Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Особливості реалізації процесів забезпечення якості в гнучких технологіях розробки програмного забезпечення

Виконав(ла): студент(ка) 6 курсу, групи САм-61
спеціальності 124 Системний аналіз
(шифр і назва спеціальності)

(підпис) Семенюк В.В. (прізвище та ініціали)

Керівник (підпис) Марценюк В.П. (прізвище та ініціали)

Нормоконтроль (підпис) Дуда О.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) Загородна Н.В. (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«27» грудня 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

студенту Семенюк Володимирович Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи **Особливості реалізації процесів забезпечення якості в гнучких технологіях розробки програмного забезпечення**

Керівник роботи **д.т.н., проф. Марценюк В.П.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*24*» *листопада* 2023 року № *4/7-1096*

2. Термін подання студентом завершеної роботи 27 грудня 2023 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)

1.2 Гнучка розробка програмного забезпечення 1.3 Формулювання задач дослідження 2 МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ 2.1 Аналіз літературних джерел для проведення дослідження 2.2 Якісний підхід до вирішення поставлених задач 2.3 Загальна характеристика гнучких процесів розробки 2.4 Нові підходи до розробки 2.5 Розуміння якості в контексті Agile-розробки 2.6 Забезпечення якості продукту через якість процесів 3 ПРОПОНОВАНЕ РІШЕННЯ З ВПРОВАДЖЕННЯ SQA В AGILE-МЕТОДОЛОГІЇ РОЗРОБКИ 3.1 Agile-технології тестування 3.2 Парне програмування 3.3 Рефакторинг 3.4 SPI та гнучка методологія 3.5 Пропоноване рішення 3.6 Застосування додаткового рівня контролю якості 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ ВИСНОВКИ СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ДОДАТКИ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С, к.т.н., доц.		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викл.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	14.11.23-15.11.23	*Виконано*
2.	Підбір наукових джерел по темі роботи	16.11.23-20.11.23	*Виконано*
3.	Переклад та опрацювання наукових джерел	20.11.23-23.11.23	*Виконано*
	по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо теми	24.11.23-10.12.23	*Виконано*
	Кваліфікаційної роботи		
5.	Оформлення першого розділу	11.12.23-12.10.23	*Виконано*
6.	Оформлення другого розділу	12.12.23-13.12.23	*Виконано*
7.	Оформлення третього розділу	13.12.23-14.12.23	*Виконано*
8.	Виконання завдання до підрозділу «Охорона праці»	08.12.23-09.11.23	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	10.12.23-12.12.23	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	12.12.23-31.12.23	*Виконано*
11.	Нормоконтроль	14.12.23-15.12.23	*Виконано*
12.	Перевірка на плагіат	15.12.23	*Виконано*
13.	Попередній захист кваліфікаційної роботи	16.12.23	*Виконано*
14.	Захист кваліфікаційної роботи	26.12.2023	

Студент _____ Семенюк В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марценюк В.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Особливості реалізації процесів забезпечення якості в гнучких технологіях розробки програмного забезпечення" // Кваліфікаційна робота освітнього рівня «Магістр» // Семенюк Володимир Володимирович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група САм-61 // Тернопіль, 2023 // с. – 67, рис. – 7, табл. – 2, джерел – 37.

Ключові слова: Agile-розробка, забезпечення якості, QA, вдосконалення програмного процесу, SPI)

Діяльність забезпечення якості (QA) у розробці програмного забезпечення є невід'ємною складовою проекту. Це не лише відповідальність за якість готового продукту, але й за якість усього процесу його створення. У звичайному варіанті розробки ПЗ, QA зазвичай виконується експертною групою з якості. Однак у гнучкій розробці більшість цих завдань виконується самими розробниками. Співпраця між усіма учасниками процесу розробки, орієнтована на тестування, виступає як один із підходів у гнучкій розробці для досягнення високої якості продукту.

Ця кваліфікаційна робота ставить перед собою завдання представити принципи забезпечення якості в гнучкій розробці. Хоча уявлення про гнучку розробку сфокусоване на якості продукту, проте важливо активно працювати над забезпеченням якості процесу гнучкої розробки, зробивши його стандартизованим і краще організованим. Діяльність забезпечення якості залишається акцентованою на тестуванні. Проте в контексті практик, таких як SPI (Software Process Improvement) та дотримання певних стандартів, бракує гнучких методологій.

У цьому дослідженні ми розглядаємо можливість додаткового контролю якості у гнучких проектах. Мета введення додаткового рівня контролю полягає в залученні експертів забезпечення якості для підвищення якості процесу розробки, що в свою чергу призведе до підвищення якості виробленого продукту.

ANNOTATION

“Peculiarities of quality assurance processes in Agile software development technologies” // Master’s degree qualification paper // Semeniuk Volodymyr Volodymyrovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group САм-61 // Ternopil, 2023 // p. – 67, fig. – 7, tables – 2, references – 37.

Key words: Agile development, quality assurance, QA, software process improvement, SPI

The activity of Quality Assurance (QA) in software development stands as the foundation of any project. This function is responsible not only for the quality of the product but also for the quality of the development process itself. In traditional software development, QA is carried out by a team of quality experts. However, with the shift towards agile development, the quality assurance activity has also evolved.

In Agile development, most of these tasks are performed by developers themselves. Close collaboration among stakeholders, testing-oriented development are approaches in agile development aimed at achieving superior product quality. This research paper presents the principles of ensuring quality in agile development. While the concept of agile development focuses on product quality, there is a need to put significant effort into ensuring the quality of the agile development process, to standardize and organize it better. Quality assurance activities remain focused on testing. In practices like Software Process Improvement (SPI) and adherence to certain standards, there's a lack of agile methodologies.

In this study, we propose adding an additional layer of quality control in agile projects. The purpose of introducing this added layer is to leverage the expertise of quality assurance professionals to enhance the development process's quality, ultimately leading to a higher level of product quality.

ЗМІСТ

ВСТУП

Актуальність задачі.

Забезпечення якості програмного забезпечення (Software Quality Assurance – SQA) має високу важливість у сучасній індустрії програмного забезпечення. Ми можемо знайти достатню дослідницьку роботу, проведену в цій галузі. SQA тлумачиться різними способами та словами. Центр забезпечення якості програмного забезпечення NASA описує SQA так: «Software Assurance is defined as the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in an intended manner» [1]. Тобто це гарантування достатнього рівня довіри до продукту за рахунок планового і систематичного підходу до оцінки якості та дотримання стандартів, процесів і процедур програмного продукту.

Кінцевою метою процесів забезпечення якості є досягнення кращої якості програмного продукту. У цій дисципліні дотримуються різних підходів і кількох моделей якості. Діяльність SQA практикується під час проекту, і ця діяльність включає контроль процесів, документацію, аудити та верифікацію та валідацію.

Мета роботи.

Основна мета цієї роботи полягає в тому, щоб висвітлити деякі прогалини в гнучкій діяльності для реалізації SQA та висунути пропозиції щодо покращення.

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

Предмет дослідження: якість програмного продукту.

Наукова новизна отриманих результатів. Запропоновано додатковий рівень контролю якості в Agile-проєктах. В гнучкій розробці розробники також можуть відповідати за діяльність з контролю якості [3]. Гнучкі проекти розробки складаються з короткої ітераційної розробки та випуску продукту. А проекти, які дотримуються гнучких принципів розробки, розвиваються навколо розробника

та клієнта, які відповідають за підтримку якості продукту [4]. Якщо відповідальність за якість у гнучкій розробці перекладається на замовника та розробника, тоді необхідно визначити допоміжну роль QA. SQA відповідає не тільки за конкретний проект, але й підтримує процеси та культуру компанії.

Практичне значення отриманих результатів. Дан робота дозволяє модифікувати процес управління проєктами з розробки ПЗ та інтегрувати SPI-процеси d Agile-розробку.

Апробація результатів та особистий внесок здобувача. Основні положення роботи доповідались, розглядались та обговорювались на науковій конференції Тернопільського національного технічного університету імені Івана Пулюя. Результати кваліфікаційної роботи опубліковані у тезах студентської наукової конференції "Інформаційні моделі системи та технології – 2023", та "Актуальні задачі сучасних технологій – 2023", які проводились у ТНТУ.

1 ЗАДАЧІ ІНЖЕНЕРІЇ ВИМОГ В AGILE-ПРОЄКТАХ

Кінцевою метою виробництва послуг або продуктів є задоволення споживача. Якість продукту чи послуги може бути безпосередньо пов’язана із задоволеністю клієнта. Щоб досягти необхідного рівня якості, необхідно встановити певні стандарти для продукту чи послуги. Щоб відповідати визначеним і необхідним стандартам, слід дотримуватись певних процедур і процесів.

Окрім досягнення задоволеності клієнтів, організації також націлені на свою конкурентну перевагу в галузі. Щоб отримати добру репутацію та прибутковість у галузі, компанія потребує постійного вдосконалення якості. Управління якістю (Quality Management – QM) є важливим і життєво необхідним для організації. Управління якістю – це процес забезпечення того, щоб необхідний рівень якості досягався не лише в продуктах, але й у процесі виробництва цих продуктів. Це передбачає визначення відповідних стандартів якості та процедур, щоб гарантувати їх дотримання. Метою має бути розвиток «культури якості», де досягнення якості розглядається як відповідальність кожного. Забезпечення якості, планування якості та контроль якості є видами діяльності, які включені в управління якістю [5].

У індустрії програмного забезпечення концепція якості виражається та визначається багатьма різними способами, деякі визначення зосереджені на безпомилковій функціональності програмного продукту, тоді як деякі визначення наголошуються на задоволенні споживача, але насправді важко навести абсолютне визначення якості навіть після перегляду літератури. Якщо ми концептуалізуємо якість, розуміючи термін «корисність» з економіки, може бути легше розвинути значення та чіткі судження про якість. Оскільки корисність вважається станом або мірою задоволення, яке хтось досягає від використання товарів, таким же чином якість програмного забезпечення також може бути концептуалізовано, як стан задоволення в певному середовищі або способі використання. Є багато атрибутів, які можна охопити, щоб

сформулювати повне визначення якості; тому пошук причини відсутності якості може привести до кращого розуміння, оскільки в [6] ми знаходимо твердження, що нестача якості – це втрати, які продукт завдає суспільству з моменту його відправлення замовнику. Якщо в цьому визначенні ми поставимо перед словом «програмне забезпечення» слово «продукт» і замінимо «відвантаження» на «доставити» для нашого розуміння, це дасть нам ширше та конкретніше бачення якості щодо розробки програмного забезпечення та продукту.

Для постійного вдосконалення необхідна постійна оцінка та моніторинг. Оцінка продукту та моніторинг процесу є основними обов'язками забезпечення якості програмного забезпечення (SQA) під час розробки програмного забезпечення. SQA також контролює та оцінює якість:

- стандартів;
- процесів;
- процедур.

Для розробки програмного забезпечення можна застосувати звичайну та гнучку методології. Обидві методики відрізняються за підходом. Відповідно, вони по-різному впроваджують процеси SQA.

1.1 SQA при традиційній розробці програмного забезпечення

У розробці програмного забезпечення під традиційними вважають методології, орієнтовані на процес. В традиційній розробці структура процесу розробки програмного забезпечення або життєвого циклу строго дотримується. Практикуються заздалегідь визначені дії, які включають [1]:

- Збір вимог і аналіз вимог.
- Специфікація.
- Архітектура.
- Проектування..
- Розробка.
- Тестування та впровадження.

– Технічне обслуговування.

Існує багато моделей розробки програмного забезпечення, орієнтованих на процес. Кожна модель визначає власні кроки, процеси, процедури та дії, яких слід дотримуватися протягом життєвого циклу розробки програмного забезпечення. Говорячи про звичайну розробку ПЗ, модель водоспаду та модель спіралі можна назвати більш зрілими та практичними. У звичайній розробці програмного забезпечення замовник взаємодіє з інженерами вимог, щоб визначити ці вимоги. Політики та стандарти рішень визначаються на управлінському рівні організації.

Діяльність із забезпечення якості працює як об’єкти моніторингу та оцінки за будь-якої моделі розробки програмного забезпечення. Ця діяльність також зосереджена на якості продукту/послуги та процесу. Крім усіх філософій якості, існує кілька моделей якості. Ці моделі якості надають нам якісні характеристики, атрибути та фактори, на які слід орієнтуватися під час розробки програмного забезпечення. Модель якості Макколла, представлена в 1977 році [7], і модель якості Боема, представлена в 1978 році [8], ці моделі якості стали основою для сучасних моделей якості. Хоча визначення чи підрахунок характеристик якості здається нескінченною петлею, але є деякі загальні атрибути та характеристик якості:

– Коректність.
– Надійність.
– Цілісність.
– Юзабіліті.
– Ефективність.
– Ремонтопридатність.
– Сумісність.
– Гнучкість.
– Багаторазове використання.
– Портативність.

- Модифікованість.
- Документація.
- Стійкість.
- Зрозумілість.
- Термін дії..
- Функціональність.
- Економіка.

Оскільки індустрія програмного забезпечення розвивається, програмні продукти стають складнішими, а вимоги користувачів зростають. Ці фактори, зрештою, збільшують складність проектів розробки програмного забезпечення. Для усунення складнощів складання програмного продукту дотримуються кількох моделей і стандартів. Деякі з найбільш практично використовуваних моделей:

- ISO 9000.
- ISO 9126.
- CMM (модель зрілості можливостей)..
- CMMI (Інтеграція моделі зрілості можливостей)
- SPICE (визначення можливостей удосконалення процесу програмного забезпечення) .
- Шість сигм.

Окрім прицілу на якість продукту/послуги, ці моделі також зосереджені на якості процесу розробки програмного забезпечення на рівні організації. Але з часом обов'язки QA збільшуються. Діяльність із забезпечення якості набула значення хребта організації. Ці дії відповідають за процес розробки, якість продукту, і ці дії забезпечують підтримку проекту. На управлінському рівні встановлюються стандарти та процедури розробки програмного забезпечення. Роль SQA полягає в тому, щоб переконатися, що визначені стандарти належним чином задокументовані, а процедури дотримані. З цією метою проводиться

оцінка продукту, аудити та зустрічі, щоб контролювати та оцінювати, чи процеси відповідають визначеним процедурам.

Що стосується розробки продукту, то виключно верифікація та валідація є основною відповідальністю SQA та діяльністю з підтримки та оцінки якості продукту. SQA проводить офіційну перевірку програмного забезпечення, інспекцію програмного забезпечення та перевірку документації для моніторингу розробки програмного забезпечення. SQA гарантує, що якість програмного забезпечення відповідає необхідним стандартам якості та вимогам. Забезпечення якості є життєво важливим на всіх етапах процесу розробки програмного продукту для досягнення вищого та необхідного рівня якості як продукту, так і процесу. Для гарантії використовуються численні інструменти. Ці інструменти включають аудити, контрольні списки перевірок, показники та автоматизовані стандартні аналізатори коду.

Діяльність SQA набула надзвичайного значення. Уявити успішний проект без залучення SQA практично неможливо. Ефективне забезпечення якості – запорука успішного проекту. Крім того, всі ці факти діяльність SQA стає все більш складною. Хоча ця діяльність веде організацію до вищого рівня якості, але ця діяльність споживає достатньо багато ресурсів, часу та зусиль. Важка документація, покрокова оцінка та моніторинг – це суворий процес.

1.2 Гнучка розробка програмного забезпечення

Гнучкий підхід до розробки програмного забезпечення повністю змінив спосіб розробки. На відміну від традиційного підходу до розробки, Agile віддає перевагу коротким ітераціям у процесі розробки програмного забезпечення. Цей підхід є коротким ітераційним, поступовим і орієнтованим на людей. Визначаючи гнучку розробку, відправною точкою має бути маніфест гнучкої розробки програмного забезпечення [2]. В ньому зазначається що цінністю є:

- Члени команди та взаємодія процесів та інструментів.
- Працююче програмне забезпечення а не повнота документації.

– Співпраця з клієнтом над узгодженням контракту.

– Реагування на зміни цінніше за дотриманням плану.

Гнучка розробка програмного забезпечення заохочує людей до співпраці в рамках проекту. У порівнянні зі звичайним способом розробки програмного забезпечення, він ефективно реагує на зміни, оскільки є поступовим та ітеративним. На рисунку 1.1 представлено одну ітерацію гнучкого підходу до розробки.

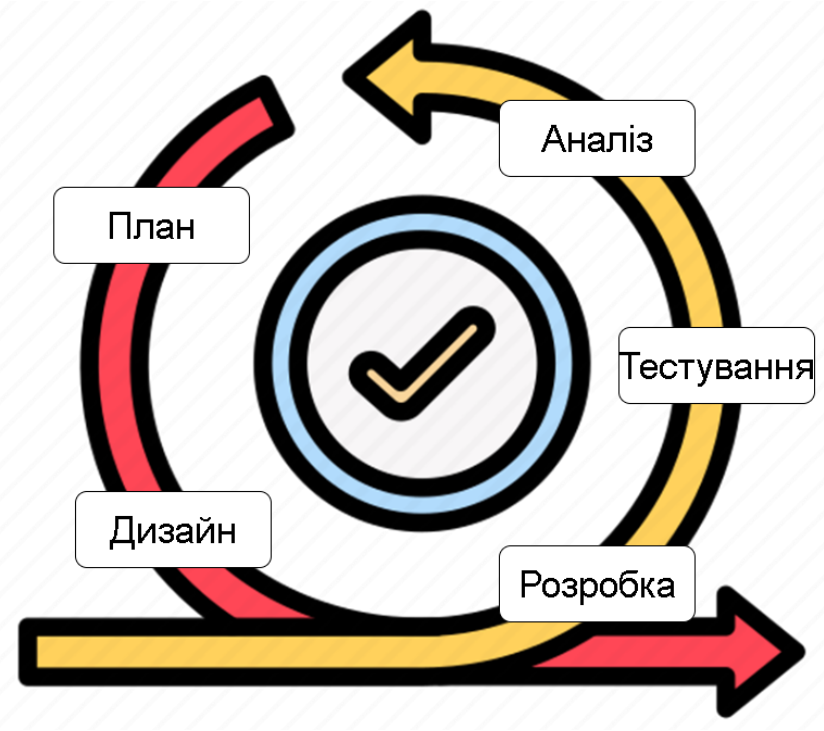


Рисунок 1.1 – Одна ітерація процесу Agile розробки

Як ми бачимо, в одній короткій ітерації всі необхідні фази розробки програмного забезпечення (планування, проектування, розробка, тестування та аналіз) відбуваються одна за одною. Після завершення однієї ітерації програмний продукт надсилається на перевірку користувачам/замовникам. Наступна ітерація починає додавати необхідні функції відповідно до відгуків користувачів/клієнтів.

Існує багато гнучких методологій, але найчастіше використовуються гнучкі методи:

– Екстремальне програмування (XP).

– Scrum.

У природі всі гнучкі методології мають схожість. На рисунку 1.2 показано характер гнучких методологій, взятих із [9].

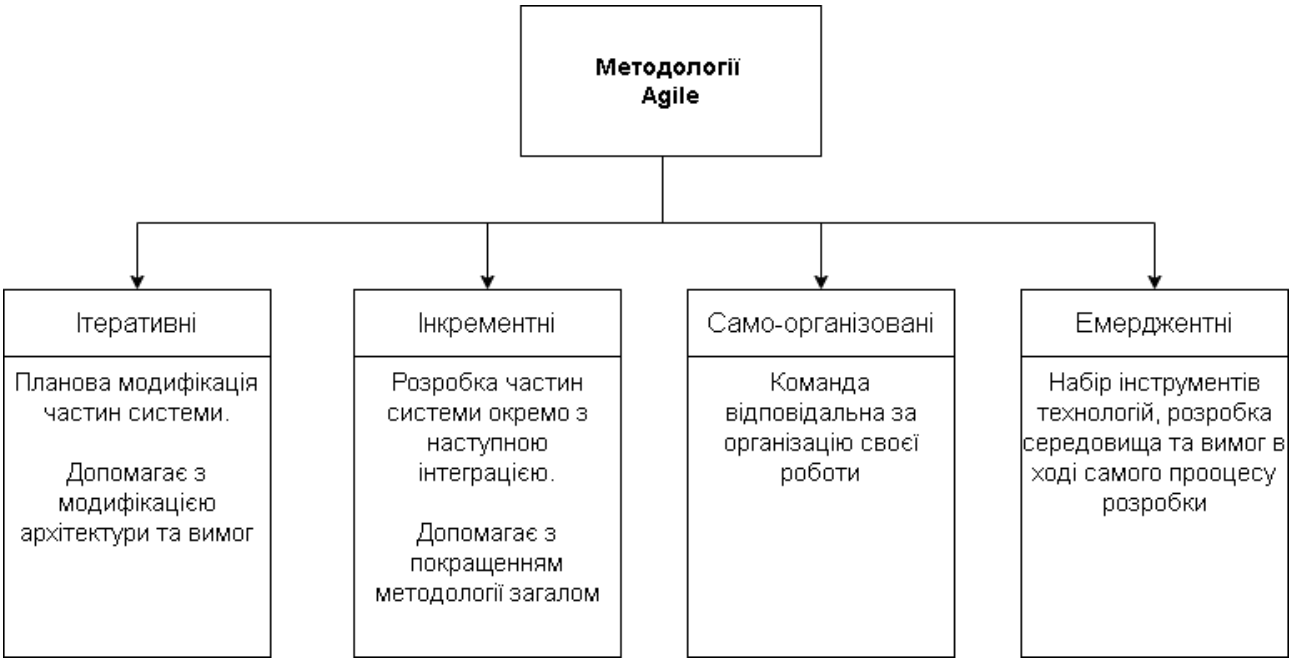


Рисунок 1.2 – Природа Agile методологій

Гнучкі методології відомі як легкі, оскільки вони не потребують інтенсивних процесів. Гнучкі методології також змінили спосіб діяльності SQA. Документація не надто масштабна, але складається лише відповідно до вимог замовника/користувача.

Тестування інтегровано в кожну ітерацію, що допомагає скоротити час релізу та забезпечує кращу якість. Фокусом нашого дослідження залишатиметься тема діяльності SAQ у гнучких методологіях; тому всі аспекти, пов’язані з цією темою, будуть розглянуті глибше в наступних розділах. Ми розглянемо різницю в поглядах теорій і практиків.

Менталітет змінився, коли гнучкі методології широко розповсюдились галузі розробки програмного забезпечення. Спостерігаються різкі зміни в діяльності з контролю якості. Agile переніс розробку з процесу, орієнтованого на людей. Тестування не є обов’язком спеціаліста із забезпечення якості,

розробники повинні тестувати свої програми та виправляти помилку щоразу, коли її знаходять. На практиці немає більш суворого та тривалого етапу тестування. Помилки виправляються на тому ж етапі, на якому вони знайдені, і незалежно від того, хто їх знаходить і виправляє. Тісна співпраця з клієнтами поєднується, щоб отримати ефективний і швидкий відгук про розроблену систему перед початком нової ітерації, щоб підвищити якість продукту.

Оскільки гнучкі методології цінуються за інтегрований підхід до тестування, швидку реакцію на зміни та орієнтацію на людей, деякі заяви також спонукали до того, що ці методології недостатньо зрілі, щоб застосовувати їх у великих масштабах. Ми детально обговоримо ці питання та спробуємо висунути пропозиції щодо покращення деяких прогалин у діяльності SAQ в Agile.

1.3 Формулювання задач дослідження

Якість програмного продукту завжди була основною ціллю компанії чи проекту, і бездоганний продукт можна розглядати як найкращу характеристику якості програмного продукту. Але щоб розробити якісний програмний продукт, його потрібно розробити якісно. Будь-яка програмна методологія дотримується, щоб правильно розробити програмний продукт. Діяльність із забезпечення якості керує органами влади в організації. Світ програмної інженерії змінився, перейшовши від процесно-орієнтованої інженерії до гнучкої інженерії. Гнучка розробка має за мету швидко реагувати на зміни.. Окрім обговорення того, як досягається якість у гнучкій розробці, ми також висвітлимо та спробуємо заповнити деякі прогалини в цій сфері інтересів.

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 спосіб виконання програмних проєктів. Але цю методологію не можна вважати зрілою та практичною, як традиційна розробка. Діяльність SQA контролює всі процеси в традиційній розробці.

Завдяки більшому залученню клієнтів і зацікавлених сторін гнучка розробка запровадила новий підхід до задоволення вимог замовника. Гнучкість і швидка реакція на зміни також роблять цю методологію більш ефективною для розробки. Незважаючи на всю свою ефективність, ця методологія зазнає критики та стикається з проблемами. Ми вказали на деякі проблеми, з якими стикається гнучка розробка в QA, і в цьому розділі ми обговоримо саме їх.

Залучення клієнта на місці та співпраця з розробником є хорошим підходом до розробки системи за потреби. Відповідно до [11], доступність клієнта на місці робить оцінку системи ефективною, а клієнт надає корисний зворотний зв’язок, коли робота ще свіжа в пам’яті кожного. У цьому сценарії ігнорується, що, коли клієнт технічно не обізнаний про параметри якості та атрибути системи, то це може призвести до неправильного розуміння системи. Більше того , якщо клієнт недоступний весь час на місці, тоді повинен бути хтось, хто представлятиме потреби клієнта, щоб підтримувати якість і гнучкість протягом усього проекту. Швидкий зворотний зв’язок є одним із важливих елементів гнучкості, і клієнт повинен надавати відгук після кожної ітерації.

Гнучка розробка програмного забезпечення запропонувала новий підхід, щоб позбутися великого обсягу документації, мінімізувавши її до найменшого рівня. Гнучкість не заохочує до великої кількості документації, але кілька практиків висунули свою критику щодо того, що документацією не можна нехтувати. Гнучка розробка мінімізує технічну документацію, але документація є частиною розробки програмного продукту [11]. Документація потрібна для

підтримки якості програмного забезпечення в майбутньому. За документацію протягом усього проекту відповідає QA. Але якщо гнучкість не заохочує до документування, а з іншого боку, документація вважається важливою, тоді в гнучкості необхідно перевизначити роль QA.

Тестування – це діяльність із забезпечення якості. У традиційному підході до розробки тестування проводиться як окремий і повний етап. Зараз на практиці застосовуються як автоматизовані, так і ручні методи тестування. Гнучка розробка змінила весь сценарій тестування програмного забезпечення. Agile переклала відповідальність за тестування на розробників, застосувавши такі методи, як парне програмування та тестове програмування. Але чи означає це, що ми скоротили потребу в QA-тестерах, і програмісти також отримали можливість брати на себе відповідальність тестувальників? Відповідно до [12] програмісти не можуть мати справу з усіма видами дефектів під час розробки програмного продукту. Щоб подолати цю проблему, було в [13] запропоновано окрему команду тестування для гнучкої розробки. Наша мета полягає в тому, щоб тримати під увагою цілісне забезпечення якості, а не тестування.

У міру того, як програмні продукти стають складнішими, а великі ручні системи потребують автоматизації, проблеми з якістю та тестуванням стають суворішими. Хоча предметом цієї роботи більшою мірою є гнучкий підхід, також фактом є те, що численні організації отримують переваги від формальних підходів, і гнучкі підходи у них не є широко поширеними. Оскільки якість програмного забезпечення не залежить від тестування програмного забезпечення, тестування програмного забезпечення також не залежить від інструментів тестування.

Хоча є багато тверджень, що гнучка розробка забезпечує нам вищу якість, але шляхом порівняння наукових праць із роботами практиків щодо гнучкої діяльності SQA ми запропонуємо покращення для усунення прогалин у діяльності з забезпечення якості в гнучких проектах. Будуть проаналізовані практики реальних проєктів, щоб зрозуміти, як організації практикують діяльність SQA в проектах гнучкої розробки, а думки респондентів допоможуть

зрозуміти обмеження та переваги гнучкості в SQA, що необхідно для висунення пропозицій для покращення.

2МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

Перш ніж розпочинати будь-яку дослідницьку роботу необхідно визначити, яка методологія, підхід чи план дій будуть дотримуватись, що пов’язує методи з результатами? Відповідно до оксфордського визначення слово «методологія» можна визначити як «систему методів, що використовуються в певній галузі». Необхідно прийняти будь-яку методологію для дослідження, щоб продовжувати дослідження, контролювати чи дослідження на правильному шляху, вимірювати, структурувати та організовувати процес дослідження. Найбільш часто використовувані методології або підходи:

1. Виконання якісного дослідження.
2. Виконання кількісного дослідження.
3. Змішаний підхід до дослідження.

Дослідницький підхід або методологія, яка застосовувалася в поточному дослідженні, був якісним підходом. Визначивши найбільш критичні проблеми в нашій галузі дослідження, ми мали на меті використати ці літературні висновки для розробки та пропозиції рішень для подолання критичних проблем у гнучкій діяльності SQA. Тому наше дослідження можна класифікувати як якісне.

2.1 Аналіз літературних джерел для проведення досліження

Огляд літератури було здійснено, щоб зібрати теоретичні дані різних досліджень у сфері нашого інтересу та отримати уявлення про перспективу розвитку предметної області та для кращого розуміння процесів розробки за гнучкими підходами.

Щоб зібрати потрібний перелік літературних джерел, ми використали різні ресурси та бази даних, доступні безкоштовно в Інтернеті. В основному використовуються такі ресурси:

- IEEE Explorer.
- Цифрова бібліотека ACM.

– Пошукова система Google.
– Електронні бібліотеки.

Ми зібрали інформацію, особливо зосереджену на діяльності з контролю якості та пов’язаних з нею процесах у гнучких проектах. Було проаналізовано декілька наукових статей, журналів та матеріалів. Щоб знайти відповідну літературу з досліджуваної області, ми використовували різні ключові слова у вищезазначених пошукових системах і дослідницьких базах даних. Анотація, вступ і завершення статей були переглянуті для оцінки актуальності. З відібраної літератури необхідна інформація була отримана та представлена під час написання цієї роботи.

2.2 Якісний підхід до вирішення поставлених задач

Тема дослідження та проблематика в поточній роботі пов’язані з теоріями, а не з числовими та статистичними даними. Потрібно було вивчити концепцію та роль QA в процесах гнучкої розробки, щоб виявити прогалини в сфері забезпечення якості в гнучких технологіях розробки. Зважаючи на характер дослідження, був прийнятий якісний підхід, оскільки саме у якісному дослідженні автор описує дослідницьку проблему, яку можна найкраще зрозуміти, досліджуючи концепцію чи явище.

Чітко сформульовані питання для дослідження відіграють важливу роль. Це веде та допомагає побудувати результати або теорії. Однією з основних цілей нашого дослідження було запропонувати рішення для виявлених прогалин. Для побудови нашого рішення замість математичних результатів була потрібна думка практиків, які працювали у суміжній галузі. Таким чином, наше дослідження можна класифікувати як якісне дослідження, оскільки використовуючи цей дослідницький підхід, дослідник збирає відкриті нові дані з головною метою розробки тем на основі даних.

Коли під час вивчення літератури були виявлені прогалини та критичні проблеми, здійснювалась робота над пропозицією щодо рішення задачі та

покращення. Але щоб запропонувати рішення, потрібна була думка практиків щодо діяльності SAQ. Щоб дізнатися думку практиків, було досліджено результати анкетування.

Було розповсюджено відкриту анкету. Анкета дозволяє отримати думку респондентів на основі їх досвіду. Хоча дослідження зосереджено на гнучкій діяльності SQA, але щоб зробити спостереження та запропоноване рішення обґрунтованим та ефективними, також було надіслано анкету практикам, які працюють у звичайному середовищі розробки програмного забезпечення.

Для розповсюдження опитувальника було вибрано п'ять різних організацій, які займаються розробкою програмного забезпечення різного типу.

Було досліджено проєкти, розподілені відповідно до списку нижче.

- Сховища даних і бізнес-аналітика.
- Розробка ПЗ на замовлення..
- Впровадження та модифікація пакета
- Телекомунікації.
- Система управління людськими ресурсами.
- Система управління документами.
- Фінансовий пакет.
- Інструмент електронного урядування.
- Програма Інтернет-банкінгу.
- Електронна медична карта.
- 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 [14] 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 замовника та всіх людей, залучених до проекту, залишається корисною для забезпечення якісного продукту. У [15] автори заявили, що гнучкі методології створені для того, щоб «охоплювати, а не відкидати, вищі темпи змін». У гнучкій розробці процеси поділяються на короткі працездатні ітерації. Коли відбувається нова ітерація, змінені вимоги об’єднуються в систему відповідно до загальних вимог. Таким чином, цей підхід, також робить процес розробки програмного забезпечення гнучким.

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

У таблиці 2.1 гнучкі та звичайні підходи розрізняються за змінними та проектним середовищем. Ми можемо помітити, що гнучкий підхід більше орієнтований на людей, оскільки він цінує участь розробника та клієнта. Крім того, він охоплює та швидко реагує на зміни на кожному рівні проекту.

Таблиця 2.1 – Порівняння Agile- та традиційних методологій розробки

Середовище проєкту		Характеристики проєкту	
Категорія	Змінна	Agile	Non-Afile
Команда розробників	Стиль комунікації	Постійна співпраця	За необхідності
	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 [16] є однією з найпопулярніших і часто використовуваних методологій гнучкої розробки програмного забезпечення. Основоположником цієї методики є Кент Бек; він представив і визначає ряд принципів і практик для підтримки продуктивності команди розробників і підвищення точності і якості створеної системи. XP – це легкий, передбачуваний, ефективний і гнучкий метод. Він був розроблений, щоб задовольнити потреби невеликої команди, яка має справу з неточними та мінливими вимогами щодо кращої розробки програмного забезпечення. XP містить набір дисциплін і практик для процесу розробки програмного забезпечення. Щоб застосувати методологію XP, є певні практики, яких необхідно дотримуватися в процесі розробки (рис. 2.1).

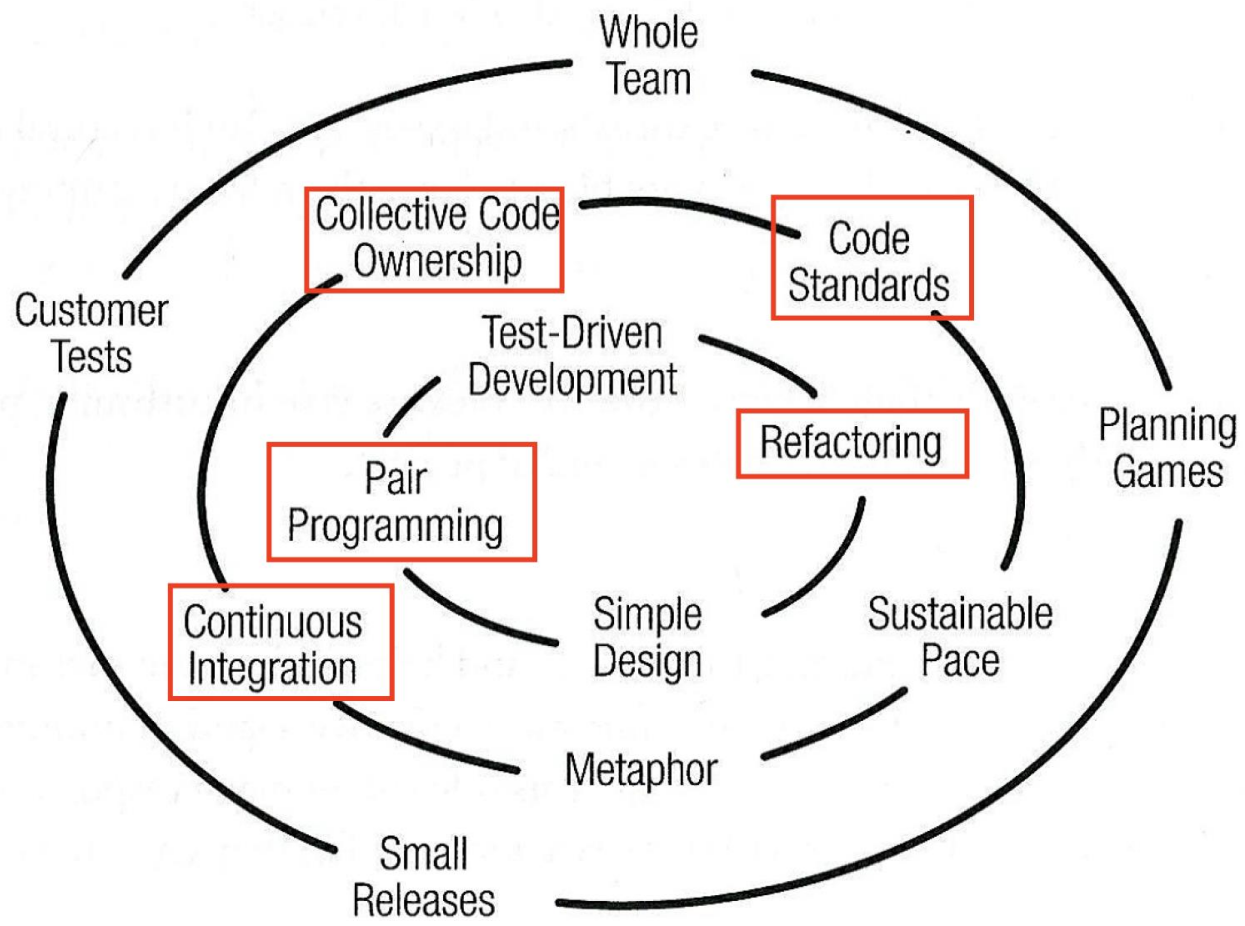


Рисунок 2.1 – Принципи екстремального програмування

Короткий виклад термінів і практики XP обговорюється відповідно до [16], [17]:

– Планування – програмісти оцінюють зусилля, необхідні для впровадження історій користувачів (юзер сторіз), а замовник вирішує час релізу та обсяг на основі оцінок.

– Метафора – робота системи визначається набором метафор між клієнтом і програмістами.

– Малі/короткі релізи – розробка додатків виконується серією невеликих версій, які часто оновлюються.

– Рефакторинг – при рефакторингу система реструктурується таким чином, щоб усунути дублювання, покращити зв’язок, спростити та додати гнучкості, але функціональність програми не повинна бути змінена.

– Простий дизайн – XP наголошує на розробці найпростішого можливого рішення, яке потрібно реалізувати, і усунути непотрібну складність і зайвий код.

– 40-годинний робочий тиждень – у XP жоден член команди не може працювати понаднормово, має бути 40-годинний робочий тиждень. Якщо робота перевищить ці терміни, то це означає, що на проєкті бути проблеми з плануванням.

– Парне програмування – на етапі розробки програмісти працюють у парах, пишуть код на одному комп'ютері.

– Стандарти кодування – у XP існують певні правила кодування та стандарти, які забезпечують послідовність і покращують спілкування між командою розробників.

– Колективна власність – жодна окрема особа не несе відповідальності за сегменти коду, кожен може змінити будь-яку частину коду в будь-який час.

– Безперервна інтеграція – код інтегрується з поточною системою, коли вона готова. Цей код має пройти всі тести після або до змін.

– Клієнт– у XP необхідна наявність клієнта з командою розробників.

У XP кожен учасник проекту має свою невід'ємну частину в команді. Команди XP формуються навколо представника бізнесу, який називається «Замовник». Зосереджуючись на бізнес-цінності, команди XP використовують просте планування та відстеження, щоб вирішити та передбачити, що слід робити далі та коли проект буде завершено. Команда створює невеликі версії програмного забезпечення, які проходять всі тести, визначені замовником [18].

Розробка, орієнтована на тестування, рефакторинг, системна метафора та парне програмування відіграють головну роль у реалзіації QA (забезпечення якості). Ці основні практики гармонізуються одна з одною. Розробка, керована тестуванням, перевіряє, що написаний код не містить помилок. Рефакторинг завжди забезпечує простоту коду, щоб уникнути складності розроблюваної системи. Системна метафора забезпечує базове розуміння архітектури системи, вона зменшує ймовірність збою системи, якщо робота з розробки виконується

відповідно до архітектури. Парне програмування є найпопулярнішою практикою XP, у якій два програмісти діляться своїми ідеями та спільно виявляють помилки, що допомагає розробити систему без помилок. Тож ми можемо сказати, що ці практики відіграють свою роль у розробці продукту кращої якості з мінімальним ризиком помилок [16].

2.3.2 Методологія Scrum

Scrum також є широко використовуваною методологією в Agile, її спочатку розробив Кен Швабер. Термін «Scrum» походить від стратегії гри в регбі, де він означає «повернення м'яча, що залишився поза грою, у гру» за допомогою командної роботи. Scrum забезпечує управління проектом із структурою, яка включає такі завдання розробки, як збір вимог, дизайн і програмування (рис. 2.2).

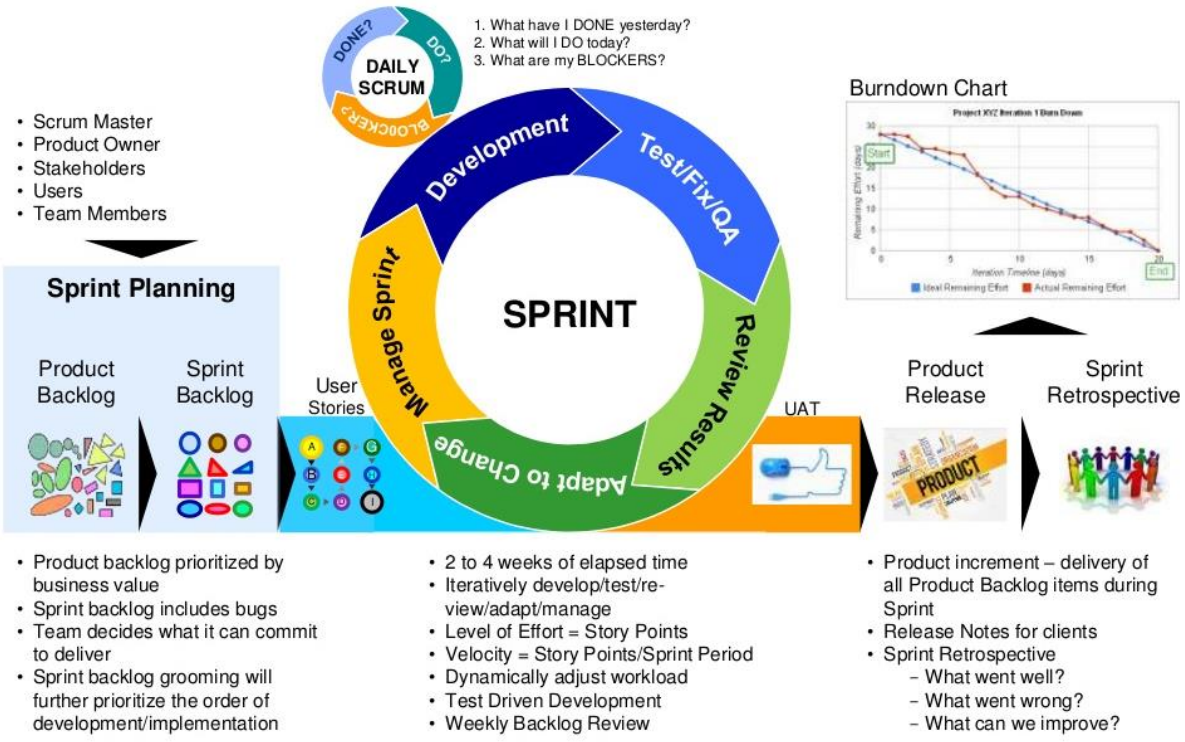


Рисунок 2.2 – Принцип методології Scrum

Він не передбачає жодного конкретного методу для застосування; він керує тим, як має функціонувати команда, щоб підтримувати гнучкість системи, у застосуванні змін навколишнього середовища. У процесі розробки існує багато

технічних і екологічних змінних, які постійно змінюються, наприклад: вимоги, час, ресурси та технологія. Через ці змінні процес розробки стає непередбачуваним і складним. Має бути процес, який може вирішити ці проблеми в системі. У Scrum часто проводяться заходи, які можуть допомогти менеджменту на проєкті досягти кращого результату [19].

Scrum містить процеси управління та розробки. Scrum передбачає швидке створення прототипів, простий перегляд системних вимог від замовника. Ці вимоги не тільки неповні, але й можуть бути змінені в процесі розробки. Основною практикою в Scrum є щоденні 15-хвилинні зустрічі для координації та інтеграції питань розробки [19]. Деякі ключові практики Scrum обговорюються нижче.

– Беклог продукту. Команда записує всі визначені на даний момент завдання в список, що називається Беклог, майже всі учасники можуть змінити його, зокрема клієнти, маркетинг і продажі, а також команда проекту. Scrum-майстер веде Scrum-зустрічі, визначає початкове відставання, яке потрібно завершити в спринті.

– Спринти. Спринти триваютьвід 10 до 30 днів. Розробникам призначається номер завдання для виконання спринту. Під час спринту заборонено жодні зміни з боку команди. У спринті основними робочими інструментами команди є зустрічі з планування спринту, беклог спринту та щоденні зустрічі Scrum.

– Зустріч з планування спринту – на ранніх зустрічах з планування спринту беруть участь клієнти, користувачі, керівництво, власник продукту та команда Scrum, на яких вони вирішують цілі та функціональність системи. Після цього Scrum Master і команда Scrum зосереджуються на розробці продукту.

– Daily Scrum – команда Scrum проводить часті зустрічі. Ці щоденні зустрічі тривають приблизно 15 хвилин, головною метою цих зустрічей є підтримання прогресу та обговорення проблем команди під час розробки. Ці групові зустрічі допомагають підняти моральний дух команди та покращити спілкування між членами команди.

Scrum – це методологія управління з деякими важливими правилами та практиками. Це не інженерний процес, який передбачає певну діяльність із забезпечення якості. Це залежить від керівництва організацій, щоб запровадити та стежити за діяльністю в процесі Scrum, щоб отримати кращу якість продукту. Переважно організації поєднують практики XP і Scrum для покращення процесу розробки. Scrum зосереджується на комунікації та зворотному зв'язку, що базується на ітеративних і поступових практиках, щоб керувати розробкою. У практиках Scrum передбачається щоденні зустрічі команди, безперервна інтеграція та приймальне тестування для гарантування потрібного рівня якості.

2.3.3 Методологія Crystal Clear

Crystal – це сімейство різних методологій, створених Алістером Кокберном. Назва «Кристал» походить від характеристики проектів за двома вимірами: розміром і критичністю. Кожна кристалічна методологія позначена відповідним кольором (жовтий, помаранчевий, червоний...), який вказує на важкість методології. Вибір відповідного кольору методології в основному залежить від розміру та критичності проекту. У великому проекті ми обираємо більш темний колір методології, тому що він вимагає більше ресурсів і координації, а не маленький [20].

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 [20].

Crystal Clear розроблено для невеликих проектів із 6-8 розробниками. Члени команди працюють у спільному офісі або в одній кімнаті, щоб підтримувати кращий зв'язок під час проекту. Crystal orange призначений для великого проекту з 10-40 членами команди і проект триває від 1 до 2 років.

Crystal завжди використовує поетапний цикл розробки; довжина кожного приросту становить від 1 до 3 тижнів. Існує багато особливостей і значень, які є

загальними для кожної методології Crystal. Його основний акцент робиться на спілкуванні та співпраці з людьми. Це не обмежує жодних інструментів чи практик, воно також дозволяє застосовувати XP і практики Scrumming для підтримки продуктивності системи [20].

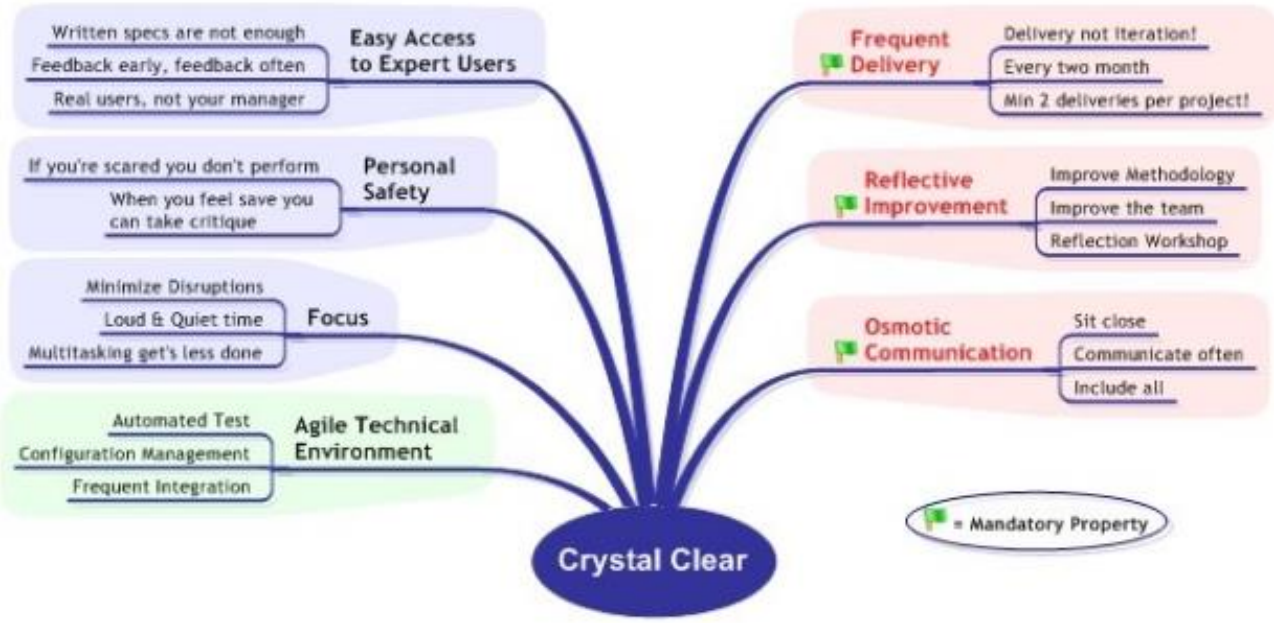


Рисунок 2.3 – Властивості методу Crystal Clear

Ключові особи, які потрібні в Crystal: старший дизайнер, дизайнер-програміст і користувач. Існують також певні практики та стандарти, яких необхідно дотримуватися або застосовувати в процесі розробки та обговорюються нижче (рис. 2.3):

– Програмне забезпечення постачатиметься поступово, протягом 2-3 місяців.

– Прогрес відстежуватиметься на проміжних етапах (mile stones) на основі виконання проектів або основних рішень, а не письмових документів.

– Crystal вимагала безпосередньої участі користувача в процесі розробки

– Повинна бути певна кількість автоматизованого регресійного тестування функціональності програми.

– Для кожного релізу має бути два перегляди користувачами.

– У середині або на початку кожного кроку повинні бути семінари для налаштування методології.

Практики Crystal є обов’язковими для дотримання, але їх можна замінити будь-якою іншою практикою, оскільки Crystal не обмежує застосування будь-якої практики для підтримки процесу розробки. Автоматизоване тестування, пряме залучення користувачів і парна робота з програмування в основному зосереджена командою розробників для досягнення та підтримки якості кінцевого продукту.

2.4 Нові підходи до розробки

У досягненні кожного завдання та у створенні або розробці кожного продукту основним фактором, на якому зосереджується уся діяльність, є точність і якість. Щоб досягти цих двох основних характеристик, ми повинні застосовувати та дотримуватися певних процедур та практик, серед яких деякі є стандартні, а деякі гнучкі. Як ми знаємо, у цьому світі немає нічого досконалого, тому кожна процедура та процес потребують удосконалення, для цього необхідне прийняття нових змін..

Відповідно до [9] якість – це придатність для використання, що означає наступні дві речі:

1) якість складається з тих характеристик продукту, які задовольняють потреби клієнтів і, таким чином, забезпечують задоволення від використання продукту;

2) Якість полягає у відсутності недоліків.

Agile процес має здатність підтримувати та досягати заданого рівня якості продукту завдяки своїй безперервній зосередженості на вимогах клієнта та вдосконаленні процесу щодо активної системи розробки визначених функціональних властивостей. Це долає рутину традиційного процесу розробки програмного забезпечення, у якому співпраця з клієнтами є мінімальною та передбачає високий ризик невдоволення клієнтів. Члени команди Agile

проводять регулярні зустрічі, щоб обговорити відгуки клієнтів і планувати наступний процес ітерації. Постійний зворотний зв'язок від клієнтів і членів команди, тісна співпраця під час процесу розробки відіграє життєво важливу роль для досягнення кращої якості продукту. Agile встановив нову тенденцію в процесі розробки програмного забезпечення та залучив людей прийняти його для досягнення своїх цілей якості.

Розробка програмного забезпечення є складним завданням. Щоби зменшити складність розробки, витримується ряд процесів. Але в останні кілька років було помічено, що гнучка розробка програмного забезпечення все частіше реалізується в організаціях. Agile приваблює організації, що займаються розробкою програмного забезпечення, завдяки швидкому відгуку та ітераційному характеру роботи. Agile прийшов з багатьма способами зменшити складність процесу розробки.

В результаті опитування, проведеного австралійською консалтинговою компанією з інформаційних технологій отримали результати опитування, які показують, що Scrum був найпопулярнішою практикою в організаціях (56 % або вище) [21]. Загальні результати показано на рисунку 2.4.

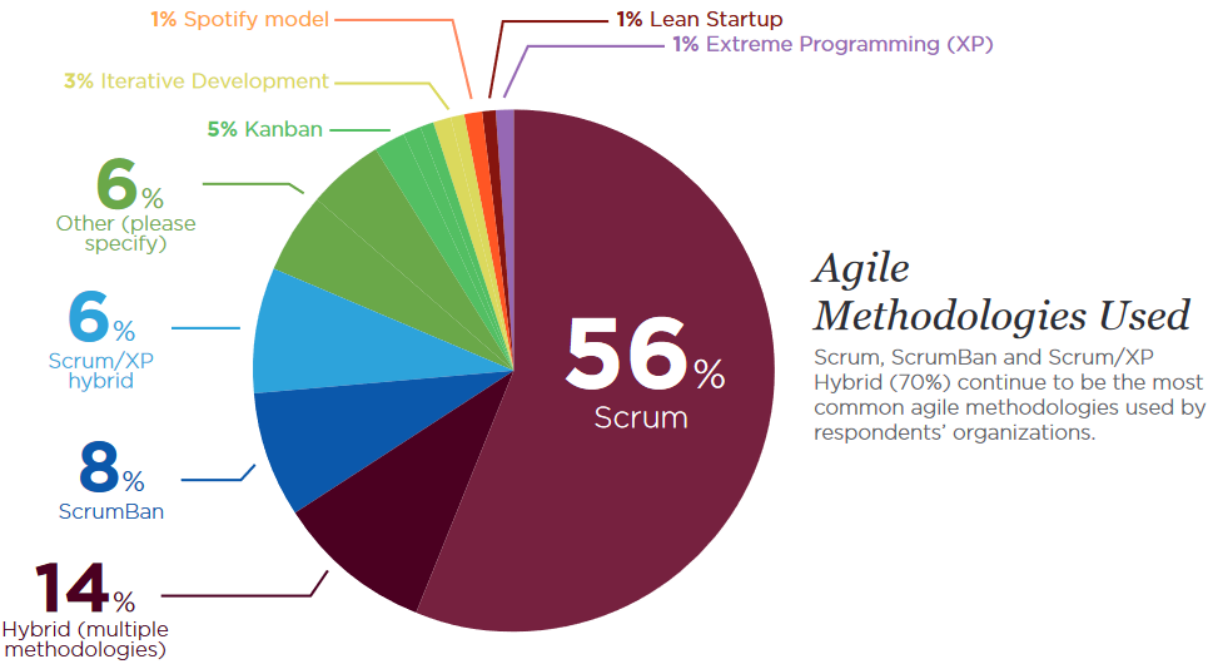


Рисунок 2.4 – Використання гнучких методів розробки ПЗ

Це опитування показує, що завдяки застосуванню гнучких методів програмного забезпечення вони отримали якість програми, простоту ведення бізнесу та зниження вартості проекту. У більшості відповідей респонденти вважають, що акцент на людях над процесами є позитивною рисою гнучкої розробки. Але гнучкі методології означають відсутність структурного планування, а документація є недоліками гнучкості. Але організації мають намір продовжувати використання гнучкого процесу або планують прийняти його в найближчому майбутньому [21].

У 2005 році компанія Digital Focus провела онлайн-опитування, проведене консалтинговою компанією в галузі IT, у якому взяли участь 137 осіб у 128 організаціях із 17 різних країн світу. Загалом 90% респондентів володіють базовими знаннями гнучких практик. Основним мотивом організації для впровадження гнучкої розробки було оброблення вимог і збільшення швидкості процесу розробки програмного забезпечення для виробництва продуктів кращої якості. Практики вказують, що здатність приймати зміни є ключовою цінністю гнучкості, але для гнучкого впровадження потрібні організаційні знання та навички.

Відповідь практиків і дослідницька робота, проведена щодо гнучкої розробки, дають чіткі докази того, що гнучкі методології впливають на організацію роботи для того, щоби подолати складності процесу розробки. Організації вірять і відчувають, що після застосування гнучких характеристик вони розробляють продукти кращої якості та досягають вищого рівня відсутності помилок. Підтримка якості та точності програмного забезпечення є головним питанням цілей розвитку організацій [22]. Незважаючи на те, що впровадження гнучкого процесу також має проблеми та недоліки, основний акцент на взаємодію між людьми та клієнтами та мінімізація процесу документування приносить кращі результати та вищий рівень задоволеності клієнтів. Це означає, що гнучкість допомагає організаціям досягти кінцевої мети якості та точності програмного забезпечення.

2.4.1 Переваги використання Agile-технологій

Опитування підкреслили переваги та недоліки гнучкого процесу розробки ПЗ. Прихильники гнучкого процесу кажуть, що індустрія програмного забезпечення значно вплинула на впровадження Agile в їхньому робочому середовищі. Гнучка розробка включає низку корисних факторів, які впливають на те, щоб організації розробляли програмне забезпечення кращої якості. Перерахуємо деякі з цих корисних факторів, з якими також стикаються практики Agile.

– Гнучкість – здатність стати гнучким і надання якісного програмного забезпечення, яке відповідає вимогам замовника, визнано ключовою перевагою гнучкої розробки.

– Підвищення продуктивності – частий зворотний зв’язок замовника зі зміною вимог дозволяє команді розробників створювати продукт кращої якості з мінімальним ризиком помилки.

– Раннє виявлення помилок. Гнучкий процес виконує проектування, аналіз і впровадження в повторюваних ітераціях, що дає чітку видимість прогресу проекту. Залежно від прогресу замовник може прийняти рішення про продовження або скасування, якщо він виявить, що все йде не так, як очікувалося, щоб заощадити додаткові інвестиції.

– Висока якість програмного забезпечення – короткі ітерації, часті відгуки та розробка, керована тестуванням, допомагають покращити загальну якість програмного забезпечення.

– Контроль проекту. Основна увага гнучкості зосереджена на людях над процесом, а менше навантаження на документацію дає розробникам можливість покращити діяльність процесу, як-от: коротка ітерація, обмін знаннями, безперервна інтеграція та зворотній зв’язок із повним контролем проекту.

– Передача знань – учасники Agile-команди діляться своїми знаннями та спостереженнями на регулярних зустрічах, під час яких вони в основному обговорюють питання розробки проекту. Ця діяльність покращує спілкування між командою, і вони залишаються в курсі поточних сценаріїв.

2.4.2 Проблеми, пов’язані з впровадженням гнучких технологій розробки

Незважаючи на те, що гнучкість має ключові переваги для досягнення кращих результатів, але впровадження нового процесу також є складним завданням, погодитися з тим, що організації, які займаються змінами, повинні змінити свої попередні налаштування, практики та противники гнучкості також вказують на проблеми впровадження гнучкості. Тут ми хотіли б висвітлити деякі загальні проблеми та виклики, які стають перешкодою для гнучкої реалізації, описані о прикладу в [23].

– Послідовне мислення, Очевидний факт, що з останніх 50 років методології розробки програмного забезпечення використовують послідовний підхід до розробки. Практики цих підходів повинні виконувати послідовний набір завдань , наприклад: збір вимог, проектування системи, а після цього вони починають кодувати. Ці люди потребують відповідної підготовки, достатньо часу та постійного моніторингу, щоб навчитися гнучким практикам. Організації, які планують застосувати гнучкі методології, мають бути впевнені, що послідовний спосіб мислення не зашкодить впровадження нового процесу.

– Індивідуальний опір. Є багато професіоналів програмного забезпечення, які не зацікавлені вивчати нові методології. Ці люди поділяються на дві групи, перша група сприймає agile як ніщо, друга група є анти-Agile, вони навмисно поширюють невірну інформацію про Agile, щоб запобігти впровадженню нових методологій у своїй організації. Цій групі людей також потрібна була детальна освіта щодо майбутніх методологій.

– Страх перед змінами. Відсутність довіри до нових методів у спеціалістів із програмного забезпечення також стає перешкодою на шляху впровадження змін. Страх змін зазвичай асоціюється з відчуттям втрати. Більшість людей побоюються, що вони не зможуть використовувати нові методи та навички для виконання вимог гнучкого використання.

– Великомасштабні організації. Організації, що працюють у великих масштабах, стикаються з проблемою впровадження гнучкого процесу, оскільки

для цього потрібна наявність клієнта на місці. Можливо, тому що різні спільноти та деякі організації не можуть дозволити собі індивідуального перекладача щоразу, коли їм доводиться спілкуватися з клієнтом. Вони все ж вважають за краще збирати вимоги відразу.

– Складне мислення щодо документації. Гнучкість зосереджена на розробці коду, а не на створенні дизайну UML. Але деякі професіонали програмного забезпечення досі вважають, що ефективність програмного забезпечення залежить від комплексної документації та розробки вимог.

– Організаційні сумніви. Люди вважають, що краще прийняти відповідні варіанти та методи з нових методологій. Багато великих організацій вважають, що поєднання гнучких і традиційних практик найкраще підходить для них; тому що впровадження загального процесу вимагатиме більшої освіти, підвищення кваліфікації та високого ризику стабільності.

2.5 Розуміння якості в контексті Agile-розробки

Не буде неправильним сказати, що гнучкий підхід до розробки не такий зрілий, як звичайний підхід до розробки. З боку практиків і дослідників було багато критики щодо гнучкого підходу до розробки. Але, крім критики, гнучкий підхід використовується багатьма організаціями для розробки програмного забезпечення. Практика в індустрії програмного забезпечення є доказом того, що цей підхід є продуктивним і корисним у кількох аспектах. Кожен із підходів до розробки програмного забезпечення має забезпечувати якість продукту та самого процесу.

«Якщо щось відповідає специфікації, воно хорошої якості » [24].

Відповідно до вищезазначеної цитати, якщо продукт/послуга відповідає вищому рівню вимог клієнтів користувачів, це означає якість. У розробці програмного забезпечення вимоги не є статичною сутністю. Вони продовжують змінюватися навіть під час розробки продукту. Традиційні підходи до розробки багато критикували за те, що вони не реагують на зміни більш ефективно. Крім

того, кілька досліджень висунули тезу про неправильне тлумачення вимог. Таким чином, важко забезпечити якість продукту/послуги без розуміння вимог. Відсутність спілкування між клієнтом і технічними спеціалістами, пов’язаними з розробкою програмного забезпечення, може призвести до неправильного тлумачення вимог [25].

У гнучкому підході розробник і співпраця з клієнтом намагалися заповнити прогалину в комунікації. Розробник постійно взаємодіє з клієнтом і розробляє систему відповідно до бачення клієнта. Таким чином організація наближається до функціональності вимог замовника, що призводить до кращої якості програмного продукту. Шостий принцип Agile Manifesto заохочує навіть особисту співпрацю.

Ще однією перевагою залучення клієнта є подолання проблем зміни вимог. Як і в гнучких проектах, наприкінці ітерації функціонуюча частина робочого програмного забезпечення випускається для зворотного зв’язку. Коли відбувається наступна ітерація, змінені вимоги об’єднуються в попередній реліз; тому це не впливає на проект у ході розробки. Прототипи розробляються на початкових стадіях проекту, що допомагає уникнути конфліктів щодо того, як система буде виглядати або працювати. На відміну від традиційного підходу до розробки, у гнучких проектах продукт скоріше еволюціонує, а не розвивається, і еволюція здебільшого призводить до підвищення якості продукту/послуги.

Якість програмного продукту також можна оцінити з точки зору бездоганності та зменшення або нульової кількості помилок і дефектів. Щоб зробити програмний продукт без помилок і оцінити його ефективність, проводиться тестування. Там, де розробка програмного забезпечення рухається до гнучкої розробки, в проектах також змінюються підходи до тестування. У будь-якій розробці програмного забезпечення діяльність із забезпечення якості (QA) безпосередньо пов’язана та відповідає за якість процесів розробки програмного забезпечення, а також програмного продукту. Гнучка розробка програмного забезпечення переосмислює діяльність і практику забезпечення якості.

У гнучкій розробці програмного забезпечення тестування та розробка проходять паралельно, щоб досягти вищого рівня якості. Тестування можна оцінити як основу діяльності з контролю якості та важливий крок для досягнення якості програмного продукту. У гнучкому процесі розробки помилка або баг виправляються, як тільки про них стає відомо, і незалежно від того, хто їх виправляє [26]. Застосовуючи цей спосіб, баги та помилки виявляються та виправляються на ранніх стадіях. Швидше виправлення помилок означає економію часу, коштів, ресурсів і підвищення якості.

2.6 Забезпечення якості продукту через якість процесів

Оскільки компанія націлена на якість програмного продукту, вона також прагне скоротити витрати, час і ресурси. У великих і складних проектах програмного забезпечення головна увага приділяється часу та ефективності, щоб якість не була під загрозою. Можна сказати, що якщо процес розробки забезпечує максимальне використання мінімальних ресурсів, це відображає якість. У цьому розділі ми обговоримо переваги гнучкого підходу до розробки для компанії в процесі розробки.

У [24] 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.

Стверджується, що гнучкий процес розробки є більш гнучким до змін. Отже , за своєю природою гнучкий підхід ефективно реагує на зміни, і додаткові зусилля та час не потрібні. І будучи ітераційними, ресурси організації не замикаються в лінійних процесах. Кілька досліджень виявили, і багато практиків стверджували, що помилки та дефекти, виявлені після релізу, коштують більше часу та ресурсів організації [27]. Гнучкі практики, як-от парне програмування та розробка, керована тестуванням, спрямовані на виявлення та усунення недоліків програмного забезпечення до остаточного релізу, і в кінцевому підсумку такий характер розробки приносить користь організації з точки зору часу та ресурсів.

Гнучкі методи не орієнтовані на процес, вони, як стверджується, орієнтовані на людей. У принципах Agile Manifesto йдеться про надання людям такого середовища, яке переконує їх працювати вільно, а не бути обмеженими процесами. Якщо під час проекту розширюється співпраця, а залученим людям довіряють виконати роботу, це може призвести до значного підвищення якості та обсягу роботи [24].

Хоча гнучкі методи є принципово адаптивними до змін, поступовими та самоорганізованими, але багато роботи виконується в області вдосконалення процесів щодо гнучкої розробки. Написано кілька книг, щоб застосувати CMMI в гнучких проектах. Крім того, стандарти Six Sigma та ISO також визнають важливість гнучкої розробки в проектах і заохочують цю методологію.

3 ПРОПОНОВАНЕ РІШЕННЯ З ВПРОВАДЖЕННЯ SQA В AGILE-МЕТОДОЛОГІЇ РОЗРОБКИ

Гнучкі методології розробки (також відомі як методології легкої ваги) все частіше використовуються та приймаються для проектів розробки програмного забезпечення. Ці методології є адаптивними до змін, гнучкими, а такі функції, як рефакторинг, залучили велику кількість практиків до прийняття гнучких методологій розробки. Деякі практики виступають за гнучку розробку, інші виступають проти неї, де деякі практики рекомендують прийняти змішані гнучкі та керовані планом практики [25]. У цьому розділі ми більш конкретно обговоримо забезпечення якості в Agile-розробці та менеджменті. Тут більше до уваги береться погляд практиків на гнучку розробку. Крім того детально обговоримо запропоновані методики, зіставляючи їх із результатами опитувань професіоналів-практиків.

3.1 Agile-технології тестування

Парне програмування та тестова розробка (Test Driven Development – TDD) є одними з ключових практик гнучкої розробки для досягнення якості програмних продуктів. Використовуючи ці підходи, гнучка розробка робить тестування невід’ємною частиною проекту. Але Agile кардинально змінила практику тестування, переклавши відповідальність за тестування від тестувальника до розробника. В Agile розробники повинні писати тести та тестувати свій код або код один одного, виконуючи парне програмування. Передбачається, що замовник бере активну участь у гнучкій розробці протягом усього проекту. Приймальні випробування є відповідальністю замовника, який бере участь у проекті. Прихильники гнучких технологій стверджують, що залучення клієнтів до проекту та тестування призводить до розробки програмного забезпечення, яке краще відповідає вимогам.

Розробка на основі тестування (TDD) стає популярною з кожним днем у гнучкому співтоваристві. TDD зосереджується на написанні тестів перед кодуванням і частій інтеграції нового коду. У TDD новий код означає фрагмент коду або фрагмент коду, який уже існує та інтегрований після кількох змін з метою реалізації змінених вимог (за відгуками споживачів) [28]. На TDD впливає підхід Test-First Development (TFD), згідно з яким розробники повинні писати всі тести перед початком фактичного програмування. Можна сказати, що

$$TDD = Рефакторинг + TFD. \quad (3.1)$$

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

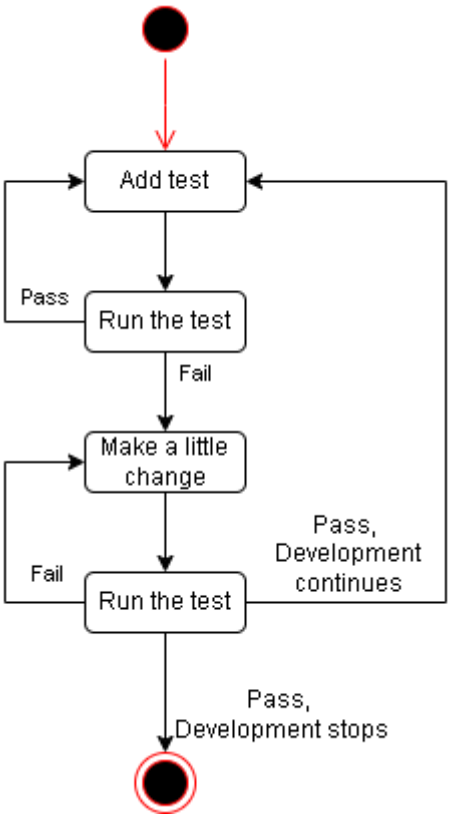


Рисунок 3.1 – Test-First розробка

Можна сказати, що TDD – це не лише підхід до тестування, але й підхід до поступового та поступового проектування програмного забезпечення. Як ми

бачимо на наведеній вище діаграмі, що запроваджуючи TFD, кожна частина коду має пройти всі тест-кейси, перш ніж її написати.

TDD змінив роль забезпечення якості в розробці програмного забезпечення, оскільки тестування проводиться розробником. Практики Agile стверджують, що це постійне тестування призводить до створення програмного продукту високої якості. В [29] автор визначає гнучке забезпечення якості як «розробку програмного забезпечення, яке може реагувати на зміни, коли клієнт вимагає цих змін. Це означає, що часта доставка перевіреного, робочого та схваленого клієнтом програмного забезпечення наприкінці кожної ітерації є важливим аспектом гнучкого забезпечення якості».

На практиці контроль якості в гнучкій розробці розвивається навколо відгуків клієнтів і розробника, який одночасно є тестувальником. Окрім досягнення вищої якості завдяки безперервному тестуванню, Agile відкрив нові розділи критики, розроблені практиками та дослідниками. Але перш ніж обговорювати проблеми та критику, з якою гнучка система контролю якості стикається в промисловості, давайте подивимося ще на кілька заходів із забезпечення якості гнучкої розробки.

3.2 Парне програмування

Відповідно до [30] парне програмування (Pair Programing – PP) означає, що весь робочий код пишеться двома людьми на одному екрані/клавіатурі/миші. PP є одним із основних видів діяльності XP. Метою впровадження PP є постійний моніторинг і навчання один у одного під час написання коду. Моніторинг розробки та процесу програмного забезпечення є обов’язком персоналу QA, але PP покладає цю відповідальність на розробників, змушуючи їх працювати разом.

У промисловості спостерігаються твердження, що для досягнення високої якості використовується підхід PP, оскільки в [31] стверджує, що парне програмування може покращити якість дизайну та зменшити кількість дефектів. Але суть дискусії полягає в тому, як довіряти продуктивності реального

програмного продукту розробникам, а не професійним тестувальникам. Безсумнівно, гнучка розробка наголошує на підборі хорошого та досвідченого персоналу; але якщо якість досягається через найкращий персонал, тоді де продуктивність методології розробки?

Учасники визнають, що для того, щоб гнучкі проекти були успішними, необхідний певний рівень досвіду персоналу. Був певний консенсус, що 25%-33% персоналу проекту мають бути компетентними та досвідченими. Тут під компетентністю та досвідом розуміються досвід побудови подібних систем, попередні знання в галузі технологій і достатні навички міжособистісного спілкування та спілкування. Суть у тому, що для успішного проекту досвідчені люди, які розробили ті самі системи, важливіші, ніж люди, які працювали в гнучкому середовищі.

3.3 Рефакторинг

Рефакторинг є невід'ємною практикою гнучких методологій. Одне з поширених визначень рефакторингу коду представлено в [32] як техніка реструктуризації існуючого тіла коду, зміна його внутрішньої структури без зміни зовнішньої поведінки. Його основа – це серія невеликих трансформацій, що зберігають поведінку. Кожне перетворення (так званий «рефакторинг») робить мало, але послідовність перетворень може призвести до значної реструктуризації. Оскільки кожен рефакторинг невеликий, менша ймовірність, що він піде не так. Система також підтримується в повній працездатності після кожного невеликого рефакторингу, зменшуючи ймовірність того, що вона може отримати серйозний дефект під час реструктуризації.

Відповідно до [30] дизайн системи розвивається шляхом трансформацій, і це досягається рефакторингом у XP. Мета рефакторингу – зробити код більш зрозумілим, зменшити його складність і спростити потік коду. Дослідження та практика показали, що рефакторинг коду позитивно впливає на якість програмного забезпечення, оскільки зменшує ймовірність помилок. Це також

розглядається як підхід до перегляду коду, тому що коли розробник проходить код для рефакторингу, це також допомагає йому усунути помилки, окрім реструктуризації коду.

Було представлено дослідження [34], яке спостерігало еволюцію розробки програмного забезпечення за допомогою XP протягом більш ніж двох років. У цьому дослідженні представлені порівняльні результати щодо XP (гнучка розробка) і розробки, керованої планом. Результати цього дослідження показали, що використання рефакторингу (в XP) є помітна відсутність складності в коді порівняно з плановою розробкою. У керованій планом розробці код можна реструктурувати, коли він весь написаний і протестований. Але при застосуванні підходу рефакторингу код постійно реструктуризується та інтегрується, що робить його менш складним і допомагає виявляти помилки на ранніх стадіях розробки.

Рефакторинг має позитивний і важливий вплив на забезпечення якості. Це допомагає зменшити кількість помилок, що зрештою економить час. У гнучкій системі інтеграція та рефакторинг коду здійснюються постійно, тому проблеми сумісності можна ефективно подолати на ранніх етапах розробки програмного забезпечення. Практики також наводять аргумент, що рефакторинг коду врешті-решт зменшує потребу у великій документації.

3.4 SPI та гнучка методологія

Удосконалення процесів розробки програмного забезпечення (Software Process Improvement – SPI) надає компанії послідовний метод виконання завдань, а SPI впливає на організацію адміністративно, а не технічно [34]. У сучасну епоху компетентності компанії вірять, що вдосконалюючи процес розробки програмного забезпечення, вони можуть розробляти якісні програмні продукти.

Протягом епохи впроваджувалися різні комерційні стандарти та моделі, щоб покращити процес розробки програмного забезпечення, зазвичай використовуваними стандартами та моделями є стандарти ISO та інтеграцію

моделі зрілості можливостей компанії (Company Maturity Model Integration – CMMI). Ці комерційні моделі та стандарти можуть відрізнятися за своєю природою, але їх кінцевою метою є постійне вдосконалення процесу розробки програмного забезпечення.

Щоб отримати сертифікацію за певним стандартом або моделлю до певного рівня, організація повинна повністю відповідати стандартам в рамках свого процесу. Щоб досягти цього, необхідно детально документувати розробку продукту протягом усього процесу, стандартизовані процедури повинні дотримуватися та контролюватись, категорично та часто проводитися аудити та процедури оцінки. Дотримуючись цих стандартів і моделей, SQA відіграє найактивнішу роль. Персонал із забезпечення якості відповідає за впровадження, моніторинг і керівництво обраною моделлю або стандартами для SPI на рівні організації, а також на рівні проекту.

Через брак часу та масштабів поточного дослідження ми не зможемо обговорити всі стандарти та моделі для SPI. У цьому розділі ми обговорюємо CMMI, оскільки це широко використовувана модель SPI. У промисловості деякі вважають, що CMMI – це набір стандартів, але, окрім назви, експерти CMMI стверджують, що це модель для SPI. Безсумнівно, якщо якась організація постійно вдосконалює свій метод розробки, це позитивно вплине на її продукти. Але всі ці філософії та практики вітаються в плановій розробці програмного забезпечення. Стверджується, що гнучкі методології несумісні з цими стандартами та моделями, і дебати щодо SPI та Agile тривають.

Неймовірні звичайні методології розробки. Гнучка розробка інтегрує практики забезпечення якості в діяльність з розробки, а не практикує їх незалежно та окремо. Іноді через управлінські та/або організаційні проблеми замовник або проект вимагає дотримання деяких стандартів. А для відповідності стандарту генерація документації має важливе значення.

Відповідно до [34] розглядаються проблеми, які роблять несумісними CMMI та гнучкі методології. Відповідно до цього документу, одна з важливих причин, через яку CMMI та гнучка методологія несумісні, полягає в тому, що

організації, які прийняли CMMI, були великими з високим рівнем менеджменту, організаційної ієрархії та управління; в свою чергу організації, які прийняли гнучкі методології, зосереджувалися на невеликих або однокомандних проектах.

У 2005 році було проведено інше тематичне дослідження [63], яке було зосереджено на використанні CMMI та гнучких методологій для зрілості організації. У цьому прикладі висувається претензія: «На даний момент існуючим гнучким методам проектного навчання, здається, бракує засобів для сприйняття організаційного аспекту SPI. Наприклад, вони не стосуються важливих аспектів систематичного визначення, перевірки, упаковки та зберігання результатів SPI гнучких проектів . ”[63]. Можна помітити, що всі аспекти, розглянуті в цій претензії, є обов’язками SQA. Будь-які вдосконалення всередині організації, проекту чи розробки спрямовані на досягнення вищого рівня програмних продуктів. Якщо гнучкі методики застосовуються для виробництва якісних продуктів без CMMI, тоді є величезний простір для вдосконалення.

Дослідження в компанії Ericsson було проведено з метою подолання прогалин у SPI та гнучкої розробки [35]. Дослідники, які брали участь у цьому дослідженні, заявили, що промисловість потребує розробки та вдосконалення практик гнучкого ІЗШ. Щоб досягти своєї мети , вони спостерігали за 18 різними проектами, впровадженими двома різними тактиками SPI. Вони розробили дві тактики SPI для SPI і назвали їх Supertank Tactic і Motorboat Tactic. Основою першої тактики було мислення CMMI, а базою другої тактики було Agile мислення. Після завершення дослідження вони зрозуміли, що друга тактика для SPI принесла кращі результати та позитивний вплив. Але у своєму дослідженні дослідники також визнали, що були доступні детальні вказівки щодо матеріалів для відпрацювання першої тактики, оскільки вона була розроблена на основі мислення CMMI. Незважаючи на кращі результати, дослідники стверджують, що гнучка розробка зосереджена на розробці програмного забезпечення на рівні проекту, тоді як мислення CMMI полягає в зосередженні на розробці програмного забезпечення через процес розробки на рівні організації.

У розробці, керованій планом, SPI охоплює всі види діяльності, пов’язані із забезпеченням якості та розробкою програмного забезпечення чітко визначеним , організованим і стандартизованим способом. Хоча люди, залучені до проекту, є учасниками SPI, але персонал із забезпечення якості відповідає за діяльність SPI. Персонал SQA проводить опитування, оцінку, зустрічі та внутрішні аудити. Вони вважаються експертами в організації, оскільки навчають і впроваджують у ній SPI. Крім того, поєднання діяльності з контролю якості в розробці програмного забезпечення, гнучкі методології скорочують організаційну роль контролю якості. Розробники можуть знати про тестування та проектування, але вони можуть бути менш обізнані про SPI на рівні організації. Літературні дані показують, що необхідно переглянути роль SQA в проектах гнучкої розробки, щоб підвищити знання організації та зрілість для досягнення максимального результату.

3.5 Пропоноване рішення

Ми зібрали доступну літературу та дослідницькі дані, провівши огляд літератури в предметній області Agile-розробки та процесів QA. Виявилось, література в цій галузі дослідження має такі особливості:

– Претензії на вищу якість завдяки гнучкій розробці (керівництва, результати опитувань, прихильності та критики).

– Інтегрована діяльність із забезпечення якості в гнучких методологіях (рекомендації, переваги та критика).

– Твердження про відсутність визначених стандартів гнучкої розробки (критики, опитування).

– Дослідження, проведені для підвищення продуктивності проектів гнучкої розробки (результати дослідження та пропозиції).

Після цього огляду літератури ми вважаємо, що роль практик забезпечення якості в гнучких проектах необхідно переглянути. Хоча наше фундаментальне

спостереження залежить від літератури, але ми також провели емпіричне дослідження.

Відповідно до цього твердження QA має бути інтегрованим, а не зміщеним, тут зміщення означає, що більшість дій із забезпечення якості в гнучких проектах виконується розробником або персонал із забезпечення якості повинен виконувати роль розробника. Немає сумніву, що гнучкі методології досягли вищої якості завдяки своїй інкрементній та тестовій природі. Виробництво більш високої якості та відсутність систематичних, організованих і чітко визначених процедур і стандартів показують, що гнучка діяльність SQA дає багато можливостей для вдосконалення. Основним напрямком запропонованого нами рішення є «перевизначення діяльності SQA, а не зміна».

Діяльність із забезпечення якості в гнучких проектах розвивається навколо тестування та зворотного зв’язку. Доступна література щодо гнучкого забезпечення якості залишається зосередженою на тестуванні та його різних підходах. Гнучкі методології є поступовими, адаптивними до змін і самоорганізованими. Але в будь-якій ситуації потрібне лідерство, оскільки самоорганізація не може успішно працювати, коли вимоги та система стають складнішими. Оскільки ці методології називають орієнтованими на людей, тож, зосереджуючись на цьому пункті, гнучкі методології, схоже, використовують досвід людей не для тих цілей, для яких їх навчали.

Різні дослідницькі роботи в галузі програмного забезпечення та практики стверджують, що гнучкі методології менш сприятливі для розробки складних програмних систем. Предметні методології вимагають створення команди від найкращих людей, і при цьому експерти та аналітики з дизайну в деяких ситуаціях відіграють роль розробника, а з іншого боку, розробник також повинен проводити діяльність із забезпечення якості. У плановій розробці програмного забезпечення QA активує моніторинг і окремі сутності. Agile інтегрує діяльність із забезпечення якості в розробку програмного забезпечення, вимагаючи від деяких спеціалістів із забезпечення якості виконувати роль розробника в проектах.

У конвенційному підході діяльність SQA спрямована не лише на якість продукту, але й на якість процесу на рівні організації. SPI є життєво важливою частиною стандартної практики SQA. Промисловість не погоджується уникати важливості та переваг SPI, тому було проведено достатньо досліджень, щоб отримати CMMI та Agile, сумісні один з одним.

Отже, гнучкий підхід до розробки потребує переосмислення ролі та діяльності SQA в проектах. Підвищення ролі SQA може призвести до того, що гнучкі методології стануть ефективними на рівні організації.

У запропонованому рішенні застосування додаткового рівня контролю якості означає залучення експертів із забезпечення якості до команди розробників. Прийнявши цей підхід, організація може підтримувати стандарти якості. Усі типові дії із забезпечення якості, як-от тестування, збір вимог і оцінка, мають бути обов’язками експерта з забезпечення якості, а не розробника. Щоб підтримувати гнучкість, слід заохочувати спілкування між командами розробників і всередині команди. Застосування додаткового рівня забезпечення якості в організації, забезпеченого зворотним зв’язком, буде обговорено в наступному розділі, щоб отримати детальне розуміння.

Запровадження культури тестування в гнучких проектах може підвищити цінність якості продукту, але якість процесу не може бути знижена для успішного бізнесу. Практиків Agile менше приваблює сертифікація CMMI або інші комерційні стандарти якості продуктів і процесів. За допомогою огляду літератури виявлено, що Agile більше зосереджується на тестуванні, яке може призвести до нехтування іншими видами діяльності з контролю якості. Якщо відповідальність за аналіз вимог і тестування перекладається на розробника, це може перевантажити роботу розробника, спричинивши недостатню якість процесу та продукту. Немає сумнівів щодо ролі спеціалістів із забезпечення якості в проектах розробки програмного забезпечення.

Ми спостерігаємо, що роль фахівців із забезпечення якості в проектах гнучкого розвитку потребує переосмислення, щоб отримати вищу якість, а не покладати свої обов’язки на розробників. Запропонована методика полягає в

тому, щоб виділити професіоналів із забезпечення якості в команду розробників як членів. І вся діяльність із забезпечення якості всередині команди має контролюватися фахівцем із забезпечення якості, а взаємна співпраця цих спеціалістів із забезпечення якості має заохочуватися та підтримуватися, щоб підтвердити гнучкість проекту.

У гнучких методологіях або розробник працює як тестувальник, або навпаки, це суттєво змінює відповідальність і може розглядатися як виклик експертів. Прийнявши запропонований підхід, організація зможе отримати якісну роботу, оскільки всі ресурси будуть працювати в своїй компетенції. Розробник не повинен тестувати свій власний написаний код, але в гнучкій розробці більша частина тестування виконується розробником.

Ми вважаємо, що гнучким методологіям бракує стандартів для доступу до їхніх можливостей і зрілості на відміну від традиційної розробки. Це може бути пов’язано з тим, що всі експерти беруть участь у розробці та не розглядають моніторинг та вимірювання проекту на організаційному рівні. Застосовуючи запропонований підхід, організація може розробляти та приймати стандарти та ефективно контролювати якість складних проектів.

Замість того, щоб навчати замовника, краще використовувати досвід тих людей, які мають здатність інтерпретувати потреби замовника, які необхідно технічно реалізувати. Але безперервна взаємодія з клієнтом також є позитивною цінністю для якості продукції та гнучкості. Єдина різниця в нашому підході – це не розробник . Персонал із забезпечення якості повинен регулярно взаємодіяти з клієнтом, щоб забезпечити якість продукції.

Під час огляду літератури ми виявили, що багато практиків стверджують, що потрібне ведення детальної документації. Документація не тільки потрібна клієнту, але й необхідна для підтримки якості продукту в майбутньому. Крім того, відгуки змусили нас повірити, що розробник не може виконати всі вимоги щодо якості та процедури без допомоги персоналу з контролю якості. Ми вважаємо, що якщо знання та досвід традиційної діяльності з забезпечення якості передаються в гнучкий проект через додатковий рівень забезпечення якості, це

може привести нас до максимізації якості продукту та зрілості гнучких методологій.

3.6 Застосування додаткового рівня контролю якості

Відповідно до запропонованого підходу принаймні двом командам розробників має бути призначено одного експерта з якості, який буде працювати як член кожної групи розробників. Цей експерт із забезпечення якості виконуватиме роль не розробника, а контролюватиме всю роботу команд, і цей експерт із забезпечення якості називатиметься вузлом забезпечення якості. Цей додатковий рівень забезпечення якості може служити процесу розробки таким чином, щоб підтримувати 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.

Збір вимог/історій користувачів у гнучких методологіях здійснюються шляхом співпраці користувача та розробника. Ми пропонуємо, щоб QA-вузол і керівник групи розробників виконували це завдання, а потім повинні пояснити ці вимоги розробникам. Прийняття цього підходу допоможе ефективніше реалізувати вимоги користувачів, навіть якщо вони досить складні. З іншого боку, присутність керівника групи розробників дозволить також збирати вимоги користувачів з технічної точки зору. Як і в гнучкій системі, вимоги недостатньо задокументовані, оскільки написані історії користувачів, а розробник інтерпретує ці історії, щоб реалізувати їх як функціональність програмного забезпечення.

Співпраця є фундаментальним елементом гнучких проектів. Але коли співпраця здійснюється між людьми з різними підходами, це може призвести до непорозумінь і втрати часу. Ми пропонуємо, щоб вузли забезпечення якості були

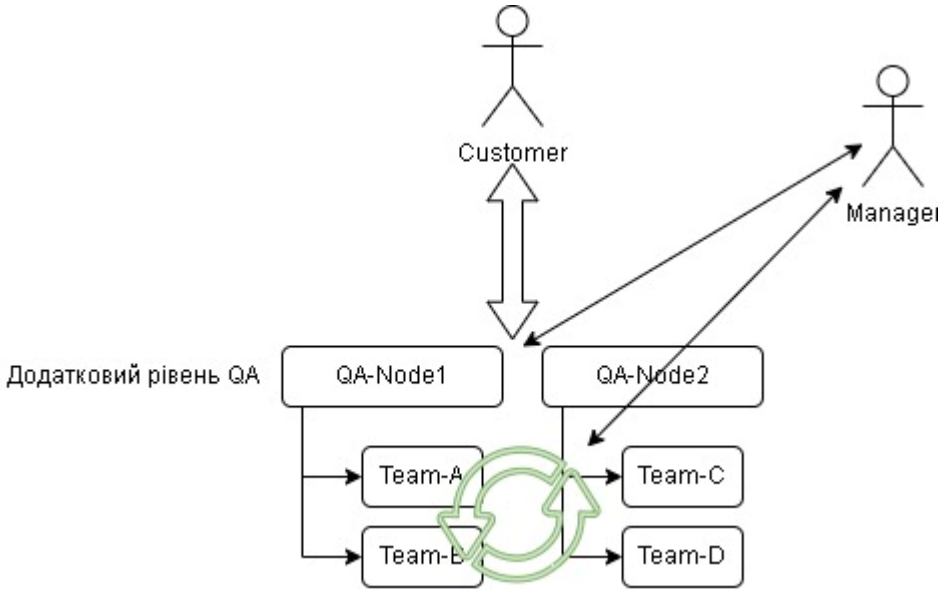
на передньому плані для співпраці з клієнтами, оскільки професіонали з забезпечення якості – це люди, які мають технічний і соціальний досвід.

Документація була критичною проблемою серед звичайних і гнучких методологій. Гнучка розробка не проти документації, але заохочує комплексну документацію, щоб заощадити час і ресурси. У деяких ситуаціях, коли попит клієнтів або програмне забезпечення є складним, документація повинна бути великою. Оскільки в документації, що керується планом, відповідає QA, тут у запропонованому нами підході QA-вузли повинні виконувати це завдання лише в межах своїх команд розробників. Це може уникнути додаткового навантаження на розробника.

Взаємодія на додатковому рівні контролю якості має бути однією з процедур проекту, як щоденні зустрічі. Команди розробників повинні взаємодіяти одна з одною та зі своїми вузлами забезпечення якості. Ці вузли забезпечення якості повинні підтримувати взаємну взаємодію, щоб вимірювати прогрес, обговорювати критичні питання, ділитися знаннями та ресурсами в рамках проекту. У сукупності вузли QA розглядаються як додатковий рівень QA проекту.

SPI вимагає постійного вивчення конкретних стандартів або моделей. Персонал із забезпечення якості є активним ресурсом для ініціювання та підтримки SPI в організації. Вони є експертами та мають достатні знання SPI. Крім того, вузол контролю якості 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).



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

На наведеному вище рисунку, коли графічно введено додатковий рівень якості, кожному вузлу забезпечення якості було призначено дві команди розробників (A, B, C і D) відповідно. Кількість команд або QA-вузлів може відрізнятися від організаційних ресурсів і рівня досвіду кожного вузла. Ці вузли забезпечення якості підтримують гнучку розробку в своїх командах і беруть участь у роботі команд як члени команди. Не порушуючи основного мислення проекту розробки, запропонований додатковий рівень QA практично відокремлений і повинен підтримувати співпрацю з зустріччю всіх QA-вузлів. Замовник повинен взаємодіяти з QA-вузлом і керівником команди взаємно. Роль керівництва полягає в тому, щоб продовжувати спостерігати та виконувати відповідні рішення, не порушуючи технічне рішення, оскільки вони приймаються командою для підтримки гнучкості.

Під час цього дослідження виявлено, що запропонована методика може мати можливі недоліки, перш ніж вона стане зрілою шляхом постійних досліджень. Деякі з очікуваних недоліків наведені нижче:

– Витрати часу на комунікацію з рівнем QA-вузлів.

– Перенесення частини обов’язків від розробника до персоналу з контролю якості може сповільнити темп проектів на початку.

– Розробка нових інструментів для дотримання запропонованої техніки може зайняти багато часу та коштів.

Переваги та недоліки будь-якої техніки в реальному середовищі можна спостерігати, реалізувавши її. Через обмеження нашого дослідження ця методика не реалізована для надання результатів у реальному середовищі. Застосування запропонованої нами методики може призвести до інших недоліків, крім перелічених вище, які можна використати для вдосконалення цього підходу.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Методи та засоби психофізіологічного розвантаження як допоміжний процес в розробці ПЗ

Для галузі ІТ інформаційна культура є необхідною умовою виживання, тому що зміна технологій в розробці програмного забезпечення відбувається кожні 6-8 місяців, а інвестиції на підготовку персоналу і освоєння нової технології величезні і великих компаніях варіюються від 1,5 до 2 млрд. доларів на рік [36].

Аналіз свідчить, що інформатизація та інтеграція комунікаційного простору України сприяє різкому підвищенню інформаційної та професійної компетентності, ділової активності, стимулюванню конкуренції, створенню інноваційних підприємств та організацій, нових робочих місць, зниженню витрат на утримання управлінського апарату [36]. Поряд із задачами і здобутками окреслилися негативи використання інформаційних технологій:

1) надмірне інформаційне навантаження, суть якого полягає у тому, що кількість корисної інформації, яка надходить до мережі, перевищує психофізіологічні можливості її сприйняття людиною;

2) велика кількість інформації, яка сприймається, але не є корисною для фахівців в даний момент;

3) інформаційний голод, причиною якого є саме надлишок інформації, викликаний інформаційним перенавантаженням;

4) «інформоманія» як хвороба людини, яка робить останню знеособленою, залежною від перебування в інформаційному просторі і роботи з комп’ютером і чому вона віддає перевагу, уникаючи «живого» спілкування з людьми;

5) поява «кіберспільнот», що за своїми соціокультурними характеристиками набагато ближчі до представників інших культур у

глобальному інформаційному просторі, ніж до своєї етнонаціональної спільноти чи решти населення, не охопленого Інтернетом;

6) індивідуалізм і дегуманізація способу життя «мешканців» Інтернету – відсутність готовності ділитися своїми знаннями.

Слід розуміти, що комп'ютерні технології істотно впливають на життєдіяльність людини, припускаючи глобалізацію і технократизацію суспільства. Але в ще більшій мірі цей вплив поширюється безпосередньо на центральну нервову систему, яка звикає працювати в дуже інтенсивному режимі багатозадачності, де вже переважають не тривалі логічні роздуми, а інтуїтивно-реактивні ланцюжки розумових формулювань у зв'язку з величезним обсягом оброблюваної щодня інформації, кількість якої зростає за експоненціальною швидкістю. Виникає припущення, що саме збільшення обсягу інформації та прискорення її обробки людиною може згубно вплинути на розвиток розумових здібностей людини.

У [36] наведено перелік протипоказань з боку органів зору та загальних (соматичних) протипоказань, які забороняють роботу на ЕОМ, а також комплекс вправ для поліпшення здоров’я і підвищення працездатності.

Таким чином, за умови високого рівня робіт з ЕОМ рекомендується психофізіологічне розвантаження у спеціально обладнаних приміщеннях (кімнати психофізіологічного розвантаження) під час регламентованих перерв або в кінці робочого дня.

При проведенні сеансів психофізіологічного розвантаження рекомендується використовувати деякі елементи методу аутогенного тренування, який ґрунтується на свідомому застосуванні комплексу взаємопов’язаних прийомів психічної саморегуляції й виконанні нескладних фізичних вправ зі словесним самонавіюванням.

У рекомендованому сеансі, який має проводитися у кімнаті психофізіологічного розвантаження з відповідним інтер'єром та кольоровим оформленням, виділяють такі три періоди, або фази:

Перший – абстрагування працівників від виробничої обстановки – відповідає фазі залишкового збудження. Звучить повільна мелодійна музика, пташиний спів. Обравши зручну позу, працівники адаптуються і психологічно готуються до наступних періодів.

Другий – заспокоєння – відповідає фазі відновлювального гальмування. Пропонується показ фото слайдів із зображеннями квітучого лугу, березового гаю, гладенької поверхні ставка тощо. Через навушники транслюється спокійна музика.

Як функціональне освітлення застосовують зелене світло. Яскравість світла має поступово знижуватися впродовж періоду заспокоєння, а наприкінці його світло вимикається зовсім на 1– 2 хв. Екран теж гасне.

Третій – активізація – відповідає фазі підвищеної збудженості.

На початку періоду світло вимкнене, через певний час на екрані з'являється червона пляма, розміри й яскравість якої поступово збільшуються.

Наприкінці періоду звучить бадьора музика. Вимовляються тричі мобілізуючі формули аутогенного тренування, яким мають передувати глибокий вдих та довгий глибокий видих.

Після сеансів психофізіологічного розвантаження у працівників зменшується відчуття втоми, з'являється бадьорість, хороший настрій. Загальний стан відчутно поліпшується.

4.2 Попередження аварій на виробництвах із застосуванням хлору

Хлор є частиною таблиці хімічних елементів і розташовується в ній під номером 17. У природі він зустрічається виключно у формі газу. Найчастіше він має специфічний зелений з жовтим переливом колір. Цей елемент важчий за повітря в 2,5 рази, тому накопичується в підвалах будинків, а на пересіченій місцевості в ярах і низинах. У воді ж хлор розчиняється без сліду і його наявність помітно тільки при великій концентрації (за рахунок специфічного запаху) [37і].

В організмі людини в середньому міститься 95 г хлору. За добу людина споживає 5-10 г хлору (кухонна сіль). Він потрібен для вироблення в шлунку соляної кислоти, яка сприяє травленню і знищенню хвороботворних бактерій. Добова потреба хлору для людини становить 800 мг.

Хлор широко застосовується на виробництві, на його основі виготовляють отрутохімікати, розчинники, засоби для дезінфекції та миття, медикаменти. Хлор використовується в кольоровій металургії, у виготовленні пластмас тощо. Також хлор з успіхом застосовується і в побуті для очищення, відбілювання, прання. Завдяки незначним витратам і досить високої ефективності дезінфекції, хлор активно використовується для очищення і знезараження води в плавальних басейнах і питної водопровідної води.

Отруєння хлором можливе в разі:

– перевищення максимально допустимих концентрацій хлору для знезараження води в трубопроводі (сильний запах хлору);

– наявність хлору у великій кількості у воді басейну і часте купання в ньому;

– відбілювання і прання в закритому не провітрюваному приміщенні;

– аварії на підприємстві;

– використання хлору в якості зброї масового ураження.

В організм хлор потрапляє через слизові оболонки дихальної і травної систем, шкіру.

Ознаки отруєння хлором. До перших ознаках отруєння хлором відносяться:

– дискомфорт і подразнення слизової дихальних шляхів;

– підвищене слиновиділення і спазм голосових зв’язок;

– кашель і утруднене дихання;

– відчуття різі та печіння в очах, сльозотеча;

– нудота і гіркота у роті;

– головні болі і можливі судоми.

При попаданні на шкірний покрив або слизові спостерігається значний свербіж і гіперемія (почервоніння), вірогідні підшкірні крововиливи без пошкодження цілісності шкіри.

Тяжкість патологічного процесу та симптоми отруєння хлором знаходяться в прямій залежності від дози отруйної речовини (хлору) і тривалості його дії.

До прибуття медиків слід надати домедичну допомогу потерпілому:

– усунути джерело надходження отрути в організм – вивести або винести потерпілого поза зону дії отруйної речовини. При цьому необхідно пам’ятати про безпеку рятувальника – застосування марлевої маски або респіратора.

– забезпечити доступ чистого повітря;

– зняти забруднений одяг і теплою (не гарячою) водою промити контактуючі ділянки шкіри.

– у разі перорального надходження (проковтування) хлорвмісних рідин, потрібно промити шлунок. Промивати краще через зонд, або можна викликати блювання після рясного пиття.

– у разі пошкодження очей, промивання великою кількістю води або слабким розчином соди для зняття подразнення;

– полоскання ротової порожнини та носа содовими розчинами для мінімізації ушкодження слизових оболонок, застосування інгаляцій з додаванням соди для полегшення кашлю.

До профілактичних заходів отруєння хлором належать:

– забезпечення належних умов праці відповідно до санітарно-технічних вимог (вентиляція, провітрювання, справне обладнання);

– використання індивідуальних засобів захисту при роботі з хімікатами на виробництві;

– регулярні перевірки концентрацій хлору в повітрі робочої зони;

– проведення профілактичних медичних оглядів для виявлення схильності (доклінічних форм) і хронічних захворювань;

– дотримання вимог безпеки у використанні хлорвмісних рідин в побуті.

Суб'єкт господарської діяльності зобов'язаний забезпечити працівників хлорних об'єктів спеціальним одягом, спеціальним взуттям та іншими засобами індивідуального захисту відповідно до Положення про порядок забезпечення працівників спеціальним одягом, спеціальним взуттям та іншими засобами індивідуального захисту:

а) для захисту органів дихання – фільтруючими протигазами, ізолюючими дихальними апаратами та ізолюючими костюмами;

б) для захисту очей – захисними окулярами;

в) для захисту шкіри від їдких речовин – гумовими або прогумованими рукавицями, гумовими чоботами або шкіряними черевиками, сукняними костюмами.

Враховуючи значний ризик і широке застосування хлору, тяжкість ураження і високу можливість летального наслідку, у кожного повинен бути сформований алгоритм дій і чітка позиція – попередити отруєння легше і доцільніше, ніж лікувати і боротися з його наслідками.

ВИСНОВКИ

Аналіз даних описано з використанням результатів огляду літературних даних. Представлено повний аналіз, який відповідає питанням дослідження. Відповідь на перше запитання дослідження стосовно "ролі заходів із забезпечення якості програмного забезпечення в гнучкій розробці, згідно з існуючою літературою" представлено у відповідних розділах роботи і виявлено, що гнучка розробка програмного забезпечення є продуктивною та корисною для досягнення задоволеності клієнтів у порівнянні зі звичайними методами. Також показано, Agile-природу мають процеси перевірки якості, як-от зосередженість на безперервному тестуванні, тестування після кожного дизайну та зосередження на історіях користувачів і реалізації інтерфейсу. Але всі мають одну головну мету – досягти вищої якості.

Відповідь на друге запитання дослідження про визначення будь-якої конкретної області процесу гнучкої ітераційної культури, яку можна покращити за допомогою заходів із забезпечення якості, запропоновано переглянути роль і завдання професіоналів із забезпечення якості в команді Agile розробників, оскільки є багато проблем, які можна вирішити за допомогою цього рішення. Наприклад, якщо фахівець із забезпечення якості також збирає історії користувачів, щоб він міг краще спілкуватися з клієнтом і давати кращі пропозиції під час збору вимог щодо підтримки якості продукту.

Деякий час через обмеження часу розробники тестують свій власний код, тому ми також пропонуємо тут, щоб у команді був член із контролю якості, щоб виконувати ці завдання, тому є багато фактів щодо запропонованого рішення. Розробку не слід сприймати просто як філософію; не всі організації можуть зібрати свою команду розробників із найкращих людей. Це підкреслює необхідність залучення якісних експертів, щоб вони могли виконувати свою роль, щоб інституціоналізувати культуру SPI в організації.

Третє запитання дослідження стосовно практичного виконання процесів SQA в Agile-проекти показали, що організації працюють із різними

методологіями, як-от плановий і змішаний підхід. Але вони також зазнають впливу та приваблення гнучких методологій через гнучкість Agile методів. Пошук ресурсів є основною проблемою в організаціях. Діяльність із забезпечення якості споживає менше ресурсів у Agile-підході порівняно з іншими.

Документація також є ключовим питанням у розробці програмного забезпечення. Таким чином, фахівці-практики мотивовані підтримувати якість продукту та процесу, а гнучка діяльність SQA дає їм можливість підтримувати її кращим чином.

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.

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.

13. Malik, Ahsan Nawaz, and Kashif Masood. "Software testing process in agile development." (2008).

14. 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.

15. Williams, Laurie, and Alistair Cockburn. "Agile software development: It's about feedback and change." Computer 36.6 (2003): 39-43.

16. Lindstrom, Lowell, and Ron Jeffries. "Extreme programming and agile software development methodologies." IS management handbook. Auerbach Publications, 2003. 531-550.

17. Beck, Kent. "Embracing change with extreme programming." Computer 32.10 (1999): 70-77.

18. Jeffries, Ron. "What is extreme programming." XP magazine 11 (2001).

19. Schwaber, Ken, and Mike Beedle. Agile software development with Scrum. Prentice Hall PTR, 2001.

20. Rosenberg, Doug, Matt Stephens, and Mark Collins-Cope. "Agile development with ICONIX process." New York: Editorial Apress (2005).

21. 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.

22. 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.

23. Mahanti, Aniket. "Challenges in enterprise adoption of agile methods-A survey." Journal of Computing and Information technology 14.3 (2006): 197-206.

24. Fowler, Martin. "The new methodology." Wuhan University Journal of Natural Sciences 6 (2001): 12-24.

25. Boehm, Barry, and Richard Turner. "Using risk to balance agile and plan-driven methods." Computer 36.6 (2003): 57-66.

26. Talby, David, et al. "Agile software testing in a large-scale project." IEEE software 23.4 (2006): 30-37.

27. Parnas, David L., and Mark Lawford. "Inspection's role in software quality assurance." IEEE Software 20.4 (2003): 16.

28. Nerur, Sridhar, RadhaKanta Mahapatra, and George Mangalaraj. "Challenges of migrating to agile methodologies." Communications of the ACM 48.5 (2005): 72-78.

29. McBreen, P., and M. Consulting. "Quality assurance and testing in agile projects." McBreen Consulting (2003).

30. Beck, Kent. "Embracing change with extreme programming." Computer 32.10 (1999): 70-77.

31. Cockburn, Alistair, and Laurie Williams. "The costs and benefits of pair programming." Extreme programming examined 8 (2000): 223-247.

32. Fowler, Martin, and Kent Beck. "Refactoring: Improving the design of existing code." 11th European Conference. Jyväskylä, Finland. 1997.

33. 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.

34. Glazer, Hillel, et al. "CMMI or agile: Why not embrace both! Software Engineering Institute." Software Engineering Institute (2008).

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

36. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин МОЗ України від 10.12.1998 № 7. // Офіційний сайт Верховної Ради України. – [Електронний ресурс]. – Режим доступу https://zakon.rada.gov.ua/rada/show/v0007282-98

37. Бідяк О. Профілактика отруєння хлором. // Офіційний сайт управління держпраці в Тернопільській області. – [Електронний ресурс]. – Режим доступу https://te.dsp.gov.ua/profilaktyka-otruyennya-hlorom/

ДОДАТКИ

Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів
«АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 6-7 грудня 2023 року

УДК 004.41

В. Семенюк, В. Сенківський, В. Чичук, Б. Хоміцький, О. Кучма
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ОГЛЯД ІНСТРУМЕНТІВ БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІЇ В СУЧАСНИХ ПРОЄКТАХ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

V. Semeniuk, V. Senkivskyi, V. Chychuk, B. Khomitskyi, O. Kuchma

OVERVIEW OF CONTINUOUS INTEGRATION TOOLS IN MODERN SOFTWARE DEVELOPMENT PROJECTS

Безперервна інтеграція (Continuous Integration (CI)) – це концепція написання коду, яка змушує розробників програмного забезпечення вносити зміни та постійно переглядати код у сховищах контролю версій. Однією з головних переваг безперервної інтеграції є те, що можна легко та швидко виявляти помилки. Оскільки кожна внесена зміна зазвичай невелика, ви можете швидко визначити конкретну зміну, яка спричинила дефект. Останнім часом CI стала таким стандартним протоколом і набором базових елементів для розробки програмного забезпечення [1].

Гнучкі методології створили постійний безперервний цикл між клієнтами та командами розробників програмного забезпечення в режимі реального часу. Дотримуючись цієї ідеї, DevOps будується на принципі циклу зворотного зв'язку в реальному часі у процесі розробки SDLC, зменшуючи ризики зупинки роботи розробників через велику кількість дефектів, забезпечення якості (QA).

У минулому команда розробників могла працювати самостійно протягом тривалого часу та об'єднувати свої зміни в програмний код лише після завершення основної гілки. Це ускладнює злиття коду, а також дозволяє накопичувати помилки без виправлення протягом тривалого часу [2]. Такі фактори не сприяли швидкому наданню клієнтам оновлень.

Розглянемо основні інструменти неперервної інтеграції.

Jenkins – одне з найпоширеніших програмних безкоштовних рішень CI з відкритим кодом. Це хмарний сервіс CI, написаний на Java, який працює на веб-сервері. Тисячі користувачів у всьому світі використовують Jenkins, оскільки він дає змогу швидко створювати та виконувати автоматизовані тести. Основні властивості:

- Безкоштовне.
- Налаштування робочого процесу.
- Велика кількість плагінів.
- Легка інсталяція для основних операційних.
- Орієнтовано на розробників.
- Добре відома та авторитетна компанія.

TeamCity – це універсальне бізнес-рішення CI, яке можна використовувати безкоштовно при кількості проєктів до 100. Можна запускати паралельні конструкції за допомогою TeamCity одночасно, використовувати мітки тощо. TeamCity легко встановити завдяки зручному інтерфейсу [3]. Основні властивості:

- Безкоштовно до 100 проєктів.
- Три потоки з трьома агентами збирання коду одночасно.
- Можна імпортувати вихідний код з двох різних VCS в одній компіляції.
- Можливість замінити тестувальників програмними агентами.
- Дозволяє перевіряти зміни без фіксації VCS.

Bamboo є продуктом від Atlassian і має швидкий і ефективний графічний інтерфейс користувача. Цей інструмент популярний серед розробників, які використовують інші інструменти Atlass [4]. Bamboo дозволяє створювати та об'єднувати нові гілки після автоматичного тестування. Основні властивості:

- Ефективна інтеграція з іншими інструментами Atlassian.
- Хороший спосіб надання сповіщень.
- Просте управління масштабуванням CI компанії.
- Автоматизація тестування.
- Автоматичне розпізнавання окремих збірок.

Buddy – це інструмент автоматизації DevOps для постійної інтеграції та розгортання. Цей інструмент був розроблений для роботи з проєктами на основі коду репозиторію Bitbucket і GitHub [5]. Buddy – це бізнес-інструмент із простим і легким у використанні інтерфейсом і оптимізованим дизайном. Служба, орієнтована на клієнта, підтримується 24 години на добу без вихідних і може бути встановлена на пристрій у версії клієнта. Основні властивості:

- Інтуїтивний інтерфейс користувача.
- Інтуїтивно зрозумілий дизайн процесу розгортання.
- Підтримка докерів.
- Доступні попередні налаштування та підказки.
- Забезпечує складну автоматизацію та потребує фундаментальних знань.
- Можливість модифікувати розроблений код.
- Клонування, змінна та універсальна автоматизація приміток.

GitLab CI – це продукт із відкритим кодом і безкоштовний інструмент постійної інтеграції. API GitLab має високий ступінь масштабування, його легко встановити та налаштувати для проєктів, розміщених на GitLab. Окрім тестування та створення проєктів, GitLab CI може використовуватись, де процес розробки потребує вдосконалення. Розробники GitLab вибирають індивідуальний GitLab CI, не замислюючись, оскільки безперервна інтеграція проєкту досягається автоматично [6].

Варто звернути увагу на такі властивості цього продукту:

- Підтримка Docker.
- Конфігурація сервера швидкої збірки.
- Працює на кількох машинах одночасно.
- Сильна інтеграція продукту можлива за допомогою API.
- Опція захисту конфіденційних даних проєкту.

Коли компанія практикує CI, уся її робота регулярно інтегрується в основну вітку коду (розпізнається як магістральна або головна). Дослідження показали, що робота компанії покращується, коли розробники мають можливість об'єднувати свій програмний код з основною віткою. Перед фактичним злиттям проводиться серія автоматизованих перевірок, щоб переконатися, чи не виникають помилки регресії [7]. Якщо ці програмні продукти містять дефекти, команда зазвичай зупиняється, щоб виправити помилки.

Усі подальші процеси повинні використовувати пакети, створені збіркою CI. Ці побудови мають бути чисельними та відтворюваними. Принаймні раз на день потрібно успішно запускати процес складання проєкту з виконанням набору автоматичного тестування. Починають із написання багатьох тестів, які охоплюють пріоритетну з точки зору якості функціональність системи. Після цього перевіряють усі нові функції. Ці тести повинні бути проведені швидко для отримання оперативного зворотного зв'язку від розробників. Принаймні один раз на день тести повинні пройти успішно. Зрештою, розробники отримуватимуть інформацію щоденно, якщо тести пройдуть

успішно та код буде злитий з основною віткою. Система CI, яка виконує автоматичне тестування, також повинна візуалізувати статус команди. Не рекомендується використовувати повідомлення електронною поштою; багато людей ігнорують сповіщення електронною поштою або створюють фільтр, який приховує повідомлення. Системні сповіщення чату є кращим і популярнішим способом досягнути цього. Безперервна інтеграція часто включає додаткові дії, які також передбачають вищу продуктивність розробки програмного забезпечення.

Стиль побудови, зосереджений на основній вітці коду, де розробники будують невеликі ділянки та об'єднують свої завдання в окрему вітку принаймні щодня, а не на довготривалих вітках додатків. Для CI потрібне автоматизоване модульне тестування. Ці тести мають бути достатньо всебічними, щоб гарантувати належне функціонування програмного забезпечення. Тести також мають тривати кілька хвилин або менше. Якщо автоматизоване модульне тестування триває довше, розробники не хочуть запускати його часто. Якщо тести виконуються рідко, результати багатьох різних змін можуть ускладнити локалізацію помилок та відладку. Тести, які проводяться рідко, важко підтримувати.

Складно створити супроводжувані пакети модульних тестів. Хорошим вирішенням цієї проблеми є практика розробки, керованої тестуванням (TDD). TDD надає багато переваг: одна полягає в тому, що розробники пишуть гнучкий, зручний для тестування код, який, як наслідок, зменшує витрати на обслуговування автоматизованих наборів тестів. Багато компаній не мають пакетів модульних тестів, які можна підтримувати, і все ще не практикують TDD.

Таким чином, CI гарантує безперервну роботу команди проєкту. Впровадження CI дає більшу швидкість розгортання, надійніші системи та якісніші програми. Перевага CI є значною. Останні дослідження підтверджують це твердження, допомагаючи підкреслити зв'язки між створенням, проєктуванням і впровадженням програмного забезпечення. Постійна інтеграція в проєкти допомагає знизити ризики організації роботи команди.

Література

1. Fowler, Martin, and Matthew Foemmel. "Continuous integration." (2006).

2. Hüttermann, Michael. "Introducing DevOps." DevOps for Developers. Berkeley, CA: Apress, 2012. 15-31.

3. Melymuka, Volodymyr. TeamCity 7 continuous integration essentials. Packt Publishing, 2012.

4. Brechner, Eric. Agile project management with Kanban. Pearson Education, 2015.

5. Vanbrabant, Bart, Thomas Delaet, and Wouter Joosen. "Authorizing and directing configuration updates in contemporary IT infrastructures." Proceedings of the 3rd ACM workshop on Assurable and usable security configuration. 2010.

6. Arefeen, Mohammed Shamsul, and Michael Schiller. "Continuous Integration Using Gitlab." Undergraduate Research in Natural and Clinical Science and Technology Journal 3 (2019): 1-6.

7. Senapathi, Mali, Jim Buchan, and Hady Osman. "DevOps capabilities, practices, and challenges: Insights from a case study." Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018. 2018.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ
2023

УДК 004.41

Володимир Семенюк, Андрій Вивюркка, Олександр Нога, Богдан Хоміцький, Олександр Кучма
Тернопільський національний технічний університет імені Івана Пулюя

ПОРІВНЯЛЬНЕ ОЦІНЮВАННЯ ПРОГРАМНИХ АРХІТЕКТУР

Volodymyr Semeniuk, Andrii Vyviurka, Oleksandr Noha, Bohdan Khomitskyi, Oleksandr Kuchma

COMPARATIVE EVALUATION OF SOFTWARE ARCHITECTURES

Якість архітектурного рішення оцінюється за сукупністю критеріїв, і його вибір завжди є компромісом, оскільки для заданого набору функціональних вимог та вимог якості не існує єдиного найкращого рішення і покращення одних показників веде до погіршення інших та навпаки. Тому при виборі архітектурного рішення для ПС необхідно використовувати методи багатокритеріального оцінювання з врахуванням конфліктів між показниками якості та пошуком компромісів.

Існує раннє і пізнє оцінювання архітектур. Раннє оцінювання базується на досвіді розробників та логічному обгрунтуванні, оскільки відсутні артефакти, які дають змогу імітувати роботу ПС. Методи, які реалізують раннє оцінювання, базуються на сценаріях. До цих методів належать наступні: SAAM і ATAM [1].На основі пріоритетів зацікавлених сторін визначаються критерії якості. Для перевірки задоволення кожного атрибута якості розробляється сценарій і проводиться оцінка рівня задоволення даного атрибуту варіантом архітектури.

Метод ATAM подібний до SAAM, але в ньому на основі аналізу сценаріїв для відібраних архітектур проводиться оцінка ризиків задоволення атрибутів якості. Оцінку ризиків проводить група експертів, яка також ранжує альтернативні варіанти за рівнем ризику і визначає так звані точки чутливості у компонентах чи зв'язках архітектури, також аналізуються компроміси між критеріями якості.

Для обгрунтованого вибору рішення в методі SAAM/ATAM вибрані альтернативні архітектури аналізуються на ефективність витрат методом CBAM [2]. Цей метод забезпечує економічний аналіз ПС, яка базується на вибраних в попередніх методах варіантах архітектури та сценаріях моделювання. Експерти призначають оцінки критеріям якості в балах від 1 до 100 і ранжують архітектури за значенням, яке ці архітектурні рішення забезпечують для атрибуту якості. Оцінка кожного варіанта архітектури обчислюється за формулою:

$$B(A_i) = \sum_{j=\overline{1,K}} \left(Cont_{i,j} \cdot Q_j\right) \; i = \overline{1,n}. \qquad (1)$$

Тут $Cont_{ij}$ – вага і-ї архітектури відносно j-го атрибута;

Q_j – пріоритет j-го атрибута.

Метод забезпечує оцінку витрат на реалізацію кожної альтернативи і дає можливість обчислити показник бажаності як відношення прибутку до витрат. На основі отриманих даних проводиться вибір кращого рішення.

Спільним недоліком розглянутих методів є послідовне оцінювання архітектури по одному критерію якості що викликає необхідність кожного разу розробляти новий сценарій і проводити експертне оцінювання ризиків, що робить процес вибору трудомістким і неформалізованим. Тут також неможливо отримати порівняльні оцінки для множини альтернатив.

Подальші дослідження в цій області показали ефективність методу аналізу ієрархій при рішенні цих задач. Однією з перших публікацій по застосуванню MAI до

оцінювання архітектур ПС по множині показників якості є робота [2]. В ній приведений покроковий алгоритм вирішення задачі оцінювання множини альтернативних архітектур по сукупності показників якості та вибору найкращої архітектури ПС. Застосування методу продемонстроване на конкретному прикладі.

Для подолання недоліку MAI, пов'язаного з неузгодженістю матриці парних порівнянь при великій кількості альтернатив (n≥7) в роботі пропонується коригувати елементи матриці. А це приводить до неповного використання та перекручування експертних даних, що зменшує правдивість отриманих результатів.. В роботі [3] для коректного використання MAI у випадку великої кількості альтернатив (n>9) було використано модифікований метод аналізу ієрархій.

В методі MAI використовується порівняльне оцінювання альтернатив по їх реалізації атрибутів якості. Він дає змогу визначити відносні ваги альтернатив по кожному атрибуту якості і проранжувати їх. За призначеними, зацікавленими сторонами, пріоритетами атрибутів якості обчислюється їх усереднене значення і визначаються ваги альтернатив відносно сукупності атрибутів якості.

Отримані відносні оцінки альтернатив можуть використовуватись для аналізу конфліктів між атрибутами якості і пошуку компромісного рішення.

Пропонований підхід базується на використанні методу аналізу ієрархій з оптимізаційним алгоритмом визначення ваг альтернатив, що дає змогу розширити межі застосування методу на більшу кількість порівнюваних альтернатив (n>9). Також слід дослідити чутливість рішення до зміни пріоритетів критеріїв якості і проаналізувати конфлікти і компроміси між критеріями якості.

Для розширення меж коректного застосування MAI слід застосувати оптимізаційний метод обчислення (визначення) ваг альтернатив, який базується на моделі мінімізації неузгодженості матриці парних порівнянь [3].

Література

1. Kazman, R. ATAM: Method for Architecture Evaluation [Text] / Rick Kazman, Mark Klein, Paul Clements. – Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, August 2000. – CMU/SEI-2000-TR-004, ADA377385. – 83 p

2. Kazman, R. Quantifying the costs and benefits of architectural decision [Text]/ Kazman, R., Asundi, J., and Klein // Proceedings of the 23rd International Conference on Software Engineering (ICSE), 2001. – pp. 297–306

3. Harchenko Alexandr, Bodnarchuk Ihor, Halay Iryna. Stability of the Solutions of the Optimization Problem of Software Systems Architecture // Proceeding of VIIth International Scientific and Technical Conference CSIT 2012. pp. 47–48, Lviv, 2012