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: Огляд проблем задачі управління вимогами у великих Agile-проєктах

Виконав(ла): студент(ка) 6 курсу, групи СНм-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

Чичук В.І,
(підпис) (прізвище та ініціали)

Керівник Боднарчук І.О.
(підпис) (прізвище та ініціали)

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

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

Рецензент
(підпис) (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)
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

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

студенту Чичук Володимирович Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи **Огляд проблем управління вимогами у великих Agile-проєктах**

Керівник роботи **к.т.н., доц. Боднарчук І.О.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*24*» *листопада* 2023 року № *4/7-1099*

2. Термін подання студентом завершеної роботи 26 грудня 2023 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП 1 ЗАДАЧІ ІНЖЕНЕРІЇ ВИМОГ В AGILE-ПРОЄКТАХ 1.1 Означення великих розподілених проєктів та задачі дослідження 1.2 Інтеграція інженерії вимог програмного забезпечення в Agile-проєкти 1.3 Проблеми масштабування Agile 1.4 Методи інженерії вимог для Agile проєктів 2 ОСОБЛИВОСТІ РЕАЛВЗАЦІЇ ПРОЦЕСУ ІНЖЕНЕРІЇ ВИМОГ ДЛЯ КОМПАНІЙ РІЗНОГО РОЗМІРУ 2.1 Аналіз причин виникнення проблем розробки вимог для Agile проєктів великого масштабу 2.2 Виклики в галузі інженерії вимог великомасштабних програмних проєктів 2.3 Аналіз механізмів виникнення проблем інженерії вимог 3 РЕКОМЕНДАЦІЇ ЩОДО ВДОСКОНАЛЕННЯ ПРОЦЕСУ ІНЖЕНЕРІЇ ВИМОГ В AGILE-ПРОЄКТАХ 3.1 Ведення wiki-сторінки з припущеннями щодо вимог 3.2 Використання декількох резервних копій продукту для включення вимог від різних стейхолдерів 3.3 Резервування частини часу спринта для важливих вимог якості 3.4 Планування спринтів по декількох беклогах 3.5 Створенням підготовчої групи 3.6 Створення команди для роботи з вимогами якості (QR) 3.7 Зіставлення проблем та викликів з відповідними практиками їх вирішення 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 _____ Чичук В.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Боднарчрук І.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Огляд проблем управління вимогами у великих Agile-проєктах" // Кваліфікаційна робота освітнього рівня «Магістр» // Чичук Володимир Ігорович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНм-61 // Тернопіль, 2023 // с. – 46, рис. – 2, табл. – 3, джерел – 22.

Ключові слова: Agile, інженерія вимог, гнучка розробка, управління проєктами

У гнучких методах розробки програмного забезпечення існує невирішена проблема в узгодженні вимог до якості. Хоча професійна спільнота визнає це, наразі не вистачає достатньо досліджень і публікацій, щоб вирішити це питання.

У цьому контексті ця робота здійснює огляд гнучких моделей, зосереджуючись на аспекті розробки вимог. Отримані висновки та результати можуть виявитися надзвичайно корисними для індустрії програмного забезпечення, сприяючи поліпшенню процесу розробки, а також для науковців, які мають намір подальшої роботи у цьому напрямку.

ANNOTATION

“Overview of the requirements management problem in large scale Agile projects” // Master’s degree qualification paper // Chychuk Volodymyr Ihorovych// Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group СНм-61 // Ternopil, 2023 // p. – 46, fig. – 2, tables – 3, references – 22.

Key words: Agile, requirements engineering, adaptive development, project management

The requirements for quality in Agile projects do not seamlessly align with flexible methods. Despite recognition within professional circles, there is currently insufficient empirical research and publication of results on this matter.

This work provides an overview of flexible models concerning requirement development. The conclusions and findings could prove highly beneficial for the software industry, enhancing the development process, and for researchers intending to delve further into this area.

ЗМІСТ

ВСТУП

Актуальність задачі.

Agile – це гнучкий підхід до розробки програмних продуктів, а значний рівень успіху проєктів зробив цей підхід популярним для використання командами розробників різного розміру. Від невеликих (до 5 чоловік), так і для великих розподілених географічно команд з десятками учасників. Ці технології обговорюються також в роботах працівників ТНТУ, наприклад в [1], [2], [3].

За останні роки було опубліковано також багато інших публікацій щодо того, чи підходять гнучкі методи для великомасштабних розподілених проєктів, наприклад [4] [5]. Дослідження конкретних випадків надає все більше доказів того, що фундаментальні положення гнучких методів розробки піддаються сумніву при застосуванні цих методів у великомасштабному розподіленому контексті. Ця робота зосереджена на одному конкретному аспекті гнучкого масштабування, а саме на розробці вимог до якості (Quality Requirements – QR), таких як безпека, зручність використання, надійність, продуктивність.

Відповідно до опублікованих наукових статей, існує ряд проблем, пов’язаних із QR, які могли призвести до провалу проєкту. Однак більшість літератури зосереджено на результатах використання методів QR та бажаному стані проєкту після впровадження конкретної вимоги якості (безпека, зручність використання чи продуктивність), тоді як точні аспекти, які є складними для інженерії вимог не пояснюються детально та залежать від конкретного контексту кожної вимоги. Результати огляду літературних джерел були підставою для проведення дослідження щодо розуміння проблем, з якими стикаються команди в гнучких великомасштабних розподілених проєктах під час розробки QR, і практик, які вони використовують для протидії цим викликам.

Мета роботи.

Метою роботи є дослідження, яке пояснює складні ситуації, з якими стикаються практики, розробляючи вимоги до якості в контексті великомасштабних розподілених гнучких проєктів. Крім того, це дослідження

описує практики, які гнучкі розподілені команди зараз використовують і які можуть сприяти при вирішенні проблем, що виникають в таких проєктах.

Об’єкт дослідження: Процеси управління вимогами до програмного забезпечення за Agile–методологіями.

Предмет дослідження: моделі життєвого циклу проєктів з Agile-методологіями.

Наукова новизна отриманих результатів. Запропоновано методи інженерії вимог для проєктів і команд різних розмірів в Agile проєктах.

Практичне значення отриманих результатів. Ця робота подає огляд проблем інженерії вимог ПЗ з точки зору практиків. Дослідження виявило ряд проблем проблем, класифікованих за п’ятьма категоріями:

1) координація команди і комунікація,

2) забезпечення якості,

3) визначення вимог до якості,

4) концептуальні проблеми,

5) архітектура програмного забезпечення.

Дослідження також розкрило механізми та практики, які можуть пом’якшити вплив цих проблем.

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

1.1 Означення великих розподілених проєктів та задачі дослідження

Для дослідження даної кваліфікаційної роботи введемо деякі означення. Зокрема, Agile-проєкт «розподіленим», якщо він складається з більш ніж однієї команди, а його команди розподілені відповідно до моделей розподілу, а саме команди з декількома сайтами – групи працюють на різних локаціях, але кожна команда є окремим сайтом, а розосереджені команди – команди працюють на різних локаціях, і кожна команда є включає декілька сайтів. Використовуючи це визначення, було виконано огляд літературних джерел з даної предметної області. Практикуючі спеціалісти залучаються з широкого спектру бізнес-процесів (наприклад, банківська справа, громадський транспорт, державні служби тощо) і сфер професійного досвіду (наприклад, тестувальники, архітектори, SCRUM-майстри, менеджери, розробники). Застосовуючи методи аналізу якісних даних було виявлено задачі та проблеми інженерії вимог ПЗ гнучкого великомасштабного розподіленого проєкту та методи протидії їм з точки зору тих, хто працює в цій галузі. Ключові результати дослідження є відповідями на наступні три питання:

1) складання каталогу описів питання QR та на характеристики цих питань;

2) складання переліку базових механізмів, відповідальних за проблеми QR;

3) систематизація методів вирішення, які використовувалися для пом’якшення ефекту викликів.

Це дослідження розширює попередні емпіричні дослідження інженерії вимог програмного забезпечення щодо гнучкої розробки вимог якості кількома способами. По-перше, до емпіричних досліджень RE добавляються контекст гнучких великомасштабних розподілених проєктів, які досі недостатньо розглянуті.

По-друге, поточна робота є спробою дати практичні рекомендації командам, що працюють за Agile-методами. У сфері гнучкої розробки було

запропоновано багато великомасштабних структур (наприклад, DAD [6], SAFe [7] , LeSS [8] і Scrum of Scrums [9]), однак уроки, отримані з їх застосування, рідко публікуються. Крім того, збір доказів у дослідженні QR у гнучкому широкомасштабному розподіленому проєкті сприяє розмові, нещодавно розпочатій у спільнотах архітектури програмного забезпечення та RE щодо міждисциплінарної природи QR.

Як буде показано, отримані результати свідчать про те, що QR можуть мати кілька концептуалізацій з точки зору гнучких команд, які працюють разом у гнучкому великомасштабному розподіленому проєкті. Ця робота показує, що в той час як деякі методи, які були висунуті в запропонованих структурах (наприклад, DAD, SAFe), можуть бути доречними для вирішення деяких проблем QR, інші вимагають переоцінки того, що ми знаємо про QR у гнучкому великому розподіленому проєкті шляхом запозичення теорій з інших дисциплін.

Відповідно до даних, отриманих під час огляду літератури, багато промислових проєктних груп працюють у гнучкому середовищі, щоби забезпечити швидку доставку високоякісного програмного забезпечення як для малих, так і для великих проєктів. Схема запропонованої методики огляду літератури представлена на рисунку 1.1.

1.2 Інтеграція інженерії вимог програмного забезпечення в Agile-проєкти

Agile, фактично, не є методом, а підходом, що базується на маніфесті. Його основна увага зосереджена на 12 принципах та 4 ключових цінностях. Основні цінності Agile маніфесту полягають у створенні працездатного програмного забезпечення, акценті на взаємодії між людьми та інструментами, співпраці з клієнтами та швидкій реакції на зміни. Гнучкі методи розробки відрізняються від традиційних планових підходів тим, що акцентують на продуктивності, а не на жорсткому дотриманні процесів. За даними [10], понад 90% практик гнучкої

методології використовують історії користувачів (user stories) для фіксування вимог.

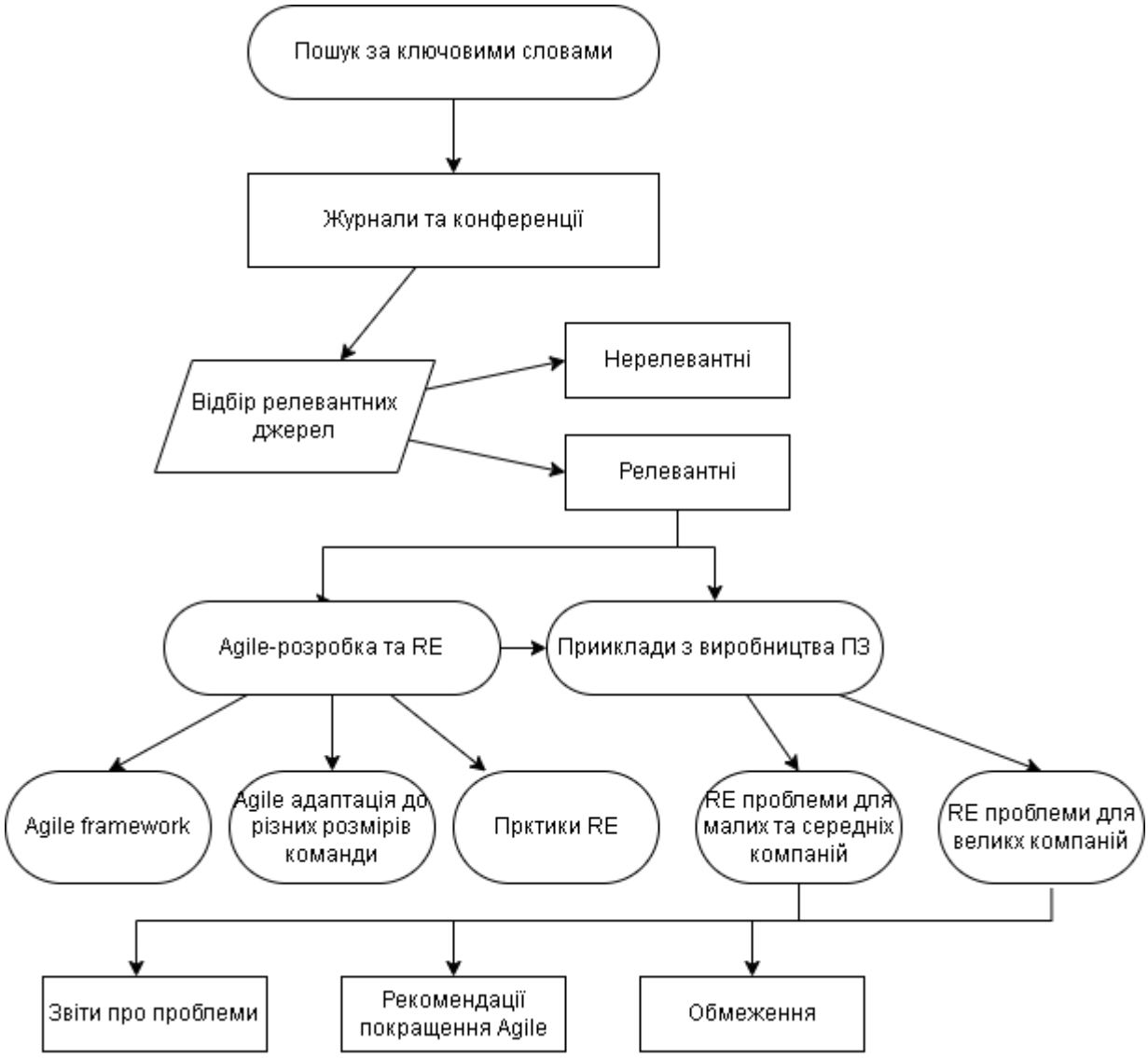

Рисунок 1.1 – Методика запровадження Agile для великих компаній

ASD (Adaptive Software Development) - це процес, що покладає акцент на адаптивність, має обмежену кількість документації, використовує обмежені ресурси та базується на висококваліфікованих та мотивованих членах команди у проєктах малих і середніх розмірів. Застосування гнучких методів дозволяє розробляти надійне та корисне програмне забезпечення через кілька ітерацій. Agile вирішує питання, які виникають у традиційній моделі розробки вимог (RE), оскільки основні проблеми пов'язані з управлінням вимогами та зміною бізнес-

процесів. Гнучка розробка переборює проблеми послідовної моделі, але має деякі обмеження, пов'язані з нею.

Далі наведено огляд адаптивних моделей: FDD (розробка, керована функціями), екстремальне програмування (XP), SCRUM та адаптивне моделювання (AM) з точки зору розробки вимог.

AM (Adaptive Modeling) - це методика, яка надає вказівки, пов'язані з моделями проєктування, і спрямована на мінімізацію документації та моделей. Вона базується на підході до розробки вимог, що ґрунтується на брейнштормінгу, підтримується деякими додатками AM.

FDD (Feature Driven Development) - це процедура, яка складається з п'яти етапів із конкретними критеріями входу та виходу. Вона основана переважно на етапах конструювання та проєктування, списку функцій побудови та плануванні, що ітеративно проєктуються та створюються за кроками функцій, де кожна функція представляє цінність для клієнта. Продукт розробляється шляхом розгляду окремих ознак, а не всього або повного продукту.

XP (Extreme Programming) – це відомий гнучкий метод, який використовується для успішного розвитку програмного забезпечення. XP підтримує легкість, комунікацію, зворотній зв'язок і простоту, незалежно від постійних змін вимог. Звичайні практики XP включають парне програмування, швидкі ітерації з невеликими випусками, швидкий зворотний зв'язок, тісну взаємодію та участь клієнтів. XP фокусується на кодуванні та розробці, а не на зборі вимог, при цьому команда приймає відповідальність за зручність використання продукту зацікавленим сторонам.

SCRUM – це гнучкий та адаптивний підхід до управління проєктами з циклом спринтів тривалістю приблизно 30 днів. Щоденно проводяться короткі (15-хвилинні) зустрічі для координації та інтеграції. У SCRUM розроблений продукт завжди знаходиться в робочому стані, і після кожної ітерації збільшується функціональність продукту.

Такі види діяльності, як виявлення, аналіз, конкретизація та документування, мають велике значення. Пріоритезація вимог у гнучкій

розробці здійснюється за допомогою точок зору. Високоякісні проєкти розробляються за менший час за допомогою гнучких методологій розробки, включаючи ітераційну розробку, збільшення проєкту, адаптацію та співпрацю.

1.3 Проблеми масштабування Agile

Визначення "великого проєкту" може варіюватися в залежності від критеріїв, які використовує конкретна компанія, або за певними абсолютними показниками, такими як обсяг залучених розробників, розмір бюджету, складність або кількість команд розробників. Позитивні результати та успіх застосування Agile в малих проєктах викликали зацікавленість команд розробників програмного забезпечення стосовно впровадження Agile у великих проєктах, оскільки гнучкий підхід швидко витісняє традиційні методи.

Проте прийняття широкомасштабного впровадження Agile у великому проєкті може стати причиною проблем у проєктному менеджменті. Незважаючи на це, зростає швидкість впровадження Agile в компанії великих розмірів для великомасштабних проєктів [11]. Дослідження свідчить про успішність впровадження Agile методології у проєктах всіх розмірів [12]. Результати впровадження Agile в проєктах різного рівня представлено на рис. 1.2.

Застосування Agile на великий масштаб проєктів може мати свої обмеження у певних ситуаціях. Наприклад, SCRUM, який часто використовується для невеликих проєктів з невеликою кількістю учасників, може стати менш ефективним, коли застосовується в масштабних проєктах.

У дослідженні [13] було визначено різні причини для впровадження Agile саме для великомасштабних проєктів. Окрім цього, їхній аналіз та класифікація, зокрема, за критичними факторами, були здійснені на основі певних критеріїв. Виявлені мотиватори були підтверджені через опитування, анкетування та емпіричні дослідження в присутності та за допомогою практикуючих фахівців з розробки програмного забезпечення.

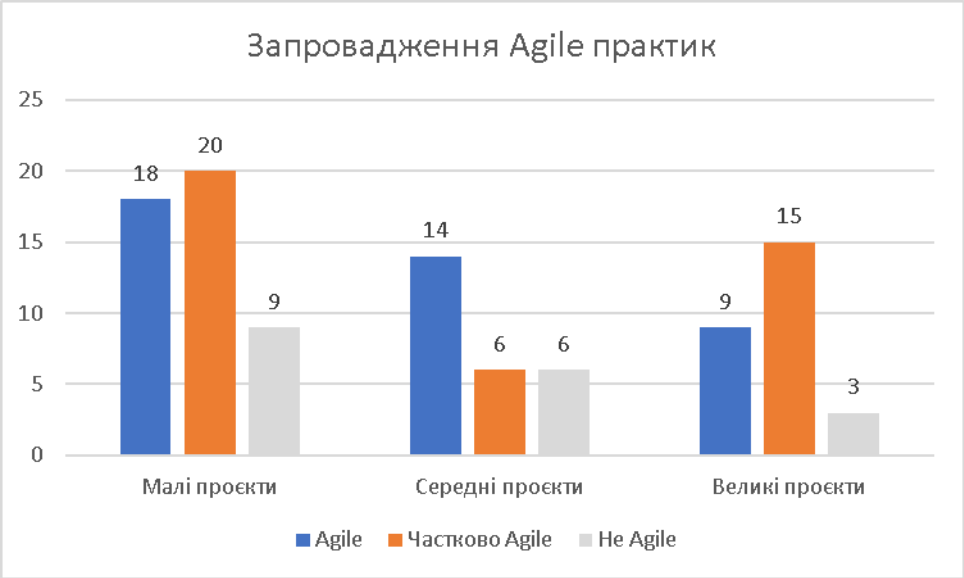

Рисунок 1.2 – Запровадження Agile у різних проєктах

Отже, вищезгадані факти підтверджують та прокладають шлях до широкомасштабного впровадження Agile у програмних проєктах. Однак, варто враховувати, що при впровадженні Agile саме для великих проєктів можуть виникнути певні виклики, які потребують уважного аналізу та адаптації методів для кращої ефективності.

1.4 Методи інженерії вимог для Agile проєктів

Згідно [14], розробка вимог для гнучких технологій створення ПЗ виконується упродовж всього циклу розробки, але таке виконання має особливості. Порівняно з традиційними методами тут вимоги розробляються невеликими порціями для кожної ітерації. Наведемо список основних практик інженерії вимог ПЗ для Agile-проєктів:

– комунікація між усіма стейкхолдерами є основною запорукою успіху проєкту;

– на успіх проєкту впливає пріоритезація вимог, тобто черговість їх формулювання та, відповідно, виконання;

Таблиця 1.1 – Перелік проблем при роботі з вимогами в Agile-проєктах.

Етап розробки	**Відповідна практика Agile**	**Проблема (-и)**	**Вплив проблеми**
Виявлення	Інтерв'ю та анкетування	Розмір задачі, неоднозначність формулювань	Неправильно сформульовані вимоги
Виявлення	Мозковий штурм	Мозковий штурм для великої групи осіб	Багато неоднозначностей
Виявлення	Прототипування	Надійність, масштабованість і безпека	Проблема з супроводом продукту
Аналіз	Пріоритезація	Конфлікти від стейкхолдерів	Невизначеність
Документування	User Stories (історії користувачів)	Нерелевантні user stories, відсутність відгуку замовника	Неправильна інформація для розробників
Документування	Беклог	Недостача документації	Втрата інформації про продукт
Перевірка вимог	Відгуки замовника	Відсутність інструментів для перевірки	Невідповідна якість продукту
Менеджмент	Контроль версій та управління змінами	Вибір правильного інструменту	Перевитрата часу
Менеджмент	Трасування	Немає ефективного відстеження вимог та їх реалізації (трасування)	Відсутність трасування, проблеми з локалізацією помилок

– Agile-методологія вимагає неперервної роботи з управління вимогами в розумінні виявлення, внесення змін. Ці дії виконуються на основі спільних концепцій для всього Agile-проєкту;

– виготовлення прототипу є типовим для Agile-проєкту. Він виготовляється для отримання зворотного зв'язку від замовника і з метою уточнення вимог та їх пріоритезації.

Для підходу Agile властивими є раннє тестування продукту, рефакторинг коду, парне програмування, випуск невеликих ітерацій та активна участь замовника. Тоді імовірність успішного завершення проєкту значно виростає. Крім цього, розбиття великих завдань на менші, написання лише основних тестів, ефективна комунікація, застосування об'єктно-орієнтованого проєктування та використання передових інструментів для швидкого циклу розробки вважаються важливими компонентами гнучкої методології розробки.

Загальні виклики, з якими стикаються в Agiel-проєктах при роботі з вимогами до ПЗ відображені в таблиці 1.1.

1. Загалом на цьому етапі можна сформулювати наступні проблеми RE (Requirements Engineering) для великомасштабних Agile-проєктів.
2. Мінімальна документація та труднощі з тим, щоб команди задокументували вимогу.
3. Недоступність клієнтів для обговорення вимог, роз’яснення та зворотного зв’язку.
4. Невідповідна архітектура, коли виникають нові вимоги.
5. Труднощі з оцінкою трудомісткості і плануванням на основі вимог.
6. Пріоритезація вимог на основі одного виміру.
7. Слабкі практики забезпечення якості, пов’язані з вимогами (наприклад, тестування QR, автоматизовані та інтеграційні тести вимог).
8. Важко виявити залежності між підсистемами на ранній стадії.
9. Нехтування QR.
10. Відсутність знань домену клієнта під час створення історій користувачів.
11. Відсутність чіткої картини вимог на ранніх етапах циклу розробки та зосередження на загальній картині.
12. Забезпечення достатніх компетенцій у командах.
13. Труднощі з включенням інноваційних ідей від команд під час написання історії користувача.

14. Використання коду прототипу у виробництві для підтримки перевірки вимог.

15. Труднощі з процесом уточнення вимог.

16. Керування вимогами високого рівня значною мірою відсутнє в гнучкості.

17. Труднощі з документуванням QR таким чином, щоб виявляти їх залежності.

18. Залучення всіх зацікавлених сторін до всього проєкту Agile.

19. Неправильне розуміння вимог і те, що розробники приймають рішення на основі свого досвіду або вгадують вимоги.

2 ОСОБЛИВОСТІ РЕАЛВЗАЦІЇ ПРОЦЕСУ ІНЖЕНЕРІЇ ВИМОГ ДЛЯ КОМПАНІЙ РІЗНОГО РОЗМІРУ

2.1 Аналіз причин виникнення проблем розробки вимог для Agile проєктів великого масштабу

З гнучкою розробкою пов'язані переваги порівняно з послідовною моделлю розробки, але все ж таки вона має свої недоліки. Проблеми, що виникають у процесі інженерії вимог, включають несумісний інтерфейс, недооцінку нефункціональних вимог та відсутність чіткості у формулюванні вимог.

Agile-підходи не мають жодної чіткої структури для адекватного перерахування або документування вимог користувачів. Внаслідок цього зміни вимог часто вимагають великого обсягу роботи, що виконується багатократно.

Іноді зміна вимог замовника між ітераціями може призводити до проблем зі сумісністю інтерфейсів у розробленому продукті. Також відсутність чітких правил або методик для збору та оцінки нефункціональних вимог може впливати на якість та функціональність продукту.

Загалом, проблеми гнучкої розробки можна сформулювати у вигляді наступного списку:

– Несумісний інтерфейс.
– Недооцінка нефункціональних вимог.
– Відсутність чіткості у формулюванні вимог.
– Відсутність чіткої структури для документування вимог.
– Великий обсяг роботи при зміні вимог.
– Проблеми сумісності інтерфейсів через зміну вимог.
– Відсутність чітких правил для збору та оцінки нефункціональних вимог.

Відсутність оформленої належним чином документації може створювати проблеми для команди, тому важливо призначити в команді фахівців для

створення мінімального обсягу документації, використовуючи відповідні інструменти для моделювання та розробки методів реінжинірингу. Недостатня пряма комунікація та відсутня документація можуть викликати проблеми з відстеженням вимог, обмежити зворотний зв'язок та взаємодію з клієнтом.

Обробка змін вимог може вимагати більше роботи та виходу за рамки планового графіка, а в той же час творчі та інноваційні вимоги в гнучкій системі можуть створювати труднощі через відсутність уваги до нефункціональних вимог.

У випадку використання SCRUM виникають проблеми з відсутністю участі команди в початковій оцінці проєкту, впливом неоднозначних вимог на якість та графік робіт, а також у мінімальній кількості документації. Різні рівні абстракції та неузгодженість історій користувачів у беклозі продукту після призначення власника продукту можуть ускладнити розуміння вимог.

В результаті змін вимог гнучкі проєкти можуть стати витратними. Недостатня документація в процесі збору вимог може ускладнити супровід та не надасть достатньої підтримки методам зворотного проєктування. Нарешті, відсутність документації може спричинити проблеми при передачі проєкту на обслуговування, при зміні персоналу або при роботі з новою командою.

У гнучкому підході до інженерії вимог (RE) увага більше акцентується на перевірці вимог, але в Agile відсутня формальна модель для детальної перевірки вимог. При визначенні пріоритетів вимог основним критерієм є бізнес-цінність, оскільки швидкість введення на ринок вважається ключовим фактором, який може в подальшому створити серйозні проблеми.

У гнучкій RE використання прототипування може призводити до неправильного розуміння зацікавленими сторонами щодо того, як швидко може бути розроблений продукт. Занадто інтенсивне застосування прототипів на початкових етапах розробки може призвести до проблем з якістю, спричинених використанням коду з прототипів у наступних етапах. Крім того, це може вплинути на прийняття зацікавленими сторонами інших, більш масштабованих та надійних циклів розробки.

Проблеми RE також включають в себе недостатню чіткість у комунікації та обміні інформацією, обмежені знання домену та зменшену кількість документації, що стосується вимог до програмного забезпечення.

Головна причина полягає в тому, що замовник та команда розробників більше не можуть проводити щоденні особисті зустрічі при великому розмірі проєкту. Брак регулярного спілкування ускладнює ситуацію, тому мінімальні специфікації стають джерелом багатьох проблем. Наприклад, це призводить до труднощів у команди забезпечити високу якість продукту та може призвести до втрати знань про предметну область або продукт.

За викладеним у джерелі [15], модель гнучкої розробки SCRUM, використана в веб-проєктах малого та середнього розміру в одній компанії, стикалася з деякими викликами і проблемами. Одна з проблем полягала в тому, що частина складного програмного коду вимагала додаткових пояснень і змін, але деякі розробники стверджували, що можуть зрозуміти цей код без необхідності в документації.

Розробники висловлювали питання стосовно важливості детальної специфікації вимог, оскільки в команді відсутні загальні знання про продукт. Це могло створити проблему, якщо особа, що володіє всією необхідною інформацією, залишить компанію, і інший член команди не зможе замінити її.

Це свідчить про потребу в більш ефективній передачі знань та документації для забезпечення доступності важливої інформації в разі втрати ключових членів команди або необхідності спільного розуміння складного програмного коду.

Так, виявлення вимог до продукту може бути не повністю завершеним на початковому етапі проєкту в гнучких методологіях розробки. Це може призвести до відсутності чіткого інтерфейсу між вимогами, а це, у свою чергу, може призвести до необхідності додаткових змін та переробок у наступних ітераціях розробки продукту.

Зазначені проблеми включають відсутність активності з інженерії вимог (RE) та недостатню увагу до нефункціональних вимог у процесі розробки

продукту. Недолік уваги до нефункціональних вимог може відігравати важливу роль у підвищенні загальної якості продукту на кожній ітерації розробки. Якщо нефункціональні вимоги не приділяються належної уваги, це може призвести до проблем у сферах безпеки та масштабованості, що значно позначиться на остаточній версії продукту.

При пріоритезації вимог важливу роль відіграють користувачі, які вносять свої відомості через історії збирання проєкту (засобами CI/CD, наприклад), які містять важливі технічні деталі та створюються у співпраці з користувачами. Але нездатність користувачів точно виразити свої потреби через вплив зовнішніх чинників, таких як влада, особисті фактори або обмежена представництво в їхній групі, може створювати проблеми у точному визначенні вимог і утруднювати розробку продукту.

У гнучкому підході Agile, участь клієнтів у прийнятті рішень відсутня, що може створювати проблеми у процесі пріоритезації вимог. Визначення пріоритетів вимог після кожної ітерації ускладнюється відсутністю конкретних методик, особливо коли кожен аспект вимоги потребує окремого уваги.

У зусиллях підвищити швидкість та знизити вартість розробки програмного продукту, компанії, що спеціалізуються на програмному забезпеченні, активно застосовують гнучкі методи в розподілених середовищах, відомих як розподілена розробка програмного забезпечення (DSD). Дослідження, присвячене цій темі, зосереджується на гнучких практиках у розподілених середовищах та визначенні передбачуваних викликів та їхнього пріоритету у контексті гнучкого розвитку у такому оточенні.

В ході огляду літератури були виявлені чотири категорії проблем: командні, процесні, пов'язані з сучасною технологією та менеджментом. Здійснено порівняльний аналіз завдань двох досліджень, а також встановлено пріоритетність завдань в межах цих чотирьох категорій.

2.2 Виклики в галузі інженерії вимог великомасштабних програмних проєктів

Гнучкі стратегії стали популярними у програмній індустрії завдяки їхнім значним перевагам. У складних проєктах розробки програмного забезпечення, таких як управління поставками, де вимоги можуть бути нестабільними та невизначеними, використання передбачуваних моделей процесу може ускладнювати розробку. Гнучкі стратегії, як адаптивний підхід, можуть бути корисними в таких ситуаціях, однак їх використання у великомасштабних проєктах розробки програмного забезпечення може створити проблеми в інших аспектах.

Одна з відомих проблем, пов'язаних з інженерією вимог (RE), при використанні гнучких практик у великомасштабних проєктах програмного забезпечення, це велика тривалість процесу через складнощі у прийнятті рішень. Також ускладнюється створення та підтримка списку пріоритетів вимог, а також час очікування на уточнення вимог.

Це вказує на те, що в ході застосування гнучких стратегій до розробки великих проєктів зі створення програмного забезпечення, особливо в складних проєктах з нестабільними вимогами, може виникати ряд викликів і проблем, пов'язаних з інженерією вимог, які вимагатимуть уваги та пошуку ефективних рішень.

Проблеми, які виникають у зв'язку з інженерією вимог (RE), великими проєктами та використанням гнучких методів, можуть бути значними. Деякі з основних проблем включають:

1. Брак високорівневого управління вимогами: Великі проєкти потребують високорівневого керування вимогами для забезпечення відповідності процесів розробки вимогам бізнесу та стратегічним цілям компанії.

2. Суперечливість та уточнення вимог: У складних проєктах уточнення вимог може бути складним через їхню неоднозначність та змінюваність.

3. Розрив між коротко- та довгостроковим плануванням: Зазвичай гнучкі методи працюють на коротких ітераціях, але у великих проєктах необхідне також довгострокове планування, що може створювати проблеми у взаємодії цих двох рівнів планування.

4. Складність створення та оцінювання User Stories: Великі проєкти можуть мати значну кількість User Stories, що ускладнює їх створення та оцінювання.

У великих гнучких проєктах, які використовують SCRUM, проблемами може бути масштабування ролі Product Owner (замовника, власника), оскільки потрібно забезпечити переклад історій користувачів у точні вимоги для команд розробників, які можуть мати складні взаємозалежності та розподіл цих вимог між різними командами розробників. Одним із ключових аспектів у вирішенні цих проблем є створення чіткої комунікаційної системи та встановлення високих стандартів для управління вимогами та їх уточненням у великих гнучких проєктах.

Так, в гнучкій розробці інженерія вимог (RE) відрізняється від традиційних моделей тим, що вона не обмежується лише початковою фазою проєкту, але знаходиться в центрі уваги протягом всього життєвого циклу продукту. Гнучкі методології створені для того, щоб дозволити гнучкіше реагувати на зміни вимог, змінні технології та середовище розробки, забезпечуючи швидкі виправлення помилок та швидкі постачання релізів, що відповідають потребам клієнтів.

Проте у великомасштабних гнучких проєктах можуть виникати проблеми координації. Спілкування в таких проєктах може бути складним через велику кількість учасників, відсутність формальних структур та взаємодії. Неформальний спосіб спілкування може призвести до проблем з розумінням, а також до втрати інформації. Крім того, складні та нестабільні вимоги, завдання зі складними взаємозалежностями та технічні проблеми можуть ускладнити процес прийняття рішень та виконання завдань.

Для подолання цих проблем у великомасштабних гнучких проєктах, важливо активно працювати над покращенням комунікаційних каналів, створенням більш структурованих методів спілкування, збільшенням взаємодії між учасниками проєкту та визначенням чітких процесів управління змінами та вимогами.

Яскравим прикладом є робота [16], в яksq порівнюються виклики інженерії вимог в гнучкій розробці систем для великомасштабних проєктів у чотирьох компаніях різних секторів: виробники автомобілів, телекомунікаційна компанія і технологічна компанія.

За цим описом виявлені основні проблеми у гнучкій розробці великомасштабних систем включають:

1. Проблеми з комунікацією та управлінням знаннями. Нестача ефективної комунікації може ускладнювати розуміння вимог і спричиняти помилки в їх реалізації.

2. Проблеми, пов'язані зі сферами знань вимог. Важливість розуміння користувачів та підтримка системи можуть бути складними завданнями при реалізації вимог.

3. Проблеми взаємодії зацікавлених сторін з різних областей. Вимоги з різних сфер можуть бути різними та складними для вирішення, особливо коли стикаються з взаємодією різних зацікавлених сторін.

Великі компанії, відповідно, можуть мати схожі проблеми у гнучкій розробці систем через їхню складну структуру та потребу в пристосуванні поточних технологій та процесів до нових технологічних вимог. Очікування результатів змін та трансформацій є важливим фактором для їхньої прийняття нових методологій, включаючи гнучку розробку систем.

Розглянуті компанії Nokia, ABB, Daimler Chrysler і Motorola використовують досвід, зібраний на основі аналізу, для подальшого поширення у своїх власних внутрішніх проєктах. У цих компаніях детально розглядаються їх сфери інтересів, а також вплив екстремального програмування. Екстремальне

програмування надає можливість дослідження впровадження гнучких методів у великих компаніях через широке його застосування.

У процесі розгляду питань впровадження Agile великі компанії визначили, що проблеми з вимогами виступають як потужні драйвери. Одна з основних складнощів полягає в тому, що вимоги важко розбити на детальні специфікації, щоб передати їх командам розробників на високому рівні. Це може стати перешкодою для ефективної доставки вимог і вплинути на роботу команд у процесі розробки програмного забезпечення.

2.3 Аналіз механізмів виникнення проблем інженерії вимог

У нашому дослідженні багатьох прикладів ми помітили, що гнучкі команди несвідомо роблять припущення щодо здійсненності QR, особливо коли впровадження цих QR залежить від ресурсів клієнта, для якого була створена система. Наприклад, у одному з проєктів гнучкі команди мали впровадити високодоступну систему для загального використання (наприклад, 24/7 днів). Система залежала від збору своїх даних на джерелі даних клієнта, яке було розташоване за брандмауером. Гнучкі команди не знали про проксі-мережевий екран і припустили, що джерело даних клієнта буде доступним для використання. Наприкінці циклу розробки гнучкі команди виявили, що через інші QR (безпека) джерело даних клієнта було доступне лише в робочий час. Потім команди були змушені тимчасово перепроєктувати систему, представивши копію джерела даних клієнта з вищою доступністю. Однак цей спеціальний обхідний шлях спричинив появу вимог узгодженості даних.

Більше того, у багатьох гнучких проєктах виявилося, що нові QR можуть з’являлися під навіть час циклу розробки. Однак ці вимоги залишалися неоднозначними, доки Власник продукту (PO) не визнав необхідність їх впровадження і, отже, не визначив їх чітко та недвозначно. У ситуаціях, коли PO не міг вказати ці QR, гнучкі команди мали продовжити їх впровадження на основі свого рішення.

QR часто пов’язані з конкретними функціональними вимогами (FR) і рідко впливають на весь набір FR для проєкту [17]. У гнучкому контексті нерозпізнані та суперечливі QR можуть потрапити в Backlog Product (PB) [18] з різними пріоритетами через їхні пов’язані FR. Впровадження QR, пов’язаного з FR із вищим пріоритетом, може призвести до обмеження впровадження QR, пов’язаного з FR із нижчим пріоритетом, або взагалі його скасування. У ще одному проєкті гнучкі команди мали дві важливі QR, а саме безпеку та продуктивність.

Вимоги безпеки стосувалися функцій автентифікації та авторизації, які мали високий пріоритет. Тоді як вимоги до продуктивності (наприклад, час відповіді не більше трьох секунд) стосувалися функцій пошуку даних кінцевих користувачів із нижчим пріоритетом. Вимоги безпеки були реалізовані однією гнучкою командою, що призвело до впровадження кількох фільтрів безпеки, через які дані повинні були пройти перед надсиланням кінцевим користувачам. Інша команда мала отримати дані та зробити їх доступними для кінцевих користувачів протягом трьох секунд. Однак до того часу, коли FR, пов’язаний з продуктивністю, отримав вищий пріоритет, неможливо було отримати дані протягом трьох секунд. Зрештою, вимогу до ефективності було скасовано, оскільки її впровадження призвело б до непомірних затрат.

Розробка програмного забезпечення є діяльністю, пов’язаною з рішенням, тоді як виявлення та визначення вимог є діяльністю, пов’язаною з вирішенням конкретної задачі [19]. Гнучка розробка заохочує аналіз і реалізацію вимог Just-In-Time (JIT), що призводить до діяльності з архітектури програмного забезпечення JIT [19]. Таким чином, гнучкі команди повинні постійно шукати відповідні рішення (наприклад, перебудову програмного забезпечення), коли розуміння проблемної області змінюється (наприклад, поява нових вимог). Виявлено, що команди в проведеному дослідженні визначили та погодили загальну архітектуру системи на початку циклу розробки на основі обмеженого знання вимог. Такий підхід може призвести до того, що QR стануть неможливими, коли з’являться потреби в них. У згаданому вище проєкті гнучкі

команди домовилися на початку проєкту про використання архітектури, керованої подіями. Вибір цієї архітектури був мотивований кількістю існуючих систем, з якими нова система повинна спілкуватися. Крім того, команди погодилися зробити події якомога меншими, щоб уникнути накладних витрат на мережу та підвищити продуктивність. На наступних стадіях проєкту потреба в більш складних подіях з більшою кількістю даних зросла, але не могла бути задоволена через архітектурні обмеження. Як обхідний шлях, команди кілька разів запитували базу даних, щоб отримати необхідні дані послідовними невеликими фрагментами, що негативно вплинуло на продуктивність як однієї з найважливіших вимог якості (QR) проєкту.

QR за своєю природою є наскрізними вимогами. Таким чином, правильна реалізація QR може вимагати добре структурованої взаємодії різних компонентів системи (див. рис. 2.1).

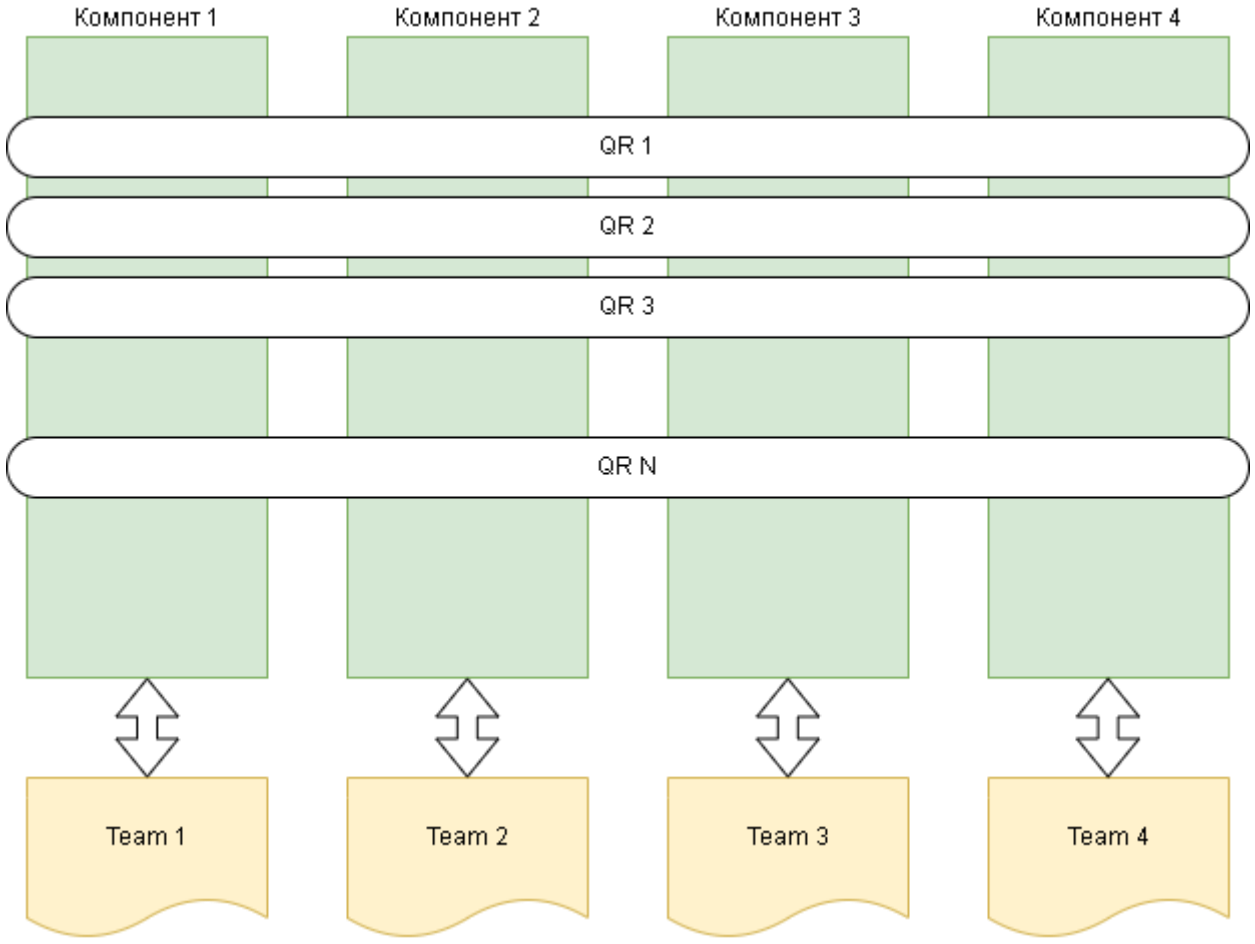

Рисунок 2.1 – Наскрізні вимоги якості (QR)

Гнучкі команди, відповідальні за впровадження конкретних компонентів, повністю володіють відповідними компонентами, але можуть недбало ставитися до загальних характеристик системи.

Гнучка філософія розробки на відміну від традиційної водоспадної моделі заохочує взаємодію з наступними за ходом виконання процесами. Тому гнучким командам необхідно встановити добре структуровані та однозначні канали комунікації, щоб забезпечити правильне впровадження необхідних QR. Зрілість залучених команд має вирішальне значення для встановлення та ведення власних каналів комунікації. [20] визначає дві моделі для координації спілкування між командами:

1. Координація між командами здійснюється вищим органом.
2. Координація між командами здійснюється самими залученими командами.

Перший варіант припускає, що зрілість команд недостатня для ефективної координації комунікації, і тому потрібна вища влада у вигляді менеджерів.

Зрілість команд слід перевірити заздалегідь, щоб визначити відповідну модель координації. У нашому дослідженні багатьох прикладів ми не спостерігали жодного систематичного процесу, який використовується гнучкою організацією для визначення відповідної моделі координації. Спорадичний вибір моделі координації, яка не відповідає рівню зрілості команд, у свою чергу, може призвести до відсутності QR.

Коли мова йде про вимоги, невидимі для кінцевих користувачів, то PO, а також замовник виявляються незацікавленими в них. Це переміщує фокус команд з невидимих QR на видимі вимоги. Крім того, тиск, який організації чинять на гнучкі команди щодо забезпечення функціональності разом із відсутністю інтересу клієнтів до внутрішніх QR, дає гнучким командам право нехтувати внутрішніми QR.

У проведеному дослідженні виявлено, що гнучкі організації, які відповідають як за впровадження програмного забезпечення, так і за його

підтримку, приділяють набагато більше уваги внутрішнім QR, ніж гнучкі організації, які були найняті лише для впровадження бажаної системи.

Щоб забезпечити правильне впровадження QR, команди повинні в якийсь момент перевірити існування бажаних QR. Однак, щоб перевірити виконання QR, їх потрібно однозначно вказати. У гнучких проєктах, де виникають (суперечливі) вимоги, гнучкі команди разом із замовником не можуть чітко визначити обсяг бажаних QR. У цих випадках перевірка задоволеності QR стала справою відчуття, а не вимірювання.

Крім того, необхідно перевірити вплив суперечливих QR, щоб визначити прийнятний рівень задоволення кожного QR для клієнта. Не роблячи цього, гнучкі команди фактично залишають тестувальників у невіданні.

У каскадному підході вимоги (FR та QR) збираються заздалегідь, документуються та передаються розробникам програмного забезпечення для їх реалізації. Розробники не повинні виявляти необхідні вимоги чи перевіряти їх. З іншого боку, Agile не робить жодних відмінностей між видами вимог (FR та QR) і визначає обидва як історії користувача (найбільш використовувана техніка документування). Очікується, що гнучкі розробники зберуть необхідні вимоги, розроблять їх далі під час сеансів особистого спілкування та вчасно запровадять їх. Але деякі гнучкі розробники все ще мали водоспадне мислення. Вони впроваджують історії користувачів, які запитує PO, але не обтяжують себе з’ясуванням того, які QR можуть знадобитися, якщо система не починає виявляти жорсткість.

Результати розділу відображені в таблиці 2.1.

Таблиця 2.1 – Проблеми великих Agile-проєктів та механізми їх виникнення

Проблема	Механізми виникнення
Проблеми координації та комунікації команд	- Впровадження QR на основі невикладених припущень. - Пріоритет, призначений суперечливим QR, виявляється неоптимальним. - Неусвідомлений вибір невідповідної моделі спілкування перешкоджає реалізації QR. - Зосередження на власному компоненті та втрата загальної картини.
Проблеми із забезпеченням якості	- Клієнти не зацікавлені у внутрішніх QR. - Критерії тестування не чітко визначені. - Зосередження на QR певної точки зору та нехтування іншими точками зору. -Перехід до Agile з водоспадним мисленням. - Відсутність зворотного зв'язку щодо стандартів якості. -Ігнорування пріоритетів історій користувача під час ітенратції.
Проблеми виявлення QR	- Клієнти не зацікавлені у внутрішніх QR. - Зосередження на QR певної точки зору та нехтування іншими точками зору. - Проєктне мислення заважає правильному застосуванню QR. - Перехід до спритності з водоспадним мисленням.
Концептуальні проблеми QR	- Реалізація QR на основі невикладених припущень. - Неусвідомлений вибір невідповідної моделі спілкування перешкоджає реалізації QR. -Клієнти не зацікавлені у внутрішніх QR. - Зосередження на QR певної точки зору та нехтування іншими точками зору. - Перехід до спритності з водоспадним мисленням.
Архітектурні проблеми.	- Пріоритет, призначений конфліктним QR, виявляється неоптимальним. - Зосередження на власному компоненті та втрата загальної картини. - QR, що виникають, важко або неможливо реалізувати у вибраній архітектурі. - Прийняття застарілих архітектурних рішень ускладнює впровадження QR нової системи.

Таким чином час перейти до огляду пропозицій щодо вирішення наведеної викликів та проблем управління вимогами для великих Agile-проєктів.

3 РЕКОМЕНДАЦІЇ ЩОДО ВДОСКОНАЛЕННЯ ПРОЦЕСУ ІНЖЕНЕРІЇ ВИМОГ В AGILE-ПРОЄКТАХ

На основі проведеного аналізу проблем інженерії вимог програмного забезпечення для великих проєктів, що виконуються за Agile-технологіями, пропонується декілька механізмів їх вирішення.

3.1 Ведення wiki-сторінки з припущеннями щодо вимог

У ситуаціях, коли PO не може надати ясність щодо специфікації QR, гнучкі команди роблять власні припущення. Ці припущення також робляться, коли команди вважають, що QR є самоочевидним і не вартий перевірки. Ці припущення зібрано на доступній вікі-сторінці та реалізовано разом із FR у різних спринтах.

3.2 Використання декількох резервних копій продукту для включення вимог від різних стейхолдерів

За результатами огляду літератури деякі гнучкі команди чітко визнають, що різні точки зору мають різні інтереси та QR. Тому розподілені команди виявили корисність мати різні думки щодо документування вимог від різних зацікавлених сторін. Таким чином було забезпечено відстеження між кожним QR та його відповідною точкою зору та інтересом зацікавлених сторін. Це було важливо для гнучких команд розробників, щоб передати можливі проблеми QR відповідним зацікавленим сторонам і розподілити ці QR по спринтах на основі пріоритетів, виявлених від відповідних зацікавлених сторін.

3.3 Резервування частини часу спринта для важливих вимог якості

Вимогами якості (QR), які, на думку PO, не мають комерційної цінності, можна знехтувати легше, ніж іншими. Нехтування цими QR може призвести до

негнучкості та складності в обслуговуванні системи. Тоді можна зібрати ці внутрішні QR в іншому беклозі продукту та досягнути згоди з PO про впровадження кожного з елементів з цих QR в кожного спринті, коли дозволяє час.

3.4 Планування спринтів по декількох беклогах

Розподілені гнучкі команди можуть використовувати кілька беклогів продукту (PB – Product Backlog) для включення QR різних точок зору. Таким чином, вони поділяють спринт відповідно до кількості та важливості різних PB. Наприклад, можна вести проєкт по трьох PB, що представляють бізнес-вимоги, архітектурні QR і вимоги до безперервної доставки. Потужність спринту була розподілена між різними PB відповідно (40%, 40% і 20%). Наприклад, якщо гнучка команда має швидкість спринту в 20 сторі-поінтів, тоді спринт матиме 8 сторі-поінтів бізнес-вимог, 8 сторі-поінтів QR, пов’язаних з архітектурою, і 4 сторі-поінти вимог безперервної доставки.

3.5 Створенням підготовчої групи

Команда підготовки – це команда, яка складається зі старших інформаційних аналітиків, старших спеціалістів з архітектури програмного забезпечення та представників бізнесу. Ця команда працює паралельно з іншими розподіленими гнучкими командами та відповідає за розробку PB, підготовку елементів PB до впровадження, визначення загальної архітектури програмного забезпечення і направлення готових елементів із PB до розподілених гнучких команд. Підготовча команда починає з так званого «нульового спринту», збираючи найважливіші вимоги та визначаючи загальну архітектуру. Команда продовжує збирати необхідні вимоги, готуючи їх до впровадження та вдосконалюючи архітектуру програмного забезпечення під час усіх наступних спринтів. Команда розподіляє визначені історії користувачів між розподіленими

командами на основі характеру історій користувачів і доступних наборів навичок у різних командах.

3.6 Створення команди для роботи з вимогами якості (QR)

За результатами огляду літературних джерел виявилось, що гнучкі великомасштабні розподілені проєкти надають привілеї власника на важливі QR командам, які добре знають про цю конкретну QR.

Наприклад, якщо безпека є важливою характеристикою якості для проєкту, то буде створено команду спеціалістів із безпеки, якій буде призначено відповідальність за вимоги безпеки. Ця команда повинна забезпечити впровадження вимог безпеки в розподілених групах. Більше того, у випадках, коли система не відповідає очікуванням клієнта щодо певної QR, для усунення несправності може бути створена спеціальна команда, яка має серйозний досвід у цьому конкретному напрямку.

3.7 Зіставлення проблем та викликів з відповідними практиками їх вирішення

Ми зіставили кожну з практик, про які повідомлялося в цьому розділі вище, із проблемами, про які повідомлялося в попередньому розділі. Таблиця 3.1 узагальнює це відображення. Перший стовпець таблиці представляє повідомлені категорії викликів. Другий стовпець показує поточні гнучкі практики в розподіленому контексті.

Кожна з описаних практик у таблиці 3.1 може (частково) пом’якшити вплив однієї чи кількох із зазначених проблем. Наприклад, для вирішення питань координації використовується практика створення підготовчої групи. Це допомагає координувати співпрацю між розподіленими командами, усуваючи будь-яку неоднозначність, яка могла виникнути через неправильне спілкування команди. Крім того, створення групи підготовки використовується для контролю архітектурних змін і запобігання некерованим архітектурним змінам.

Таблиця 3.1 – Зіставлення практик Agile з викликами великих проєктів

Категорії викликів	Практики
Проблеми координації та комунікації команд	- Підтримуйте вікі-сторінку спільних тверджень. - Створити підготовчу групу.
Проблеми із забезпеченням якості	- Використання засобів автоматичного моніторингу. - Створити команди QR спеціалістів. - Ітерація інновацій та планування (IP).
Проблеми виявлення QR	- Створити команди компонентів. - Створити підготовчу групу. - Резервування частини спринту для важливих QR.
Концептуальні проблеми QR	- Використовуйте автоматизовані засоби моніторингу - Використовуйте кілька резервних копій продуктів, щоб включити вимоги з різних точок зору. - Підтримувати вікі-сторінку припущення. - Розподіл спринтів на основі кількох PB.
Архітектурні виклики	- Створити підготовчу групу. - Створити команди QR спеціалістів. - Ітерація інновацій та планування (IP).

Крім того, вплив повідомлених проблем можна пом’якшити шляхом впровадження однієї чи кількох практик. Наприклад: проблеми категорії Проблеми виявлення QR можна пом’якшити шляхом впровадження іншої практики, а саме:

1) створення команд компонентів для підвищення внутрішньої якості різних компонентів,

2) створення групи підготовки для збору вимог і випуску щоб PO не був єдиним джерелом вимог, і

3) зарезервувати частину спринту, яка буде використовуватися для реалізації важливих QR.

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 [21].

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 [21]. 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 [21] 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) [22].

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

Таким чином виявлено, що спеціалісти-практики сприймають QR як архітектурні вимоги. Вони були стурбовані запізнілим роз'ясненням QR у своїх проєктах і хотіли, щоб QR були ідентифіковані якомога раніше та проаналізовані на предмет їх архітектурного впливу. Це можна розглядати як сигнал до якнайшвидшого з'ясування ролі архітектури в гнучких великомасштабних розподілених проєктах та її рушійних вимог.

Залишається відкритим питання щодо того, чи має сенс у гнучкому великомасштабному розподіленому проєкті визначати архітектурні вимоги (наприклад, QR) наперед. Наші результати змушують нас думати, що це має сенс, оскільки поява цих вимог на просунутій стадії призводить до зміни архітектури програмного забезпечення, що призводить до втрати часу та ресурсів, яких можна уникнути.

Висновки показали зв'язок між викликами QR і тим, як практикуючі спеціалісти застосовують їх в Agile. Гнучка розробка – це безперервний процес навчання, який включає неявні знання. Старші розробники накопичують міцні теоретичні та практичні знання, які дозволяють їм легко зорієнтуватися в динамічному середовищі та допомагають приймати правильні рішення. Молодші розробники, з іншого боку, не мають такого неявного знання. Єдині знання, які вони мають, це ті, які вони отримали в своїх навчальних закладах, які не включають імпровізацію в динамічному середовищі.

Хоча гнучкі практики, які зараз використовуються, можуть пом'якшити проблеми, вони також можуть створити інші проблеми. У прикладі гнучких великомасштабних розподілених проєктів з кількома розподіленими командами організовують команди навколо певних підсистем або компонентів (наприклад, платіжний компонент, компонент реєстрації). Згідно з досвідом учасників, коли певна команда володіє певним компонентом, команда також вживає заходів для захисту якості свого власного компонента. Це пояснюється тим, що технічний дефект, виявлений у певному компоненті, зрештою повертається до

відповідальної групи. Крім того, кожен компонент демонструватиме стабільність і чіткість, оскільки лише одній команді дозволено торкатися коду, що належить цьому конкретному компоненту. Тому команди компонентів можуть бути ефективним пом’якшенням проти низької внутрішньої якості та неоднозначності.

Виявлено також, що гнучкі команди, особливо менш зрілі, все ще потребують ролі вищого керівництва для координації співпраці між розподіленими командами, що повністю суперечить духу гнучкості [19]. Гнучке мислення базується на концепції самоорганізації команди, яка передбачає, що команда здатна сама організувати необхідні дії для виконання необхідної роботи.

У поточному дослідженні встановлено, що гнучкі команди використовують практики, які передають елементи самоорганізації команді вищого керівництва (наприклад, команда підготовки). Це призводить до гнучких команд, які більше не є самоорганізованими. Створення групи підготовки пом’якшує деякі проблеми, наприклад, некеровані архітектурні зміни, координує зв’язок між розподіленими командами та покращує процеси виявлення вимог і специфікації.

Мета цієї роботи полягала в тому, щоб зрозуміти виклики RE, базові механізми та практики вирішення в гнучкому великомасштабному розподіленому проєкті. Однак, як ми бачимо, деякі проблеми, які ми виявили, також поширені в традиційних підходах у великих проєктах. Зокрема, пізнє виявлення нездійсненності QR, проблеми з впровадженням QR у різних компонентах, неадекватна специфікація тесту QR, недогляд джерел QR, недостатня видимість QR та проблема некерованої архітектури, припущення щодо співпраці між командами та неоптимальна організація між командами є загальними як для гнучких, так і для негнучких контекстів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 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.

2. Програмна архітектура в розподілених командах гнучких проєктів / О. Гузеляк, Ю. Шевчук, Б. М. Береженко, Ігор Орестович Боднарчук // Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології“, 7–8 грудня 2022 року. – Т. : ТНТУ, 2022. – С. 110–112.

3. Проєктування архітектури програмних систем в проєктах з гнучкими методами управління / І. Боднарчук, О. Харченко, Б. Хоміцький, Г. Шимчук // Матеріали XXI наукової конференції ТНТУ ім. І. Пулюя, 16-17 травня 2019 року. – Т. : ТНТУ, 2019. – С. 46–48.

4. T. Dingsøyr , N.B. Moe , T.E. Fægri , E.A. Seim , Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation, Empir. Softw. Eng. 23 (1) (2017) .

5. K.H. Rolland , B. Fitzgerald , T. Dingsøyr , K.-J. Stol , Problematizing Agile in the Large: Alternative Assumptions For Large-Scale Agile Development, ICIS, 2016.

6. Ambler, Scott W., and Mark Lines. Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise. IBM press, 2012.

7. Duncan, Scott. "SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering. 2017." The Quality Management Journal 25.1 (2018): 66-66.

8. Larman, Craig, and Bas Vodde. Large-scale scrum: More with LeSS. Addison-Wesley Professional, 2016.

9. Larman, Craig, and Bas Vodde. Practices for scaling lean & Agile development: large, multisite, and offshore product development with large-scale scrum. Pearson Education, 2010.

10. Dalpiaz, F., & Brinkkemper, S. (2018, August). Agile requirements engineering with user stories. In 2018 IEEE 26th International Requirements Engineering Conference (RE) (pp. 506-507). IEEE.

11. Bodnarchuk, I., Lisovyi, V., Kharchenko, O., & Galai, I. (2018, September). Adaptive Method for Assessment and Selection of Software Architecture in Flexible Techniques of Design. In 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 292-297). IEEE.

12. Jørgensen, M. (2018, May). Do agile methods work for large software projects? In International Conference on Agile Software Development (pp. 179-190). Springer, Cham.

13. Abrar, M. F., Khan, M. S., Ali, S., Ali, U., Majeed, M. F., Ali, A., ... & Rasheed, N. (2019). Motivators for large-scale agile adoption from management perspective: A systematic literature review.

14. Kovitz, B. (2003). Hidden skills that support phased and Agile requirements engineering. Requirements Engineering, 8(2), 135-141.

15. Cho, J. J. (2010). An exploratory study on issues and challenges of agile software development with scrum. All Graduate theses and dissertations, 599.

16. Kasauli, R., Liebel, G., Knauss, E., Gopakumar, S., & Kanagwa, B. (2017, September). Requirements engineering challenges in large-scale agile system development. In 2017 IEEE 25th International Requirements Engineering Conference (RE) (pp. 352-361).

17. Kassab, M., Ormandjieva, O., & Daneva, M. (2009, September). An ontology based approach to non-functional requirements conceptualization. In 2009 Fourth International Conference on Software Engineering Advances (pp. 299-308). IEEE.

18. K. Schwaber, J. Sutherland, The Scrum Guide, 2016. http://www.scrumgui des.org/index.html

19. Fægri, T. E., & Moe, N. B. (2015, May). Re-conceptualizing requirements engineering: findings from a large-scale, agile project. In Scientific Workshop Proceedings of the XP2015 (pp. 1-5).

20. Appelo, J. (2011). Management 3.0: leading Agile developers, developing Agile leaders. Pearson Education.

21. 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

22. 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

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 Чичук, Леонід Мариненко, Володимир Сенківський, Богдан Хоміцький, Тимофій Ланевич
Тернопільський національний технічний університет імені Івана Пулюя

ОЦІНЮВАННЯ АЛЬТЕРНАТИВНИХ ПРОГРАМНИХ АРХІТЕКТУР МЕТОДОМ АНАЛІЗУ ІЄРАРХІЙ

Volodymyr Chychuk, Leonid Marynenko, Volodymyr Senkivskyi, Bohdan Khomitskyi, Oleksandr Noha, Tymofii Lanevych

EVALUATION OF ALTERNATIVE SOFTWARE ARCHITECTURES WITH ANALYTIC HIERARCHY PROCESS

З ускладненням програмних систем стає все важче оперативно дотримуватись відповідності вимогам якості на етапі їх створення. Для зменшення цього негативного впливу цей процес переносять на початкові стадії проектування, особливо під час формування архітектури. Архітектура у цьому контексті означає сукупність елементів, які містять в собі основні обчислювальні завдання та забезпечують їх взаємодію, створюючи конфігурацію.

Процес проектування архітектури включає кілька етапів [5]:

1. Визначення вимог до програмного забезпечення, як функціональних, так і якісних, здійснюється на основі аналізу потреб усіх зацікавлених сторін.

2. Вибір альтернативних проектних рішень. На основі вимог створюються різні варіанти проектування, які потім порівнюються для знаходження найбільш відповідного.

3. Аналіз і оцінка проектних рішень. Кожен варіант проектного рішення повинен бути оцінений та порівняний з іншими, з урахуванням їх впливу на виконання якісних аспектів.

4. Загальний аналіз архітектури та прийняття рішення. З урахуванням попередніх етапів архітектор обирає оптимальний варіант, який задовольняє всі вимоги якості. Якщо такого варіанту немає, то досліджуються конфлікти між критеріями якості, шукаються компроміси для вибору оптимального рішення. Розглянемо питання оцінювання якості архітектури по множині показників якості методом аналізу ієрархій (MAI) з використанням оптимізаційного алгоритму визначення ваг альтернатив.

Процес порівнювального оцінювання архітектур з використанням MAI представлено на рис.1.

Вибір архітектури повинен виконуватись таким чином, щоб побудована на її основі програмна система (ПС) задовольняла вимогам якості. Тому тут представлено два рівні взаємопов'язаних критеріїв якості:

– критерії якості ПС у відповідності зі стандартом ISO/IEC 25010;
– критерії якості архітектури;
– альтернативні архітектурні рішення.

Множина критеріїв якості ПС $\{K_i^1\}$, та обмеження на них визначаються при розробці вимог до ПС. А множина критеріїв якості архітектури $\{K_i^2\}$ визначається з використанням методу QFD (Quality Function Deployment, або методу парних порівнянь [1].

Необхідно вибрати з наявних альтернатив таку, яка б найкраще забезпечувала якість ПС, тобто треба вирішити задачу оптимізації за сукупністю критеріїв $\{K_i^1\}$, $\{K_i^2\}$. Це задача багатокритеріальної ієрархічної оптимізації і для її розв'язання будемо використовувати метод аналізу ієрархій Сааті [2].

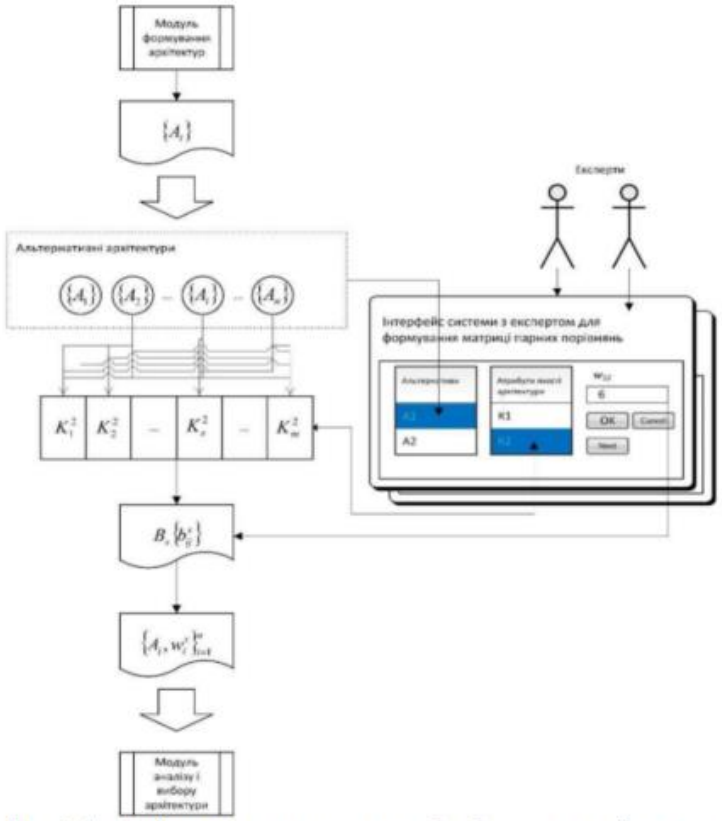

Рис. 1. Ієрархічне представлення задачі оцінювання архітектури

При використанні MAI для рішення таких задач відносні оцінки критеріїв (ваги) $\{\omega_i^S\}$ для альтернатив на кожному рівні знаходяться з використанням матриць парних порівнянь $B^S(b_{ij}^S)$, які заповнюють експерти (тут b_{ij} звизначає перевагу i-тої альтернативи над j-ю по реалізації s-го критерію).

Коефіцієнти матриць повинні бути узгодженими, тобто $b_{ij} = w_i / w_j \quad \forall b_{ij} \in B$. Ваги в цьому випадку знаходяться як компоненти власного вектору матриці парних порівнянь, які відповідають максимальному характеристичному числу матриці. Обчислення власного вектору матриці є досить трудомісткою процедурою. Тому користуються як правило наближеними співвідношеннями [2]

$$w_i = \frac{1}{n}\sum_{j=1}^{n}\frac{b_{ij}}{\sum_{j=1}^{n} b_{ij}} \qquad (1)$$

Але при значній кількості альтернатив, в силу дії на експертів різних негативних факторів, матриця є неузгодженою і її ранг буде відмінним від одиниці, тобто матриця буде мати декілька власних значень. А.Павловим в роботі [3] для розв'язку даної задачі запропоновано моделі для різних форм представлення міри неузгодженостей і метод обчислення ваг альтернатив з умови мінімізації неузгодженості матриці. А в роботах [4],

[5] цей метод було використано в задачі вибору архітектури ПС з врахуванням показників якості. В даній задачі було використано міру неузгодженості наступного виду.

$$\left|\frac{w_i}{w_j} - b_{ij}\right| \le \delta_{доп} \cdot b_{ij}, \quad \delta_{доп} \ge 0, \tag{2}$$

де $\delta_{доп}$ – задане порогове значення.

Тоді ваги, які мінімізують (2), знаходяться з рішення задачі лінійного програмування:

$$\begin{aligned} &\min_{\{w_i\}} \sum_{i=1}^{n}\sum_{j=1}^{n}\left(y_{ij}^{+} - y_{ij}^{-}\right) \\ &w_i \ge a_i, \quad i = \overline{1,n}, \\ &w_i - b_{ij} w_j = y_{ij}^{+} - y_{ij}^{-}; \qquad y_{ij} \ge 1; \end{aligned} \tag{3}$$

$$\begin{aligned} &-\delta_{доп} \cdot b_{ij} \cdot w_j \le w_i - b_{ij} \cdot w_j \le \delta_{доп} \cdot b_{ij} \cdot w_j, \\ &y_{ij}^{+}, y_{ij}^{-} \ge 0; \quad i, j = \overline{1,n}. \end{aligned} \tag{4}$$

Таким чином, знаходження ваг альтернативних архітектур зводиться до рішення задачі (3), (4). При розв'язуванні даної задачі може виявитися несумісність обмежень (4), в такому випадку потрібно збільшити $\delta_{доп}$. Використання наведеного алгоритму дає змогу отримати ваги альтернатив як по реалізації кожного критерію якості, так і по їх сукупності. Однак, при прийнятті рішення з вибору архітектури по отриманих оцінках ω_i^S для врахування обмежень та пріоритетів розробників потрібно аналізувати конфлікти між критеріями якості та шукати компромісні рішення.

Література

1. Харченко О.Г. Інструментальний засіб розробки та комунікації вимог якості до програмних систем [Текст] / О.Г. Харченко, В.В. Яцишин, І.Е. Райчев // Інженерія програмного забезпечення. – 2010. – № 2. – с. 29–34.

2. Saaty T. Decision Making with the Analytic Network Process./ Saaty T. Vargas L.// – N.Y.: Springer, 2006. 278 p

3. Павлов А.А. Математические модели оптимизации для нахождения весов объектов в методе парных сравнений. Павлов А.А, Лищук Е.И., Кут В.И. // Системні дослідження та інформаційні технології – К.: ІПСА,– 2007. №2, с. 13 –21.

4. 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.

5. Harchenko, Alexandr, Ihor Bodnarchuk, and Iryna Halay. "Decision support system of software architect." 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS). Vol. 1. IEEE, 2013.

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.