ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «УПРАВЛІННЯ ПРОЦЕСАМИ ТЕСТУВАННЯМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ AGILE ПІДХОДІВ»

Ступінь вищої освіти – магістр
Спеціальність – 073 «Менеджмент»
Освітня програма «Agile-технології розробки програмного забезпечення»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Керівник: к.т.н., доцент
Веніамін ГІТІС

Керівник: викладач
Олег МУШИНСЬКИЙ

Виконав: здобувач групи
МЕН/Agile-23м
Василь СТАВИЦЬКИЙ

Київ, 2024 р.

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри інформаційного менеджменту, математики та статистики
_____Денис БАЛДИК
«__»____20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
СТАВИЦЬКИЙ ВАСИЛЬ ОЛЕКСІЙОВИЧ

Тема роботи	Управління процесами тестуванням програмного забезпечення з використанням agile підходів
Номер та дата наказу про затвердження теми	№56-3 від 27.06.2024
Коротка постановка завдання	Впровадження гнучкого підходу до управління тестуванням програмного забезпечення
Посилання на джерела інформації (не більше п’яти найменувань, які рекомендує науковий керівник)	Fowler, Martin, and Kent Beck. "Refactoring: Improving the design of existing code." 11th European Conference. Jyväskylä, Finland. 1997. Capiluppi, Andrea, et al. "An empirical study of the evolution of an agile-developed software system." 29th International Conference on Software Engineering (ICSE'07). IEEE, 2007.
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне та/або практичне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблематику в галузі управління та адміністрування, яка характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів Agile технологій.

Дата видачі завдання «14» липня 2024 р

Керівник Веніамін ГІТІС

Керівник Олег МУШИНСЬКИЙ

Здобувач Василь СТАВИЦЬКИЙ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
Підготовчий етап			
1	Вибір напряму дослідження та керівника	01.07.2024 р.	Виконано
2	Формування теми та призначення керівника	08.07.2024 р.	Виконано
3	Затвердження теми кваліфікаційної роботи	09.07.2024 р.	Виконано
4	Затвердження завдання на кваліфікаційну роботу	15.07.2024 р.	Виконано
Основний етап			
5	Розробка концепції кваліфікаційної роботи	22.07.2024 р.	Виконано
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів.	29.07.2024 р.	Виконано
7	Затвердження розширеної постановки завдання. Підготовка та подання керівнику розділу 1 кваліфікаційної роботи	18.09.2024 р.	Виконано
8	Проєктування інформаційної системи. Підготовка та подання керівнику розділу 2 кваліфікаційноїроботи	18.09.2024 р.	Виконано
9	Реалізація інформаційної системи. Підготовка та подання керівнику розділу 3 кваліфікаційної роботи	25.09.2024 р.	Виконано
10	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	01.10.2024 р.	Виконано
11	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівнику доопрацьованого варіанту кваліфікаційної роботи	04.10.2024 р.	Виконано
Завершальний етап			
12	Представлення рукопису для перевірки на плагіат	07.10.2024 р.	Виконано
13	Підготовка презентації та доповіді на передзахист	07.10.2024 р.	Виконано
14	Передзахист кваліфікаційної роботи	08-11.10.2024 р.	Виконано
15	Технічна самоекспертиза роботи на відповідність вимогам до оформлення та виправлення недоліків	08-11.10.2024 р.	Виконано
16	Експертиза роботи керівником та зовнішнім експертом	14.10.2024 р.	Виконано
17	Доопрацювання доповіді та презентації для захисту	18.10.2024 р.	Виконано
18	Захист кваліфікаційної роботи	21-25.10.2024 р.	Виконано

Керівник Веніамін ГІТІС

Керівник Олег МУШИНСЬКИЙ

Здобувач Василь СТАВИЦЬКИЙ

Ставицький В.О. Управління процесами тестуванням програмного забезпечення з використанням Agile підходів.

Кваліфікаційна випускна робота на здобуття ступеня вищої освіти магістра за спеціальністю 073 – Менеджмент. – ВНЗ «Університет економіки та права «КРОК», Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра математичних методів та статистики, Київ, 2024.

У кваліфікаційній роботі досліджено актуальну проблему в управлінні процесами тестування програмного забезпечення з використанням Agile підходів, використовуючи методи та інструменти гнучкого управління проєктами. Проведено аналіз ключових проблем, що виникають під час інтеграції тестування в гнучкі процеси розробки, та запропоновано шляхи їх оптимізації для досягнення вищої якості продукту.

Ключові слова: менеджмент, управління, гнучке управління, Agile менеджмент, гнучка розробка програмного забезпечення, планування, спринт, ретроспектива, фреймворк Scrum, Agile-технології тестування, Рефакторинг, Екстремальне програмування (XP).

Табл. 2. Рис. 8. Бібліограф.: 49 найм.

Stavytskyi V.O. Software Testing Process Management Using Agile Approaches.

Qualifying final work for obtaining a master's degree in higher education by specialty 073 – Management. – «KROK» University, Educational and Scientific Institute of information and communication technologies, Department of Mathematical Methods and Statistics, Kyiv, 2024.

This qualification paper explores the relevant issue of managing software testing processes using Agile approaches, employing methods and tools of flexible

project management. The analysis addresses key challenges that arise during the integration of testing into Agile development processes and proposes optimization solutions to achieve higher product quality.

Keywords**:** management, process management, Agile management, flexible software development, planning, sprint, retrospective, Scrum framework, Agile testing technologies, refactoring, Extreme Programming (XP). Tabl. 2. Fig. 8. Bibliography: 49 Items.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасна розробка програмного забезпечення стрімко розвивається, вимагаючи впровадження інноваційних підходів до управління проектами та забезпечення якості кінцевого продукту. Управління процесами тестування програмного забезпечення (Software Testing Management) є ключовим елементом забезпечення якості (QA) у процесі розробки ПЗ. Його мета – гарантувати, що програмний продукт відповідає вимогам замовника, працює безпомилково та готовий до використання в реальних умовах.

Забезпечення якості, орієнтоване на тестування, охоплює всі аспекти управління якістю: планування, проведення тестів, а також аналіз отриманих результатів. Це дає змогу не тільки виявляти потенційні проблеми, але й запобігати їх появі, зменшуючи ризик критичних помилок на всіх етапах розробки. Використання як ручного, так і автоматизованого тестування дозволяє повністю контролювати якість продукту. Основною метою тестування в SQA є зниження ймовірності помилок і забезпечення стабільності та надійності програмного забезпечення.

Управління процесами тестування стає критично важливим для успішної реалізації Agile проектів, оскільки це дозволяє своєчасно виявляти дефекти, знижувати ризики та підвищувати загальну якість продукту. Використання автоматизованого тестування, DevOps-практик, а також постійне вдосконалення процесів тестування допомагають командами досягати високої продуктивності.

Тісний зв'язок тестування з іншими аспектами забезпечення якості робить процес створення програмного забезпечення більш структурованим і керованим. Автоматизоване тестування є регресійним тестуванням, що значно прискорює перевірку великих компонентів, тоді як ручне тестування дозволяє глибше аналізувати нові складні сценарії. Такий комплексний підхід

не тільки підвищує якість кінцевого продукту, але й забезпечує його безпеку, надійність та відповідність ринковим вимогам.

Таким чином, дослідження управління процесами тестування програмного забезпечення з використанням Agile підходів є актуальним для забезпечення конкурентоспроможності ІТ-компаній, покращення якості розробки продукту та оптимізації ресурсів.

Мета роботи. Основною метою цієї роботи є виявлення недоліків у застосуванні гнучких методологій при впровадженні тестування як важливої складової забезпечення якості програмного забезпечення (SQA) та надання рекомендацій для підвищення ефективності цього процесу. Аналізуються ключові проблеми, що виникають під час інтеграції тестування в гнучкі процеси розробки, і пропонуються шляхи їх оптимізації для досягнення вищої якості продукту.

Об’єкт дослідження: Процеси розробки ПЗ за гнучкими (Agile) технологіями.

Предмет дослідження: Якість програмного продукту через систематичне тестування.

Наукова новизна отриманих результатів. Запропоновано вдосконалення підходів до тестування в Agile-проєктах. У гнучких методологіях тестування інтегрується на кожному етапі життєвого циклу продукту. Agile-проєкти характеризуються короткими ітераціями, де тестування відіграє ключову роль в ідентифікації дефектів і забезпеченні якості продукту. Тісна співпраця між розробниками, тестувальниками та клієнтами допомагає забезпечити, що продукт відповідає вимогам ще на ранніх етапах розробки. Роль фахівця з якості (QA) має бути чітко визначена, щоб він виконував функції координації, а також гарантував ефективність процесів тестування, не перекладаючи повну відповідальність за якість на розробників чи клієнтів.

Практичне значення отриманих результатів. Робота пропонує інтеграцію тестування як безперервного процесу в Agile-методологіях. Завдяки запропонованим підходам можна більш ефективно управляти проєктами, знижуючи кількість помилок і підвищуючи продуктивність команди. Систематичне тестування на кожній ітерації сприяє швидкому виявленню проблем і їх усуненню, що підвищує якість кінцевого продукту. Також це дозволяє командам швидше реагувати на зміни у вимогах клієнтів, забезпечуючи гнучке управління розробкою програмного забезпечення.

Структура та обсяг роботи. Робота складається зі вступу, трьох розділів, які поділені на підрозділи, висновків до кожного розділу, загального висновку, списку використаних джерел та додатків. Загальний обсяг роботи 78 сторінок, обсяг основного тексту 66 сторінок.

РОЗДІЛ 1. ОСНОВИ AGILE-ПІДХОДІВ ДО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Аспекти управління процесами тестування програмного забезпечення

Головною метою розробки програмного забезпечення є створення високоякісного продукту, що відповідає очікуванням клієнтів та забезпечує їх задоволеність. У сучасних умовах ринок вимагає не лише функціональних продуктів, а й таких, що є надійними, стабільними та безпечними. Досягти цих якостей можна за допомогою ретельно організованих процесів тестування, які є невід'ємною частиною забезпечення якості програмного забезпечення (SQA). Проте в контексті Agile-методологій роль тестування набуває ще більшого значення, оскільки тут воно стає безперервним процесом, що інтегрується на всіх етапах життєвого циклу розробки.

Agile-підходи до розробки програмного забезпечення, які зосереджені на гнучкості та швидкій адаптації до змін, вимагають особливих підходів до тестування. Традиційні моделі тестування, такі як "водоспадна" модель, передбачали проведення тестування на завершальних етапах проекту, коли продукт вже майже готовий. Однак у Agile-процесах тестування інтегрується на кожній ітерації, що дозволяє виявляти та виправляти дефекти ще на ранніх етапах розробки. Це зменшує ризики накопичення помилок та допомагає підтримувати високий рівень якості протягом усього циклу розробки [5].

Однією з ключових характеристик Agile-підходів є інкрементальний та ітеративний характер розробки. Протягом кожної ітерації команда розробляє, тестує та випускає частини продукту, що дозволяє швидко отримати зворотний зв’язок від користувачів та клієнтів. У цьому контексті тестування стає інструментом, який забезпечує впевненість у тому, що кожен інкремент продукту відповідає вимогам і функціонує належним чином. Такий підхід дозволяє уникнути ситуацій, коли помилки накопичуються до останніх етапів проєкту, створюючи значні ризики для успішного випуску.

Існує багато атрибутів, які потрібно враховувати, щоб сформулювати повне уявлення про якість. Але для кращого розуміння варто також звернути увагу на причини її відсутності. Як зазначено в одному з джерел, «відсутність якості – це втрати, які продукт завдає суспільству з моменту доставки клієнту». Якщо в цьому визначенні замінити слово «продукт» на «програмне забезпечення», а «відвантаження» на «доставку», ми отримаємо більш чітке та широке розуміння якості в контексті розробки програмного забезпечення.

Цей підхід перегукується з думкою Стіва Джобса: «Будьте еталоном якості. Деякі люди не звикли до середовища, де досконаліть є нормою». Якість у розробці — це не лише функціональність, але й загальне відчуття, яке продукт залишає у користувача.

Однією з найважливіших особливостей тестування в Agile є автоматизація. Автоматизоване тестування є необхідним елементом для того, щоб забезпечити швидке та ефективне тестування на кожній ітерації, особливо коли обсяги коду постійно зростають. Автоматизовані тести дозволяють перевіряти регресію, функціональність, інтеграцію та інші важливі аспекти продукту, зберігаючи при цьому швидкість розробки. Завдяки автоматизації можна виконувати повторювані завдання швидше та з меншими витратами часу, що дозволяє командам зосередитися на важливіших аспектах розвитку продукту.

Не менш важливим у Agile є і співпраця між командами. Успішне тестування в Agile-проектах неможливе без тісної взаємодії між розробниками, тестувальниками та іншими членами команди. В Agile-командах тестування не є окремим етапом, яким займаються лише тестувальники. Навпаки, відповідальність за якість розподіляється між усіма членами команди, що дозволяє швидко і ефективно вирішувати проблеми. Тестувальники працюють разом з розробниками над написанням тестових сценаріїв, проведенням тестів та аналізом результатів. Така взаємодія сприяє більшій прозорості та прискоренню процесів, що веде до кращих результатів у тестуванні та забезпеченні якості.

Ще одним важливим аспектом є активне залучення клієнтів у процес тестування. В Agile-проектах клієнти мають можливість безпосередньо брати участь у визначенні вимог та тестуванні продукту. Це дозволяє уникнути непорозумінь і забезпечує, що кінцевий продукт буде максимально відповідати очікуванням користувачів. Клієнти можуть надавати зворотний зв'язок на кожній ітерації, а команди — відповідно адаптувати свої рішення. Це дозволяє підтримувати високий рівень гнучкості та швидко реагувати на зміни, що особливо важливо у динамічних ринкових умовах.

Однією з ключових переваг Agile-підходів до тестування є швидкість реакції на зміни. Оскільки тестування проводиться на кожній ітерації, команда має можливість швидко виявляти проблеми, коригувати їх і випускати оновлений продукт. Це особливо важливо в умовах постійної зміни вимог клієнтів чи ринкових умов. Гнучке тестування дозволяє команді бути впевненою у якості продукту на кожному етапі розробки та забезпечувати його відповідність постійно змінюваним очікуванням ринку.

У підсумку, Agile-підходи до тестування програмного забезпечення ставлять тестування в центр всього процесу розробки, роблячи його безперервною та інтегрованою частиною кожної ітерації. Автоматизація, тісна співпраця між командами, активне залучення клієнтів та здатність швидко реагувати на зміни є основними елементами, які забезпечують високу якість продукту в Agile-проектах. Тестування стає не лише інструментом для виявлення помилок, а важливим засобом для досягнення конкурентоспроможності, надійності та стабільності кінцевого продукту.

Управління процесами тестування програмного забезпечення є критично важливим етапом у забезпеченні якості кінцевого продукту. Ефективне тестування допомагає не тільки виявити дефекти в коді, але й запобігти їхньому впливу на роботу системи в майбутньому, забезпечуючи надійність і функціональність продукту. У сучасних умовах швидкого розвитку технологій управління тестуванням стає все більш складним і багатогранним процесом, який включає планування, виконання, контроль, аналіз результатів і

управління дефектами. Розуміння основних аспектів цього процесу дозволяє оптимізувати ресурси, зменшити час на тестування та підвищити загальну якість продукту, що є важливою умовою для його успішного впровадження на ринок. Розглянемо ключові елементи управління процесами тестування:

1. Планування тестування

Це початковий етап, який передбачає визначення стратегії тестування, вибір методологій та інструментів для тестування, а також визначення критеріїв прийняття. Важливо розробити детальний план тестування, який включатиме:

- Опис цілей тестування.
- Оцінку обсягу роботи (які частини продукту потребують тестування).
- Ресурси, необхідні для проведення тестування (тестувальники, інструменти).
- Календар тестування (графік виконання етапів тестування).
- Критерії виходу з тестування (як визначити, коли тестування завершене).

2. Організація та координація тестування

Ефективне управління процесами тестування вимагає чіткого розподілу обов'язків серед членів команди тестування. Кожен тестувальник має свої завдання, такі як розробка тестових сценаріїв, виконання тестів, аналіз результатів та документування виявлених проблем. Крім того, важливо підтримувати постійну координацію між розробниками, тестувальниками та іншими зацікавленими сторонами, щоб забезпечити своєчасне виявлення та виправлення дефектів.

3. Виконання тестів

На цьому етапі здійснюється фактичне тестування програмного забезпечення. Воно може бути ручним або автоматизованим, залежно від обраної стратегії. Тестування може включати:

- Функціональне тестування (перевірка відповідності функцій програмного забезпечення специфікаціям).

- Нефункціональне тестування (тестування продуктивності, безпеки, навантаження тощо).
- Регресійне тестування (перевірка, чи не вплинули останні зміни на вже існуючий функціонал).

4. Моніторинг і контроль

Процеси тестування повинні контролюватися для забезпечення відповідності плану. Це включає моніторинг прогресу тестування, аналіз результатів та виявлених дефектів. Для цього часто використовуються спеціалізовані інструменти управління тестуванням (наприклад, JIRA, TestRail), які дозволяють відстежувати статус тестів, їх виконання та фіксування помилок.

5. Аналіз та звітування

Після завершення тестування здійснюється аналіз його результатів. Головна мета цього етапу – визначити, наскільки програмне забезпечення відповідає вимогам, чи були виявлені критичні дефекти, які впливають на роботу продукту, і чи готове програмне забезпечення до випуску. Результати тестування документуються у вигляді звітів, які включають:

- Виявлені дефекти та їх пріоритет.
- Відсоток покриття тестами.
- Рекомендації щодо випуску чи необхідності подальшого тестування.

6. Управління дефектами

Важливо мати чіткий процес управління дефектами, який включає реєстрацію виявлених проблем, їхній аналіз, пріоритизацію та моніторинг процесу виправлення. Дефекти класифікуються за рівнем критичності, а процес їхнього усунення повинен контролюватися до повного виправлення.

7. Аналіз ефективності процесу тестування

Після завершення проекту або великої його частини проводиться ретроспектива процесу тестування, щоб виявити, які аспекти можна покращити в майбутньому. Це може стосуватися як вдосконалення

методологій тестування, так і процесів управління командою тестування або технічних рішень.

Загалом, управління процесами тестування – це багатоступеневий процес, який вимагає точного планування, координації та постійного аналізу для забезпечення високої якості програмного забезпечення та своєчасного випуску продукту.

1.2 Забезпечення якості програмного забезпечення при традиційній розробці програмного забезпечення

У розробці програмного забезпечення традиційними вважаються методології, які зосереджені на процесах. У таких підходах структура процесу розробки або життєвого циклу продукту суворо дотримується. Зазначені етапи включають [1]:

- Збір і аналіз вимог;
- Специфікація;
- Архітектурне проектування;
- Проектування системи;
- Розробка коду;
- Тестування і впровадження;
- Технічне обслуговування.

Існує багато моделей розробки програмного забезпечення, які орієнтовані на процеси. Кожна модель має свої етапи, процеси, процедури та дії, які потрібно дотримуватися протягом життєвого циклу розробки. У традиційній розробці програмного забезпечення моделі водоспаду та спіралі вважаються найбільш зрілими і практичними. У таких підходах замовник взаємодіє з інженерами вимог для визначення конкретних вимог до продукту. Політики та стандарти рішень формуються на управлінському рівні організації.

Тестування є ключовим елементом забезпечення якості у будь-якій моделі розробки програмного забезпечення. Воно спрямоване на перевірку

якості продукту, процесу розробки та відповідності до визначених вимог. Попри різні підходи до тестування, існує кілька моделей, що визначають важливі характеристики, які потрібно оцінювати під час розробки програмного забезпечення. Моделі якості Макколла (1977) [7], та Боема (1978) [8] лягли в основу сучасних стандартів тестування. Важливо звертати увагу на такі аспекти якості під час тестування:

- Коректність;
- Надійність;
- Цілісність;
- Юзабіліті;
- Ефективність;
- Ремонтопридатність;
- Сумісність;
- Гнучкість;
- Можливість повторного використання;
- Портативність;
- Модифікованість;
- Документація;
- Стійкість;
- Зрозумілість;
- Термін служби;
- Функціональність;
- Економічність.

З розвитком індустрії програмного забезпечення програми стають дедалі складнішими, а вимоги користувачів зростають. Це призводить до збільшення складності проектів розробки програмного забезпечення. Для управління цією складністю використовуються різні моделі та стандарти. Серед найбільш широко застосовуваних моделей можна виділити:

- ISO 9000;
- ISO 9126;

- CMM (Модель зрілості можливостей);
- CMMI (Інтеграція моделі зрілості можливостей);
- SPICE (Оцінка можливостей удосконалення процесів програмного забезпечення);
- Шість сигм.

Окрім фокусу на якості продукту чи послуги, ці моделі також орієнтовані на забезпечення якості процесу розробки програмного забезпечення на рівні організації. З часом роль фахівців з забезпечення якості (QA) розширюється, і їх діяльність стає ключовою для організації. QA відповідає за управління процесом розробки, забезпечення якості продукту та підтримку проекту. На управлінському рівні встановлюються стандарти та процедури розробки програмного забезпечення. Роль SQA полягає в забезпеченні правильного документування визначених стандартів та дотриманні встановлених процедур. Для цього проводяться оцінки продукту, аудити та зустрічі, які сприяють контролю та перевірці відповідності процесів визначеним процедурам.

Основні завдання тестувальників полягають у верифікації та валідації продукту, що дозволяє оцінити якість на кожному етапі розробки. Важливими інструментами для цього є аудити, контрольні списки, показники якості та автоматизовані аналізатори коду, які допомагають підтримувати стандарти та вимоги. Процес тестування забезпечує відповідність розробленого програмного забезпечення встановленим стандартам і допомагає своєчасно виявляти та виправляти недоліки.

Акцент на тестуванні дозволяє гарантувати, що програмний продукт відповідає всім технічним і функціональним вимогам. Без ефективного тестування неможливо забезпечити успішне впровадження програмного забезпечення, тому ця діяльність є критично важливою на всіх етапах проекту. Тестування вимагає значних ресурсів і часу, але є основою для досягнення високої якості кінцевого продукту

1.3 Гнучка розробка програмного забезпечення

Гнучкий підхід до розробки програмного забезпечення радикально змінив традиційні методи розробки. У Agile командах роль QA-фахівців кардинально відрізняється від традиційної моделі, де тестування зазвичай проводиться після завершення розробки. В Agile QA-фахівці стають повноправними учасниками всього життєвого циклу розробки, взаємодіють з іншими членами команди з самого початку проекту і відіграють важливу роль у забезпеченні якості на кожному етапі.

Однією з основних задач QA-фахівців в Agile є активна участь у плануванні. З перших ітерацій QA спеціалісти беруть участь у визначенні вимог до продукту разом з розробниками та стейкхолдерами. Це дозволяє виявляти можливі проблеми та уточнювати вимоги ще до початку розробки. Завдяки такому підходу тестування інтегрується в процес розробки, а не стає його завершальною частиною, як це відбувається у традиційних моделях.

Основою гнучкої розробки є маніфест гнучкої розробки програмного забезпечення [2], який визначає такі основні цінності:

- Взаємодія між членами команди та процесами і інструментами важливіші, ніж просто використання інструментів.
- Функціональне програмне забезпечення цінується більше за детальну документацію.
- Співпраця з клієнтом важливіша за формальності в контракті.
- Гнучкість у реагуванні на зміни важливіша за суворе дотримання плану.

Гнучка розробка програмного забезпечення сприяє активній співпраці учасників проекту. У порівнянні зі звичайним способом розробки програмного забезпечення, він ефективно реагує на зміни, оскільки є поступовим та ітеративним. На рисунку 1.1 зображена одна ітерація гнучкого підходу до розробки.

Рисунок 1.1 – Одна ітерація процесу Agile розробки

Джерело: [9]

Як ми бачимо, в одній короткій ітерації всі необхідні фази розробки програмного забезпечення (вимоги, проектування, розробка, тестування, розгортання, оцінка) відбуваються одна за одною. Після завершення кожної ітерації програмний продукт передається на перевірку користувачам або замовникам. В наступній ітерації реалізуються нові функції на основі отриманих відгуків. Існує безліч гнучких методологій, але найбільш поширеними є:

- Екстремальне програмування (XP);
- Scrum.

Всі гнучкі методології мають спільні риси. На рисунку 1.2 показано особливості цих методологій, згідно з [9].

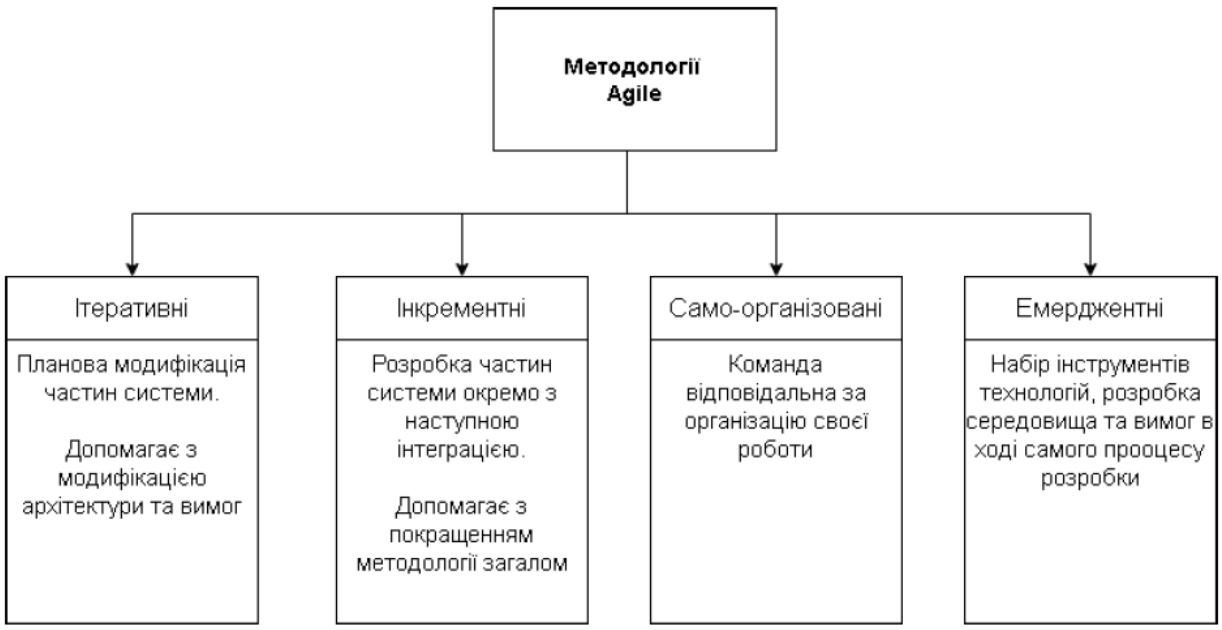

Рисунок 1.2 – Природа Agile методологій

Джерело: [9]

Гнучкі методології вважаються "легкими", оскільки не вимагають складних процесів. Вони також трансформували діяльність SQA, зменшивши обсяг документації, яка тепер створюється лише у відповідності до вимог замовника або користувача.

Тестування є частиною кожної ітерації, що сприяє скороченню часу до релізу і підвищує якість. Наша увага буде зосереджена на діяльності SAQ у гнучких методологіях, тому всі аспекти, пов'язані з цією темою, будуть детально розглянуті в наступних розділах. Ми також проаналізуємо різницю між теоретичними підходами та практичними аспектами.

Менталітет змінюється з поширенням гнучких методологій у сфері розробки програмного забезпечення. Діяльність з контролю якості зазнала суттєвих змін: Agile змістив акцент з процесів, орієнтованих на людей. Тестування більше не є виключно обов'язком спеціаліста з забезпечення якості; тепер розробники повинні самостійно тестувати свої програми і виправляти помилки, як тільки їх виявлять. В практиці немає окремого, тривалого етапу тестування. Помилки виправляються одразу на тому етапі, де їх знайдено,

незалежно від того, хто їх виявив і виправив. Тісна співпраця з клієнтами забезпечує швидкий і ефективний зворотний зв'язок щодо розробленої системи перед початком нової ітерації, що сприяє підвищенню якості продукту.

Ще однією важливою характеристикою є підхід до тестування, який називається тестуванням, що супроводжує розробку (test-driven development, TDD). У цьому підході тестування є невід'ємною частиною процесу написання коду. Тестувальники і розробники спільно створюють тест-кейси перед написанням коду, що дозволяє виявляти помилки на ранніх стадіях і забезпечує високу якість ще до початку інтеграції компонентів.

Один з ключових прикладів ефективного залучення QA-фахівців можна спостерігати в компанії «Spotify», яка відома своїми інноваційними підходами до Agile. У «Spotify» роль QA інтегрована в кросфункціональні команди, що працюють над розробкою нових функцій. Тут QA-фахівці не тільки тестують продукт, але й активно беруть участь у плануванні, обговоренні вимог та розробці тест-кейсів на ранніх етапах. Це дозволяє команді отримувати зворотний зв'язок щодо якості продукту на кожному етапі розробки та швидко вносити необхідні зміни.

«Amazon» також є прикладом компанії, яка ефективно інтегрувала QA в Agile процеси. У Amazon тестувальники тісно співпрацюють з розробниками і залучені в процес автоматизації тестування, особливо в рамках безперервної інтеграції та доставки (CI/CD). Компанія використовує автоматизовані тести для кожного коду, який додається до продукту, що дозволяє забезпечувати високу якість при швидких релізах. QA-фахівці «Amazon» допомагають створювати автоматизовані тести, які запускаються при кожному оновленні коду, що дає можливість виявляти помилки практично в реальному часі.

Гнучкі методології відзначаються інтегрованим підходом до тестування, швидкою реакцією на зміни та орієнтацією на людей. Однак деякі критики стверджують, що ці методології недостатньо зрілі для масштабного впровадження. У нашій роботі ми детально розглянемо ці питання та

запропонуємо можливі рішення для усунення існуючих прогалин у діяльності SAQ в контексті Agile.

1.4 Формулювання задач дослідження

Якість програмного продукту завжди була основною метою для будь-якої компанії чи проекту, і бездоганний продукт вважається найбільшою характеристикою якості. Однак для створення якісного програмного продукту необхідно забезпечити високий рівень розробки. Будь-яка програмна методологія слідує певним принципам, щоб забезпечити правильну розробку продукту. Діяльність із забезпечення якості контролюється відповідними органами в організації. Світ програмної інженерії зазнав змін, перейшовши від процесно-орієнтованих підходів до гнучких методів. Гнучка розробка спрямована на швидку адаптацію до змін. У нашій роботі ми не тільки обговоримо, як досягається якість у гнучкій розробці, але й висвітлимо та спробуємо усунути деякі прогалини в цій області.

Обмежимо нашу дослідницьку роботу пошуком відповідей на наступні питання:

1. Яка роль заходів із забезпечення якості програмного забезпечення в гнучкій розробці згідно з існуючою літературою?

2. Визначити прогалини гнучкої ітеративної культури, які можна покращити щодо діяльності із забезпечення якості і запропонувати підхід для усунення прогалин.

3. Як організації практикують діяльність SQA, використовуючи гнучку розробку в проектах?

Agile розробка змінила підхід до виконання програмних проєктів, але її не можна вважати настільки зрілою та практичною, як традиційні методи розробки. В традиційній розробці діяльність SQA відповідає за контроль всіх процесів.

Завдяки активнішому залученню клієнтів і зацікавлених сторін, гнучка розробка ввела новий підхід до задоволення вимог замовника. Гнучкість та

швидка реакція на зміни роблять цю методологію більш ефективною для розробки. Проте, незважаючи на свою ефективність, вона стикається з критикою і проблемами. У цьому розділі ми розглянемо деякі з проблем, що виникають у гнучкій розробці в контексті забезпечення якості.

Залучення клієнта на місці та активна співпраця з розробником є ефективним підходом до створення системи, коли це необхідно. Як зазначається в [11], присутність клієнта на місці робить оцінку системи більш ефективною, оскільки він може надати корисний зворотний зв’язок, поки робота ще свіжа в пам'яті. Однак важливо враховувати, що коли клієнт не має достатнього технічного розуміння параметрів якості та атрибутів системи, це може призвести до неправильного розуміння системи. Крім того, якщо клієнт не завжди доступний на місці, необхідно мати представника, який зможе донести потреби клієнта, щоб забезпечити якість і гнучкість проекту протягом усіх етапів. Швидкий зворотний зв’язок є критично важливим для гнучкості, і клієнт повинен надавати відгук після кожної ітерації.

Гнучка розробка програмного забезпечення запровадила новий підхід, що передбачає скорочення обсягу документації до мінімуму. Хоча гнучкість не підтримує надмірне документування, деякі практики висловлюють критику, стверджуючи, що ігнорування документації не є правильним. Хоча гнучка розробка зменшує обсяг технічної документації, вона залишається важливою для підтримки якості програмного забезпечення в майбутньому [11]. QA несе відповідальність за документацію протягом всього проекту. Оскільки гнучкий підхід не заохочує детальне документування, а документація має значення, роль QA в гнучких методологіях потребує перегляду.

Тестування є ключовим аспектом забезпечення якості. У традиційних методах розробки тестування проводиться як окремий і завершений етап. Наразі використовуються як автоматизовані, так і ручні методи тестування. Гнучка розробка суттєво змінила підхід до тестування програмного забезпечення, переклавши відповідальність за тестування на розробників через такі методи, як парне програмування та тестове програмування. Але чи

означає це, що потреба в QA-тестувальниках зменшилася, а програмісти тепер також беруть на себе функції тестувальників? Відповідно до [12], програмісти не можуть впоратися з усіма типами дефектів під час розробки продукту. Щоб вирішити цю проблему, в [13] було запропоновано створити окрему команду тестування для гнучкої розробки. Наша мета – забезпечити всебічне управління якістю, а не обмежуватися лише тестуванням.

Зі збільшенням складності програмних продуктів і необхідністю автоматизації великих ручних систем, проблеми з якістю та тестуванням стають дедалі серйознішими. Хоча ця робота більше зосереджена на гнучких підходах, слід зазначити, що багато організацій також отримують вигоду від формальних методів, і гнучкі підходи не завжди є широко впровадженими. Якість програмного забезпечення не обмежується лише тестуванням, і тестування не залежить виключно від інструментів, що використовуються.

Хоча багато хто стверджує, що гнучка розробка забезпечує високу якість, ми плануємо проаналізувати наукові праці та практичні дослідження діяльності SQA у гнучких проектах, щоб запропонувати вдосконалення для усунення існуючих прогалин. Аналіз реальних практик у проектах гнучкої розробки дозволить нам краще зрозуміти, як організації реалізують SQA, а відгуки респондентів допоможуть оцінити обмеження та переваги гнучкого підходу в забезпеченні якості, що стане основою для подальших пропозицій щодо покращення.

Висновки до розділу 1

В Agile-методологіях тестування є невід'ємною частиною кожної ітерації розробки, що дозволяє оперативно виявляти і усувати помилки на ранніх етапах, забезпечуючи високу якість продукту. Гнучкий підхід сприяє тісній взаємодії між замовником і командою розробників, що дозволяє швидко адаптувати продукт до змін. Водночас, роль фахівців із забезпечення якості (QA) змінюється: вони більше зосереджуються на процесах, ніж на

документації, а розробники беруть активнішу участь у тестуванні своїх продуктів. Незважаючи на численні переваги Agile, залишається необхідність у подальшому вдосконаленні методів забезпечення якості для усунення існуючих прогалин, що може включати більш чітке розмежування відповідальності між розробниками і тестувальниками, а також впровадження додаткових процесів для забезпечення надійного контролю якості.

РОЗДІЛ 2. МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ У КОНТЕКСТІ AGILE-ПІДХОДІВ

2.1 Аналіз літературних джерел для проведення досліження

Перед початком будь-якого дослідження важливо визначити методологію, підхід або план дій, які будуть використовуватися для зв'язку методів з очікуваними результатами. Згідно з визначенням Оксфордського словника, термін «методологія» позначає «систему методів, що застосовуються в певній сфері». Вибір методології для дослідження є критично важливим для забезпечення ефективного процесу дослідження, контролю за його правильним напрямком, а також для вимірювання, структурування і організації дослідження. Найпоширеніші методології або підходи включають:

- Якісне дослідження;
- Кількісне дослідження;
- Змішаний підхід до дослідження.

В нашому дослідженні було використано якісний підхід. Визначивши основні критичні проблеми в досліджуваній області, ми намагалися використовувати висновки з літератури для розробки і пропозиції рішень, спрямованих на вирішення цих проблем у гнучкій діяльності SQA. Тому наше дослідження можна віднести до категорії якісних.

Огляд літератури був проведений для збору теоретичних даних з різних досліджень у нашій сфері інтересів. Це дозволило отримати уявлення про перспективи розвитку предметної області та краще зрозуміти процеси розробки в контексті гнучких підходів.

Важливість інтеграції тестування на всіх етапах життєвого циклу розробки в межах Agile підтверджують роботи як міжнародних дослідників, так і українських науковців. У роботах Джо Саттлера та Елізабет Хендерсон акцентується увага на тому, що тестування є невід’ємною частиною кожної ітерації, що сприяє ранньому виявленню та усуненню дефектів [14] [15].

Українські дослідники також вивчають вплив Agile на тестування. Зокрема, Романенко А. (2020) у своїй роботі *"Адаптація процесів тестування в умовах Agile розробки"* зазначає, що гнучкі підходи дозволяють забезпечити вищу якість продукту завдяки постійній співпраці між тестувальниками та розробниками на всіх етапах. Він також підкреслює важливість комунікації в команді для досягнення високих результатів [22].

Дослідження Пола Грейна та колег (2019) демонструє, що інтеграція тестування в Agile процеси, як у XP та Scrum, дозволяє покращити прозорість розробки та зменшити кількість помилок [16]. В Україні подібні дослідження провели Мартинюк О. і Кравченко О. (2019), які зазначають, що раннє тестування допомагає досягати вищої якості продукту та скоротити терміни розробки [23].

Важливим елементом Agile підходів є автоматизація тестування, що підтверджується дослідженнями як міжнародних, так і українських вчених. Мері Джонс (2020) наголошує, що автоматизація значно підвищує ефективність Agile-процесів, зменшуючи потребу в ручному тестуванні та прискорюючи ітерації [17]. Дослідник Семенчук Ф. (2021) у своїй статті "Автоматизація тестування в умовах гнучкої розробки програмного забезпечення" описує позитивний досвід українських IT-компаній, які завдяки використанню інструментів, таких як Selenium і Jenkins, змогли підвищити продуктивність і якість тестування [24]. Ці висновки узгоджуються з глобальними тенденціями, які описував Фаулер (2015), акцентуючи увагу на важливості автоматизації у контексті Continuous Integration (CI) та Continuous Testing (CT) [18].

Попри численні переваги Agile тестування, існують і певні виклики. За словами Нікласа Меллера (2020), основним викликом є підтримка адекватного покриття тестами у швидких ітераціях [20]. Аналогічно, в українських реаліях, Довженко П. (2019) у статті *"Проблеми тестування в Agile процесах в Україні"* зазначає, що часто через стислий графік команди можуть ігнорувати деякі аспекти тестування, що може негативно вплинути на якість продукту [26].

Катріна Клопп (2019) додає, що одним з найбільших викликів є документування процесу тестування в умовах швидких ітерацій [21]. Подібні проблеми відзначають і українські дослідники, зокрема Левченко Д. (2021), який у своїй статті *"Тестування без документації: реалії Agile в Україні"* підкреслює важливість балансування між якістю тестування та швидкістю розробки [27].

Ми зібрали інформацію, зосереджуючи увагу на контролі якості та супутніх процесах у гнучких проектах. Проаналізовано ряд наукових статей, журналів та інших матеріалів. Для пошуку релевантної літератури ми використовували різні ключові слова в зазначених пошукових системах і дослідницьких базах даних. Оцінювання статей здійснювалося через перегляд анотацій, вступів і висновків для визначення їхньої актуальності. Відповідну інформацію було відібрано і використано при написанні цієї роботи.

2.2 Якісний підхід до вирішення поставлених задач

У даному дослідженні акцент робиться на теоретичних аспектах, а не на числових чи статистичних даних. Необхідно було дослідити концепцію і роль забезпечення якості (QA) у процесах гнучкої розробки, щоб виявити прогалини в забезпеченні якості в гнучких технологіях. Через специфіку теми був обраний якісний підхід, оскільки саме він дозволяє детально описати дослідницьку проблему, краще зрозумівши концепції та явища через глибокий аналіз.

Чітко сформульовані питання дослідження мають ключове значення, оскільки вони направляють та допомагають у створенні результатів або теорій. Однією з основних цілей нашого дослідження було знайти рішення для виявлених проблем. Для розробки цих рішень ми орієнтувались на думки практиків, які працюють у відповідній галузі, а не на математичні результати. Таким чином, наше дослідження можна класифікувати як якісне, оскільки в ньому використовувався підхід, що дозволяє збирати нові дані з основною метою розробки тем на їх основі.

Під час аналізу літератури були виявлені прогалини та критичні проблеми, що вимагали розробки пропозицій для їх вирішення та покращення. Для формулювання ефективних рішень була необхідна точка зору практиків щодо діяльності SAQ. З метою збору думок практиків було проведене анкетування.

Була розповсюджена відкрито доступна анкета, яка дозволяє зібрати думки респондентів на основі їхнього досвіду. Хоча основна увага дослідження зосереджена на гнучких підходах до SQA, для забезпечення обґрунтованості та ефективності спостережень і запропонованих рішень також було проведене анкетування практиків, які працюють у традиційних середовищах розробки програмного забезпечення.

Для розподілу анкети було обрано п’ять різних організацій, що спеціалізуються на розробці програмного забезпечення різного типу. Проекти, які досліджувалися, охоплюють такі сфери:

- Сховища даних і бізнес-аналітика;
- Розробка програмного забезпечення на замовлення;
- Впровадження та модифікація пакетних рішень;
- Телекомунікації;
- Системи управління людськими ресурсами;
- Системи управління документами;
- Фінансові пакети;
- Інструменти електронного урядування;
- Програми для Інтернет-банкінгу;
- Електронні медичні карти;
- Медична документація.

2.3 Загальна характеристика гнучких процесів розробки

Гнучкість, що стосується розробки програмного забезпечення, можна виразити як гнучкий, готовий до змін і швидкий характер процесу розробки програмного забезпечення.

В [42] автор визначив гнучкість як «усунення якомога більшої частини тяжкості, зазвичай пов’язаної з традиційними методологіями розробки програмного забезпечення, для сприяння та швидкого реагування на зміну середовища, зміни у вимогах користувачів, прискорені терміни виконання проекту тощо». Процеси розробки програмного забезпечення, впроваджені з використанням Agile, сприяють коротким релізам і не сприяють створенню детальної документації, що скорочує втрату часу та чітке бачення продукту, який потрібно розробити.

Тісна співпраця між замовником та усіма учасниками проекту є важливою для забезпечення високої якості продукту. Як зазначають автори [43], гнучкі методології розроблені для того, щоб «приймати, а не ігнорувати, швидкі зміни». У гнучкій розробці процеси розбиваються на короткі, функціональні ітерації. Під час кожної нової ітерації змінені вимоги інтегруються в систему відповідно до загальних специфікацій. Це робить процес розробки програмного забезпечення більш гнучким.

Представлення короткого порівняння між звичайним і гнучким підходами до розробки може привести нас до кращого розуміння гнучкої філософії.

Тісна співпраця між замовником та усіма учасниками проекту є важливою для забезпечення високої якості продукту. Як зазначають автори [43], гнучкі методології розроблені для того, щоб «приймати, а не ігнорувати, швидкі зміни». У гнучкій розробці процеси розбиваються на короткі, функціональні ітерації. Під час кожної нової ітерації змінені вимоги інтегруються в систему відповідно до загальних специфікацій. Це робить процес розробки програмного забезпечення більш гнучким.

Таблиця 2.1 – Порівняння Agile- та традиційних методологій розробки

Середовище проєкту		Характеристики проєкту	
Категорія	Змінна	Agile	Non-Afile
	Стиль комунікації	Постійна співпраця	За необхідності

Команда розробників	Локація	Зосереджена	Розподілена
	Локація	Зосереджена	Розподілена
	Розмір	До 50 чоловік	Понад 50 чоловік
	Постійне навчання	Властиве	Не притаманне
Управління проєктом	Культура менеджменту	Реактивна	Вказівки та контроль
	Залучення команди	Обов'язкове	Не вітається
	Планування	Постійне	На початку
	Механізм зворотного зв'язку	Декілька різних	Недоступний
Замовник	Залучення	Протягом проєкту	На етапі аналізу
	Доступність	Легко досяжний	Важко доступний
Процеси та інструменти	Роль команди	Вирішальне слово за командою	Команді вказують, що робити
	Кількість	Необхідний мінімум	Більше, ніж необхідно
	Адаптивність	Можливі зміни	Зміни неможливі
Контракт	Вимоги та дати	Гнучкі	Фіксовані
	Вартість	Час та матеріали	Фіксовані

У наступних розділах ми обговоримо різні гнучкі методології, щоб зрозуміти, як концепція гнучкості використовується для розробки програмного забезпечення різними способами та підходами.

2.3.1 Екстремальне програмування (XP)

Екстремальне програмування (XP) [30] є однією з найпопулярніших та найпоширеніших методологій гнучкої розробки програмного забезпечення. Кент Бек, засновник цієї методики, визначив низку принципів і практик,

спрямованих на підвищення продуктивності команди розробників та покращення точності і якості створеної системи. XP — це легка, передбачувана, ефективна та гнучка методологія, розроблена для невеликих команд, що працюють з невизначеними та змінними вимогами до програмного забезпечення. XP включає набір дисциплін і практик, яких необхідно дотримуватися під час розробки програмного продукту (рис. 2.1).

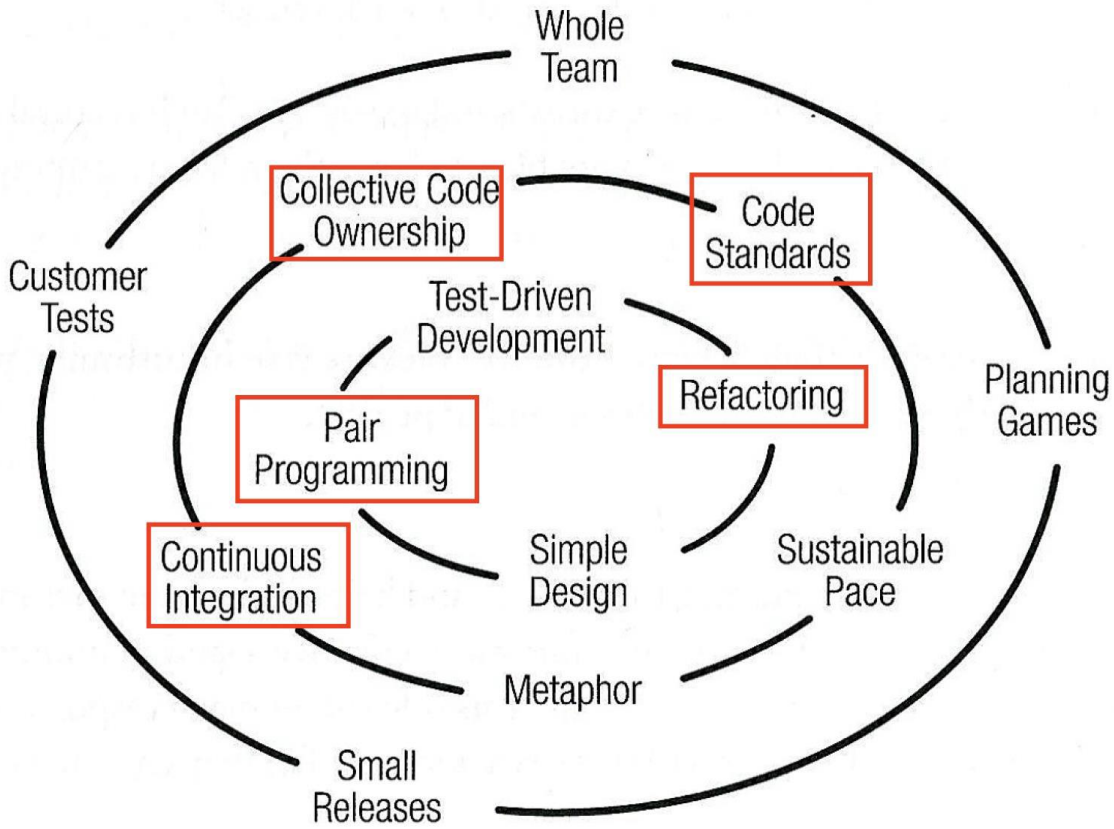

Рисунок 2.1 – Принципи екстремального програмування

Джерело: [30]

Ключові терміни та практики XP розглянуті відповідно до [30], [31]:

- **Планування**: програмісти оцінюють зусилля для реалізації користувацьких історій (user stories), а замовник визначає час релізу та обсяг роботи на основі цих оцінок. **Метафора**: робота системи описується за допомогою метафор, зрозумілих як для клієнтів, так і для програмістів.

Малі релізи: додатки розробляються серією частих і невеликих версій, які регулярно оновлюються.
Рефакторинг: система реструктурується, щоб зменшити дублювання, покращити зв’язки та спростити структуру без зміни функціональності.
Простий дизайн: XP зосереджується на розробці максимально простого рішення та уникненні зайвої складності й непотрібного коду.
40-годинний робочий тиждень: жоден член команди не працює понаднормово; перевищення цього ліміту може вказувати на проблеми з плануванням.
Парне програмування: програмісти працюють у парах на одному комп'ютері під час написання коду.
Стандарти кодування: існують визначені стандарти коду, які забезпечують послідовність і полегшують спілкування між розробниками.
Колективна власність: ніхто не є виключним власником певних частин коду, кожен може змінювати будь-який код у будь-який момент.
Безперервна інтеграція: код інтегрується в систему після кожного етапу, проходячи всі необхідні тести.
Клієнт: у XP обов’язково присутній клієнт, який працює разом з командою розробників.

У XP кожен член команди відіграє важливу роль у проекті. Команди організовуються навколо бізнес-представника, якого називають «Замовником». З орієнтацією на бізнес-цінність, команди XP застосовують прості методи планування та відстеження для визначення наступних кроків і прогнозування завершення проекту. Вони розробляють невеликі версії програмного забезпечення, які проходять усі тести, визначені замовником [32].

Один з ключових підходів у XP — це тестування на рівні одиниць (unit testing). Розробники пишуть тести для кожного класу або методу ще до того, як код створений, перевіряючи функціональність кожної його частини. Цей

підхід тісно пов'язаний із практикою розробки через тестування (TDD). Спочатку створюється тест, який визначає очікувану поведінку функції, а потім пишеться код, що повинен цей тест пройти. Така методологія дозволяє запобігти багатьом проблемам на ранніх етапах, орієнтуючи розробку на досягнення конкретних результатів.

Безперервна інтеграція також відіграє важливу роль у процесі тестування в XP. Код постійно зливається в головну гілку проекту, і будь-які зміни негайно проходять автоматизовані тести. Це допомагає швидко виявляти конфлікти або помилки, що виникають через інтеграцію нового коду.

Крім тестів на рівні одиниць, у XP важливо проводити функціональне тестування, яке перевіряє роботу програми з точки зору сценаріїв використання. Такі тести переконуються, що програма працює відповідно до вимог і очікувань користувачів.

Рефакторинг — ще одна важлива практика, тісно пов'язана з тестуванням у XP. Він дозволяє покращувати архітектуру системи, роблячи код чистішим і зрозумілішим. Постійне вдосконалення коду вимагає наявності тестів, які гарантують, що зміни не впливають на вже існуючу функціональність і не призводять до нових помилок.

Важливою частиною XP є також тісна співпраця з клієнтами, які залучаються до розробки приймальних тестів (acceptance tests). Ці тести допомагають переконатися, що кінцевий продукт відповідає початковим вимогам і задовольняє очікування клієнтів.

Тестування, рефакторинг, системна метафора та парне програмування є ключовими елементами забезпечення якості (QA) в XP. Ці практики взаємопов’язані та підтримують одна одну. Тестування гарантує, що код не містить помилок, а рефакторинг забезпечує його простоту, щоб уникнути складнощів у системі. Системна метафора допомагає краще зрозуміти архітектуру, зменшуючи ризик збоїв, якщо розробка виконується відповідно до цієї структури. Парне програмування дозволяє двом розробникам обмінюватися ідеями та спільно знаходити помилки, що сприяє створенню

безпомилкової системи. Таким чином, ці практики допомагають розробляти якісний продукт із мінімальними ризиками [30].

2.3.2 Методологія Scrum

Scrum також є однією з найпоширеніших методологій у Agile, спочатку розробленою Кеном Швабером. Назва «Scrum» походить із термінології регбі, де вона позначає командну взаємодію для повернення м'яча в гру. Scrum пропонує структуру управління проєктом, яка охоплює ключові етапи розробки, такі як збір вимог, проєктування та програмування (рис. 2.2).

Він не пропонує жодного конкретного методу для використання, але регулює, як має працювати команда, щоб забезпечити гнучкість системи в умовах змін зовнішнього середовища. Під час розробки існує багато змінних, як технічних, так і екологічних, що постійно змінюються, таких як вимоги, час, ресурси та технології. Через ці змінні процес розробки стає складним і непередбачуваним. Необхідний процес, який здатен вирішувати ці питання в системі. У Scrum часто застосовуються практики, що допомагають менеджменту проєкту досягти кращих результатів [33].

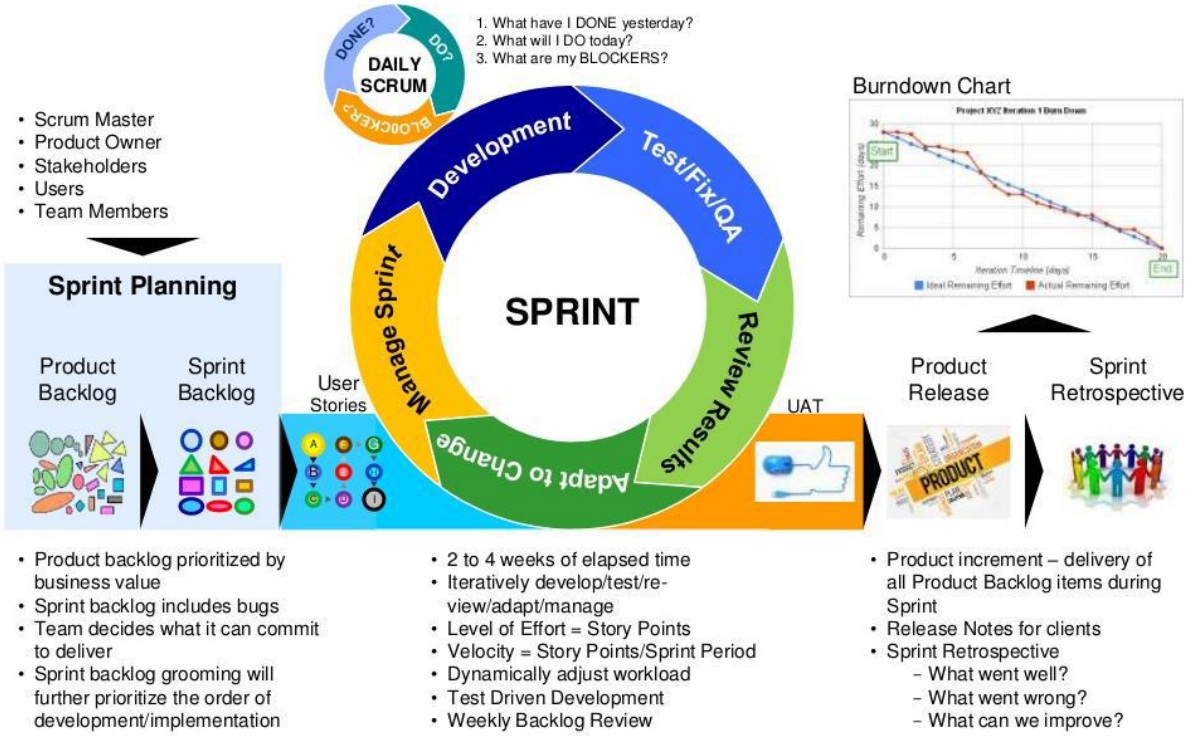

Рисунок 2.2 – Принцип методології Scrum

Джерело: [33]

Scrum включає процеси як управління, так і розробки. Ця методологія передбачає швидке створення прототипів і спрощений перегляд вимог від замовника, які можуть бути неповними та змінюватися протягом розробки. Однією з ключових практик Scrum є щоденні 15-хвилинні зустрічі для координації та вирішення питань розробки [33]. Деякі з основних практик Scrum включають:

- **Беклог продукту**: У команді фіксуються всі поточні завдання в списку під назвою Беклог. Змінювати його можуть не лише учасники проєкту, а й клієнти, відділи маркетингу та продажів. Scrum-майстер керує зустрічами Scrum і визначає завдання для спринту.
- **Спринти**: Тривалість спринтів становить від 10 до 30 днів. Розробники отримують завдання для виконання в межах спринту. Протягом спринту будь-які зміни з боку команди заборонені. Основними інструментами є зустрічі з планування спринту, беклог спринту і щоденні зустрічі Scrum.
- **Зустріч з планування спринту**: У початкових планувальних зустрічах беруть участь клієнти, користувачі, керівництво, власник продукту і команда Scrum. Вони визначають цілі та функціональність системи, після чого Scrum-майстер і команда зосереджуються на розробці продукту.
- **Щоденний Scrum**: Команда проводить короткі щоденні зустрічі тривалістю близько 15 хвилин, щоб підтримувати прогрес і вирішувати проблеми, що виникають під час розробки. Ці зустрічі покращують комунікацію та підвищують моральний дух команди.

Scrum – це методологія управління з чіткими правилами та практиками, яка не є інженерним процесом із забезпечення якості. Вона покладається на керівництво організацій для впровадження і контролю процесів у Scrum з метою покращення якості продукту. Часто організації комбінують практики XP

і Scrum для оптимізації процесу розробки. Основна увага в Scrum приділяється комунікації та зворотному зв’язку через ітеративні та поступові підходи, що передбачають щоденні зустрічі, безперервну інтеграцію і приймальне тестування для забезпечення необхідного рівня якості.

Тестування програмного забезпечення в рамках методології Scrum є інтегрованим і безперервним процесом, який охоплює всі етапи розробки продукту. У Scrum команда розробки працює за ітераційною моделлю, що називається спринтами, тривалістю зазвичай від одного до чотирьох тижнів. Тестування ПЗ тут відбувається в кожному спринті і має на меті забезпечити якість програмного забезпечення на всіх етапах його створення.

Основні особливості тестування у Scrum:

- **Інкрементальне тестування:** У Scrum тестування виконується паралельно з розробкою функціональності. Команда тестувальників (якщо такі є) або самі розробники перевіряють продукт на кожному етапі розробки, що дозволяє швидко виявляти помилки і дефекти в процесі створення нового інкременту.
- **Автоматизація тестування:** Для забезпечення швидкого циклу зворотного зв'язку в Scrum часто використовується автоматизація тестів. Тестові сценарії автоматизуються, що дозволяє повторювати тести при кожному новому збірці продукту, перевіряючи його на відповідність вимогам без витрат часу на ручну перевірку.
- **Роль тестувальника у Scrum:** У Scrum немає чіткого поділу на ролі "тестувальників" та "розробників". Всі члени команди відповідають за якість продукту. Проте деякі команди можуть мати спеціалістів з тестування (QA), які зосереджені на створенні тестових планів, виконанні ручних та автоматизованих тестів і валідації вимог.
- **Тестування в рамках спринту:** Кожен спринт починається з планування, де визначаються вимоги та функціональність, яку команда має реалізувати і протестувати. Важливо, що всі тести мають бути завершені до кінця спринту, щоб інкремент продукту був готовий

до релізу (або "Done"). Це включає тестування нової функціональності, регресійне тестування та інші види перевірок.

- **Acceptance-тестування (Приймальні тести):** Вимоги у Scrum зазвичай записуються у вигляді історій користувача (User Stories). До кожної історії додаються критерії прийняття, які допомагають зрозуміти, що саме і як має бути протестовано. Приймальні тести виконуються після того, як функціональність розроблена, щоб переконатися, що вона відповідає бізнес-вимогам.
- **Ітеративне тестування:** Програмне забезпечення регулярно вдосконалюється протягом кількох спринтів. Усі нові функції мають бути протестовані, і часто проводиться регресійне тестування для того, щоб переконатися, що нові зміни не спричинили поломок старого функціоналу.
- **Тестування продуктивності та безпеки:** У деяких командах Scrum, особливо коли продукт вже перебуває на пізніших стадіях розробки, можуть виконуватись спеціальні тести на продуктивність, навантаження або безпеку. Ці тести можуть бути включені до кожного спринту або проводитись періодично залежно від специфіки продукту.

Загалом, підхід до тестування в Scrum є гнучким та інкрементальним. Він дозволяє команді швидко адаптуватись до змін, підвищувати якість продукту та зменшувати кількість дефектів на кожному етапі його створення. Така організація тестування сприяє безперервному вдосконаленню програмного забезпечення та підтримує високий рівень якості в умовах швидких ітерацій розробки.

2.3.3 Методологія Crystal Clear

Crystal – це набір методологій, розроблених Алістером Кокберном. Назва «Crystal» походить від класифікації проєктів за двома параметрами: розміром та критичністю. Кожна методологія в цьому сімействі має свій колір (жовтий, помаранчевий, червоний тощо), який відображає складність методології. Вибір

кольору залежить від розміру і критичності проєкту: для більш масштабних проєктів використовують темніші відтінки, оскільки вони вимагають більше ресурсів та координації, на відміну від менших проєктів [34].

В основному розроблено три методології кристалів: Crystal Clear, Crystal Orange і Crystal Orange web. Усі ці методології надають інструменти та стандартні ролі для впровадження в процес розробки програмного забезпечення [34].

Crystal Clear призначений для невеликих проєктів з командами, що складаються з 6-8 розробників. Учасники працюють в одному офісі або в одній кімнаті для забезпечення кращої комунікації під час роботи. Crystal Orange підходить для масштабніших проєктів, де команда налічує від 10 до 40 учасників, а тривалість проєкту становить від 1 до 2 років.

Crystal завжди використовує поетапний цикл розробки з тривалістю кожного етапу від 1 до 3 тижнів. Незважаючи на різноманітність методологій Crystal, вони мають спільні риси та цінності. Основна увага приділяється спілкуванню та співпраці між людьми. Методологія не накладає обмежень на використання інструментів або практик, а також допускає інтеграцію XP та Scrum для підвищення ефективності системи [34].

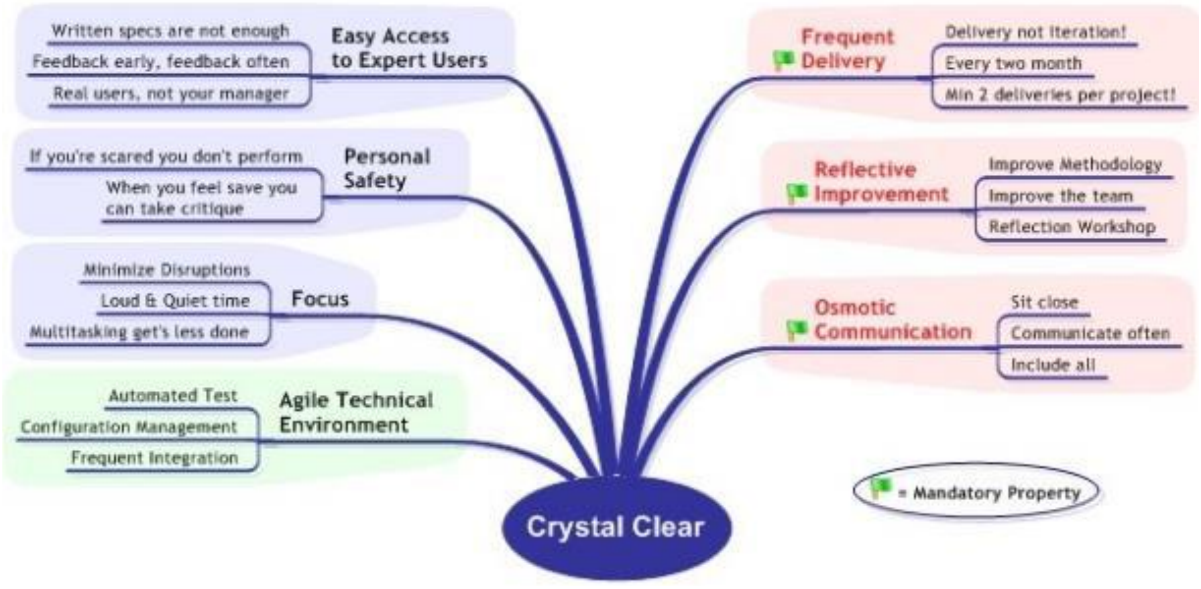

Рисунок 2.3 – Властивості методу Crystal Clear

Джерело: [34]

У Crystal важливу роль відіграють такі ключові особи: старший дизайнер, дизайнер-програміст і користувач. Є також певні практики та стандарти, яких потрібно дотримуватися під час розробки, зокрема:

1. Поступове постачання програмного забезпечення кожні 2-3 місяці;
2. Прогрес оцінюється на основі досягнення ключових етапів (milestones) або прийняття важливих рішень, а не на підставі документів;
3. Crystal передбачає активну участь користувача в процесі розробки;
4. Необхідно проводити автоматизоване регресійне тестування функціональності;
5. Кожен реліз має проходити через два користувацькі огляди;
6. На початку або в середині кожного етапу мають проводитися семінари для налаштування методології.

У методології Crystal Clear тестування програмного забезпечення тісно інтегроване в усі етапи розробки та базується на постійному зворотному зв’язку і співпраці в команді. Головний акцент робиться на комунікації та прозорості процесів, де кожен учасник проекту відповідає за якість. Це передбачає регулярні перевірки коду через неформальні огляди і тісну взаємодію між розробниками та тестувальниками.

Автоматизоване тестування займає важливе місце в Crystal Clear, дозволяючи виявляти помилки на ранніх етапах. Команда використовує юніт-тести для забезпечення правильності роботи компонентів до їх інтеграції. У зв'язку з тим, що методологія орієнтована на невеликі команди, процес тестування залишається простим і ефективним, без перевантаження складними документами чи надлишковими процесами. Велика увага приділяється візуалізації прогресу через, наприклад, використання дошок для відстеження виконаних завдань.

У Crystal Clear немає чіткого поділу ролей, тому всі члени команди активно залучені до тестування, що підсилює відповідальність за кінцеву якість продукту.

2.4 Нові підходи до розробки

У процесі тестування програмного забезпечення головним пріоритетом є точність виявлення помилок та забезпечення якості кінцевого продукту. Для досягнення цих ключових цілей необхідно дотримуватись чітко визначених процедур і практик, деякі з яких є стандартними, а інші — більш гнучкими та адаптивними до змін у розробці. Оскільки в будь-якому програмному продукті можуть бути недоліки, тестування потребує постійного вдосконалення, що вимагає впровадження нових підходів та технік.

Згідно з [9], якість у тестуванні програмного забезпечення — це відповідність продукту до його вимог і призначеного використання, і вона включає два основні аспекти:

1. Якість означає наявність характеристик продукту, які відповідають вимогам клієнтів і забезпечують задоволення від його використання;
2. Якість передбачає мінімізацію або повну відсутність дефектів у програмному забезпеченні.

Agile-тестування забезпечує досягнення необхідного рівня якості програмного продукту завдяки постійному фокусу на вимогах користувача та вдосконаленню процесу тестування функціональних властивостей системи. Це дозволяє уникнути недоліків традиційних методів, де тестування часто відбувається на пізніх стадіях розробки, що збільшує ризики виявлення критичних помилок вже після впровадження продукту. Учасники Agile-команди, включаючи тестувальників, регулярно проводять зустрічі для обговорення зворотного зв'язку від клієнтів та коригування тестових сценаріїв на основі цього фідбеку. Постійний обмін думками між клієнтами, розробниками і тестувальниками, а також тісна співпраця під час процесу розробки є ключовими для забезпечення високої якості продукту. Agile-тестування пропонує новий підхід до процесу контролю якості в розробці програмного забезпечення, який стимулював його широке впровадження для

досягнення цілей якості через безперервну інтеграцію і тестування на кожній ітерації.

Розробка програмного забезпечення – це складне завдання, для спрощення якого використовуються різні процеси. Однак за останні кілька років дедалі більше організацій почали впроваджувати гнучкі методології розробки. Agile привертає увагу завдяки своїй здатності швидко реагувати на зміни та ітераційному підходу до роботи. Ця методологія запропонувала багато способів для зменшення складності процесу розробки.

За результатами опитування, проведеного австралійською консалтинговою компанією в сфері інформаційних технологій, було встановлено, що Scrum є найпоширенішою практикою в організаціях, яку використовували 56% компаній або більше [35]. Загальні результати відображені на рисунку 2.4.

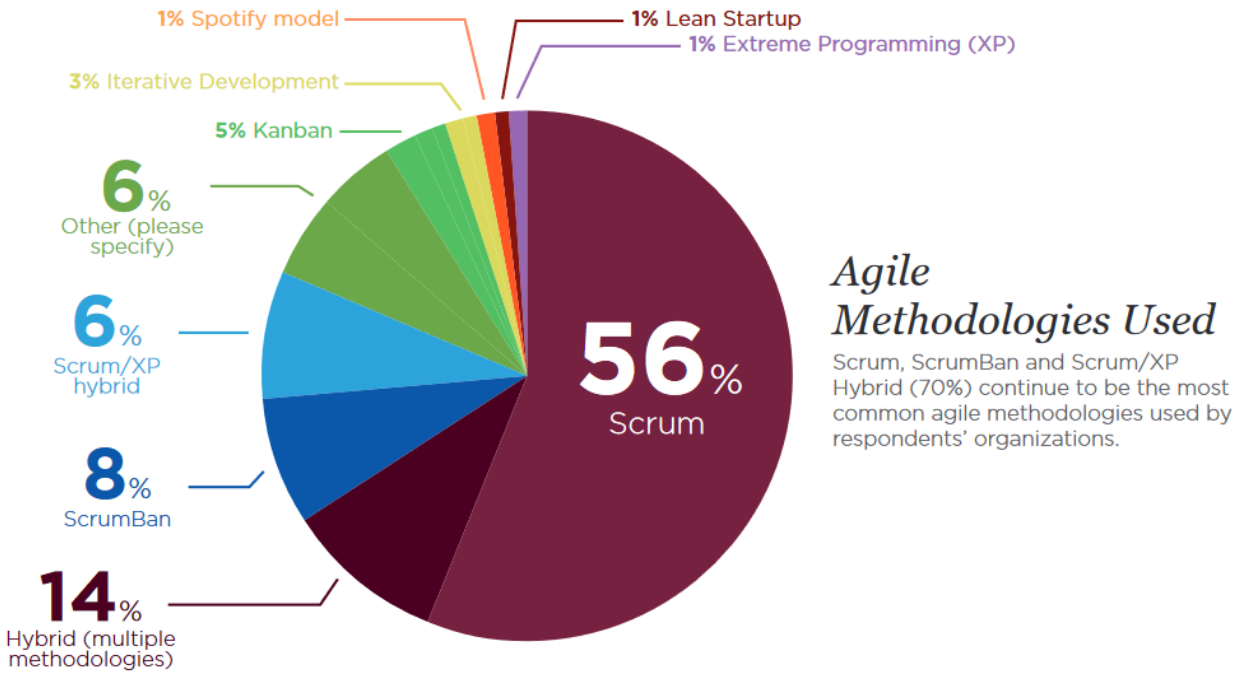

Рисунок 2.4 – Використання гнучких методів розробки ПЗ

Джерело: [35]

Опитування показало, що впровадження гнучких методів розробки програмного забезпечення призвело до покращення якості продукту, спрощення ведення бізнесу та зниження вартості проєкту. Більшість

респондентів відзначили, що акцент на людях замість процесів є позитивною рисою гнучкої розробки. Однак, гнучкі методології часто відзначаються відсутністю структурного планування, а також недостатньою документацією, що є їхнім недоліком. Тим не менш, організації мають намір продовжувати використовувати гнучкі процеси або планують їх впровадження в найближчому майбутньому [35].

У 2005 році компанія Digital Focus організувала онлайн-опитування, яке провела консалтингова фірма в IT-індустрії. В опитуванні взяли участь 137 осіб з 128 організацій у 17 країнах. Згідно з результатами, 90% респондентів мають базові знання про гнучкі практики. Основною мотивацією для впровадження гнучкої розробки у організаціях стало покращення обробки вимог і збільшення швидкості розробки програмного забезпечення для створення якісніших продуктів. Практики зазначають, що здатність адаптуватися до змін є ключовим аспектом гнучкості, але для успішного впровадження необхідні організаційні знання та навички.

Відповіді практиків і дослідження щодо гнучкої розробки чітко демонструють, що гнучкі методології позитивно впливають на організацію роботи, дозволяючи подолати складнощі процесу розробки. Організації відзначають, що застосування гнучких підходів дозволяє їм створювати продукти вищої якості і з меншим рівнем помилок. Забезпечення якості та точності програмного забезпечення є основною метою розвитку для багатьох організацій. Хоча впровадження гнучких процесів супроводжується певними проблемами і недоліками, акцент на взаємодії між людьми та клієнтами та зменшення обсягу документації зазвичай призводить до кращих результатів і підвищення рівня задоволеності клієнтів. Це свідчить про те, що гнучкі підходи сприяють досягненню високих стандартів якості та точності програмного забезпечення.

2.4.1 Порівняння ефективності Agile тестування з класичними підходами

У порівнянні з класичними підходами до тестування, такими як модель «водоспаду» (Waterfall), Agile тестування відрізняється своєю гнучкістю, швидкістю та інтеграцією в усі етапи розробки. Класичні підходи, такі як Waterfall, передбачають чітку послідовність етапів, де тестування відбувається тільки після завершення розробки. Це часто призводить до виявлення помилок на пізніх стадіях проекту, коли їхнє виправлення може бути складним і дорогим. У свою чергу, Agile методології передбачають ітеративний підхід, де тестування є безперервним процесом, інтегрованим у кожну ітерацію. Це забезпечує швидке виявлення та виправлення дефектів, а також дозволяє командам адаптуватися до змін вимог на кожному етапі розробки.

Однією з головних переваг Agile тестування є те, що воно починається на ранніх етапах розробки і триває безперервно протягом всього життєвого циклу проекту. У класичних методологіях тестування відбувається після завершення основних етапів розробки, що створює ризик виявлення критичних помилок на фінальних стадіях, коли їх виправлення може вплинути на терміни релізу. В Agile цей ризик зменшується завдяки постійному тестуванню та зворотному зв’язку, що дозволяє виявляти проблеми одразу і виправляти їх у ході розробки.

Гнучкість є ще однією перевагою Agile тестування. У класичних підходах зміни у вимогах під час розробки можуть спричинити суттєві затримки, оскільки часто потрібно переглядати вже виконані етапи. Agile ж дозволяє командам адаптуватися до змін у реальному часі, завдяки чому тестувальники можуть швидко коригувати свої плани та тести відповідно до нових вимог. Наприклад, в Scrum тестувальники працюють тісно з розробниками під час кожного спринту, що дозволяє їм швидко реагувати на зміни та інтегрувати нові вимоги до процесу тестування. У Kanban, де робота над проектом триває безперервно, швидке тестування дозволяє швидко випускати нові функції без затримок.

Автоматизація тестування є ще однією важливою складовою Agile методологій. У класичних підходах тестування часто виконується вручну, що

уповільнює процес і вимагає багато часу. У Agile автоматизовані тести виконуються після кожної зміни в коді, що дозволяє скоротити час на тестування і забезпечити постійну перевірку всіх функцій продукту. Інструменти, такі як Selenium, JUnit, Cypress та інші, дозволяють командам автоматизувати тести і виконувати їх після кожної інтеграції нових змін у код. Це особливо важливо для забезпечення стабільності продукту при частих релізах, що є типовим для Agile проектів.

Окрім цього, Agile підходи підвищують ефективність за рахунок тісної співпраці між розробниками і тестувальниками. У класичних підходах ці дві ролі часто працюють ізольовано, що може призвести до проблем із комунікацією та непорозумінь у вимогах. В Agile тестувальники залучаються до процесу розробки з самого початку, активно беручи участь у плануванні ітерацій та визначенні тестових сценаріїв. Це дозволяє команді спільно приймати рішення, що підвищує ефективність роботи та знижує ризик виникнення помилок.

Таблиця 2.2 – Порівняння Agile та класичних підходів до тестування:

Критерій	**Agile тестування**	**Класичне тестування (Waterfall)**
Час проведення тестування	Тестування відбувається безперервно на кожному етапі ітерацій	Тестування проводиться після завершення розробки
Гнучкість	Легко адаптується до змін у вимогах на кожній ітерації	Зміни у вимогах спричиняють значні затримки
Автоматизація	Широке використання автоматизованих	Переважно ручне тестування,

	тестів на кожній ітерації	автоматизація зустрічається рідко
Зворотний зв’язок	Постійний зворотний зв'язок на кожному етапі	Зворотний зв'язок після завершення розробки
Співпраця в команді	Тестувальники та розробники працюють разом протягом усього циклу	Тестувальники залучені на пізніх етапах
Ризики	Ризики мінімізуються завдяки постійному тестуванню	Високий ризик виявлення помилок на фінальній стадії

У методологіях Scrum тестування відбувається після кожного спринту, що дозволяє команді забезпечити якість кожної частини продукту перед тим, як вона буде передана клієнту або інтегрована з іншими частинами. У XP (Extreme Programming) тестування є частиною щоденної практики через підхід TDD (Test-Driven Development), що означає, що тестування відбувається ще до написання самого коду, що гарантує його відповідність вимогам. У Kanban тестування є постійним процесом, що підтримує безперервну розробку та випуск нових функцій.

Таким чином, Agile тестування є більш ефективним у середовищах, де важлива гнучкість, швидкість і можливість постійного вдосконалення продукту. Воно забезпечує швидке виявлення помилок, тісну співпрацю між усіма учасниками процесу і зменшує ризики, характерні для класичних підходів, що робить його більш придатним для сучасних динамічних проектів.

2.4.2 Проблеми, пов’язані з впровадженням гнучких технологій розробки

Хоча гнучкість має значні переваги для досягнення кращих результатів, впровадження нового процесу може бути складним завданням. Організації, які зазнають змін, повинні переосмислити свої попередні налаштування і практики. Супротивники гнучких методологій також вказують на труднощі їх впровадження. Нижче описані деякі загальні проблеми та виклики, які можуть стати перешкодою для реалізації гнучких процесів, згідно з [37]:

- Послідовний підхід: За останні 50 років методології розробки програмного забезпечення використовували послідовний підхід, що включає етапи збору вимог, проектування та кодування. Практики таких підходів потребують часу, навчання та постійного моніторингу для освоєння гнучких практик. Організації, що планують впроваджувати гнучкі методології, повинні бути впевнені, що традиційний послідовний підхід не стане перешкодою для нового процесу;
- Індивідуальний опір: Деякі професіонали не готові вивчати нові методології. Вони поділяються на дві групи: перші не бачать цінності в Agile, а другі активно поширюють дезінформацію, щоб запобігти впровадженню нових методів. Цій категорії людей також потрібне детальне навчання новим методологіям;
- Страх перед змінами: Недовіра до нових методів є ще однією перешкодою. Багато спеціалістів бояться, що не зможуть адаптуватися до нових вимог і навичок, що призводить до страху втрати;
- Великомасштабні організації: Великі організації стикаються з проблемами впровадження гнучких процесів, оскільки це часто вимагає наявності клієнта на місці. Через різноманітні громади та обмеження фінансування вони можуть не мати можливості для індивідуального перекладу при кожному контакті з клієнтом, що веде до бажання збирати вимоги одразу;
- Документація: Гнучкість акцентує увагу на розробці коду, а не на документації, такій як UML. Проте деякі спеціалісти все ще вважають,

що ефективність програмного забезпечення залежить від детальної документації та розробки вимог;

- Організаційні сумніви: Деякі організації вважають, що комбінація гнучких і традиційних практик буде для них більш ефективною. Вони вважають, що впровадження нових методів вимагатиме більшої освіти, підвищення кваліфікації та створить високий ризик для стабільності.

Впровадження Agile тестування несе значний потенціал для покращення якості програмного забезпечення, але разом із тим має низку викликів. Головними проблемами є недостатня автоматизація, потреба у кваліфікованих спеціалістах, забезпечення високого рівня комунікації в командах та інтеграція нових методологій у вже існуючі процеси. Проте, попри ці труднощі, Agile тестування має великі перспективи і може значно підвищити ефективність розробки програмного забезпечення.

Однією з найскладніших проблем при впровадженні Agile є автоматизація тестування. У традиційних моделях, як Waterfall, ручне тестування є стандартом, але в Agile воно є недостатньо ефективним через часті ітерації та релізи. Автоматизація потребує інвестицій у ресурси та розробку тестів, що може викликати затримки на ранніх етапах впровадження. Проте перспективи автоматизації у таких методологіях, як Scrum і Kanban, відкривають широкі можливості для підвищення ефективності, знижуючи навантаження на команди та пришвидшуючи процес релізу.

Іншою проблемою є нестача кваліфікованих фахівців. В Agile потрібні не тільки тестувальники, але й фахівці, які мають навички в автоматизації, DevOps, а також здатність працювати у швидкозмінних середовищах. Це створює виклики для команд, які лише починають переходити до Agile, оскільки знайти фахівців з відповідними навичками буває важко. Проте перспектива розвитку цієї сфери є позитивною – попит на таких спеціалістів постійно зростає, а компанії інвестують у навчання своїх команд, що у

довгостроковій перспективі дозволяє підвищити рівень знань та підготувати фахівців для майбутніх проектів.

Важливим аспектом Agile є комунікація в команді. У Waterfall розробники і тестувальники зазвичай працюють окремо, що може призводити до затримок і непорозумінь. В Agile, навпаки, тісна співпраця є обов'язковою умовою для успішної реалізації проекту. Це стосується таких методологій, як Scrum, де команди працюють у коротких спринтах і щодня проводять стендап-збори для обговорення прогресу. У XP (Extreme Programming) взаємодія між розробниками і тестувальниками ще тісніша завдяки таким практикам, як парне програмування та розробка через тестування (TDD), що дозволяє спільно створювати якісний

2.5 Розуміння якості в контексті Agile-розробки

Не буде перебільшенням сказати, що гнучкий підхід до розробки не досяг такої зрілості, як традиційний підхід. Існує чимало критики з боку практиків і дослідників щодо гнучкого методу. Проте, попри ці зауваження, багато організацій використовують гнучкий підхід для розробки програмного забезпечення. Практичний досвід у галузі програмного забезпечення свідчить про те, що цей метод є продуктивним і корисним у багатьох аспектах. Кожен підхід до розробки програмного забезпечення має на меті забезпечення якості як продукту, так і процесу.

«Якщо щось відповідає специфікації, воно хорошої якості » [38].

Згідно з наведеним твердженням, висока якість продукту або послуги визначається їх відповідністю високим вимогам клієнтів. У розробці програмного забезпечення вимоги не є постійними; вони можуть змінюватися навіть у процесі розробки. Традиційні методи розробки часто критикують за їхню недостатню здатність адаптуватися до змін. Крім того, кілька досліджень підкреслили проблему неправильного тлумачення вимог. Отже, забезпечення якості продукту чи послуги стає важким без чіткого розуміння вимог.

Недостатня комунікація між клієнтом і технічними спеціалістами може призвести до неправильного розуміння вимог [39].

У гнучкому підході розробник активно співпрацює з клієнтом для заповнення прогалин у комунікації. Розробник постійно взаємодіє з клієнтом, адаптуючи систему відповідно до його бачення. Це дозволяє організації краще відповідати вимогам замовника та покращує якість програмного продукту. Шостий принцип Agile Manifesto навіть заохочує особисту співпрацю.

Ще однією перевагою залучення клієнта є можливість ефективно справлятися зі змінами вимог. Як і в гнучких проектах, наприкінці кожної ітерації випускається функціонуюча частина програмного забезпечення для отримання зворотного зв’язку. У наступній ітерації нові вимоги інтегруються в попередній реліз, що дозволяє уникнути впливу на проект під час розробки. Прототипи створюються на ранніх стадіях проекту, що допомагає уникнути конфліктів щодо вигляду та функціональності системи. На відміну від традиційного підходу до розробки, в гнучких проектах продукт швидше еволюціонує, ніж розвивається, і ця еволюція зазвичай покращує якість продукту або послуги.

Якість програмного продукту можна оцінювати за бездоганністю, а також за зменшенням або повною відсутністю помилок і дефектів. Для досягнення безпомилковості і оцінки ефективності програмного продукту проводиться тестування. З переходом до гнучкої розробки програмного забезпечення змінюються і підходи до тестування. У будь-якому проекті розробки програмного забезпечення діяльність з забезпечення якості (QA) тісно пов'язана і відповідає за якість як процесів розробки, так і самого продукту. Гнучка розробка програмного забезпечення переосмислює роль і практику забезпечення якості.

У гнучкій розробці програмного забезпечення тестування і розробка відбуваються паралельно, що дозволяє досягти вищого рівня якості. Тестування є основою контролю якості і ключовим етапом для забезпечення якості програмного продукту. У гнучкому процесі помилки або баги

виправляються негайно, як тільки вони виявляються, незалежно від того, хто їх усуває [40]. Завдяки цьому підходу проблеми виявляються і вирішуються на ранніх стадіях, що сприяє економії часу, коштів і ресурсів, а також підвищує загальну якість продукту.

2.6 Забезпечення якості продукту через якість процесів

Оскільки компанія орієнтується на високу якість програмного продукту, вона також намагається зменшити витрати, час і ресурси. У великих і складних проектах програмного забезпечення основна увага приділяється ефективності та управлінню часом, щоб не ставити під загрозу якість. Можна стверджувати, що якщо процес розробки забезпечує максимальну ефективність при мінімальних витратах ресурсів, це є показником високої якості. У цьому розділі ми розглянемо переваги гнучкого підходу до розробки для компанії в процесі розробки.

У [38] автори стверджують, що у гнучкому проекті особливо важливо використовувати прості підходи, оскільки їх легше змінити, легше щось додати до надто простого процесу, ніж відняти щось із надто складного.

Зазначається, що гнучкий процес розробки краще адаптується до змін. Це означає, що гнучкий підхід ефективно реагує на зміни без потреби в додаткових зусиллях і витратах часу. Оскільки гнучка розробка є ітераційною, ресурси організації не застрягають у лінійних процесах. Дослідження показали, і багато практиків підтвердили, що виправлення помилок і дефектів після релізу може вимагати значних витрат часу та ресурсів [41]. Гнучкі практики, такі як парне програмування та тестування, орієнтоване на розробку, допомагають виявити і усунути дефекти до фінального релізу, що в кінцевому підсумку вигідно для організації з точки зору часу та ресурсів.

Гнучкі методи не зосереджені на процесах, а на людях. Принципи Agile Manifesto акцентують увагу на створенні середовища, яке мотивує людей працювати вільно, а не бути обмеженими процесами. Коли під час проекту

розширюється співпраця і залученим особам надається довіра в виконанні завдань, це може суттєво підвищити якість і обсяг роботи [38].

Хоча гнучкі методи є суттєво адаптивними до змін, поступовими та самоорганізованими, значна частина роботи зосереджена на вдосконаленні процесів у гнучкій розробці. Написано кілька книг, які досліджують застосування CMMI в гнучких проектах. Крім того, стандарти Six Sigma та ISO також визнають важливість гнучкої розробки і підтримують цю методологію.

Висновки до розділу 2

Розглянуті у розділі методології розробки програмного забезпечення, такі як екстремальне програмування (XP), Scrum та Crystal Clear, демонструють ефективність використання гнучких підходів до забезпечення якості програмного продукту. Основні принципи, такі як тісна співпраця з клієнтами, короткі ітерації, швидка адаптація до змін та постійна інтеграція, дозволяють значно підвищити якість як самого процесу розробки, так і кінцевого продукту.

Особливу увагу приділено ролі забезпечення якості (QA) у гнучких процесах розробки, де паралельне тестування та розробка дозволяють швидко виявляти та виправляти помилки, мінімізуючи витрати та зусилля на пізніших етапах проєкту. Рефакторинг, парне програмування та безперервна інтеграція є ключовими практиками, що сприяють підвищенню якості коду та зменшують ризик виникнення дефектів.

Застосування гнучких методологій довело свою ефективність у сучасних проєктах, що підтверджується дослідженнями та відгуками практиків. Вони дозволяють адаптуватися до змінних умов, підвищують продуктивність команд та забезпечують високу якість програмного продукту, що робить їх популярними у багатьох організаціях.

РОЗДІЛ 3. РЕЗУЛЬТАТИ РОБОТИ З ВПРОВАДЖЕННЯ SQA В AGILE-МЕТОДОЛОГІЇ РОЗРОБКИ

3.1 Agile-технології тестування

Гнучкі методології розробки, також відомі як легкі методології, все частіше використовуються та приймаються в проектах розробки програмного забезпечення. Ці методології характеризуються адаптивністю до змін і гнучкістю, а такі функції, як рефакторинг, привертають увагу багатьох практиків до гнучкого підходу. Деякі спеціалісти підтримують гнучку розробку, інші виступають проти неї, а деякі рекомендують комбінувати гнучкі методи з традиційними практиками управління проектами [39]. У цьому розділі ми детальніше розглянемо забезпечення якості в Agile-розробці та управлінні проектами, зокрема погляди практиків на гнучкий підхід. Також будемо детально обговорювати запропоновані методики, порівнюючи їх з результатами опитувань професіоналів.

Парне програмування та тестова розробка (Test Driven Development – TDD) є одними з основних практик гнучкої розробки для забезпечення якості програмних продуктів. Використовуючи ці методи, гнучка розробка інтегрує тестування як невід'ємну частину проекту. Agile суттєво змінила підходи до тестування, передавши відповідальність за нього від тестувальників до розробників. В Agile розробники повинні писати тести та перевіряти свій код або код один одного через парне програмування. Очікується, що замовник активно бере участь у проекті протягом усього його циклу, включаючи приймальні випробування, які є його відповідальністю. Прихильники гнучких методик вважають, що залучення клієнтів до проекту та тестування сприяє створенню програмного забезпечення, яке краще відповідає вимогам.

Розробка на основі тестування (TDD) набирає популярності в гнучкому середовищі. TDD акцентує увагу на написанні тестів до початку кодування і частій інтеграції нового коду. У TDD новий код може бути як новим фрагментом коду, так і зміненим існуючим кодом, який інтегрується після

внесення змін для реалізації оновлених вимог (згідно з відгуками користувачів) [42]. TDD черпає вплив з підходу Test-First Development (TFD), який передбачає написання всіх тестів до початку фактичного програмування.

Щоб візуалізувати концепцію TFD зараз, давайте представимо діаграму UML для TFD (див. рис. 3.1).

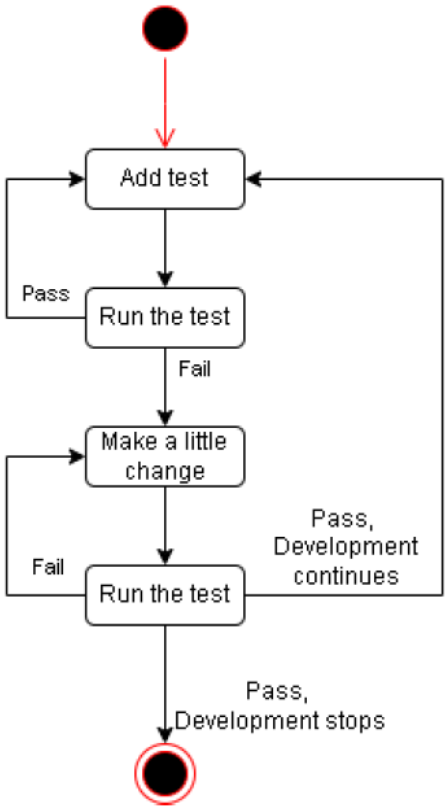

Рисунок 3.1 – Test-First розробка

Джерело: [42]

TDD можна розглядати не лише як метод тестування, а й як підхід до поетапного та поступового проектування програмного забезпечення. Як показано на наведеній діаграмі, при впровадженні TFD кожен фрагмент коду має проходити всі тест-кейси до того, як його буде написано.

TDD трансформував роль забезпечення якості в розробці програмного забезпечення, адже тестування тепер виконується розробником. Практики Agile стверджують, що це постійне тестування веде до створення продукту

високої якості. В [43] автор визначає гнучке забезпечення якості як «розробку програмного забезпечення, яке здатне адаптуватися до змін на вимогу клієнта. Це передбачає, що часта доставка перевіреного, функціонуючого та затвердженого клієнтом програмного забезпечення в кінці кожної ітерації є ключовим аспектом гнучкого забезпечення якості».

На практиці контроль якості в гнучкій розробці формується на основі відгуків клієнтів та розробників, які також виконують роль тестувальників. Хоча безперервне тестування сприяє досягненню високої якості, Agile також піднімає нові питання і критику, які обговорюють практики та дослідники. Перед тим як розглянути проблеми та критику, з якими стикається система контролю якості в гнучкій розробці, давайте розглянемо ще кілька аспектів забезпечення якості в цьому підході.

3.2 Парне програмування

Згідно з [44], парне програмування (Pair Programming – PP) передбачає, що весь робочий код пишеться двома людьми за одним комп'ютером: один виступає як "водій" (той, хто пише код), а інший як "спостерігач" або "навігатор" (той, хто аналізує код, пропонує ідеї та стежить за загальною якістю). PP є однією з основних практик XP.

Завдяки спільній роботі двох програмістів код перевіряється одразу на етапі написання. Спостерігач (навигатор) відстежує можливі помилки, потенційні баги або недоліки в логіці, що робить процес своєрідним "живим" тестуванням. Спільна робота дозволяє уникнути типових помилок, як-то невраховані випадки або забудькуватість щодо тестів. Парне програмування заохочує кращі практики, такі як дотримання стилю кодування та правильне структурування коду, що позитивно впливає на загальну якість ПЗ.

Парне програмування часто інтегрується з підходом розробки через тестування (TDD), коли тести пишуться до написання коду. Один з учасників пари може зосередитися на написанні тестів, тоді як інший — на створенні функціональності, що відповідає цим тестам.

У розробці висловлюються думки, що парне програмування (PP) може сприяти досягненню високої якості, оскільки, як зазначає [44], ця практика може поліпшити якість дизайну та зменшити кількість дефектів. Проте головна дискусія полягає в довірі до продуктивності розробленого програмного продукту, створеного розробниками, а не професійними тестувальниками. Гнучка розробка, безсумнівно, підкреслює важливість наявності висококваліфікованого та досвідченого персоналу, але питання залишається: якщо якість забезпечується завдяки кращим кадрам, то як оцінюється продуктивність самої методології розробки?

Учасники визнають, що для успішної реалізації гнучких проектів потрібен певний рівень досвіду серед персоналу. Існує загальна думка, що від 25% до 33% членів команди повинні бути компетентними та досвідченими. Компетентність і досвід тут включають знання в розробці подібних систем, попередній досвід у відповідних технологіях та навички міжособистісного спілкування. Головне в тому, що для успішного проекту важливіші ті, хто вже працював над схожими системами, ніж ті, хто має досвід лише в гнучкому середовищі.

3.3 Рефакторинг

Рефакторинг є важливою складовою гнучких методологій. Згідно з [46], рефакторинг коду визначається як процес реструктуризації існуючого коду, який змінює його внутрішню структуру, не впливаючи на зовнішню поведінку. Це здійснюється шляхом серії невеликих змін, які зберігають функціональність системи. Кожна така зміна (називана «рефакторингом») має невеликий ефект, але сукупність таких змін може призвести до значної реструктуризації. Оскільки кожний окремий рефакторинг є малим, ймовірність помилок зменшується. Крім того, система залишається повністю працездатною після кожної невеликої зміни, що знижує ризик серйозних дефектів під час реструктуризації.

Згідно з [44], дизайн системи розвивається шляхом трансформацій, які досягаються через рефакторинг у XP. Основною метою рефакторингу є підвищення зрозумілості коду, зменшення його складності та спрощення його структури. Дослідження та практика доводять, що рефакторинг позитивно впливає на якість програмного забезпечення, знижуючи ймовірність помилок. Крім того, рефакторинг також слугує як метод перегляду коду, оскільки процес перегляду дозволяє розробникам виявляти та усувати помилки разом із реструктуризацією коду.

Під час рефакторингу можуть бути застосовані різні техніки, такі як:

- **Red-Green-Refactor**: популярна методика, що включає три етапи — зупинитися і розглянути, що потрібно розробити (Red), забезпечити проходження базового тестування (Green), впровадити поліпшення (Refactor);
- **Рефакторинг шляхом абстрагування**: зменшення надлишкового коду через створення підкласів або згортання ієрархії;
- **Метод композиції**: впорядкування коду для зменшення дублювання, що може включати екстракцію та вбудований рефакторинг.

Після застосування цих технік тестування допомагає перевірити, чи залишилася функціональність ПЗ незмінною і чи не з'явилися нові дефекти. Це включає як ручне, так і автоматизоване тестування. Ручне тестування дозволяє виявити дефекти, які можуть бути пропущені автоматизованими тестами, особливо ті, що стосуються користувацького інтерфейсу та досвіду користувача

В Дослідженні [48] проаналізовано розвиток програмного забезпечення з використанням XP протягом понад двох років. В ньому порівняно результати XP (гнучка розробка) з розробкою, що керується планом. Виявилося, що рефакторинг у XP значно знижує складність коду порівняно з плановою розробкою. У плановій розробці код може бути реструктуризований лише після завершення написання та тестування. Натомість у XP код постійно

реструктуризується і інтегрується, що зменшує його складність і допомагає виявляти помилки на ранніх етапах розробки.

Рефакторинг має значний і позитивний вплив на забезпечення якості, оскільки допомагає зменшити кількість помилок і, таким чином, економить час. У гнучкій системі код постійно інтегрується та рефакторується, що дозволяє ефективно вирішувати проблеми сумісності на ранніх етапах розробки програмного забезпечення. Крім того, практики стверджують, що рефакторинг коду зменшує необхідність у великій документації.

3.4 SPI та гнучка методологія

Удосконалення процесів розробки програмного забезпечення (Software Process Improvement – SPI) надає компаніям систематичний підхід до виконання завдань, впливаючи на організацію в адміністративному, а не технічному аспекті [49]. Сьогодні компанії вірять, що покращення процесів розробки програмного забезпечення дозволяє їм створювати високоякісні програмні продукти.

Однією з головних цілей SPI є покращення загальної якості продукту, а тестування відіграє критичну роль у виявленні дефектів на різних етапах розробки. SPI допомагає командам удосконалити свої методи тестування, зокрема автоматизувати тести, збільшити покриття тестами та інтегрувати тести на ранніх етапах розробки. У контексті SPI тестування стає не ізольованим етапом, а інтегрованою частиною кожної фази розробки. Використовуються практики, як-от безперервна інтеграція та тестування, що дозволяють рано виявляти дефекти та уникати великих витрат на їх виправлення на пізніх стадіях.

SPI включає ретельний аналіз результатів тестування, щоб визначити, які аспекти процесу розробки потребують покращення. Наприклад, якщо тестові сценарії часто виявляють одні й ті ж помилки, це може свідчити про проблеми в плануванні чи проектуванні продукту. Аналіз тестових метрик допомагає керівникам проектів ухвалювати рішення про поліпшення процесів розробки.

Протягом різних етапів розвитку були впроваджені різні комерційні стандарти та моделі для покращення процесу розробки програмного забезпечення, зокрема стандарти ISO та модель інтеграції зрілості можливостей компанії (Company Maturity Model Integration – CMMI). Ці моделі допомагають оцінити ефективність існуючих тестових процесів і визначити, на яких етапах вони потребують покращення. Зрілі процеси тестування характеризуються високим рівнем автоматизації, гарним плануванням і організацією тестування, а також ефективним зворотним зв'язком між розробниками та тестувальниками.

Для отримання сертифікації за певним стандартом або моделлю на певному рівні, організація повинна повністю відповідати цим стандартам у своєму процесі. Це вимагає детальної документації розробки продукту протягом всього процесу, дотримання і контролю стандартних процедур, а також регулярного проведення аудитів та оцінювальних процедур. У цьому контексті забезпечення якості відіграє ключову роль, оскільки персонал з якості відповідає за впровадження, моніторинг та управління обраними моделями чи стандартами для вдосконалення процесу розробки як на рівні організації, так і на рівні окремих проектів.

Через обмежений час і масштаби поточного дослідження ми не можемо охопити всі стандарти та моделі для вдосконалення процесів розробки програмного забезпечення (SPI). У цьому розділі ми зосередимося на CMMI, оскільки ця модель є широко використовуваною в сфері SPI. У галузі деякі вважають CMMI набором стандартів, але експерти CMMI визначають її як модель для SPI. Безсумнівно, постійне вдосконалення процесів розробки позитивно вплине на якість продуктів організації. Проте ці підходи та практики більше властиві плановій розробці програмного забезпечення. Гнучкі методології, на думку деяких, не зовсім сумісні з цими стандартами і моделями, і дискусії щодо SPI та Agile тривають.

Традиційні методології розробки часто відокремлюють забезпечення якості від основного процесу розробки. На відміну від цього, гнучка розробка

інтегрує практики забезпечення якості безпосередньо в розробку, а не як окрему активність. Іноді через управлінські чи організаційні вимоги замовник або проект можуть вимагати дотримання певних стандартів, де важливу роль відіграє генерація документації для відповідності цим стандартам.

Згідно з [48], існують проблеми, які ускладнюють сумісність між CMMI та гнучкими методологіями. Однією з основних причин є те, що організації, які впровадили CMMI, зазвичай були великими з розвиненою управлінською ієрархією та структурою. Натомість організації, що використовують гнучкі методології, часто орієнтовані на менші проекти або команди.

У 2005 році було проведено тематичне дослідження [48], яке аналізувало використання CMMI та гнучких методологій для підвищення зрілості організації. Це дослідження стверджує, що існуючі гнучкі методи проектного управління не забезпечують необхідних інструментів для врахування організаційних аспектів SPI. Зокрема, вони не охоплюють важливі аспекти, такі як систематичне визначення, перевірка, упаковка та зберігання результатів SPI в гнучких проектах [49]. Зазначені аспекти є відповідальністю SQA. Вдосконалення в організації, проекті чи розробці спрямовані на підвищення якості програмних продуктів. Якщо гнучкі методики використовуються для створення якісних продуктів без застосування CMMI, існує значний потенціал для подальшого вдосконалення.

У дослідженні компанії Ericsson, яке було спрямоване на подолання розбіжностей між SPI та гнучкою розробкою, виявилося, що індустрія потребує вдосконалення практик гнучкого SPI [49]. У цьому дослідженні, яке охоплювало 18 різних проектів, застосовували дві тактики SPI: Supertank Tactic, що ґрунтується на принципах CMMI, і Motorboat Tactic, що спирається на гнучке мислення. Результати показали, що Motorboat Tactic забезпечила кращі результати і позитивніші ефекти. Однак дослідники відзначили, що детальні інструкції для Supertank Tactic були доступні, оскільки вона базувалася на CMMI. Незважаючи на успішність другої тактики, дослідники зазначили, що гнучка розробка фокусується на розробці на рівні проекту, тоді

як CMMI зосереджена на вдосконаленні процесів розробки на рівні організації.

У плановій розробці SPI охоплює всі аспекти забезпечення якості та розробки програмного забезпечення через чітко визначені, організовані та стандартизовані процеси. Хоча всі учасники проекту беруть участь у SPI, саме персонал з забезпечення якості відповідає за його виконання. Спеціалісти SQA здійснюють опитування, оцінки, зустрічі та внутрішні аудити, і є експертами в організації, оскільки навчають та впроваджують SPI, тестувальники стають ключовими учасниками процесу покращення, оскільки вони не тільки виконують тести, а й активно беруть участь у вдосконаленні методологій і процесів тестування. Вони також допомагають у виявленні кореневих причин проблем і пропонують рішення для їхнього уникнення в майбутньому. У випадку з гнучкими методологіями, організаційна роль контролю якості зменшується, оскільки розробники, хоч і мають знання про тестування та проектування, можуть бути менш обізнані про SPI на рівні організації. Література свідчить про необхідність перегляду ролі SQA у проектах гнучкої розробки, щоб покращити знання організації та її зрілість для досягнення максимальних результатів.

3.5 Оцінка результатів тестування в Agile проектах

В Agile методологіях тестування не обмежується лише кінцевими етапами розробки, а відбувається безперервно, що дозволяє швидко виявляти та виправляти дефекти, забезпечуючи стабільність продукту та високу швидкість його випуску.

Одним із ключових аспектів оцінки результатів тестування є аналіз кількості виявлених дефектів. Це дозволяє команді зрозуміти, наскільки якісно були протестовані функції продукту під час ітерації, і чи є необхідність у додаткових заходах для поліпшення якості. Оцінка кількості критичних і некритичних дефектів допомагає краще розподіляти пріоритети та визначати

проблемні ділянки, що потребують більш ретельного тестування у наступних ітераціях.

Покриття тестами також є одним із основних показників якості процесу тестування. В Agile командах прагнуть забезпечити високий рівень покриття коду тестами, особливо за допомогою юніт- та регресійного тестування. Це дозволяє впевнитися, що більшість функцій була протестована на різних рівнях, зменшуючи ймовірність виникнення непомічених помилок. Водночас важливо пам'ятати, що покриття тестами не є єдиною гарантією якості: необхідно оцінювати і те, наскільки тести охоплюють реальні сценарії використання продукту.

Час на виконання тестів є критично важливим у проектах з частими релізами, таких як Scrum або Kanban, де ефективність кожної ітерації залежить від швидкості тестування. У Scrum, де кожен спринт має фіксовану тривалість, важливо забезпечити, щоб автоматизовані тести могли бути виконані швидко та не затримували реліз. В Kanban, де робота над новими функціями може бути безперервною, швидке тестування дозволяє швидше випускати нові версії продукту, не накопичуючи нерелізованих змін.

Час на виправлення дефектів є важливою метрикою, що відображає, наскільки швидко команда реагує на виявлені помилки. В ідеалі, команда повинна виявляти та виправляти дефекти ще під час поточної ітерації, щоб вони не вплинули на випуск продукту. У методологіях, таких як Extreme Programming (XP), цей процес ще більш інтегрований, оскільки команди використовують підхід розробки через тестування (TDD), що дозволяє виправляти помилки безпосередньо під час розробки коду.

Метрика пропущених дефектів є важливою для оцінки того, наскільки тестування ефективно виявляє критичні помилки. Якщо після завершення ітерації та релізу в продакшн виявляються суттєві проблеми, це свідчить про те, що тестування не повністю охопило важливі аспекти продукту. Це може вимагати перегляду процесів регресійного тестування або збільшення

покриття автоматизованими тестами для зменшення ризику пропуску критичних помилок у майбутньому.

Для кожної з методологій Agile, таких як Scrum, Kanban, або XP, оцінка результатів тестування виглядає дещо по-різному. У Scrum, метрики, такі як кількість виявлених дефектів, покриття тестами та час на виправлення помилок, використовуються для оцінки якості кожного спринту. У Kanban більш важливим стає швидкість тестування і час до виявлення дефекту, що дозволяє постійно випускати нові функції. В XP більший акцент робиться на швидке виявлення та виправлення помилок за допомогою тестування, що супроводжує розробку (TDD) і безперервну інтеграцію.

Таким чином, оцінка результатів тестування в Agile проектах дозволяє командам гнучко адаптувати процеси тестування, швидко виправляти помилки та забезпечувати високу якість продукту протягом всього циклу розробки. Аналітика метрик тестування допомагає приймати обґрунтовані рішення щодо вдосконалення тестування в наступних ітераціях та підвищення стабільності продукту.

3.6 Запропоноване рішення

Ми зібрали та проаналізували доступну літературу та дослідницькі дані через огляд літератури в галузі Agile-розробки та процесів забезпечення якості. Виявлено такі ключові аспекти літератури в цій області:

- Покращення якості через гнучку розробку: Обговорюються як переваги, так і критика гнучких методологій, включаючи результати опитувань, керівництва та відгуки практиків;
- Інтеграція тестування в Agile-команди: Огляд рекомендацій, переваг і критики, що стосуються інтеграції QA в гнучкі підходи;
- Відсутність визначених стандартів для гнучкої розробки: Проблематика та критика, що виникає через відсутність чітких стандартів, а також результати опитувань;

- Дослідження для підвищення продуктивності в гнучких проектах: Результати досліджень і пропозиції щодо поліпшення продуктивності проектів, що використовують гнучкі методології.

На основі цього огляду літератури ми дійшли висновку, що роль практик забезпечення якості в гнучких проектах потребує перегляду. Хоча наше фундаментальне спостереження базується на літературних даних, ми також провели емпіричне дослідження, щоб підтвердити чи уточнити ці результати.

Відповідно до цього твердження, QA має бути інтегрований у процес розробки, а не замінений, де "замінений" означає, що більшість завдань з забезпечення якості у гнучких проектах виконується розробниками або персонал із QA повинен виконувати роль розробника. Безсумнівно, гнучкі методології досягають високої якості завдяки своїй інкрементній та тестовій природі. Однак відсутність систематичних, організованих і чітко визначених процедур та стандартів вказує на те, що діяльність SQA у гнучких проектах має значний потенціал для вдосконалення. Основним напрямком нашого підходу є «перегляд діяльності SQA, а не її заміна».

Діяльність з забезпечення якості в гнучких проектах зосереджена на тестуванні та зворотному зв’язку. Наявна література в основному акцентує увагу на тестуванні та його різних підходах у контексті гнучкого забезпечення якості. Гнучкі методології відрізняються поступовістю, адаптивністю до змін та самоорганізацією. Однак будь-яка система потребує лідерства, оскільки самоорганізація не завжди ефективна при ускладненні вимог і системи. Хоча ці методології позиціонуються як орієнтовані на людей, зосередження на цьому аспекті вказує на те, що досвід людей може використовуватися не завжди за призначенням, для якого вони були навчені.

Дослідження в області програмного забезпечення і практичні спостереження свідчать, що гнучкі методології можуть бути менш ефективними для розробки складних програмних систем. Ці методології часто вимагають формування команди з найкращих спеціалістів, де експерти та аналітики з дизайну можуть виконувати роль розробників, а розробники, в

свою чергу, займаються тестуванням для забезпечення якості. У традиційній плановій розробці програмного забезпечення QA функціонує як окрема сутність з моніторингом. В Agile ж забезпечення якості інтегрується безпосередньо у процес розробки, що може вимагати від спеціалістів з QA виконання ролі розробників у проектах.

Незважаючи на адаптивність Agile, важливо не нехтувати стандартизацією процесів забезпечення якості. Успішне управління тестуванням передбачає поєднання гнучких методів із системним підходом, таким як вдосконалення процесів розробки (SPI). SPI спрямоване на поліпшення якості не лише кінцевого продукту, але й самого процесу розробки. Це дозволяє підвищити ефективність як на рівні команди, так і на рівні організації. Вдосконалення процесів розробки програмного забезпечення (SPI) є ключовим елементом стандартних практик SQA. Оскільки галузь визнає важливість та переваги SPI, було проведено чимало досліджень, щоб забезпечити сумісність між CMMI та Agile.

Таким чином, для гнучкого підходу до розробки необхідно переглянути роль та функції управління тестуванням, для забезпечення якості (SQA) у проектах. Підвищення значення SQA може сприяти підвищенню ефективності гнучких методологій на рівні організації.

Пропоноване рішення з оптимізації якості продукту передбачає інтеграцію експертів із забезпечення якості безпосередньо в Agile-команди, що дозволяє ефективніше контролювати та підтримувати стандарти якості на всіх етапах розробки. Це може бути досягнуто через регулярне тестування, рефакторинг, парне програмування, а також використання гнучких підходів для безперервного вдосконалення продукту. Для збереження гнучкості важливо стимулювати активне спілкування як між командами, так і всередині них. Наступний розділ детально розгляне, як застосування цього додаткового рівня забезпечення якості разом із зворотним зв’язком може бути реалізовано в організації.

Впровадження культури тестування в гнучких проектах може значно підвищити якість продукту, але не можна знижувати якість процесу для досягнення успіху в бізнесі. Практики Agile часто менш зацікавлені в сертифікації CMMI або інших комерційних стандартах якості продуктів і процесів. Огляд літератури показав, що Agile переважно фокусується на тестуванні, що може призвести до недооцінювання інших аспектів контролю якості. Перекладання відповідальності за аналіз вимог і тестування на розробника може перевантажити його, що негативно вплине на якість як процесу, так і продукту. Роль спеціалістів з забезпечення якості в проектах розробки програмного забезпечення є безсумнівною.

Ми вважаємо, що роль фахівців із забезпечення якості в гнучких проектах потребує перегляду для підвищення якості, замість того щоб перекладати їх обов'язки на розробників. Пропонована методика передбачає інтеграцію фахівців із забезпечення якості безпосередньо в команду розробників. Усі аспекти забезпечення якості повинні контролюватися цими фахівцями, а також важливо заохочувати і підтримувати їх співпрацю, щоб зберегти гнучкість проекту.

У гнучких методологіях розробник часто виконує роль тестувальника, або навпаки, що суттєво змінює їхні обов'язки і може створити певні виклики для експертів. Запропонований підхід дозволяє організації досягти високої якості, оскільки всі ресурси працюватимуть у межах своїх спеціалізацій. Розробники не повинні тестувати свій власний код, хоча в гнучких проектах значна частина тестування виконується саме ними.

Ми вважаємо, що гнучким методологіям не вистачає стандартів для досягнення їхніх можливостей і зрілості, що відрізняє їх від традиційних підходів до розробки. Це може бути пов'язано з тим, що всі експерти активно залучені до розробки і не приділяють достатньої уваги моніторингу та вимірюванню проектів на організаційному рівні. Впровадження запропонованого підходу дозволить організації розробити та впровадити стандарти і ефективно контролювати якість складних проектів.

Замість того, щоб навчати замовника, краще використовувати досвід тих, хто має навички технічної реалізації потреб замовника. Постійна взаємодія з клієнтом також позитивно впливає на якість продукції та гнучкість. Наша пропозиція полягає в тому, щоб не залучати розробника до цього процесу. Персонал із забезпечення якості повинен регулярно взаємодіяти з клієнтом для забезпечення високої якості продукту.

Під час огляду літератури ми виявили, що багато практиків вважають ведення детальної документації важливим аспектом. Документація необхідна не лише для клієнта, але й для підтримки якості продукту в майбутньому. Крім того, відгуки вказують на те, що розробник не може самостійно виконати всі вимоги щодо якості та процедури без підтримки фахівців з контролю якості. Ми вважаємо, що інтеграція знань і досвіду традиційних практик забезпечення якості в гнучкий проект через додатковий рівень контролю якості може сприяти підвищенню якості продукту та зрілості гнучких методологій.

3.7 Застосування додаткового рівня контролю якості

Згідно з пропонованим підходом, кожній команді розробників слід призначити принаймні одного експерта з якості, який буде працювати як частина кожної з цих команд. Цей експерт з якості, який буде виконувати функції контролю, а не розробки, буде називатися вузлом забезпечення якості. Такий додатковий рівень контролю якості може підтримувати процес розробки, забезпечуючи при цьому гнучкість проекту.

Тестування – це виявлення рівня якості. Розробник може виконувати значну частину тестування, застосувавши автоматизоване тестування в організації. Але саме QA-вузли повинні писати тести, а не розробники. Крім того, ці вузли забезпечення якості повинні проводити ручне тестування, оскільки вони мають досвід у тестуванні та мають ширше бачення цієї сфери.

У гнучких методологіях процес збору вимог або історій користувачів відбувається через співпрацю між користувачем і розробником. Ми пропонуємо, щоб цю задачу виконували QA-вузол та керівник групи

розробників, а потім вони повинні передати ці вимоги розробникам. Такий підхід забезпечить ефективніше втілення вимог користувачів, навіть якщо вони є складними. Крім того, присутність керівника групи розробників дозволить також оцінити вимоги з технічної точки зору. Оскільки в гнучких системах вимоги часто недостатньо документовані і представлені у вигляді історій користувачів, розробники повинні інтерпретувати ці історії для реалізації їх у функціональність програмного забезпечення.

Співпраця є ключовим аспектом гнучких проектів. Проте, коли в ній беруть участь люди з різними підходами, можуть виникати непорозуміння та втрати часу. Ми пропонуємо, щоб вузли забезпечення якості грали провідну роль у взаємодії з клієнтами, оскільки QA-фахівці володіють як технічним, так і соціальним досвідом.

Документація була важливою проблемою як у традиційних, так і в гнучких методологіях. Гнучка розробка не відкидає документацію, але підтримує її в розумних межах, щоб зекономити час і ресурси. У випадках, коли вимоги клієнтів або складність програмного забезпечення зростають, документація повинна бути більш детальною. У традиційних методологіях за документацію відповідає QA, тому в нашому запропонованому підході QA-вузли повинні управляти документацією лише в межах своїх команд розробників, що допоможе уникнути зайвого навантаження на розробників.

Взаємодія на додатковому рівні контролю якості повинна бути такою ж регулярною процедурою, як і щоденні зустрічі. Команди розробників мають активно співпрацювати одна з одною та зі своїми QA-вузлами. Ці QA-вузли повинні підтримувати ефективну взаємодію для моніторингу прогресу, обговорення критичних питань, обміну знаннями та ресурсами в межах проекту. Таким чином, QA-вузли функціонують як додатковий рівень контролю якості в проекті.

SPI потребує безперервного вивчення конкретних стандартів чи моделей. Персонал з забезпечення якості відіграє активну роль у запуску та підтримці SPI в організації завдяки своїм експертним знанням у цій сфері.

Крім того, QA-вузол проекту також може вжити заходів для ініціювання SPI в гнучкому середовищі.

Нижче ми збираємося представити організаційний погляд на запропоноване нами рішення для подолання недоліків гнучкого SQA (див. рис. 3.2).

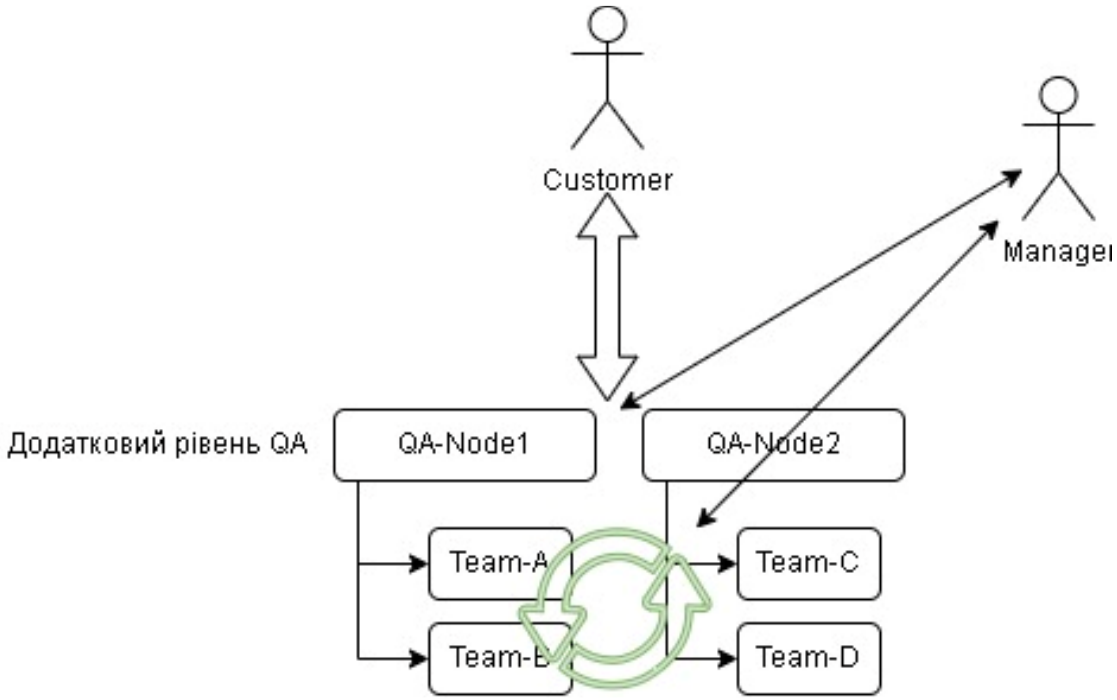

Рисунок 3.2 – Організаційний вигляд Agile-проекту після впровадження додаткового рівня контролю якості

Джерело: *розроблено автором*

На наведеному рисунку, де графічно представлено додатковий рівень якості, кожному вузлу забезпечення якості призначено по дві команди розробників (A, B, C і D). Кількість команд або QA-вузлів може варіюватися в залежності від організаційних ресурсів і досвіду кожного вузла. Ці вузли забезпечення якості підтримують гнучку розробку у своїх командах і беруть участь у діяльності як члени команди. Незважаючи на основну концепцію проекту, запропонований додатковий рівень QA є автономним і має забезпечувати співпрацю через регулярні зустрічі всіх QA-вузлів. Замовник

повинен активно взаємодіяти з QA-вузлом та керівником команди. Роль керівництва полягає у наданні нагляду та виконанні відповідних рішень без порушення технічних аспектів, щоб підтримувати гнучкість проекту.

Під час дослідження виявлено, що запропонована методика може мати потенційні недоліки, які слід враховувати до її повного впровадження через постійні дослідження. Нижче наведені деякі з можливих недоліків:

- Витрати часу на комунікацію з QA-вузлами;
- Переміщення частини обов'язків від розробників до персоналу з контролю якості може уповільнити хід проектів на початкових етапах;
- Розробка нових інструментів для впровадження запропонованої техніки може вимагати значних витрат часу та ресурсів.

Переваги та недоліки будь-якої методики можна чітко оцінити лише через її практичне впровадження. Оскільки наше дослідження не передбачало реалізації цієї методики в реальних умовах, результати її застосування залишаються невідомими. Використання запропонованого підходу може виявити додаткові недоліки, які можуть бути враховані для подальшого вдосконалення методики.

Висновки до розділу 3

У результаті аналізу було встановлено, що інтеграція практик забезпечення якості (SQA) в Agile-проекти є критичною для підвищення якості кінцевого продукту. Запропоноване рішення полягає в тому, щоб включити експертів із забезпечення якості безпосередньо в команди розробників, що дозволить більш ефективно контролювати та підтримувати стандарти якості на всіх етапах розробки. Додатковий рівень контролю якості (QA-вузли) забезпечує взаємодію між командами, надаючи фахівцям QA можливість брати активну участь у розробці, зокрема через парне програмування та рефакторинг.

Зазначено, що впровадження такого підходу сприятиме не тільки підвищенню якості програмного забезпечення, але й допоможе покращити процес управління проєктами через регулярне тестування та взаємодію між учасниками. Однак важливо зазначити, що для успішної реалізації запропонованого рішення необхідно врахувати потенційні виклики, зокрема витрати часу на комунікацію та можливі затримки через необхідність адаптації нових підходів.

Таким чином, подальші дослідження мають бути спрямовані на тестування ефективності цього підходу на практиці для забезпечення його успішної інтеграції в процеси Agile-розробки.

ВИСНОВКИ

Аналіз даних проведено на основі результатів огляду літератури. Представлено всебічний аналіз, що відповідає поставленим дослідницьким питанням. Відповідь на перше питання дослідження, яке стосується "ролі заходів з забезпечення якості програмного забезпечення в гнучкій розробці згідно з існуючою літературою", наведена у відповідних розділах роботи. Виявлено, що гнучка розробка програмного забезпечення є ефективною та корисною для досягнення клієнтської задоволеності порівняно з традиційними методами. Також показано, що процеси перевірки якості в Agile мають характер, що включає безперервне тестування, тестування після кожного етапу дизайну, а також фокус на історіях користувачів і реалізації інтерфейсу. Незважаючи на різні підходи, усі вони мають спільну мету – досягнення високої якості.

Відповідь на друге питання дослідження, яке стосується визначення конкретних аспектів гнучкої ітераційної культури, які можна покращити завдяки заходам із забезпечення якості, полягає у перегляді ролі та завдань фахівців із забезпечення якості в команді Agile розробників. Це рішення може допомогти вирішити численні проблеми. Наприклад, якщо фахівець із забезпечення якості також займається збором історій користувачів, він зможе ефективніше взаємодіяти з клієнтом та надавати кращі рекомендації при зборі вимог для підтримки якості продукту.

Оскільки через обмеження часу розробники часто тестують власний код, ми також пропонуємо включити до команди фахівця з контролю якості для виконання цих завдань. Це рішення базується на ряді факторів. Розробка не повинна розглядатися лише як філософія; не всі організації мають можливість зібрати команду з найкращих фахівців. Тому важливо залучати кваліфікованих експертів, які можуть відігравати свою роль у впровадженні культури SPI в організації.

Розгляд третього запитання дослідження, яке стосується практичного застосування процесів SQA в Agile-проектах, показав, що організації використовують різні методології, такі як плановий та змішаний підходи. Проте вони також відчувають вплив і привабливість гнучких методологій завдяки гнучкості Agile-підходів. Основною проблемою для організацій є пошук ресурсів. В Agile-підходах діяльність із забезпечення якості зазвичай вимагає менше ресурсів порівняно з іншими методами.

Документація є критично важливим аспектом розробки програмного забезпечення. Тому фахівці-практики зацікавлені в підтриманні якості як продукту, так і процесу. Гнучкі методи SQA надають їм можливість забезпечити високу якість у найефективніший спосіб.

Таким чином, тестування в умовах Agile методологій є складним, але надзвичайно важливим компонентом успішної розробки програмного забезпечення. Воно вимагає не лише впровадження автоматизації, але й тісної співпраці між розробниками, тестувальниками та іншими учасниками команди. Тестування стає безперервним процесом, інтегрованим на всіх етапах розробки, що дозволяє швидко виявляти помилки, виправляти їх на ранніх стадіях і підтримувати стабільність продукту навіть при частих релізах. Перспективи розвитку Agile тестування відкривають нові можливості для вдосконалення процесів завдяки інструментам автоматизації, DevOps практикам і системам безперервної інтеграції. Успішне впровадження цих підходів дозволить командам не тільки підвищити ефективність роботи, але й забезпечити високу якість кінцевого продукту, який відповідатиме вимогам замовників та користувачів. Таким чином, Agile тестування стає не лише інструментом забезпечення якості, але й ключовим фактором конкурентної переваги, що дозволяє компаніям залишатися гнучкими та адаптивними до змін у сучасному динамічному середовищі розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Software Assurance and Software Safety. [Електронний ресурс]. – Режим доступу: URL: https://sma.nasa.gov/sma-disciplines/software-assurance-and-software-safety, вільний. – Загол. з екрану.
2. Manifesto, Agile. "Manifesto for agile software development." (2001).
3. Kharchenko, A., Raichev, I., Bodnarchuk, I., & Matsiuk, O. (2021, October). The Survey of Global Software Design Processes. In 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T) (pp. 291-294). IEEE.
4. Mnkandla, Ernest, and Barry Dwolatzky. "Defining agile software quality assurance." 2006 International Conference on Software Engineering Advances (ICSEA'06). IEEE, 2006.
5. Saleh, F. Malik. "Software Quality Framework." Journal of Computer Science and Engineering 13.2 (2012): 1-6 p.
6. Bergman, Bo, and Bengt Klefsjö. Quality from customer needs to customer satisfaction. Studentlitteratur AB, 2010.
7. McCall, Jim A., Paul K. Richards, and Gene F. Walters. "Factors in software quality. volume i. concepts and definitions of software quality." General Electric CO Sunnyvale CA (1977).
8. Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., MacLeod, G., J. & Merit, M. J., Characteristics of Software Quality, TRW Series of Software Technology 1, North-Holland, 1978.
9. Stamelos, Ioannis G., and Panagiotis Sfetsos, eds. Agile software development quality assurance. Igi Global, 2007.
10. Cockburn, Alistair. Agile software development: the cooperative game. Pearson Education, 2006.
11. Ullah, Malik Imran, and Waqar Ali Zaidi. "Quality Assurance Activities in Agile: Philosophy to Practice." (2009).

12. Dybå, Tore, and Torgeir Dingsøyr. "Empirical studies of agile software development: A systematic review." Information and software technology 50.9-10 (2008): 833-859 p.
13. Malik, Ahsan Nawaz, and Kashif Masood. "Software testing process in agile development." (2008).
14. Suttler, J. "Agile Testing in Software Development." New York: Tech Press (2021).
15. Henderson, E. "Scrum Testing Strategies for Agile Teams." London: Agile Publishing (2018).
16. Grain, P., Smith, R., & Taylor, M. "The Integration of Testing in Agile Development: *A Case Study*. " Journal of Software Quality Assurance (2019), 15(3), 47-63 p.
17. Jones, M. "Automated Testing and Continuous Integration in Agile Environments." International Journal of Software Testing (2020), 8(2), 101-119 p.
18. Fowler, M. "Continuous Integration and Continuous Testing in Agile Software Development." New York: Agile Books (2015).
19. Crispin, L., & Gregory, J. "Agile Testing: A Practical Guide for Testers and Agile Teams." Addison-Wesley Professional (2017).
20. Müller, N. "Challenges in Agile Software Testing: A Comprehensive Review." Software Testing Journal (2020), 6(4), 32-45 p.
21. Klopp, K. "Overcoming Documentation Challenges in Agile Testing." Journal of Modern Software Development (2019), 11(2), 22-39 p.
22. Романенко А. "Адаптація процесів тестування в умовах Agile розробки." Журнал інформаційних технологій України (2020), 12(3), 56-63 с.
23. Мартинюк О., & Кравченко, О. "Інтеграція тестування в Agile процеси в українських IT-компаніях." Вісник комп'ютерних наук та технологій (2019), 15(1), 102-109 с.

24. Семенчук Ф. "Автоматизація тестування в умовах гнучкої розробки програмного забезпечення." Український журнал IT-технологій (2021), 8(2), 87-93 с.
25. Задорожна І. "Роль комунікації між тестувальниками та розробниками в Agile командах. " Вісник програмної інженерії (2020), 6(4), 44-52 с.
26. Довженко П. "Проблеми тестування в Agile процесах в Україні. " Журнал інформаційних систем (2019), 14(2), 33-40 с.
27. Левченко Д. "Тестування без документації: реалії Agile в Україні. " Вісник сучасних технологій (2021), 9(1), 23-29 с.
28. Erickson, John, Kalle Lyytinen, and Keng Siau. "Agile modeling, agile software development, and extreme programming: the state of research." Journal of Database Management (JDM) 16.4 (2005): 88-100 p.
29. Williams, Laurie, and Alistair Cockburn. "Agile software development: It's about feedback and change." Computer 36.6 (2003): 39-43 p.
30. Lindstrom, Lowell, and Ron Jeffries. "Extreme programming and agile software development methodologies." IS management handbook. Auerbach Publications, 2003. 531-550 p.
31. Beck, Kent. "Embracing change with extreme programming." Computer 32.10 (1999): 70-77 p.
32. Jeffries, Ron. "What is extreme programming." XP magazine 11 (2001).
33. Schwaber, Ken, and Mike Beedle. Agile software development with Scrum. Prentice Hall PTR, 2001.
34. Rosenberg, Doug, Matt Stephens, and Mark Collins-Cope. "Agile development with ICONIX process." New York: Editorial Apress (2005).
35. The 12th Annual State of Agile Report. [Електронний ресурс]. – Режим доступу:URL: https://www.qagile.pl/wp-content/uploads/2018/04/versionone-12th-annual-state-of-agile-report.pdf. вільний. – Загол. з екрану.

36. Vijayasarathy, L. E. O. R., and Dan Turk. "Agile software development: A survey of early adopters." Journal of Information Technology Management 19.2 (2008): 1-8 p.
37. Mahanti, Aniket. "Challenges in enterprise adoption of agile methods-A survey." Journal of Computing and Information technology 14.3 (2006): 197-206 p.
38. Fowler, Martin. "The new methodology." Wuhan University Journal of Natural Sciences 6 (2001): 12-24 p.
39. Boehm, Barry, and Richard Turner. "Using risk to balance agile and plan-driven methods." Computer 36.6 (2003): 57-66 p.
40. Talby, David, et al. "Agile software testing in a large-scale project." IEEE software 23.4 (2006): 30-37 p.
41. Parnas, David L., and Mark Lawford. "Inspection's role in software quality assurance." IEEE Software 20.4 (2003): 16 p.
42. Nerur, Sridhar, RadhaKanta Mahapatra, and George Mangalaraj. "Challenges of migrating to agile methodologies." Communications of the ACM 48.5 (2005): 72-78 p.
43. McBreen, P., and M. Consulting. "Quality assurance and testing in agile projects." McBreen Consulting (2003).
44. Beck, Kent. "Embracing change with extreme programming." Computer 32.10 (1999): 70-77 p.
45. Cockburn, Alistair, and Laurie Williams. "The costs and benefits of pair programming." Extreme programming examined 8 (2000): 223-247 p.
46. Fowler, Martin, and Kent Beck. "Refactoring: Improving the design of existing code." 11th European Conference. Jyväskylä, Finland. 1997.
47. Capiluppi, Andrea, et al. "An empirical study of the evolution of an agile-developed software system." 29th International Conference on Software Engineering (ICSE'07). IEEE, 2007.
48. Glazer, Hillel, et al. "CMMI or agile: Why not embrace both! Software Engineering Institute." Software Engineering Institute (2008).

49. Aaen, Ivan, Anna Börjesson, and Lars Mathiassen. "SPI agility: How to navigate improvement projects." Software Process: Improvement and Practice 12.3 (2007): 267-281 p.