний засіб передано у тестування, що залежить від застосованої моделі життєвого циклу розробки. Що може бути досягненням визначеного рівня показника якості чи якогось формального моменту на етапі розробки, для agile-моделей розробка та тестування можуть, взагалі, відбуватись умовно паралельно. Застосовуються усі необхідні та погоджені види тестування.

Через те, що в результаті розробки уточнюється чи додається достатньо великий об’єм інформації щодо реальної реалізації: використані технології, технічні рішення та обмеження, перед початком безпосередньо тестування потрібно переглянути актуалізувати попередні плани, спираючись на отриману інформацію.

Під час фази тестування користуючись усіма трьома моделями якості (даних, продукту та якості під час застосування), що розглянуті у пункті 2.5, перевіряються характеристики та підхарактеристики якості, що неможливо було визначити або були недостатньо визначені на етапі розробки ПЗ.

У загальному випадку наступні види тестування мають бути виконані:

- функційне тестування;
- інтеграційне тестування внутрішніх модулів;

- інтеграційне тестування із зовнішніми системами;
- визначені види нефункційного тестування, наприклад:
    - PSR/Volume/Loading;
    - тестування розгортання/roll-back;
    - тестування даних для моніторингу;
    - рецензування внутрішньої та, особливо, зовнішньої документації.

Особливу увагу потрібно звернути на якість під час застосування, тому що тестування – перший етап, де ПЗ оцінюється в цілому з точки зору кінцевого користувача. Для цього потрібно мати на увазі не лише технічні, але й бізнес-вимоги з очікуваннями та потребами зацікавлених сторін. Це дозволить уникнути ситуації, коли усі вимоги виконані формально, але кінцевий ПЗ не відповідає очікуванням та потребам стейкхолдерів. Теоретично для запобігання цій ситуації можна використовувати V-модель, але на практиці все дуже сильно залежить від інженерів з якості та тестувальників, які проводять end-to-end перевірки.

Для кількісної оцінки загального рівня якості потрібно використовувати підходи, використані у пункті 3.5.2, але застосовуючи їх до всіх визначених характеристик та підхарактеристик. Також потрібно мати на увазі, що з точки зору тестування ці характеристики та підхарактеристики можуть визначатися іншими рівнями показників якості. Наприклад, під час розробки, дотримання вимог доступності (accessibility) може визначатися відношенням кількості реалізованих вимог до загальної кількості вимог (диференційний), а під час тестування мати значення може лише факт дотримання усіх таких вимог (бінарний).

## 3.7 Супровід ПЗ. Розгортання і моніторинг

У загальному випадку супровід розглядається як розгортання та процес покращення, оптимізації та виправлення дефектів у програмному забезпеченні

протягом його експлуатації. Діяльність з покращення, оптимізації та виправлення дефектів повинна проходити по усім етапам життєвого циклу, що вказані для ПЗ, тому до неї застосовуються усі моменти розглянуті в цій роботі, з урахуванням об’єму роботи та доцільності. Тому розглянемо окремо лише розгортання та моніторинг.

### 3.7.1 Рекомендації щодо забезпечення якості процесу розгортання

Процес розгортання або впровадження (сьогодення ці терміни використовуються як синоніми) залежить від розміру і складності ПЗ. У випадку коли це додаток чи програма, яку користувач встановлює на особистий пристрій, достатньо забезпечити чітку інструкцію з встановлення з ілюстраціями або, навіть, відео, навести рекомендації та надати можливість вирішити виникаючі проблеми за допомогою наступних методів (можуть використовуватися як поодинці, так й усі разом):

- відповіді на поширені запитання (FAQ);
- механізм автоматичного виправлення помилок (troubleshooting);
- оператор-консультант.

Сьогодення, з поширенням штучного інтелекту (artificial intelligence, AI) розповсюдженою практикою є використання комбінованих механізмів з використанням AI (користувач спочатку спілкується з AI, а у разі неможливості вирішення питання на цьому рівні він перенаправляється до людини-консультанта).

У випадках комплексної системи, дуже часто сьогодні це розподілені системи з використанням хмарних технологій (cloud-based), перед початком розгортання потрібно виробити детальний план розгортання:

- необхідні операції;
- механізми, програми та методики, що будуть задіяні для кожної операції;

- ресурси потрібні для кожної операції;
- час потрібний для кожної операції;
- маркери що будуть вказувати на успіх/неуспіх кожної операції;
- послідовність операцій;
- відповідальні за кожну операцію, етап та процес загалом;
- якім чином можна визначити що ПЗ інтегрований з усіма необхідними системами;
- якім чином можна визначити що ПЗ встановлений повністю та знаходиться в робочому стані;
- дії у разі передбачуваних негативних сценаріїв;
- дії у разі непередбачуваних ситуацій.

Протягом процедури розгортання дотримуватися цього плану чітко контролюючи усі операції, етапи та дотримання цього плану.

Загальним принципом розгортання будь-якого ПЗ повинна бути транзакційність – ПЗ або повинен бути коректно встановленим й знаходитися у робочому стани, або у випадку якихось відмов чи непередбачуваних ситуацій система повинна бути повернута у стан до початку процедури розгортання.

### 3.7.2 Рекомендації щодо моніторингу

На базовому рівні логування, метрики та системи сповіщень розглянуті у пункті 2.8, тут наведені загальні рекомендації щодо організації та принципів використання цих сутностей.

Під час ведення логів необхідно враховувати баланс між деталізацією логування та продуктивністю системи. Занадто докладний логінг може призвести до зниження продуктивності, тому необхідно визначити оптимальний рівень деталізації. Для цього слід визначитися у двох моментах: для яких цілей будуть використовуватися лог-файли, і який рівень деталізації не призведе до зниження продуктивності [17].

Бажано застосовувати до логованої інформації модель якості даних (детально розглянута у пункті 2.5.1). Інформація щодо операцій користувача та їх послідовності у системі повинна бути простежуваною, так само як і внутрішні виклики модулів всередині ПЗ. Також потрібно приховувати або анонімізувати конфіденційну інформацію та чутливі дані, чи взагалі не записувати їх. Та застосовувати правила доступу до лог-файлів, щоб обмежити доступ лише до необхідної інформації. А впровадження унікальності лог-записів (для кожної операції генерується лише один запис) дозволяє суттєво зменшити потрібний об’єм дискового простору для збереження лог-файлів та підвищити швидкість їх аналізу. Гарним критерієм є: «якщо лог-запис не може бути використаний для дослідження або мониторінгу, він не повинен бути згенерованим».

При виборі методології метрик (наведені у пункті 2.8.2) слід розуміти, що в залежності від реалізації, мети та типу для кожного компоненту системи може бути необхідним збирати окремі метрики. Наприклад, для веб- або application-сервера краще підходять RED та LTES, для шини даних або обробників черг завдань – USE [31].

При настроюванні систем моніторингу потрібно розуміти, що SLO є "бажаним" станом, тому встановлювати його пороговим значенням для сповіщень не є гарною ідеєю, бо не все, що від нього відрізняється, обов'язково є позаштатною ситуацією. З великою імовірністю це може призвести до появи так званого «алертного шуму» – великої кількості оповіщень при цілком адекватній роботі системи. Але у протилежному випадку є ризик отримати алерт тільки в момент порушення SLA [31].

Бажано дотримуватися принципу: алерт має спрацьовувати не тоді, коли "все вже дуже погано", і не від кожної "перешкоди", а тоді, коли виникла проблема, але ще можна щось виправити. На практиці, щоб досягти цього балансу, внутрішні алерти найкраще встановлювати на значення, що знаходиться між SLO і Error Budget – коли поведінку системи ще можна назвати штатною, але якщо нічого не зробити, є ризик вийти за межі SLA [31].

### 3.7.2 Моніторинг як засіб підвищення стабільності системи та задоволеності зацікавлених сторін

Впроваджуючи моніторинг для ПЗ, слід орієнтуватися не лише на розглянуті вище аспекти як попередження проблемних ситуацій чи дослідження дефектів. Потрібно мати на увазі що моніторинг – це потужний елемент ідей для подальшого покращення ПЗ, підвищення стабільності та задоволеності стейкхолдерів. За допомогою нього потрібно відслідковувати тренди використання ресурсів системи та впроваджувати коригувальні дії задля уникнення її відмови. Це може бути як редизайн чи рефакторінг самої системи, так і збільшення доступних ресурсів. Також за допомогою аналізу лог-записів можна визначити реальні сценарії роботи користувачів і впровадити нові можливості (feature) ПЗ, які нададуть додаткові конкурентні переваги. Такі заходи значно підвищують стабільність, функціональну повноту та юзабіліті ПЗ, що позитивно відображається на загальному рівні його якості.

При аналізі найпоширеніших запитів до системи допомоги можна визначити незрозумілі для користувачів елементи системи та переробити їх задля підвищення очевидності, або надати додаткові начальні матеріали задля підвищення обізнаності користувачів чи оптимізувати шлях до отримання рішення такої проблеми у системі допомоги. Завдяки системі моніторингу команда проєкту може ідентифікувати наявні проблеми до того як з ними зіткнуться користувачі і розробити та впровадити її вирішення до того як користувачі поскаржаться на неї, чи значно зменшити проміжок часу від подання скарги користувачем до її вирішення. Ці заходи підвищують задоволеність та лояльність зацікавлених сторін, що в решті решт позитивно позначається на очікуваннях стейкходерів.

# ВИСНОВКИ

У роботі розглянуті сучасні теоретичні та практичні тенденції з забезпечення якості протягом життєвого циклу ПЗ. Детально розглядається кожний етап цього циклу, проводиться аналіз публікацій та наводяться рекомендації щодо підвищення якості процесів, та, де це застосовно, наводяться градації, метрики та формули задля пріоритизації, вимірювання та розрахунку показників якості.

У роботі розкриваються декілька ідей. Однією з них є впровадження механізмів забезпечення та контролю якості на якомога більш ранніх етапах життєвого циклу ПЗ, починаючи з валідації самої бізнес-ідеї. Чим раніше будуть виявлені потенційні проблеми, невідповідності або прогалини, тим дешевше їх виправляти або, взагалі, уникати.

Наступним аспектом є те що якість, хоча і стає життєвонеобхідною, але все ж ще залишається однією з характеристик проєкту, продукту чи послуги. Тому вона не може розцінюватися як самоціль та не повинна обтяжувати, гальмувати, або, взагалі, зупиняти процеси життєвого циклу ПЗ. Задля дотримання цього підходу у роботі наведені різні варіанти (від простих до комплексних) щодо забезпечення якості, у залежності від проєкту потрібно застосовувати ті, що даватимуть найбільший ефект, або їх комбінацію.

Також у роботі зроблений акцент на покращенні процесів протягом усього життєвого циклу ПЗ, тому що це дає сталий позитивний ефект з точки зору забезпечення якості.

Загалом у роботі наведені рамкові механізми забезпечення якості протягом життєвого циклу ПЗ, основний акцент зроблено на найменш формалізовані етапи життєвого циклу, такі як валідація бізнес-ідеї, визначення очікувань зацікавлених сторін, формування усіх типів вимог та використання моніторингу задля подальшого покращення ПЗ.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ ISO 9000:2015 Системи управління якістю. Основні положення та словник термінів (ISO 9000:2015, IDT)

2. ДСТУ ISO/IEC 25010:2016 Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів (ISO/IEC 25010:2011, IDT)

3. Запорожець О.В. Курс лекцій з дисципліни «Тестування та оцінювання якості програмних засобів»: Харків, ХНУРЕ, 2023.

4. Бизнес идея — что это? Понятие и суть идеи для бизнеса. Бізнес Маркет. URL: https://biznesmarket.com.ua/biznes-idei/biznes-ideya-chto-eto-ponyatie-i-sut-idei-dlya-biznesa/

5. Как оценить бизнес-идею? Основные вопросы и ответы. Бізнес Маркет. URL: https://biznesmarket.com.ua/biznes-idei/kak-otsenit-biznes-ideyu-osnovnye-voprosy-i-otvety/

6. Стейкхолдери проєкту: хто такі та чому важливо налагодити з ними комунікацію. wizeclub. URL: https://wizeclub.education/blog/stejkholderi-proyektu-hto-taki-ta-chomu-vazhlivo-nalagoditi-z-nimi-komunikatsiyu/

7. Зацікавлені строни. Що таке стейкхолдер? LibreTexts. URL: https://ukrayinska.libretexts.org/%D0%91%D1%96%D0%B7%D0%BD%D0%B5%D1%81/%D0%91%D1%96%D0%B7%D0%BD%D0%B5%D1%81/%D0%92%D1%81%D1%82%D1%83%D0%BF%D0%BD%D0%B8%D0%B9_%D0%B1%D1%96%D0%B7%D0%BD%D0%B5%D1%81/%D0%9A%D0%BD%D0%B8%D0%B3%D0%B0%3A_%D0%92%D1%81%D1%82%D1%83%D0%BF_%D0%B4%D0%BE_%D0%B1%D1%96%D0%B7%D0%BD%D0%B5%D1%81%D1%83_(Lumen)/01%3A_%D0%A0%D0%BE%D0%BB%D1%8C_%D0%B1%D1%96%D0%B7%D0%BD%D0%B5%D1%81%D1%83/1.06%3A_%D0%97%D0%B0%D1%86%D1%96%D0%BA%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D1%96_%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%BD%D0%B8

8. Jingjing Yang. A Framework of User Classification Model of Online User Innovation Communities Based on User Innovation Value / Open Journal of Social Sciences, Vol.8 No.5, 2020. URL: https://www.scirp.org/journal/paperinformation?paperid=100366

9. Stakeholder Expectations Definition. NASA. URL: https://www.nasa.gov/reference/4-1-stakeholder-expectations-definition/

10. Laura Olkkonen. Stakeholder Expectations. Conceptual Foundations and Empirical Analysis. JYVÄSKYLÄ 2015. URL: https://jyx.jyu.fi/bitstream/handle/123456789/47685/978-951-39-6386-6_vaitos28112015.pdf?isAllowed=y&sequence=1

11. АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. baklaniv. URL: http://baklaniv.at.ua/ANALIZ_VYMOG/lekcija_1-2.pdf

12. Л.А. Люшенко Я.В. Хіцко. РОЗРОБКА ТА АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. КОМПОНЕНТИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ КУРСОВЕ ПРОЄКТУВАННЯ. URL: https://ela.kpi.ua/bitstream/123456789/38101/1/Rozrobka_ta_analiz_KP.pdf

13. Best Practices for Code Review. SMARTBEAR. URL: https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

14. Тестування програмного забезпечення. Wikipedia. URL: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F

15. Леонов О. Тестування ПЗ (види тестування). URL: https://drukarnia.com.ua/articles/testuvannya-pz-vidi-testuvannya-JInS1

16. Розгортання програмного забезпечення. Wikipedia. URL: https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D0%B3%D0%BE%D1%80%D1%82%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE

_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F

17. Лог-файли як невід'ємна частина процесу розробки. foxminded. URL: https://foxminded.ua/shcho-take-loh-faily/

18. Штефан Н.В. Курс лекцій з дисципліни «Кваліметрія»: Харків, ХНУРЕ, 2022

19. Carlos Eduardo Martins Serra. Stakeholders Analysis: Power/Influence-Interest Matrix. Projectizing. URL: https://projectizing.com/stakeholders-analysis-powerinfluence-interest-matrix/

20. Mohamed Sami. Stakeholders Management, WHAT, WHY, and HOW? melsatar. URL: https://melsatar.blog/2018/03/07/stakeholders-management-what-why-and-how/

21. Mahesh. What is Plan-Do-Check-Act (PDCA) Cycle? Definition, Steps, and Pros/Cons. BokasTutor. URL: https://bokastutor.com/pdca-cycle/

22. What is pair programming? codementor. URL: https://www.codementor.io/pair-programming

23. Pair Programming. geeksforgeeks. URL: https://www.geeksforgeeks.org/pair-programming/

24. Козак О.Л. Опорний конспект лекцій з курсу «Аналіз вимог до програмного забезпечення». URL: http://dspace.wunu.edu.ua/retrieve/14135/FCIT_kKN_sPZS_dAVPZ_%20LEC.pdf

25. Що таке специфікація вимог: визначення, найкращі інструменти та методи | Гід. visure. URL: https://visuresolutions.com/uk/blog/requirements-specification/

26. Як оцінити перспективність вашої бізнес-ідеї: 10 факторів. Менеджмент@БЛОГ. URL: https://www.management.com.ua/blog/1864

27. ЗАЦІКАВЛЕНА СТОРОНА. Фінансова енциклопедія. URL: https://ua.nesrakonk.ru/stakeholder/

28. What Are Business Requirements? (With Tips And FAQs). indeed. URL: https://in.indeed.com/career-advice/career-development/business-requirements

29. НЕФУНКЦІОНАЛЬНІ ВИМОГИ: ПРИКЛАДИ, ТИПИ, ПІДХОДИ. QATestLab. URL: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/

30. ОГЛЯД ВИДІВ ТЕСТУВАННЯ. QATestLab. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/

31. Спорышев С. Мониторинг начинается с метрик, или Как не сделать из алертов белый шум. itsumma. URL: https://www.itsumma.ru/blog/alerts

32. Хто такі стейкхолдери і як із ними дружити. BrainRain. URL: https://brainrain.com.ua/uk/hto-taki-steykholderi-ua

33. What Is Pair Programming? Codecademy. URL: https://www.codecademy.com/resources/blog/what-is-pair-programming/

34. ДОСЛІДЖЕННЯ РИНКУ ТА СТРАТЕГІЯ ДЛЯ БІЗНЕС-ІДЕЇ. PRO CONSULTING. URL: https://pro-consulting.ua/ua/services/issledovanie-rynka-i-strategiya-dlya-biznes-idei

35. Зацікавлені строни. Аналіз зацікавлених сторін. LibreTexts. URL: https://ukrayinska.libretexts.org/%D0%91%D1%96%D0%B7%D0%BD%D0%B5%D1%81/%D0%9C%D0%B5%D0%BD%D0%B5%D0%B4%D0%B6%D0%BC%D0%B5%D0%BD%D1%82/%D0%9A%D0%BD%D0%B8%D0%B3%D0%B0%3A_%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF%D0%B8_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%8F/04%3A_%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BC%D1%96%D1%81%D1%96%D1%97%2C_%D0%B1%D0%B0%D1%87%D0%B5%D0%BD%D0%BD%D1%8F_%D1%82%D0%B0_%D1%86%D1%96%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9/04.6%3A_%D0%97%D0%B0%D1%86%D1%96%D0%BA%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D1%96_%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%BD%D0%B8

36. What are business requirements? educative. URL: https://www.educative.io/answers/what-are-business-requirements

37. Вичугова А. Показатели vs метрики (не только продуктовые): отличия и примеры для бизнес-аналитика с комментариями из BABOK®Guide. Babok-school. URL: https://babok-school.ru/blog/metrics-and-kpi-technique-from-babok-and-product-analytics/

38. Zosym Maxym. Значущість зацікавлених сторін (Stakeholder Salience). maxzosim. https://www.maxzosim.com/stakeholder-salience/

39. Pair programming. Wikipedia. URL: https://en.wikipedia.org/wiki/Pair_programming

40. How To Test A Business Idea. Forbes. URL: https://www.forbes.com/sites/mikekappel/2017/09/20/how-to-test-a-business-idea/?sh=75ee16b2635b

41. Стерненко М. Стейкхолдери: що таке і чому важливі. Smartik. URL: https://smartik.kiev.ua/stejkkholdery-shcho-take-i-chomu-vazhlyvi/

42. Business Requirements Analysis. MindTools. URL: https://www.mindtools.com/aa5la5e/business-requirements-analysis

43. Minimum viable product. Wikiedia. URL: https://en.wikipedia.org/wiki/Minimum_viable_product

44. Foster Patrick. How to Test a Business Idea. Patriot. URL: https://smallbusiness.patriotsoftware.com/how-test-business-idea-viability/?utm_source=FRB45&utm_medium=article&utm_campaign=GBL&utm_content=test_bizidea

45. Top tips for writing the perfect business requirements document. Lucidchart. URL: https://www.lucidchart.com/blog/tips-for-a-perfect-business-requirements-document

46. 10 Ways to Evaluate a New Business Idea. dummies. URL: https://www.dummies.com/article/business-careers-money/business/small-business/start-ups/10-ways-to-evaluate-a-new-business-idea-179321/

47. Trekhov Vladimir. How to Evaluate a Business Idea: Detailed Checklist. ATTRACT GROUP. URL: https://attractgroup.com/blog/how-to-evaluate-a-business-idea-detailed-checklist/#