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: Проблеми масштабування Agile-методів розробки програмних продуктів для великих організацій та проєктів.

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

Сенківський В.І
(підпис) (прізвище та ініціали)

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

Нормоконтроль Готович В.А.
(підпис) (прізвище та ініціали)

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

Рецензент Луцик Н.С.
(підпис) (прізвище та ініціали)

Тернопіль
2024

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 Сенківський Володимир Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи **Проблеми масштабування Agile-методів розробки програмних продуктів для великих організацій та проєктів.**

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 № *4/7-1100*

2. Термін подання студентом завершеної роботи 29 травня 2024 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
СПИСОК СКОРОЧЕНЬ ВСТУП 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 1.1 Методологія проведення дослідження за темою роботи 1.2 Передумови масштабування Agile-практик 1.3 Аналіз Agile-фреймворків для великих проєктів 2 ПРОВЕДЕННЯ АНАЛІЗУ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМОЮ РОБОТИ 2.1 Методика проведення огляду 2.2 Результати огляду 2.3 Загрози дійсності 3 ДОСЛІДЖЕННЯ МАСШТАБУВАННЯ AGILE У ВЕЛИКІЙ КОМПАНІЇ 3.1 Інформація про середовище роботи 3.2 Задачі дослідження процесів масштабування Agile в компанії 3.3 Результати дослідження процесу розробки 3.4 Результати дослідження процесу розробки 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	14.04.24-15.04.24	*Виконано*
2.	Підбір наукових джерел по темі роботи	16.04.24-20.04.24	*Виконано*
3.	Переклад та опрацювання наукових джерел	20.04.24-24.04.24	*Виконано*
	по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо теми	24.04.24-10.05.24	*Виконано*
	Кваліфікаційної роботи		
5.	Оформлення першого розділу	04.05.24-05.10.24	*Виконано*
6.	Оформлення другого розділу	05.05.24-13.05.24	*Виконано*
7.	Оформлення третього розділу	13.05.24-14.05.24	*Виконано*
8.	Виконання завдання до підрозділу «Охорона праці»	08.05.24-09.04.24	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	10.05.24-05.05.24	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	05.05.24-31.05.24	*Виконано*
11.	Нормоконтроль	14.05.24-15.05.24	*Виконано*
12.	Перевірка на плагіат	15.05.24	*Виконано*
13.	Попередній захист кваліфікаційної роботи	16.05.24	*Виконано*
14.	Захист кваліфікаційної роботи	30.05.2024	

Студент _____ Сенківський В.І.
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

"Проблеми масштабування Agile-методів розробки програмних продуктів для великих організацій та проєктів" // Кваліфікаційна робота освітнього рівня «Магістр» // Сенківський Володимир Ігорович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНнм-61 // Тернопіль, 2024 // с. – 74, рис. – 8, табл. – 3, джерел – 37.

Ключові слова: Agile, моделі процесів розробки, Agile-розробка, масштабування Agile-процесів

На сьогоднішній день гнучка розробка програмного забезпечення стала дуже поширеною. Проте, адаптація гнучкості для великомасштабних проєктів залишається складним завданням із численними проблемами. Метою даної роботи є огляд практик, викликів та факторів успіху в масштабуванні Agile, спираючись як на літературні джерела, так і на дані з проєкту великої компанії з розробки програмного забезпечення, для визначення найбільш критичних факторів.

Цілеспрямований огляд літератури здійснюється з метою визначення важливості практик масштабування, проблем та факторів успіху. Результати цього огляду застосовуються для аналізу процесів розробки у компанії, що спеціалізується на програмному забезпеченні, з метою масштабування гнучких методів. Дослідження показало, що культура компанії, попередній досвід з гнучкими методологіями, економія, підтримка керівництва та узгодженість цінностей є ключовими факторами успіху в процесі дослідження.

Основними проблемами масштабування виявилися стійкість до змін, надмірно тривалі терміни впровадження, проблеми із забезпеченням якості та інтеграція з уже існуючими негнучкими бізнес-процесами. Таким чином, процес дослідження дозволив поєднати ідеї з літератури з реальним контекстом

компанії. Масштабування Agile всередині організації не потребує конкретної схеми, а може бути адаптоване під потреби, зберігаючи при цьому основні цінності та принципи гнучких методологій.

ANNOTATION

“Problems of scaling Agile methods of developing software products for large organizations and projects” // Master’s degree qualification paper // Senkivskyi Volodymyr Ihorovych// Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group СНнм-61 // Ternopil, 2024 // p. – 74, fig. – 8, tables – 3, references – 37.

Key words: Agile, development process models, Agile-development, Agile-processes scaling

Currently, agile software development has become widespread. However, extending agility to large-scale contexts remains a challenging task with numerous issues. The goal of this work is to review the practices, challenges, and success factors for scaling Agile, based on both literature and project data from a large software development company, identifying the most critical factors.

A targeted literature review is conducted to determine the significance of scaling practices, issues, and success factors. The results of this literature review are used to investigate the development processes within a software company to scale agile practices. The research findings established that company culture, prior experience with agility, cost savings, management support, and alignment of values were key success factors during the action research process.

Resistance to change, excessively long deployment times, quality assurance problems, and integration with existing non-agile business processes were identified as critical issues in the scaling process. Thus, the research process allowed the integration of literature insights into the company's context. Scaling Agile within an organization does not require a specific framework; rather, the process can be tailored to meet the needs while maintaining the core values and principles of agile methodologies.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ 8
ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 13
1.1 Методологія проведення дослідження за темою роботи 13
1.2 Передумови масштабування Agile-практик 15
1.3 Аналіз Agile-фреймворків для великих проєктів 20
2 ПРОВЕДЕННЯ АНАЛІЗУ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМОЮ РОБОТИ 24
2.1 Методика проведення огляду 24
2.2 Результати огляду 26
2.2.1 Практики масштабування 26
2.2.2 Виклики Agile-масштабування 29
2.2.3 Фактори успіху для Agile-масштабування 35
2.3 Загрози дійсності 38
3 ДОСЛІДЖЕННЯ МАСШТАБУВАННЯ AGILE У ВЕЛИКІЙ КОМПАНІЇ 40
3.1 Інформація про середовище роботи 40
3.1.1 Команди розробників 41
3.1.2 Менеджери продукту 43
3.1.3 Agile-тренери 43
3.1.4 Директор продукту 44
3.1.5 Директор з розробки 44
3.1.6 CTO та архітектура 44
3.1.7 DevOps 45

3.2 Задачі дослідження процесів масштабування Agile в компанії 45
3.3 Результати дослідження процесу розробки..... 48
3.3.1 Зміни під час процесу дослідження розробки 48
3.3.2 Практики масштабування..... 50
3.3.3 Виклики..... 52
3.3.4 Фактори успіху..... 53
3.3.5 Пропозиції, отримані в ході дослідження розробки 55
3.4 Результати дослідження процесу розробки..... 58
4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 60
4.1 Ергономічний аналіз умов праці. Система "людина-машина"..... 60
4.2 Оцінка хімічної обстановки та розрахунок аварії на підприємстві із зберіганням аміаку 64
ВИСНОВКИ..... 69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70
ДОДАТКИ

СПИСОК СКОРОЧЕНЬ

LeSS – Large Scale Scrum, Scrum великих масштабів;

SAFe – Scaled Agile Framework, масштабована гнучка структура;

CoP – Communities of Practice, співтовариства практиків;

SoS – Scrum of Scrums;

CTO – Chief Technology Officer; генеральний директор, головний виконавчий директор;

UX – User Experience;

QA – Quality Assurance, забезпечення якості;

CMS – Content Management System, система управління контентом

ВСТУП

Актуальність задачі.

Дослідження, що стосуються використання гнучких методологій розробки програмного забезпечення (ПЗ), ведуться багатьма науковцями в області інженерії програмного забезпечення. Зокрема, серед робіт, виконаних в ТНТУ, є роботи на тематику проєктування архітектури ПЗ в Agile-проєктах [1, 2, 3, 4, 5, 6], проблеми управління Agile-проєктами [7], інтеграції Agile у CI/CD процеси [8]. Серед згаданих вище робіт лише [5] містить частково огляд проблеми застосування гнучких методів розробки ПЗ у великих компаніях. Тому для вирішення задач дослідження цієї кваліфікаційної роботи звернемося до ширшого переліку літературних джерел і на їх основі побудуємо своє дослідження.

Гнучка розробка програмного забезпечення сьогодні дуже популярна, як показує її застосування в багатьох різних контекстах [9]. Однак традиційні гнучкі методи розроблялися з думкою про одну команду й не мали на меті вирішення проблем масштабованості [10]. У цьому сенсі «масштабування» можна описати як «безперервний процес передачі, перекладу та трансформації знань між різними спільнотами та окремими особами» [11], приділяючи значну увагу комунікаційним потребам під час процесу масштабування.

Масштабування гнучкої розробки програмного забезпечення у великих організаціях є складним і створює кілька проблем. Великі проекти вимагають відповідної координації та спілкування між командами, потрібно залучати інші негнучкі підрозділи [12], залучати потрібних кваліфікованих працівників. Останні дослідження повідомляють, що більшість цілей і практик для масштабування Agile і не залежать від предметної області. Серед ключових факторів згадаються проблеми з координацією кількох команд, труднощі з управлінням вимогами, проблеми з адаптацією організаційної структури та проблеми з розумінням гнучких концепцій у ланцюжку створення вартості. Крім того, також залучення клієнтів (замовників) разом з проблемами з архітектурою

програмного забезпечення та координацією між командами зазначаються, як основні виклики великомасштабних Agile-проєктів [13].

Загалом навчання персоналу, інформування та залучення людей до процесу, а також залучення акторів, які можуть просувати процес далі, були загальними факторами успіху, знайденими в тематичних дослідженнях, пов'язаних із гнучкою масштабованістю [14]. Прийняття нових практик масштабування також є складним, оскільки існують багато наскрізних факторів, які можуть впливати та були виявлені в дослідженнях впровадження гнучких технологій. Зокрема, опір змінам і слабка підтримка керівництва можуть відігравати важливу роль [15].

Гнучкість масштабування та пошук найкращих методів гнучкості масштабування є однією з найбільш релевантних тем дослідження з точки зору інтересу для практиків. Однак у цій роботі не ставиться за мету досягнення консенсусу щодо найкращого набору практик у контексті Agile-розробки [11]. Ми знаємо, що гнучке масштабування є особливо складним для розподіленої розробки, для розробки критично важливих систем безпеки через протиріччя, які можуть виникнути у великих бюрократичних організаціях [16].

Мета роботи.

Мета цієї роботи – зібрати більше доказів щодо факторів впливу під час гнучкого процесу масштабування, оскільки нам потрібно більше відомостей про практики, які найкраще працюють у певному контексті.

З метою досягнення поставленої мети спочатку виконаємо оцінку існуючих методів для масштабування Agile-розробки. Вибираємо два з них – SAFe (Scaled Agile Framework) та LeSS (Large Scale Scrum) – і досліджуємо їх глибше. Використовуючи знання, отримані в результаті цієї оцінки, ми отримаємо загальні практики, які ці методи та фреймворки використовують для масштабування гнучкої розробки.

Об'єкт дослідження: Agile-процеси розробки програмного забезпечення у великих проєктах.

Предмет дослідження: моделі життєвого циклу проєктів з Agile у великих проєктах.

Задачі дослідження.

У цій роботі ми намагаємося відповісти на шість основних питань дослідження:

1. Які практики гнучкого масштабування, про які йдеться в літературі, використовували великі компанії (застосовуючи практики SAFe, LeSS)?

2. З якими труднощами, про які йдеться в літературі, зіткнулися великі компанії (використовуючи практики SAFe, LeSS) під час впровадження масштабної гнучкої розробки?

3. Які фактори успіху, описані в літературі, допомогли великим компаніям (використовуючи практики SAFe, LeSS) прийняти та масштабувати гнучку розробку?

4. Які практики гнучкого масштабування були найефективнішими в контексті компанії?

5. З якими труднощами зіткнулася компанія під час впровадження масштабного процесу гнучкої розробки?

6. Які фактори успіху були релевантними для компанії при адаптації процесів масштабованої гнучкої розробки?

Наукова новизна отриманих результатів. Здійснено огляд методик масштабування Agile-практик на великі, в т.ч. розподілені команди та запропоновано методики такого масштабування.

Практичне значення отриманих результатів. У цій роботі проведено дослідження за літературними джерелами з метою виявлення шляхів та методів розширення гнучких процесів розробки на великі команди.

Після проведення дослідження вивчено досвід великих компаній, які застосовують гнучкі методи розробки ПЗ. Ми визначаємо проблеми та фактори успіху цих впроваджень, а також вивчаємо практики масштабування, які використовували ці компанії. Нарешті, на основі відкритих даних ми проведемо дослідження у великій компанії [17], [18], розширюючи процеси гнучкої

розробки програмного забезпечення, щоб виявити виклики, фактори успіху та конкретні практики масштабування.

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", яка проводились у ТНТУ.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Методологія проведення дослідження за темою роботи

Ми дотримувалися процесу дослідження, який базувався на двох основних етапах. Першим кроком (крок 1, рис. 1.1) був детальний огляд літератури. Мета огляду літератури полягала в тому, щоби отримати теоретичні відомості для промислових досліджень про імпакт-фактори в масштабуванні гнучких досліджень. В роботі [19] подано ґрунтовний огляд цього питання, але наша мета не полягала в тому, щоб надати ще один розширений огляд у гнучкому контексті. Нашою метою було проінформувати розробників матеріалами, які можна було б легко зрозуміти та проконсультувати, забезпечивши хорошу відправну точку для другого кроку.

Рисунок 1.1 – Процес проведення дослідження

Огляд літератури був корисним для визначення практик, проблем і факторів успіху, які були знайдені в попередніх дослідженнях. Після початкового визначення загальних практик із фреймворків SAFe та LeSS ми проводимо огляд літератури, щоб визначити найбільш використовувані

практики, проблеми, з якими стикалися, і фактори успіху в попередніх дослідженнях. Ми використовували процес, подібний до систематичних картографічних досліджень (SMS) і систематичних оглядів літератури (SLR) [20], хоча наша мета була іншою, а застосований процес був менш масштабним. Основна подібність полягала у відображенні результатів дослідження, яке зазвичай виконується в SMS для відображення поточного статусу наукової області, щоб визначити кількість і тип досліджень і доступних результатів. У нашому випадку відображення проводилося між ідентифікованими факторами та попередніми дослідницькі статті.

Другим кроком був процес дослідження розробки (етап 2, рис. 1.1), проведений одним із авторів під час процесу розширення гнучких процесів у компанії, що займається програмним забезпеченням. Виявлені практики, виклики та фактори успіху були вхідними даними для цього процесу. Дослідження процесу розробки – це поєднання теорії та практики, оскільки воно намагається надати практичну цінність, водночас здобуваючи нові теоретичні знання дієвим і відтворюваним способом. Таким чином, можна надавати зворотний зв’язок компанії, а також спостерігати за змінами та аналізувати їх, щоб надати додаткові знання про широкомасштабну гнучку реалізацію.

Дослідження процесу розробки складається з п’яти етапів, які циклічно повторюються: діагностика, планування дій, вжиття дій, оцінка та вивчення специфікації. Були обмеження на етапі прийняття дій, оскільки коригування всього процесу розробки є тривалим, а дослідження обмежено часовими обмеженнями: деякі пропозиції не вдалося реалізувати.

На етапі діагностики розробляються теоретичні припущення про природу організації та її проблемну сферу з визначенням основних проблем. Ця діагностика була проведена за допомогою інтерв’ю з представниками організації, які представили дослідникам контекст організації та предметну область.

На етапі планування дій створюється план дій, який визначає мету змін і підхід до змін. Пропоновані заходи керуються як бажаною метою організації,

так і відповідними змінами, які б досягли мети компанії. План представлений у формі конкретних дій, а також додаткових пропозицій, які підтримували зміну до бажаного майбутнього стану.

Після того, як план дій виконано, дослідники та практики проводять оцінку результатів. Результати в основному представили представники організації. Як зазначено, виконання плану дій було обмежено можливістю змінити процес розробки в обмежений час.

Діяльність вивчення специфікації формально описується як остання фаза методу дослідження процесу розробки. Однак, як правило, це постійний процес, оскільки нові знання здобуваються та постійно обмірковуються.

1.2 Передумови масштабування Agile-практик

При переході до великомасштабних Agile-проєктів варто чітко сформулювати означення таких проєктів. Чіткої згоди стосовно цього терміну в літературі немає. У той час як гнучка методологія передбачає 7 ± 2 розробників на команду, дискримінаційним фактором для розрізнення малих і великомасштабних проєктів є кількість команд, причому 2-9 команд означають «великомасштабні», а понад 10 «дуже масштабні» [21]. Більш чітке визначення «великомасштабного» проєкту було дано в [19], що враховує 50-100 розробників і шість команд для великого проєкту розробки.

На сьогоднішній день існує декілька інфраструктур, які можна використовувати для керування процесом масштабування в організаціях. На основі потреб масштабування, що виникали протягом багатьох років, було розроблено кілька методів, практик та інфраструктур для розширення традиційних гнучких методів: Scrum of Scrums (SoS), Scaled Agile Framework (SAFe), Large-Scale Scrum (LeSS), Disciplined Agile Delivery (DAD), Lean Scalable Agility for Engineering (LeanSAFE), Recipes for Agile Governance in the Enterprise (RAGE). Серед них SoS, SAFe і LeSS вважаються більш зрілими [22], [23], [24], [25].

Згідно з опитуванням Version One 2017 про стан гнучкості [26] найбільш використовувані гнучкі методи масштабування наразі відсортовані від вищого рівня впровадження: Scaled Agile Framework (28%), Scrum/Scrum Scrums (27%), внутрішні методи (13%), Lean Management (4%), Agile Portfolio Management (4%), Large Scale Scrum (3%), Disciplineed Agile Delivery (1%), Recipes for Agile Governance in the Enterprise (RAGE) (1%), Nexus (1%). «Внутрішньо створені» або спеціальні методи відіграють важливу роль, поєднуючи кілька практик по-новому. У таблиці 1.1 зведено основні структури для гнучкого масштабування, які опишемо далі.

Таблиця 1.1 – Порівняння основних фреймворків для гнучкого масштабування

Аспект	**SAFe**	**SoS**	**LeSS**	**DAD**	**Nexus**	**RAGE**
Розмір команди	50-120 осіб у випускних поїздах	5-10 команд	10 команд Scrum, 7 членів х команду	200 осіб і більше	3-9 команд Scrum	немає конкретного розміру
Дифузія	Високий	Високий	Середня	Низький	Низький	Низький
Рівень зрілості	Високиц	Високий	Високий	Середній	Низький	Низький
Складність	Високий / Середній	Середній / низький	Середній / низький для Scrum-aware	Високий (багато практик)	Середній / низький для Scrum-aware	Середній / низький для Scrum-aware
Тип організації	Традиційні підприємства	Традиційні та гнучкі підприємства	Великі підприємства	Кілька організацій& Підприємств	Традиційні та гнучкі підприємства	Традиційні та гнучкі підприємства

SAFe – це фреймворк, а також збірка найкращих практик гнучкої розробки для великих підприємств [22]. Фреймворк був прийнятий великими підприємствами, такими як Intel, Hewlett-Packard Enterprise і Cisco [27]. Фреймворк побудований на ідеях гнучкої розробки і системного мислення. Він підтримує компанії різного розміру, від невеликих із менш, ніж сотнею співробітників, до великих підприємств із тисячами людей. Для підтримки такого рівня гнучкості SAFe має додаткові розширення для великих компаній.

Визначення SAFe є дуже детальним і іноді містить навіть конкретний розклад порядку денного окремих зустрічей. Його організаційна структура

велика, має кілька ієрархічних рівнів із багатьма визначеними ролями та обов'язками.

Архітектура SAFe складається з рівнів. Є три базові рівні: рівень портфоліо, рівень програми та рівень команди. Для великих підприємств існує додатковий рівень Value.

Рівень портфоліо керує підприємством у його місії. Він визначає та керує основними стратегічними рішеннями, які мають принести цінність організації. Тоді рівень програми реалізує визначену місію. Він керує, підтримує та синхронізує кілька гнучких команд, які створюють рішення. Командний рівень є найнижчим рівнем SAFe. Він описує, як працюють гнучкі команди: він використовує методи Scrum, Kanban і XP у процесі розробки.

Scrum of Scrums (SoS) – це гнучкий підхід до масштабування проектів із 5-10 командами, який базується на зменшенні накладних витрат на спілкування, які можуть виникнути під час масштабування проектів [28]. Основна ідея полягає в тому, щоб призначити деяких учасників щоденних зустрічей Scrum амбасадорами, які братимуть участь у зустрічах SoS разом із амбасадорами з інших команд. Такі щоденні зустрічі відбуваються як звичайні зустрічі Scrum, зосереджені на проблемах координації, які можуть перешкоджати співпраці команди, і все це керується через окремий беклог.

SoS – це здебільшого міжкомандний метод синхронізації, основною метою якого є забезпечення якості процесів Scrum у кожній команді, але його часто цитують у дослідженнях як повну структуру для масштабування Agile [25]. Однак він досягає повного потенціалу, якщо його прийняти в рамках контексту інших фреймворків (наприклад, SAFe). У решті частини цієї роботи ми розглядаємо SoS, як практику масштабування, яку можна використовувати для збільшення гнучких процесів.

LeSS використовує інший підхід, ніж SAFe [27]. Незважаючи на те, що це все ще фреймворк, він набагато легший, ніж SAFe. Основна ідея LeSS полягає в тому, що навіть для великих організацій немає потреби в надскладному процесі. LeSS намагається дотримуватися одного з основних принципів гнучкого

маніфесту – що люди та взаємодії є ціннішими, ніж процеси та інструменти. Є два варіанти: LeSS для організацій, які розробляють продукти, які потребують до восьми гнучких команд. LeSS Huge призначений для підприємств, яким потрібна більша кількість команд.

Порівняно з SAFe, LeSS є менш суворим в описі практик. LeSS залишається максимально гнучким: зосереджується на мисленні, цінностях і принципах, не вводячи занадто багато процесів і ролей. Сама структура набагато молодша і не має такої корпоративної підтримки, як SAFe.

Автори фреймворків SAFe та LeSS погоджуються, що розуміння цінностей, що лежать в їх основі, є важливим і без їхнього визначення процеси не мають сенсу. Ми можемо спостерігати деякі подібні моделі в організаційній структурі та процесах між цими фреймворками. Обидва фреймворки використовують масштабовану роль власника продукту та ієрархію невиконаних завдань. Більше того, обидва вони схиляються до Feature Teams [29] – міжкомпонентних, міжфункціональних команд, які можуть полегшити керування вимогами. Крім того, обидва фреймворки використовують деякі додаткові команди, які допомагають командам розробників у їхній роботі, так звані Undone–підрозділи, які можуть допомогти з наскрізними проблемами, такими як забезпечення якості та їхня інтеграція концепції «зроблено» в спринт/ітерацію. І останнє, але не менш важливе, LeSS і SAFe використовують масштабовані форми спільних зустрічей, такі як масштабоване планування, масштабована демонстрація та масштабована ретроспектива.

Disciplined Agile Delivery (DAD) розширює Scrum за допомогою інших гнучких методів, таких як Lean і Kanban [30]. Багато гнучких практик є частиною фреймворку: Kanban для візуалізації прогресу роботи в обмеженому часі, Scrum, Agile Modeling, Unified Process, eXtreme Programming (XP), Test Driven Development (TDD) і Agile Data – гнучкі стратегії, які можна застосувати до аспектів «даних» впровадження та розгортання програмних систем.

DAD охоплює три основні фази життєвого циклу проекту: початок (початкова фаза дослідження для розуміння обсягу проекту), будівництво (виробництво рішення) і перехід (розгортання рішення).

Фази DAD можна налаштувати відповідно до потреб: Agile/Basic (більш схоже на програму Scrum), Advanced/Lean (більш схоже на Kanban), Continuous Delivery Lifecycle (короткий перехідний період, ідеально підходить для швидких випусків, навіть щоденних), Exploratory Lifecycle (для стартапів/дослідницьких проектів з більшою кількістю етапів для подання та вимірювання релізів).

Окрім традиційних ролей Scrum, DAD представляє роль під назвою Team Lead (схожа на Scrum Master) і роль власника архітектури, отриману від методу гнучкого моделювання. DAD надав інші ролі для вирішення проблем масштабованості між командами (спеціаліст, незалежний тестувальник, доменний експерт, технічний експерт та інтегратор).

Одна відмінність DAD порівняно з іншими фреймворками полягає в тому, що вона надає командам свободу адаптувати процеси, а також дозволяє налаштовувати їх відповідно до вищезазначених життєвих циклів залежно від потреб. DAD особливо корисна для отримання вказівок у сфері архітектури, дизайну та DevOps, а також може бути інтегрований із моделями зрілості. Тим не менш, досі він не отримав широкого поширення.

Nexus було створено з наміром збільшити масштаби практик Scrum, дозволивши зменшити бар’єр для впровадження та спростити застосування [31]. Nexus дозволяє збільшити робоче навантаження від двох до дев’яти команд, а Scrum використовується для керування взаємозалежностями між командами та проблеми, які можуть виникнути при збільшенні їх числа.

У Nexus усі команди Scrum працюють над Backlog продукту, прагнучи досягти інтегрованого приросту за допомогою Nexus Sprint Planning для координації діяльності всіх команд Scrum. Один власник продукту, Scrum Master і член команди інтеграції є частиною Команди інтеграції в Nexus. Існують ретроспективи Nexus Sprint, на яких представники кожної команди Scrum обговорюють виклики спринту. Такі ретроспективи проводяться паралельно зі

збірними ретроспективами. Однак Nexus відносно менш зрілий, ніж інші фреймворки. Очікується, що він досягне більш широкого впровадження в найближчі роки.

Recipes for Agile Governance in the Enterprise (RAGE) – це структура гнучкого управління з процесами, які застосовуються на рівні проекту, програми та портфоліо будь-якого підприємства. Ключовими характеристиками є швидке прийняття рішень, легкі артефакти, налаштування для різних процесів (повністю гучкий, водоспад, гібрид).

На рівні проекту RAGE вимагає, щоб команди були схожі на Scrum-команди з традиційними видами діяльності, такими як планування спринту та ретроспективи, а на рівні управління пропонується застосовувати рейтингову оцінку та розробку історії [32]. На рівні програми, власник продукту та програмний менеджер проводить зустрічі з планування релізів, зустрічі Scrum of Scrums або перегляди випусків, при цьому рівень управління зосереджується на передачі випусків, постановці оглядів готовності, огляду готовності продукту та перевірці розробки продукту. На рівні портфоліо власник портфеля, регіональний власник продукту та керівники програм проводять головним чином наради щодо вдосконалення портфоліо для розробки стратегій, при цьому рівень управління зосереджується в основному на моніторингу, прийнятті рішень та оцінці якості.

RAGE дозволяє досить високий рівень налаштування процесів, але є відносно новим, і рівень впровадження все ще низький.

1.3 Аналіз Agile-фреймворків для великих проєктів

Ми використали інформацію про фреймворки, щоб окреслити наступний огляд літератури. Однак, щоб спростити передачу знань у рамках процесу дослідження розробки в рамках компанії, нам довелося зосередитися на підмножині структур. Охоплення їх усіх із усіма невеликими відмінностями було б непрактичним і призвело б до більш складної комунікації. Ми вибрали два

основні фреймворки: Scaled Agile Framework (SAFe) та Large Scale Scrum (LeSS). Причина в тому, що їх можна розглядати як два репрезентативні фреймворки, оскільки вони використовують різні підходи до масштабованості: SAFe є найскладнішим гнучким фреймворком для масштабування Agile, тоді як LeSS намагається бути максимально мінімалістичним: він менше покладається на процеси та конкретні організаційні структури, а більше на мислення людей і спеціальне спілкування. З цієї причини в наступних розділах ці два фреймворки (і їхні практики) розглядаються як еталонні фреймворки для гнучкого процесу масштабування.

Ми визначили вісім загальних практик масштабування, які використовують дві системи (SAFe, LeSS). Для кожної з них ми використовуємо ідентифікатор (SP = Scaling Practices), щоб полегшити читабельність:

SP1. Scrum of Scrums: підхід до масштабування Scrum у великих групах, поділ людей на групи та проведення щоденних Scrums з амбасадорами команд. Ми вже розглянули основні характеристики SoS, оскільки її можна розглядати як своєрідну окрему структуру для гнучкого масштабування в межах організації. SoS міститься як у LeSS, так і в SAFe.

SP2. CoP – практичні спільноти: група людей зі спільною турботою або пристрастю, які вчаться вдосконалюватися, діляться своїми знаннями та досвідом, взаємодіючи на постійній основі. Основна ідея полягає в тому, що участь у такій спільноті може бути способом збільшити результати навчання також на організаційному рівні. Великі гнучкі організації можуть отримати вигоду від різноманітних неофіційних міжкомандних спільнот, а спільноти практик можуть допомогти в тому, щоб лідери діяли як координатори членів спільноти, а не просто як ієрархічні керівники. у гнучких проектах. І LeSS, і SAFe настійно рекомендують використовувати спільноти практиків у великомасштабних гнучких організація.

SP3. Scaled Sprint Demo: зустріч, під час якої команди демонструють усі реалізовані функції, зосереджуючись на основному продукті, що розробляється. SAFe називає це System Demo, LeSS називає це Sprint Review.

SP4. Управління масштабованими вимогами: група методів масштабування, пов’язаних із керуванням вимогами, наприклад створення ієрархічної структури власників продукту, ієрархічної структури беклогів продукту та керування ними. LeSS Huge використовує зони вимог, власників продукту області та беклоги продукту. У SAFe ця практика використовується у формі потоків цінностей, гнучких релізів, а також їх беклогів і менеджерів.

SP5. Планування масштабного спринту: зустріч, на якій обговорюється майбутнє продукту. SAFe називає зустріч PI Planning, а LeSS називає її Sprint Planning One. Зустріч більша в контексті SAFe, ніж LeSS. Відповідно до SAFe, порядком денним зустрічі може бути бізнес-контекст і бачення продукту, бачення архітектури, готовність організації та аналіз ризиків. LeSS пропонує лише пряму та коротку зустріч, на якій власник продукту представляє пріоритетний список питань, які готові до розробки та їх уточнення разом із визначенням мети спринту. Потім команди обирають, над якими предметами вони працюватимуть.

SP6. Масштабована ретроспектива: зустріч, на якій розглядається виконання попередньої ітерації на рівні однієї команди. Команди повинні обдумати більші перешкоди, які їм заважають. Зустріч є частиною семінару Inspect and Adapt у SAFe. LeSS називає це Загальною ретроспективою.

SP7. Команди функцій: багатокомпонентні, багатофункціональні, стабільні, довготривалі та самоорганізовані команди, які можуть допомогти в розробці, плануванні та впровадженні в спринті. Таким чином, команда функцій повинна мати можливість повністю реалізувати функцію, яка охоплює будь-які компонент продукту, а в ідеалі команда має бути стабільною протягом кількох років. LeSS і SAFe пропонують використовувати команди функцій замість спеціалізованих команд для окремих компонентів.

SP8. Undone Department: група команд, які підтримують команди розробників для досягнення потенційно придатних для доставки інкрементів наприкінці кожної ітерації. І LeSS, і SAFe погоджуються, що команди повинні мати можливість створювати потенційно придатні для доставки інкременти

наприкінці кожної ітерації. Відділ Undone має допомогти із загальним визначенням «зробленого» в ітерації і може допомогти з усіма наскрізними проблемами, які є частиною розробки продукту, від архітектурного дизайну до забезпечення якості (QA).

2 ПРОВЕДЕННЯ АНАЛІЗУ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМОЮ РОБОТИ

2.1 Методика проведення огляду

У цьому розділі ми повідомляємо про цілеспрямований огляд літератури, який був проведений як перший крок до процесу дослідження розробки.

Огляд літератури проводився з використанням чотирьох цифрових сховищ (IEEE Explore, Springer, Research Gate, ACM). Мета полягала в тому, щоб відобразити практики, визначені в попередньому розділі, від SAFe та LeSS до досвіду та уроків, отриманих у процесі трансформації. Ми прагнули відповісти на три основні питання:

1. Які практики гнучкого масштабування, про які йдеться в літературі, використовували великі компанії (застосовуючи практики SAFe, LeSS)?

2. З якими труднощами, про які йдеться в літературі, зіткнулися великі компанії (застосовуючи практики SAFe, LeSS) під час впровадження масштабної гнучкої розробки?

3. Які фактори успіху, описані в літературі, допомогли великим компаніям (використовуючи практики SAFe, LeSS) прийняти та масштабувати гнучку розробку?

Було визначено рядки пошуку для трьох аспектів, щоб знайти відповідні основні дослідження:

1. Agile методи ("agile, lean, scrum, kanban, agile");
2. Рамки масштабування ("широкомасштабна scrum, АБО масштабована гнучка структура АБО безпечна АБО гнучка доставка")
3. Щоби включити одну статтю в огляд, перевірена компанія повинна мати принаймні 50 співробітників і кілька команд, які розробляють один продукт. Ми визначили такі критерії включення/виключення:

1) початкове навчання може відповідати вимогам;

2) включення сірої літератури, якщо вона знайдена (це був основний вибір для включення сховища, такого як ResearchGate)

3) тільки англійська мова;

4) виключення документів, які не стосуються практик SAFe або LeSS, визначених у попередньому розділі;

5) виключення статей, які не надають уроків, засвоєних з точки зору доказів із тематичних досліджень / опитувань;

Таблиця 2.1 – Репозиторії та результати запитів

Репо	**Запит**	**Результати**
IEEE	(Agile, Lean, Scrum, Kanban, Agile) I (великомасштабна scrum АБО scal * agile framework АБО безпечна АБО дисциплінована гнучка доставка)	105
Спрингер	agile AND lean AND scrum AND kanban AND agile AND ("широкомасштабна scrum" АБО "масштабована гнучка структура" АБО safe АБО "дисциплінована гнучка доставка")	60
ACM	"query": {content.ftsec:(+agile, +lean, +scrum, +kanban, +agile large scrum, scaled agile framework, safe, disciplined agile delivery)}	73

Результати повнотекстового пошуку в усіх репозиторіях (IEEE, SpringerLink, ACM, ResearchGate), представлені в таблиці 2.1. Після рецензування назви та реферату було до огляду включено 12 статей, що також показує відображення статей на рисунках із відповідним цитуванням.

2.2 Результати огляду

Щоб відповісти на запитання дослідження, далі представляємо три відображення (практики, виклики, фактори успіху), обговорюючи кожен висновок відповідно до схеми класифікації, яка була отримана або зі статей, або з початкового огляду систем масштабування. Якщо деякі документи не включено в одне з відображень на наступних рисунках (рис. 2.1, 2.2 і 2.3), це тому, що ми не знайшли відповідного відображення факторів у статті.

2.2.1 Практики масштабування

Беручи до уваги загальні практики, визначені в структурах SAFe і LeSS, у цьому розділі описано, які практики масштабування були знайдені в розглянутій літературі (рис. 2.1).

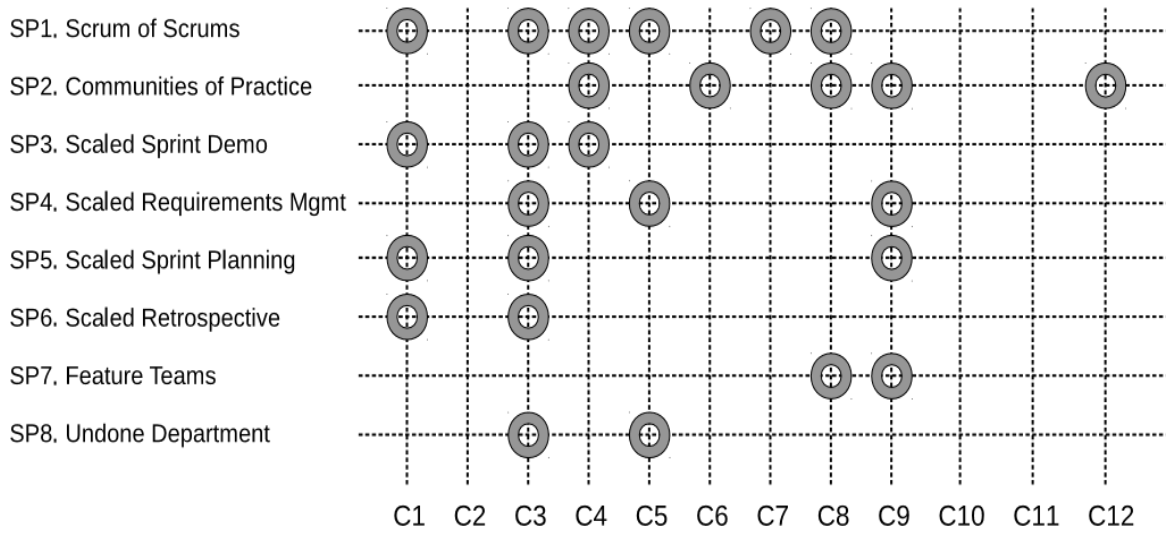

Рисунок 2.1 – Зіставлення документів з індивідуальними практиками масштабування

SP1. Scrum of Scrums: це була найбільш використовувана практика масштабування [33]. Типове використання SoS було для синхронізації між командами, але в деяких випадках його використовували як щоденну зустріч, яка може з часом призводять до проблем через велику кількість розподілених

команд, які повинні бути представлені. Спеціальну функцію SoS використовували, щоб уникнути проблеми надто великих зустрічей. Надмірні зустрічі SoS можуть призвести до того, що команди перестануть повідомляти про свої перешкоди, відчуваючи, що є недостатньо часу під час зустрічей. Як пропонують LeSS або SAFe, наявність представників команди на зустрічах може зменшити цю проблему, однак їх потрібно залучати до роботи команди, щоб внести корисні моменти в обговорення.

SP2. CoPs. Співтовариства практиків: загалом у п’яти випадках згадується використання CoPs. CoPs вирішували брак інформації про дизайн системи, гнучку розробку та інструменти або використовувалися для обміну знаннями та просування найкращих практик для полегшення переходу до функційних команд [34]. CoPs також використовувалися для навчання, координації та проектування між командами, вдосконалення програмного забезпечення та технологій, а також покращення способу роботи.

SP3. Scaled Sprint Demo: у всіх випадках демонстрація Scaled Sprint Demo використовувалася як демонстрація функцій усіх частин продукту [35]. Масштабованість демонстрації може бути проблемою, якщо її відвідує багато команд. Пропонується або дозволити Власнику продукту бути присутнім на демонстраційних сесіях або використати модель наукового ярмарку з командами, які демонструють свої результати на деяких стендах. Випадок [34] зіткнувся з кількома проблемами з демонстрацією Scaled Sprint Demo. Спочатку зустріч проходила у великій аудиторії, на ній були присутні всі команди. У міру того, як проект розростався, демо масштабований спринт став занадто великим. Таким чином, команди почали проводити окремі демонстраційні зустрічі, на яких були присутні лише власники продукту, але це не давало командам достатнього контексту для всього продукту. Наступним кроком було знову використати Scaled Sprint Demo в одному місці з усіма командами, але через обмеження часу демонстрації команд було зведено до слайдів. Автори повідомили, що організація планує повернути Scaled Sprint Demo до окремих демонстрацій окремих команд.

Інша організація пройшла подібну трансформацію Scaled Sprint Demo, як і попередня. Першою спробою було дозволити командам демонструвати свою роботу в одному місці. На другому етапі кожна команда мала лише слайд-презентацію. Останнім етапом зустрічі була так звана модель наукового ярмарку: «кожна команда встановлює експозицію, а співробітники відвідують ті експозиції, які їм цікаві. Кожна «кабіна» пропонує 15-хвилинну презентацію про нещодавню роботу». А аналогічна модель запропонована в LeSS.

SP4. Управління масштабованими вимогами: масштабування управління вимогами було виявлено в трьох випадках. Залежності між компонентами можуть бути проблемою, оскільки вони призводять до великих накладних витрат на компоненти, що не дозволяє проводити окремі зустрічі для різних областей продукту. Налаштування архітектури продукту відповідно до потреб великомасштабної розробки програмного забезпечення може бути ключовим фактором.

SP5. Масштабне планування спринту: масштабне планування спринту було виявлено в трьох випадках. Це може бути складним у розподіленому проекті з кількома сайтами, оскільки може призвести до пізнього планування, неправильних оцінок і неправильного розуміння вимог. Додаткові зустрічі мали бути проведені, щоби пояснити вимоги командам у достатній мірі. Командам також потрібні були більш детальні пояснення та роз’яснення з точки зору дослідження функції та дослідження концепції функції для кожної функції, щоб з’ясувати, чи можна виконати функцію, скільки зусиль потрібно докласти та як це можна зробити.

SP6. Масштабована ретроспектива: у двох випадках було виявлено масштабну ретроспективу. Виявлені проблеми можуть бути надто великими та складними, щоб їх можна було вирішити. Вирішення цих проблем може тривати кілька місяців. Тому ті самі проблеми були виявлені командами в кількох послідовних ітераціях зустрічі. Таким чином, команди поставили під сумнів корисність зустрічі, оскільки вони виявляли ті самі перешкоди знову і знову, без жодного прогресу.

SP7. Команди функцій: лише два випадки згадували про використання команд функцій. На жаль, жодна з цих справ не була успішною. В одному випадку організація створила міжкомпонентні, міжфункціональні команди, що спеціалізуються на конкретних бізнес-потоках. Ця спеціалізація дозволила командам зосередитись на більш вузькій галузі продукту. В іншому випадку рішенням було створити програму систематичного обміну між командами.

SP8. Undone Department: хоча в багатьох випадках доводилося мати справу з проблемами проектування архітектури та з неадекватною інфраструктурою розробки, лише два випадки згадували використання Undone Department, а саме команду архітектури, що складається з 9 технічних спеціалістів високого рівня. А в іншому випадку там були тестувальники, експерти із забезпечення якості та команди DevOps.

2.2.2 Виклики Agile-масштабування

Було кілька проблем, з якими компанії зіткнулися під час трансформації до великомасштабної гнучкої розробки (рис. 2.2). Ми підсумовуємо їх разом із коментарями щодо кожного з викликів. Для кожного завдання ми надаємо загальний ідентифікатор (SC = Scaling Challenge).

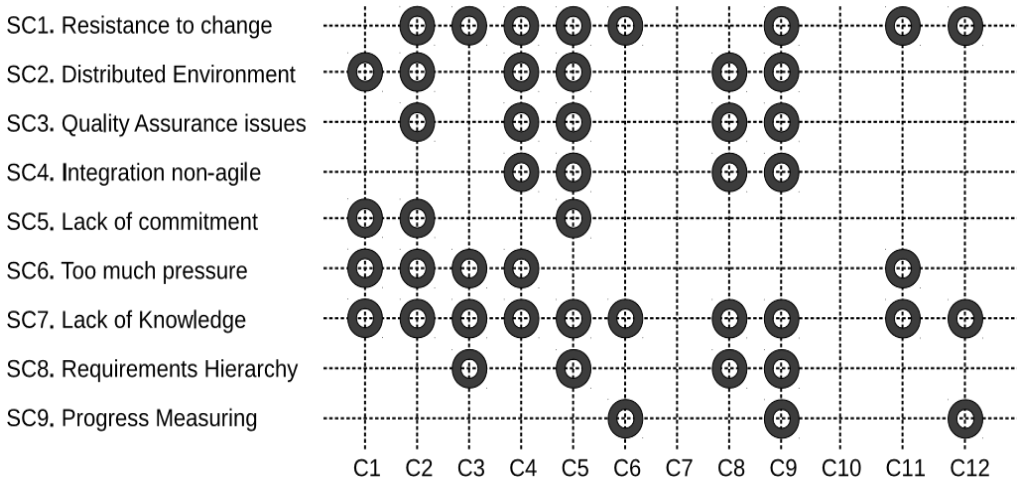

Рисунок 2.2 – Відображення проблем у відповідних роботах

SC1. Опір змінам: опір змінам і прихильність до попередніх процесів були дуже поширеними проблемами. Певна форма опору змінам була описана у ряді статей, наприклад у [34]. Опір виникав на всіх рівнях організацій, включаючи команди розробників, середнє та вище керівництво. Крім того, у великих організаціях опір змінам на вищих рівнях, таких як середнє та вище керівництво, є більш серйозною проблемою. Для успішного переходу необхідне підтримуюче управління змінами: кілька випадків зіткнувся зі значними проблемами через його відсутність.

Однією з причин, чому члени команди не хотіли переходити на гнучку розробку, була підвищена прозорість, тому що люди відчували себе під спостереженням і не хотіли ділитися своїми проблемами. Ще однією причиною опору були нові обов'язки, які гнучка розробка принесла командам. Очікується, що команди будуть самокеровані в гнучкій розробці, але не всі були задоволені. Команди не хотіли вирішувати свої нові проблеми. По-перше, у гнучкій розробці керівники середнього рівня більше не повинні керувати командами. Така передача обов'язків керівництва середньої ланки викликала плутанину, а також опір змінам, оскільки менеджери боялися, що вони більше не знадобляться.

По-друге, менеджери не поважали рішень команди та їхнього бачення процесу розвитку. Інколи менеджери намагалися мікрокерувати командами. Це мікрокерування спричинило кілька проблем, одна з них полягала в тому, що команди втратили інтерес до зустрічей і перестали їх відвідувати, оскільки вони не відповідали за синхронізацію та зв'язок, і тому вважали зустрічі марними. Також команди були немотивованими через брак ресурсів і брак визнання їхніх зусиль щодо покращення процесу розробки. Зрештою, команди кинули свої зусилля.

SC2. Розподілене середовище: будь-яка організація, яка мала свої команди розробників, розподілених на кількох сайтах, стикалася зі значними проблемами з гнучкою розробкою. Проблеми через розподілене середовище були зареєстровані в шести випадках. Гнучка розробка робить наголос на постійному спілкуванні та командному дусі. Розподілене середовище ускладнює

встановлення цих тісних зв’язків. Прозорість є одним із наріжних каменів гнучкої розробки, і без неї гнучка розробка не може бути успішною. Досягти прозорості між командами, які розподілені на кількох сайтах, дуже складно.

Про обмін інформацією можна легко забути в середовищі з кількома сайтами. Інформація з обговорень і нотаток зустрічі з одного сайту може не потрапити в інші місця. Організація повинна наголошувати на прозорості всієї інформації між сайтами та надавати необхідні інструменти, такі як вікі та відеоконференції. Важливо мати всі необхідні знання та інфраструктуру в кожному місці. Щоб пом’якшити ці проблеми, автори [34] запропонували залучити високоякісне обладнання для відеоконференцій.

SC3. Проблеми із забезпеченням якості: п’ять досліджень повідомили про проблеми із забезпеченням якості та втратою якості коду невдовзі після переходу до масштабованих гнучких процесів. Втрата якості була спричинена головним чином додатковими обов’язками, тиском на команди або неправильним визначенням поняття «готово». Команди пропускали виправлення помилок, тестування та не використовували безперервну інтеграцію, таким чином накопичувався технічний борг. На вирішення проблем із великим технічним боргом можуть знадобитися роки. Втрата якості призвела до розчарування команд. Команди почали брехати та повідомляти неправдиві дані про свою роботу, щоб приховати проблеми. Ці проблеми були в основному вирішені шляхом запровадження відділу Undone і навчання. Наприклад, організація включила до команд тестувальників і експертів із забезпечення якості. Експерти визначили допустимі рівні технічної заборгованості для окремих продуктів: нові функції не могли бути додані до продукту, рівень якого не був задоволений. Крім того запропонували командам негайно виправляти помилки. У [34] запропонували створити загальний беклог, запровадити допоміжні ролі, наприклад людей, відповідальних за певну підсистему та архітекторів, побудувати відповідне тестове середовище та навчити персонал постійної інтеграції. Відділ Undone організації включав архітектурну групу, групу

підтримки, яка усувала перешкоди, які не могли вирішити команди розробників, і експертну групу з економічного використання.

SC4. Інтеграція з негнучкими частинами організації: неправильне узгодження організаційних структур спричинило проблеми в чотирьох випадках. Наприклад, проблема полягала в тому, що негнучкі команди не хотіли покладатися на гнучкі команди, не знаючи, чи гнучкі команди виконають свою роботу вчасно. Ці проблеми можна вирішити шляхом включення каскадних частин організації в процес планування та залучення негнучких команд на ранніх стадіях процесу планування. Крім того, допомагає вдосконалення безперервної інтеграції та систем автоматизації тестування, оскільки це забезпечує швидшу та кращу інтеграцію.

SC5. Відсутність командної роботи: три випадки повідомили про проблеми з відданістю та командною роботою. Проблеми з відданістю команд були спричинені частими змінами в Історіях користувачів, які впроваджувалися, та невизначеністю через неправильну та пізню специфікацію історій користувачів. Відсутність командної роботи також була пов’язана зі спеціалізацією членів команди. Спеціалізовані люди в командах були стурбовані своїми проблемами та не хотіли дотримуватися спільного командного плану. Командне визначення «зробленого» також дуже відрізнялося між членами команди: окремі члени команди визначали пріоритетність різних проблем на основі своєї спеціалізації. Розподілене прийняття рішень також було ускладнене, члени команди не розуміли роботу інших і тому не хотіли покладатися на інших. Крім того, обмін знаннями не працював, оскільки фахівці усвідомлювали, що вони важливі для компанії. Вони не хотіли, щоб інші розробники знали, як виконувати їх особливу роботу, оскільки це робило їх більш незамінними для компанії. Все це призвело до дуже конкурентного середовища, і тому деякі люди боялися повідомляти про свої проблеми. В одному випадку командна робота була порушена, тому що люди вважали, що Scrum був запроваджений для підвищення ефективності та замінності програмістів. Крім того, вони вважали, що плоска управлінська

ієрархія пропонує менше можливостей кар’єрного зростання. Отже, люди намагалися бути більш помітними та конкурентоспроможними.

SC6. Занадто великий тиск і робоче навантаження: високий тиск і робоче навантаження викликали багато проблем. Про проблеми, пов’язані з ними, було повідомлено у п’яти випадках. Новий спосіб роботи, нові обов’язки разом із постійним тиском ринку посилили напругу в командах. Ця напруга призвела до поганої якості коду та запізнілого планування. Команди також пропустили тестування, рефакторинг, виправлення дефектів та інженерні практики, що створило багато технічних боргів. Крім того, оскільки команди, які були новачками у гнучкій розробці, не усвідомлювали важливість удосконалення процесу, занадто сильний тиск змусив їх не виконувати свої обов’язки. Дефіцит часу змусив команди не піклуватися про виявлені проблеми в їхніх ретроспективах. Недосвідчені команди ще не мали спільного бачення Scrum і процесу їх розробки, тому було важливо дати їм достатньо часу для розробки такого бачення. Крім того, високий тиск заважав комунікації в командах. Команди почали пропускати та відкладати зустрічі. Тому щоденні зустрічі стали щотижня, а потім навіть щомісяця.

SC7. Брак знань, наставництва та тренінгу: найпоширенішими викликами, які у багатьох випадках стали причиною інших труднощів і проблем, були брак знань, навчання, наставництва чи розуміння. Ці питання були описані в десяти випадках. Декілька факторів спричинили брак тренінгів і знань, здебільшого через недооцінку труднощів адаптивної трансформації, фінансові обмеження, відсутність підтримки з боку керівництва або поспішний перехід. Наприклад, організація недооцінила складність адаптивного впровадження. Організація не мала достатніх знань у своїх лавах, але відмовилася використовувати зовнішніх експертів через фінансові обмеження. Натомість менеджери, які були новачками в Agile, взяли на себе роль експертів з гнучкості. Було кілька наслідків відсутності тренувань і підготовки. Команди не могли змінити свої звички так легко, як вони очікували.

Через швидкий перехід із браком знань і досвіду багато припущень були суто теоретичними. Тому виникло багато неочікуваних проблем. Крім того було зазначено, що команди, які були під наставництвом, показали кращі результати, ніж команди без наставництва.

Наслідком браку знань стало помилкове впровадження гнучких практик. Наприклад, у трьох випадках люди не розуміли мети зустрічей. Команди не знаходили глибинних причин у ретроспективі, а лише визначали симптоми. Крім того, команди не дотримувалися основних принципів практик, які вони мали впроваджувати, наприклад поваги до ітерацій у Scrum, пріоритетності історій користувачів або важливості контакту з клієнтом.

Хоча команди мали всю інформацію про практики, вони не розуміли їх і використовували неналежним чином. Наприклад, організація зіткнулася з ситуацією, коли менеджер примушував дотримуватись практик Scrum суворо, згідно з правилами, і проти їх волі. Крім того, деякі команди дотримувалися практик занадто фанатично та втратили перспективу.

Крім того, перехід до великомасштабної гнучкості також вимагав вивчення предметної області. Організації довелося навчати команди в інших предметних областях, щоб створити команди функцій через попередню компонентну архітектуру їх продукту. Відповідний інструктаж менеджменту збільшив підтримку гнучких методів.

SC8. Ієрархія управління вимогами: у великих проектах вимоги не піддаються управлінню одній особі (власнику продукту), отже, існує необхідність розділити відповідальність за їх управління. Цей поділ у багатьох випадках був серйозною проблемою. Масштабування управління вимогами неможливо уникнути при масштабуванні Agile. Той факт, що деяким організаціям довелося коригувати всю архітектуру своїх продуктів, свідчить про такі труднощі. Той факт, що весь перехід може бути невдалим через неправильний поділ продукту на області, також демонструє важливість. Тому робити це потрібно акуратно і правильно.

SC9. Вимірювання прогресу: коли організація змінює процес розробки, вимірювання тенденції змін є важливим. Однак це виявилося складним завданням у великомасштабній гнучкій розробці, коли справа дійшла до моніторингу та вимірювання прогресу їхньої гнучкої розробки. Найбільшою проблемою було знайти правильні показники – організації не знали, що вимірювати, щоб отримати значущі результати.

2.2.3 Фактори успіху для Agile-масштабування

З усіх документів, включених до огляду, ми визначили та згрупували зареєстровані фактори успіху в сім груп (рис. 2.3) . Кожному фактору надається унікальний ідентифікатор (SF = Success Factor).

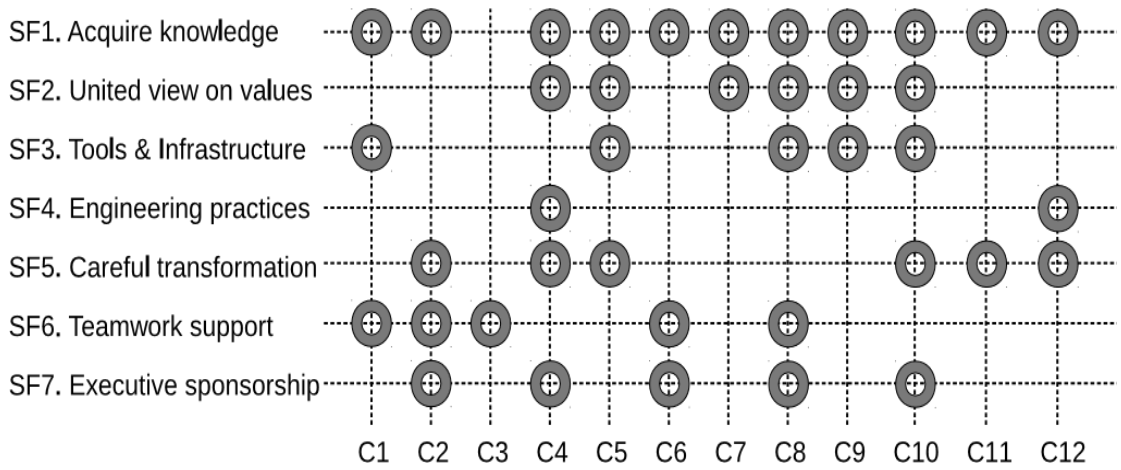

Рисунок 2.3 – Зіставлення документів на індивідуальні фактори успіху

SF1. Здобуття знань: оскільки найпоширенішою проблемою була відсутність знань, не дивно, що найпоширенішим фактором успіху була спроба підвищити рівень знань і досвіду в гнучких практиках. Ретельне, глибоке та систематичне отримання знань та обмін ними було зазначено як фактори успіху в одинадцяти випадках. Найбільш рекомендованим способом отримання знань та досвіду було найняти зовнішнього експерта з широким та глибоким знайомством з Agile-розробки.

Зовнішній експерт поділився своїми знаннями з кількома внутрішніми співробітниками, які потім поширили їх по всій організації. Однією з причин, чому було вигідніше залучити експерта, ніж інвестувати в отримання знань з інших джерел, було те, що організації усвідомлювали, що загальні та теоретичні пропозиції нелегко застосувати в їх конкретному контексті, і потрібне було глибше розуміння методів.

Інша рекомендація полягала в тому, щоб навчати людей глибоко, а не широко. Коучинг має стосуватися як цінностей, так і процесів, але, на жаль, ці цінності часто забувають. Відсутність розуміння цінностей може призвести до відсутності прагматизму та нездатності мати справу з новими та невідомими ситуаціями. Agile означає бути гнучким, але його адаптація вимагає глибокого розуміння його цінностей. Краще, якщо люди будуть віддані постійному навчанню, ніж відданим певним гнучким методам або принципам. Крім того, важливо навчити команди, як навчатися, наприклад, використовуючи двоциклове навчання, щоб вони краще знаходили основну причину проблеми.

Відповідно до двох випадків важливо навчати людей, а не диктувати їм. Тренери повинні використовувати «слід» чи «варто» замість «повинен» і дозволити трансформації здійснюватися знизу вгору. Що стосується успішної гнучкої трансформації, необхідно мати підтримку співробітників. Крім того, систематичне навчання тренерів та інших людей всередині організації було необхідне для успішного широкомасштабного впровадження гнучкої розробки. Систематичне навчання та тренінги означають регулярні заходи, презентації зовнішніх експертів або спільнот практиків. Крім того, систематичне навчання забезпечує достатньо наявних тренерів в організації. Відсутність тренерів призводить до неконтрольованого та непідготовленого прийняття гнучких методів надмірно завзятими командами. Змінити їхні неправильні звички було важче, ніж змінити команду без будь-якого досвіду. Більше того, так званий парний коучинг, який поєднує гнучких експертів із експертом із знаннями предметної галузі, щоб забезпечити більш ефективний і точний коучинг, виявився ефективним.

SF2. Єдиний погляд на цінності та практики: для успішного адаптивного переходу необхідно визначити спільний погляд на зміни. Загалом у шести випадках уніфікація цінностей, визначень, способу роботи та розуміння була корисною.

Необхідно правильно визначити ролі, їхні обов'язки та загальні визначення. Крім того, важливо вказати на неправильні ідеї та неправильні уявлення. Команди більш охоче дотримуються єдиного погляду на Scrum. Що було корисно використовувати спільну мову між командами, архітекторами та власниками продуктів, як-от встановлені гнучкі визначення, уніфікована мова моделювання (UML) або фундаментальні концепції моделювання (FMC). Є кілька способів створити спільний погляд і спосіб роботи. Наприклад, це можна зробити за допомогою інструктажу, документації, спільнот практиків або ціннісних семінарів.

SF3. Інструменти та інфраструктура: організації повинні бути готові надавати достатні ресурси командам під час переходу до гнучкої розробки. П'ять випадків стверджували, що гідні інструменти та інфраструктура були корисними. Інструменти включають, наприклад, середовище розробки та локальне тестування, безперервну інтеграцію та автоматизовані тести, комунікаційну інфраструктуру, включно з обладнанням для відеоконференцій. Загальні інструменти та інфраструктура були особливо корисними в розподіленому середовищі, де командам потрібна допомога, щоб зменшити свої комунікаційні перешкоди.

SF4. Надійна інженерна практика: багато команд зіткнулися з втратою якості свого коду під час адаптивної трансформації. Тому наявність надійних інженерних навичок має вирішальне значення. Запропонували включити QA та тестувальників до команд розробників. Крім того, було запропоновано ефективне технічне управління боргом, наприклад рівні допустимих відхилень і візуальні індикації рівнів допустимих відхилень.

SF5. Обережна трансформація: перехід організації до гнучкого розвитку означає зміну мислення людей. Ця зміна може зайняти багато часу. Отже, у

шести випадках була потрібна повільна та обережна трансформація. Краще сприймати трансформацію як довгострокову організаційну зміну. Організація повинна вдосконалювати процес розвитку поступово і займати щонайменше три місяці перед зміною методів і практик. Організаціям також слід переконатися, що вони мають усі необхідні ресурси, такі як достатньо часу, підготовлений план і досвід, перш ніж здійснювати перехід. Командам потрібен час і простір, щоб навчитися та адаптуватися до нового способу роботи та змінити свої звички. Таким чином, організація повинна давати їм достатньо часу і не тиснути на них або дозволяти їм працювати понаднормово.

SF6. Підтримка командної роботи: у семи випадках повідомлено про те, що тісні зв'язки та постійне спілкування між командами та членами команди необхідні для успішної гнучкої розробки. Організація повинна створити прозоре середовище для відкритості в команді, не боячись обговорення проблем для покращення командної роботи. Крім того, краще тримати команди невеликими. Спілкування між командами та членами команди можна покращити за допомогою зустрічей SoS та CoPs. Ці практики є «і форумами, і провокаторами дискусій».

SF7. Залучення виконавчого керівництва є найважливішим фактором успішної трансформації організації. Без належного залучення виконавчого керівництва жодні інші чинники успіху не були б реалізовані, оскільки було б виділено недостатньо ресурсів. У п'яти випадках зазначено, що залученість та підтримка керівництва є обов'язковими. Отримання залученості керівництва може бути проблематичним, якщо перехід до гнучкості є стимулом знизу вгору. Єдина пропозиція, яку ми знайшли, полягала в тому, щоб зібрати достатньо доказів ефективності гнучкої розробки за допомогою програми вимірювання.

2.3 Загрози дійсності

Існує кілька внутрішніх і зовнішніх обмежень у проведеному огляді літератури, які необхідно враховувати. Ми не дотримувалися всіх етапів

процесів систематичного картографування, оскільки наші цілі були іншими. Конкретно четвертий етап – формулювання тез – ми не проводили. Однак для такого рішення було дві причини. По-перше, ми вже мали набір категорій загальних практик масштабування з початкового визначення практик. По-друге, фактори успіху та виклики не можуть бути виведені простим ключовим словом у тезах.

Потреби в нашому огляді літератури були особливими через співпрацю з виробництвом ПЗ. Загальновідомо, що більшість практиків не читають дослідницьких статей. Це залежить або залежить частково від того факту, що дослідження дає відповіді на цікаві питання з точки зору практиків, оскільки може бути прогалина, але також від того факту, що ці статті журналів і конференцій не є хорошим каналом зв’язку між академічними колами та виробництвом. З цієї причини нам потрібен був певний цільовий огляд для аудиторії компанії: надання зацікавленим сторонам компанії попередніх чудових систематичних оглядів, опублікованих у цій галузі (наприклад,[19]) було б недостатньо для досягнення наших цілей.

Таким чином, огляд літератури в цій роботі слід розглядати в цьому сенсі, не як вичерпний, радше як підтримку подальшого процесу дослідження розробки. Наші цілі полягали в тому, щоб забезпечити організацію синтезом усього, що було відомо з літератури щодо картографування практик із SAFe та LeSS. Масштабність огляду не була основною метою: завершення огляду було необхідне вчасно, перш ніж можна було розпочати процес дослідження дії, оскільки співпраця з галуззю повинна відповідати суворим часовим вимогам.

3 ДОСЛІДЖЕННЯ МАСШТАБУВАННЯ AGILE У ВЕЛИКІЙ КОМПАНІЇ

Огляд практик, викликів і факторів успіху з літератури став теоретичним внеском для дійового дослідження [9], дані про котре доступні у відкритих джерелах, компанією з розробки програмного забезпечення. Компанія на сьогоднішній день налічує близько 200 співробітників. Основним продуктом є платформа, яка включає систему керування вмістом (CMS), а також функції електронної комерції та онлайн-маркетингу. Нещодавно компанія почала зміщувати свою увагу на хмарну платформу. Проект розпочався з однієї команди, яка мала повну свободу визначати процес розробки. Додано ще три команди. Ці команди все ще могли вільно вибирати інструменти, технології та частково визначати свій процес розробки. Команди були відокремлені від колишньої організаційної структури. Кожна команда розробила окрему частину повного хмарного рішення. Основна проблема полягала в тому, що хмарна платформа почала розвиватися, і організація хотіла виділити більше розробників і ресурсів. Тому потрібен був більш точний процес розробки..

3.1 Інформація про середовище роботи

На початку процесу дослідження дій нова хмарна платформа мала чотири команди розробників (рис. 3.1). Кожна команда мала свого менеджера продукту та Agile Coach. Керівників продуктів координував директор продукту, а Agile Coaches – директор з розробки. З командою архітекторів проводилися консультації щодо архітектури продукту. Існували додаткові спеціалізовані ролі, які підтримували команди: був спеціаліст із безпеки і координатор із забезпечення якості, який відповідав за команду DevOps. Agile-тренерів, спеціаліста з безпеки та координатора з якості координував директор з розробки. Була також команда технічних авторів, яка відповідала за технічну документацію. Крім того, була доступна команда взаємодії з користувачами.

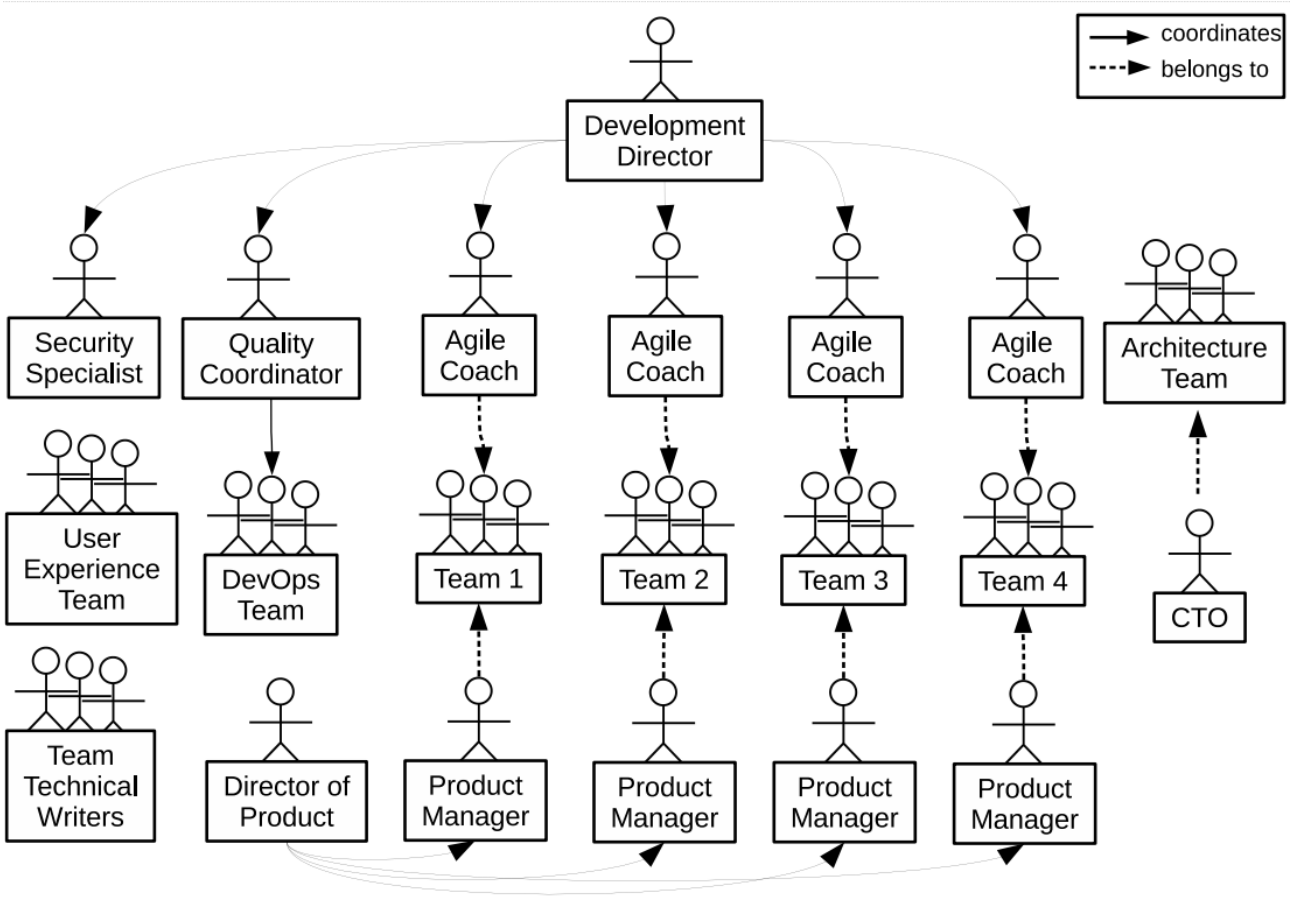

Рисунок 3.1 – Структура організації

Опишемо детальніше згадані ролі.

3.1.1 Команди розробників

На початку процесу дослідження дії всі чотири команди розробників використовували Scrum. Крім того, використовувалися принципи економного виробництва, головним чином зосереджуючись на усуненні затрат, але також розглядалися інші шість принципів економного виробництва. Кожна команда мала свого менеджера з продукції та Agile Coach. Крім того, деякі члени команди мали спеціалізовані ролі, такі як DevOps та UX Designers.

Команди контактували із замовником насамперед через співбесіди, спрямовані на перевірку рішення. Зазвичай із замовником спілкувався один розробник разом із Product Manager. Інші члени команди могли використовувати записи дзвінків. Менеджер продукту щоденно підбивав підсумки співбесід. Крім

того, кожен у командах мав доступ до так званої дошки продуктів, яка була інструментом для збору відгуків від клієнтів.

Кожного дня команди проводили зустрічі для синхронізації – щоденну зустріч Scrum. Кожного тижня відбувалася зустріч із синхронізації хмарної платформи. У ньому взяли участь директор продукту, генеральний директор та представники всіх команд. Крім того, кожна команда мала зустріч із генеральним директором, щоб обговорити свій прогрес і майбутні кроки. Інші зустрічі, такі як семінари з дизайну, зустрічі з питань синхронізації між командами, зустрічі з аналізу ризиків, організовувалися спеціально.

У трьох командах беклог належав команді, а не менеджеру з продукту. Причина полягала в тому, щоб дозволити менеджерам із продуктів більше зосередитися на своїх завданнях. Спочатку беклог підготував менеджер із продукту, але компанія вирішила, що ця роль має більше зосереджуватися на отриманні відгуків від клієнтів і аналізі ринку. Таким чином, право власності на беклог було передано командам, тоді як менеджери продуктів доносили командам ввідну інформацію ззовні. Ця зміна допомогла організації належним чином. Керівники продуктів мали більше часу для зосередження на ринку. Крім того, команди краще розуміли потреби клієнтів, оскільки їм потрібно було працювати з попередньо наданими вхідними даними. Тому команди були більш активними, ставили додаткові запитання та ретельніше аналізували дані.

Були також деякі проблеми, пов’язані з цією зміною. По-перше, на початку потрібно було належним чином пояснити мотивацію цієї зміни, щоб команди брали участь. По-друге, команди перебували на різних стадіях розробки продукту, тому, коли впроваджувалася нова функція, загальна точка зору та її впровадження були дуже різними в окремих командах. І останнє, але не менш важливе: новий спосіб роботи створив додаткові витрати для команд на створення історій користувачів.

3.1.2 Менеджери продукту

Менеджери продукту відповідали за пошук рішення проблеми та відповідності продукту ринку, повернення інвестицій, аналіз конкурентного ландшафту, аналіз виграшів/програшів, покупця та користувача, а також стратегію виходу на ринок. Відповідальність за стратегію виходу на ринок була поділена між відділами продажів і маркетингу. Керівники продукту підпорядковувалися директору з продукту.

Менеджери з продуктів були частиною команд розробників: вони сиділи в одному офісі та щодня спілкувалися з командами. Синхронізація між менеджерами продуктів була здебільшого тимчасовою. Щотижня проводилися дві зустрічі щодо синхронізації: одна для синхронізації між командами, а інша для синхронізації між Agile-тренерами та менеджерами з продуктів.

3.1.3 Agile-тренери

Кожна команда мала одного Agile-тренера. Чотири Agile-тренери разом із директором з розробки були частиною команди Scrum Master. Agile Coaches відповідали за задоволеність членів команди, їхню мотивацію, організацію команди та адміністрування команди, поширення інформації, допомогу командам із глобальними проблемами та перешкодами та навчання навичкам спілкування. Вони також брали участь у процесі найму.

Agile Coaches підвищили свою кваліфікацію, поділившись своїми знаннями, наприклад, використовуючи інструменти бази знань, такі як програмне забезпечення Confluence, і відвідуючи гнучкі та економічні конференції. Agile Coaches також відвідали інші компанії та поділилися з ними своїм досвідом. Був також так званий Scrum Club, який являв собою одногодинну зустріч для додаткового обміну знаннями.

Agile Coaches відвідали кілька зустрічей, таких як Scrum of Scrums, зустрічі щодо синхронізації хмарної платформи та зустрічі щодо стратегії продукту.

Вони щодня спілкувалися з командами, а також проводили регулярні особисті зустрічі з членами команди.

3.1.4 Директор продукту

Директор із продукту відповідав за повне виконання продукту. В його обов'язки входило своєчасне постачання продукту на ринок, комунікація продукту на ринку та прийняття продукту на ринку. Він також керував розставленням пріоритетів між командами та перевіркою результатів міжкомандних ітерацій, керуючи менеджерами з продуктів. Він відвідував щотижневі наради з синхронізації команд і щотижневі наради з синхронізації для Agile Coaches і Product managers. Крім того, він проводив особисті зустрічі з менеджерами з продуктів.

3.1.5 Директор з розробки

Директор з розробки відповідав за ефективну співпрацю між відділами розробки та іншими відділами, правильну роботу організації розроки, підвищення цінності співробітників і допомогу їм у кар'єрних шляхах, правильно інтерпретовану та реалізовану стратегію та бачення компанії, результативність та ефективність зростання розробки та впровадження механізму постійного вдосконалення в організацію. Директор з розробки щотижня відвідував зустрічі з питань синхронізації для хмарної платформи, Agile Coaches і Product Managers.

3.1.6 CTO та архітектура

Технічний директор був членом команди архітектури. Його роль полягала в консультаціях щодо архітектури запланованих функцій. Він також був членом однієї з команд розробників. Команда архітектури переглянула впроваджені

Історії користувачів з архітектурної точки зору та повідомила команди, якщо було щось, що потрібно було вирішити.

Архітектуру обговорювали під час наради щодо синхронізації хмарної платформи та під час спеціальних зустрічей, коли командам потрібно було обговорити.

3.1.7 DevOps

Команда DevOps відповідала за різні інструменти та інфраструктуру розробки: це включало інструменти контролю версій, об'єднання коду, моніторинг продуктивності додатків, конфігурацію та керування інфраструктурою, автоматизацію випусків, сховище артефактів, продуктивність тестів і результатів, інструменти безперервної інтеграції та статус збірки. Представники DevOps відвідали зустрічі з синхронізації для Agile Coaches та Scrum of Scrums. Крім того, відбувалися щотижневі зустрічі DevOps щодо синхронізації, а також спеціальні зустрічі DevOps із командами розробників.

3.2 Задачі дослідження процесів масштабування Agile в компанії

Було визначено три основні дослідницькі питання, які були ціллю для всього процесу дослідження дії. Ці дослідницькі питання імітували ті, які ми поставили для початкової частини огляду літератури – цього разу зведені до основного контексту компанії. Це дозволить визначити відмінності між тематичними дослідженнями, включеними в огляд літератури, та результатами, отриманими в результаті дослідження розробки.

4. Які практики гнучкого масштабування були найефективнішими в контексті компанії?

5. З якими труднощами зіткнулася компанія під час впровадження масштабного процесу гнучкої розробки?

6. Які фактори успіху чи було в компанії доцільно прийняти масштабований процес гнучкої розробки?

Протягом проєкту, що досліджувався підтримувався контакт з компанією та була активна участь у наданні зворотного зв’язку на основі потреб, що випливають із процесу масштабування (рис. 3.2).

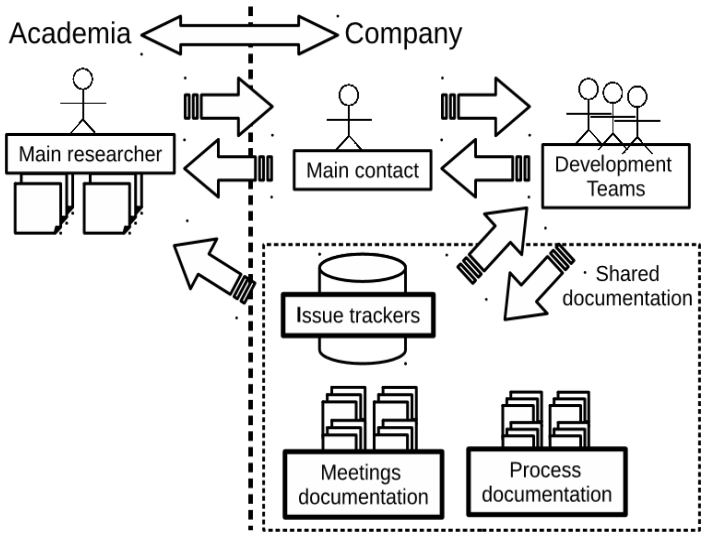

Рисунок 3.2 – Співпраця академічних кіл і промисловості під час процесу дослідження розробки

Дослідник мав доступ до онлайн-інструментів, які використовувалися для документування процесу розробки – головним чином до засобів відстеження проблем з більш обмеженим набором дозволів, ніж у розробників. Він також міг мати доступ до нотаток зустрічей, коли інформація, якою поділилися під час зустрічей, не вважалася захищеною.

Головний контактний пункт допоміг зменшити затрати на комунікацію та надати доступ до будь-якої відповідної інформації, гарантуючи, що жодна зарезервована інформація не залишить компанію. Якщо знадобилася інформація від команд, яку не можна було отримати зі спільних документів, головний дослідник зв’язався з головним контактним пунктом у компанії. Те саме працювало в іншому напрямку, від команди розробників до дослідника, як своєрідний двосторонній зв’язок. Хоча це здається лише опосередкованістю, насправді це надає швидку інформацію дослідникам. Єдиним порушенням у цій дослідницькій обстановці були прямі співбесіди головного дослідника з різними

ролями в командах розробників, які явно не могли проходити через головну контактну точку. На основі огляду літератури та нових знань, що з’явилися під час процесу, головний дослідник передав запропоновані практики головній контактній особі, яка потім надала би зворотний зв’язок досліднику на основі застосування всередині команд. Зрозуміло, що зміни в процесі розробки були на розсуд директорів з розробки / команди розробників – це було реальне середовище, у якому потрібно було поставляти продукти.

Дані збиралися/обмінювалися безперервно за допомогою кількох методів.

1. Було кілька зустрічей, під час яких було визначено проблемну область у компанії.

2. Після цього головний дослідник представив рішення з літератури в кількох ітераціях і обговорив прогрес з конкретними проблемами більш детально.

3. Коли проблеми в компанії були правильно зрозумілі, було створено анкету. Анкету було передано головному контакту, який розповсюдив її на різні посади. Ролі, які ми включили, були менеджерами, розробниками, Agile Coaches, DevOps і членами групи підтримки. Метою анкети було отримати додаткові думки з різних точок зору. Додаткові питання обговорено за допомогою онлайн-спілкування. Анкета була подана двічі, один раз на початку процесу дослідження та в кінці.

4. Основним джерелом інформації були інтерв'ю. Крім того, реалізовано постійний онлайн-спілкування для додаткового уточнення, уточнення та підтвердження інформації.

Розробка процесу дослідження створення ПЗ в компанії будувалась на основі діалогового процесу дослідження розробки, у якому знання з часом розвиваються з обох сторін дослідників і практиків. Головний дослідник запропонував би головному контакту в компанії підходи до експерименту, засновані на нових знаннях як з огляду літератури, так і з попередніх відгуків від команд розробників. Команди розробників отримували б інформацію про підходи, які можна було б випробувати в реальних умовах. Нульова ітерація

стосується надання результатів огляду літератури. Діаграми з огляду літератури (рис. 2.1, 2.2 і 2.3) виявився швидким і ефективним способом зв’язку також із головною контактною точкою. Такі знання розвивалися в процесі дослідження розробки, оскільки деякі підходи були змінені командами розробників.

3.3 Результати дослідження процесу розробки

3.3.1 Зміни під час процесу дослідження розробки

Порівняно з початковою ситуацією в компанії, про яку ми повідомляли на початку цього розділу, було внесено кілька змін.

– Зустрічі. Щоденні зустрічі Scrum все ще використовувалися в командах розробників. Загалом синхронізація команд через різні платформи залишилася (без змін). Була запроваджена зустріч керівництва для обговорення тем компанії (місія, бачення, стратегія);

– Управління беклогом. Не було змін в управлінні беклогом продукту. Були представлені можливості (Epics), визначені менеджерами з продуктів і командами UX.

– Зміни в ролях. Директор з розробки тепер тісно співпрацює з Scrum Master Leader для допомоги у вирішенні стратегічних питань. Було введено роль Керівника команди продукт-менеджера – допомога в організації команди менеджера продукту (схожа на роль керівника Scrum Master).

– Архітектура програмного забезпечення. Внесено зміни: кожна команда має підтвердити свої зміни в архітектурі за допомогою ролі архітектора. Команда має підготувати огляд можливостей та їх впливу на архітектуру, потім архітектори переглянуть його та обговорять з командами додаткові запитання, якщо це необхідно.

– Масштабована ретроспектива. Запроваджено Milestone Retrospective (проводиться 4 рази на рік) – люди можуть додавати свої теми, а після того, як усі теми будуть зібрані, з’являється механізм голосування. Питання, які

отримали найбільше голосів, потім обговорюються на окремих зустрічах – на сесію запрошуються всі виборці.

– Планування масштабування. З’явилася більша гнучкість – спочатку визначення пріоритетів відбувалося лише кожні 3 місяці. Пріоритезація тепер відбувається щомісяця, коли вводяться нові можливості, а потім розставляються за пріоритетом ті, які вже є пріоритетними.

– Ітераційні огляди. Перенесено на четвер кожні два тижні – дві частини: публічна та приватна (де публікуються новини, пов’язані з маркетингом, продажами та успіхом клієнтів).

– Scrum Scrums. концепція залишилася незмінною.

– Масштабування управління вимогами. Робота з беклогом була єдиною для команд. Є один беклог з усіма помилками для команд крос-розробників (на основі узгоджених правил команди мають взяти певну кількість помилок за один спринт і виправити їх). Існує також один загальний беклог для вдосконалення UX. Кожна ітерація має угоду про найменування для автоматизації збору даних: статус ітерацій може автоматично заповнюватися щодня без ручного втручання.

– Undone Department. Концепція залишилася незмінною.

– Практичні спільноти (гільдії). Без істотних змін. Зустрічі, на яких група програмістів збирається разом, щоб навчатися та практикуватися – проводилися щомісяця.

– Відмінності в гнучкій культурі між компаніями. Нове бачення та місія були обговорені та представлені співробітникам. Було введено нову цінність – «Прагнення до досконалості». Загалом актуальними аспектами є постійне відвідування конференцій, зустрічей та обмін досвідом з іншими компаніями, відкритість до будь-якої практики.

– Заходи процесу. Застосовується безперервний моніторинг готовності беклогу та коефіцієнта обслуговування протягом кожної ітерації. Здійснено/доставлено також контролюється.

– Нові практики розробки програмного забезпечення. Використання нотації Gherkin було введено, щоб краще задовольнити потребу в автоматизованих тестах. Поєднання гнучких, ощадливих методологій і експериментування – це аспекти, які є обов'язковими.

3.3.2 Практики масштабування

Протягом усього процесу дослідження розробки ми визначили серію практик гнучкого масштабування, які використовувала організація. Виявлено сім методів масштабування: Scrum of Scrums (SP1), спільноти практики (SP2), масштабування управління вимогами (SP4), масштабоване планування спринту (SP5), масштабована ретроспектива (SP6), Undine dept. (SP8) і масштабована демонстрація спринту / Review (SP3). Це були найважливіші практики, які дозволили запровадити масштабований гнучкий процес. Команди функцій (SP7) не були серед практик, прийнятих у компаії.

– Масштабована ретроспектива: формат масштабної ретроспективи був результатом кількох ітерацій. Компанія вперше спробувала створити теми під час ретроспектив команди, але не вистачило часу та контексту для визначення основних перешкод між командами. Інший експеримент полягав у тому, щоб дозволити Agile Coaches створювати теми, але це включало лише їхню точку зору. Додавання системи голосування разом із темами, створеними будь-ким, ще більше покращило Масштабовану ретроспективу, оскільки відвідували лише люди, зацікавлені в обговорюваній темі.

– Масштабне планування: обертається навколо так званих віх. Віха визначила бачення, цілі, бізнес і технічні ініціативи, яких компанія хотіла досягти в наступному випуску. Ітерація етапу тривала 16 тижнів із чотирма фазами (відкриття, стратегія, підстратегія та вихід на ринок, розробка). На етапі відкриття були визначені найбільш цінні напрямки продукту, враховуючи нові технології, аналіз ринку та конкуренції. На етапі стратегії були згенеровані та підтверджені рішення на основі етапу відкриття. Підетап підстратегії полягав у

створенні початкової дорожньої карти для нового релізу, визначення будь-яких перешкод, залежностей і ризиків, пов’язаних із дорожньою картою, і визначення кроків для їх пом’якшення. Вихід на ринок використовував дорожню карту з першого підетапу, налаштувавши документ, у якому підсумовано, що новий реліз пропонує, кому та як. Етап розробки стосувався реалізації поставлених раніше цілей за два тижні спринту.

– Демонстрація масштабованого спринту/огляд (так званий ітераційний огляд): проводився кожні два тижні в кінці кожної ітерації розробки. Головною метою було поширити знання серед команд розробників – зустріч була обов’язковою для всіх у компанії. Ще одна мета зустрічі полягала в тому, щоб зібрати попередні відгуки та включити їх на ранній стадії. На зустрічі також оцінювали, чи все доставлено, чи чогось не вистачає. Крім того, обговорювалися подальші кроки та майбутнє продукту.

– Scrum of Scrums: перша зустріч була зустріччю синхронізації для Scrum Masters, директора з розробки та інших спеціалістів, таких як координатор із забезпечення якості. Друга зустріч була для синхронізації всієї хмарної платформи. У ньому взяли участь тренери Agile, директор з розробки та команди розробників. Поточний формат був результатом кількох ітерацій удосконалень, наприклад, обмін знаннями було відокремлено до іншої зустрічі, яка відбулася перед зустрічами SoS.

– Масштабування керування вимогами: управління вимогами між командами не дотримувалося жодного конкретного процесу. Команди спілкувалися та синхронізували елементи, які повинні були виконуватися декількома командами, використовуючи спеціальні зустрічі та відвідуючи інші команди щодня. Команди також можуть проводити деякі командні зустрічі разом з іншими командами. Крім того, між командами можна було обмінюватися членами команди, а також було встановлено процес перегляду між командами. Той самий спеціальний процес використовувався менеджерами з продуктів. Усі функції спочатку були пріоритетними в одному беклозі. Після початкового визначення пріоритетів кожна функція була оброблена в беклог команди, яка

відповідала за впровадження. Якщо більше команд співпрацювали над однією функцією, вони використовували беклоги команди, яка володіла цією функцією.

– Undone Department: компанія використовувала багато різних спеціалізованих команд, які допомагали командам розробників у їхній роботі. Була команда архітектури, команда DevOps, команда безпеки, команда технічних авторів і одна команда взаємодії з користувачем. У кожній команді був один член команди, який також був членом однієї з цих груп експертів.

– Спільноти практиків: компанія використовувала подібну концепцію. Темами цих спільнот були, наприклад, підвищення продуктивності, графічний дизайн, коучинг, постійна інтеграція та автоматичне тестування. Кількість людей, залучених до цих спільнот, коливалася від п'яти до десяти. Спільноти збиралися в середньому кожні два тижні.

3.3.3 Виклики

Під час процесу трансформації було встановлено чотири проблеми, з якими зіткнулася компанія-кейс. Це опір змінам (SC1), проблеми із забезпеченням якості (SC3), інтеграція з негнучкими частинами організації (SC4), надто швидке розгортання.

– Опір змінам: хоча багато людей у компанії вже мали великий досвід гнучкої розробки, все одно були деякі, кому не подобався новий спосіб роботи. Проблема опору також була спричинена тим, що команди розробників, які отримали абсолютну свободу та обрали свій шлях, були змушені змінитися. Команди не хотіли відмовлятися від свого раніше створеного процесу та обраних інструментів.

– Занадто швидке розгортання: були різні ситуації, коли виникали проблеми через поспішне впровадження нового процесу. Використання пілотування або менших поступових змін з однією або двома командами могло б заощадити багато ресурсів компанії.

– Питання забезпечення якості: на початку створення нової хмарної платформи компанії командам розробників була надана абсолютна свобода. Команди розробляли окремі продукти, які ще перебували на етапі валідації. Компанія дозволила командам створити весь процес розробки, вибрати свої інструменти та технології, оскільки не було необхідності уніфікувати процеси, оскільки команди розробників майже не взаємодіяли між собою. Незважаючи на те, що це створило чудову атмосферу в командах і покращило їхню мотивацію, незабаром компанія зрозуміла, що команди розробників неправильно керують своїми процесами. Технічний борг накопичився, а інфраструктура розвитку не була належним чином налаштована.

– Інтеграція попередніх і негнучких частин організації: багато частин організації все ще впроваджували негнучкий процес розробки, оскільки новий гнучкий процес розробки створювався поруч зі старою моделлю розробки попереднього продукту. Людей переводили зі старого процесу розробки на новий у міру необхідності. Однак колишній продукт все ще потребував технічного обслуговування. Наприкінці дослідження компанія спробувала вирішити це, включивши спільні ресурси в кожну команду. Наприклад, кожна команда матиме одного технічного автора.

3.3.4 Фактори успіху

Загалом ми визначили чотири основні фактори успіху, які допомогли компанії запровадити гнучку розробку. Це були: уніфікація поглядів і цінностей (SF2), підтримка керівництва (SF7), корпоративна культура (новий) і попередній досвід гнучкості та економічного використання (новий).

– Культура компанії та попередній досвід гнучкої розробки: багато людей у компанії мали попередній досвід гнучкої розробки, особливо Agile-тренери та керівництво. Тому компанія не страждала від браку знань чи досвіду багато інших компаній знання поширювали Agile Coaches, які використовували методи економії. Вони навчали людей, слухаючи та ставлячи запитання, а не

змушуючи їх змінюватися. Багато людей у компанії були зацікавлені в Agile та Lean загалом. Вони були залучені до Agile-спільноти та стежили за останніми тенденціями у Agile-розробці. Компанія використовувала програми обміну Agile Coaches з іншими організаціями, а також співпрацювала з університетами. Крім того, деякі співробітники презентували результати компанії на лекціях в університеті. Культура компанії також допомогла. Через прозорість люди не відчували спостереження, оскільки середовище було дуже відкритим і невимушеним.

– Виконавче залучення/підтримка керівництва: керівництво ініціювало зміни: це допомогло послідовно прийняти. Усі необхідні ресурси були доступні. Крім того, менеджери не намагалися задушити співробітників, а допомагали їм рости. Керівництво також активно залучалося до команд під час етапу тестування нових процесів. Деякі члени керівництва, наприклад генеральний директор і технічний директор, були членами команди розробників, яка випробувала новий процес. Таке залучення керівників покращило мотивацію інших членів команди, оскільки вони бачили зацікавленість керівництва. Це також допомогло керівництву, оскільки вони краще зрозуміли практику та процеси.

– Уніфікація поглядів і цінностей: компанія визначила спільні цінності: це було досягнуто шляхом запровадження «Маніфесту гнучкості та економії». Маніфест містив дванадцять тверджень про основні цінності та принципи, яких компанія хотіла дотримуватися. Крім того, він містив кілька правил, якими керувався спосіб роботи в компанії. Ці правила визначили пріоритети в організації, наприклад, 1 – міжгрупова співпраця та узгодженість перед автономією та 2 – спеціалізація з можливістю заміни перед розподілом відповідальності команди. Маніфест допоміг компанії змусити прийняти цінності.

3.3.5 Пропозиції, отримані в ході дослідження розробки

Під час процесу дослідження розробки було висунуто декілька пропозицій щодо процесу масштабування, але їх неможливо було оцінити з огляду на часові рамки дослідження. Це пропозиції, які були надані компанії, але вони будуть частиною майбутніх циклів релізу, якщо вважатимуться доречними.

Перша пропозиція полягає в сприянні кращим інженерним практикам і управлінню боргом. Ми побачили, що технічна заборгованість і погана інженерна практика спричинили багато проблем у великомасштабній гнучкій роботі. Тому наявність надійної інженерної практики та управління боргом є одними з передумов для успішної великомасштабної гнучкої розробки. Надійна інфраструктура розробки з належною постійною інтеграцією та автоматизованими тестами дозволяє командам більше зосередитися на новому способі роботи.

Команди також матимуть більше часу, щоб подумати про вдосконалення свого процесу розробки. Якщо технічний борг і погана якість коду надто сильно тиснуть на команди розробників, вони не зможуть зосередитися на інших обов’язках, які несе гнучка розробка. Команди також можуть відчувати загальне розчарування: вони можуть почати пропускати виправлення помилок або повідомляти про помилковий статус своєї роботи. Можна покращити управління боргом, запровадивши візуальну індикацію стану своїх продуктів, щоб визначити, чи продукт готовий до нової функції, чи спочатку потрібно усунути дефекти.

Документ також додає додаткову мотивацію командам для їхнього технічного управління боргом, вводячи елементи гейміфікації. Крім того, було запропоновано додаткове навчання, використання міжкомандного беклогу та команд DevOps. Однак вирішити цю проблему нелегко: на це можуть піти роки.

Друга пропозиція полягає в тому, щоб бути обережними щодо великої спеціалізації без обміну знаннями Комунікація відіграє ключову роль у групах гнучких розробників та між ними. Компанія, яка розглядала справу,

використовувала спеціалізацію в командах розробників. Надмірна спеціалізація в командах може значно зашкодити командній роботі і створити конкурентне середовище. Це знизило загальну продуктивність команд і спричинило кілька інших проблем. Команди не мали чіткого, єдиного уявлення про розробку і командний план, оскільки спеціалізовані члени команди більше зосереджувалися на своїй конкретній сфері. Члени команди виконували лише свою роботу. Вони не розуміли роботи інших членів команди, тому розподілене прийняття рішень було неможливим. Крім того, окремі члени команди стали незамінними . Тому вони не хотіли ділитися своїми знаннями та ноу-хау зі своїми колегами. Ми запропонували розглянути можливість завчасного впровадження процесу обміну знаннями між членами команди, наприклад, за допомогою семінарів або створення CoP. На наступних етапах існує ризик того, що члени команди можуть вважати свої знання надто важливими, щоб ними ділитися.

Третя пропозиція полягає в тому, щоб використовувати команди функцій і розділити продукт на зони розробки, орієнтовані на клієнта. І LeSS, і SaFE сприяють командам функцій. Однією з характеристик команд функцій є те, що вони є міжкомпонентними. Причина створення міжкомпонентних команд полягає в тому, що вони дозволяють набагато легше масштабувати гнучку розробку, оскільки команди функцій мінімізують залежності між командами. Вони також забезпечують набагато гнучкішу структуру організації, оскільки команди можуть працювати над будь-яким компонентом і не обмежені технічними знаннями. Таким чином, команди можна легко переміщати між різними напрямками розробки. Оскільки немає технічних обмежень, команди можна розділити на зони розробки, які зосереджені навколо клієнта: це дозволяє командам бачити всі функції з точки зору клієнта, а не лише частину функцій, які впроваджуються в їхніх компонентах.

Впровадити команди функцій непросто. Пропонований спосіб переходу до груп функцій полягає в тому, щоб створити одну команду функцій із осіб, які знають про кожен компонент. Члени команд таких груп функцій діляться своїми знаннями та працюють разом над функцією, яка охоплює декілька компонент.

Організація не повинна додавати додаткові команди функцій, доки перша не буде добре працювати. Однак існує ризик, пов’язаний із цією стратегією, коли члени першої команди функцій мали центральні ролі у своїх попередніх командах.

Четверта пропозиція полягає в підтримці та просуванні спільнот практиків: CoPs використовуються як у SAFe, так і в LeSS.3 Вони є інструментом, який дозволяє добровільну співпрацю в довільній сфері. Найпоширенішим використанням CoPs є обмін знаннями. Таким чином, вони можуть бути використані, щоб забезпечити кращий перехід до функційних команд, сприяти кращим інженерним практикам або уникнути надмірної спеціалізації в командах. CoPs є самоорганізованими і не є частиною процесу чи організації. Однак, щоб створити робочі CoPs, організація повинна активно підтримувати та просувати їх, виділяючи ресурси: це означає, що організація повинна забезпечити фасилітаторів, IT-інфраструктуру та бюджет. CoP також потребує здібного лідера, який координує та мотивує спільноту. Тому організації повинні намагатися знайти, мотивувати та підтримувати лідерів CoP. Є багато способів використання CoPs: з метою тренування, покращення майстерності написання програмного забезпечення, дослідження нових технологій, покращення роботи та покращення координації та дизайну між командами.

Остання пропозиція полягає в кращому розгортанні індикаторів і показників для вимірювання прогресу процесу розробки: дослідження існуючих впровадження Agile показали, що впровадження є безперервним процесом з довгостроковою метою. Ця трансформація вимагає зусиль, ресурсів і часу. Тому існує потреба в кращому розумінні та оцінці. Цього можна досягти шляхом кількісного вимірювання зміни. Трансформацію також необхідно поступово адаптувати до конкретного контексту організації. Таким чином, показники можна використовувати як дійсний аргумент для цього коригування процесу. Багато організацій намагалися знайти значущі показники для вимірювання своєї гнучкої трансформації. Причина полягає в тому, що в традиційному каскадному процесі розробки існують чіткі вхідні та вихідні дані потоків ресурсів процесу.

З іншого боку, гнучка розробка має набагато сильніший характер: потоки ресурсів процесу можуть змінюватися щодня. Тому знайти відповідні показники дуже складно. Однак існують показники, які можна використовувати для вимірювання гнучкого процесу трансформації в різних сферах, наприклад зміна швидкості реагування, зміна пропускної здатності, зміна розподілу робочого процесу та зміна якості. Наявність таких показників і індикаторів може бути корисним для відстежувати процес адаптивного масштабування.

3.4 Результати дослідження процесу розробки

Загалом ми визначили багато практик, проблем і факторів успіху для масштабування гнучкості у великих організаціях (рис. 3.3, на якому ми порівнюємо висновки з літератури та частини дослідження розробки). Ми побачили, що для успішного масштабування Agile у великій компанії не потрібно дотримуватися певної схеми. Навпаки, кейс-компанія пристосувала процес розробки до потреб, зберігаючи при цьому основні цінності та принципи гнучкої розробки.

Трансформація була в основному успішною завдяки гнучкому мисленню та попередньому досвіду гнучкого розвитку осіб, які відповідали за процес трансформації. Розповсюдження нового мислення за допомогою коучингу з принципами економії дозволило поширити нові цінності іншим членам команди. Попередній досвід інших прикладів гнучкого масштабування може слугувати корисним посиланням для оцінки кількох альтернатив – однак результати здаються дуже специфічними для контексту. Фактори, які ми ідентифікували, загалом були визнані позитивними для масштабованості гнучких процесів, але відносно легко знайти випадки, коли сприятлива практика не є успішною в інших контекстах. Тим не менш, ми вважаємо, що фактори, які ми витягли як з огляду літератури, так і з процесу дослідження дій, разом із узагальненим досвідом, можуть бути цінними для інших організацій, які потребують масштабування процесів розробки програмного забезпечення.

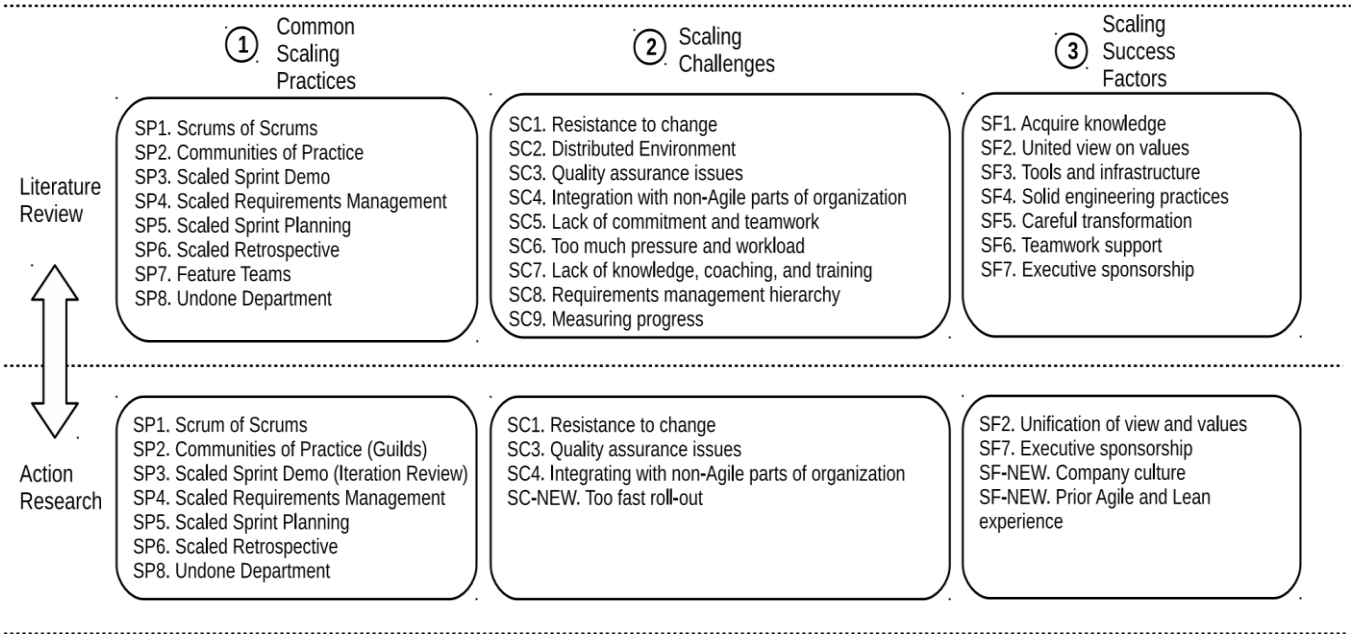

Рисунок 3.3 – Загальні результати дослідження

Ми визначили виклики та фактори успіху «великомасштабного» гнучкого впровадження в дослідженні дій у компанії. Ми з'ясували, що компанія зіткнулася з проблемами, факторами успіху, подібними до тих, які ми визначили в літературі. Найбільшими проблемами, з якими зіткнулася компанія, були стійкість до змін (SC1), проблеми із забезпеченням якості (SC3), інтеграція попередніх і негнучких частин організації (SC4) і надто швидке розгортання.

Існували чотири основні фактори успіху, які були найбільш сприятливими для впровадження в конкретну компанію: уніфікація поглядів і цінностей (SF2), виконавче спонсорство / підтримка керівництва (SF7), культура компанії (новий) і попередній досвід Agile та Lean (новий).

Основними прийнятими практиками були: Scrum of Scrums (SP1), Спільноти практиків (SP2), масштабування управління вимогами (SP4), планування масштабованого спринту (SP5), масштабована ретроспектива (SP6), Undone dept (SP8) і масштабована демонстрація спринту (SP3).

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Ергономічний аналіз умов праці. Система "людина-машина"

На певному етапі свого розвитку для задоволення своїх зростаючих матеріальних і духовних потреб людина починає створювати штучні знаряддя праці – машини. Одержавши у своє розпорядження величезні запаси енергії, нову техніку й технології, вона змінила своє життя, але разом з тим постала перед складним завданням – забезпечити ефективне, стійке та безпечне керування цією технікою.

При вирішенні завдань пов’язаних з поліпшенням умов праці, необхідне детальне вивчення системи „людина-машина” (СЛМ). СЛМ – це складна багатофункціональна система, яка включає в себе людський і технічний фактори (рис. 4.1) і має такі складові:

– машина – усе те, що штучно створено руками людини для задоволення своїх потреб (технічні пристрої, інформаційне забезпечення);

– людина – людина-оператор, при взаємодії з машиною виконує деякі функції для досягнення поставленого завдання;

– навколишнє середовище – визначається такими параметрами, як освітленість, шум, випромінювання, температура, вологість тощо;

– робоче місце – окреслюється положенням оператора при виконанні своїх обов’язків;

– органи керування (ОК) – за допомогою їх людина керує об'єктами;

– засоби відображення інформації (ЗВІ) – завдяки їм людина слідкує за станом машини (виробничого процесу).

Одним із важливих завдань СЛМ є розподіл функцій між людиною і машиною, який повинен ураховувати їх можливості. Однак загальне рішення складно отримати, оскільки кожна система характеризується своїми особливостями. На основі порівняння можливостей людини і машини в системах керування можна запропонувати наведений далі варіант розподілу функцій.

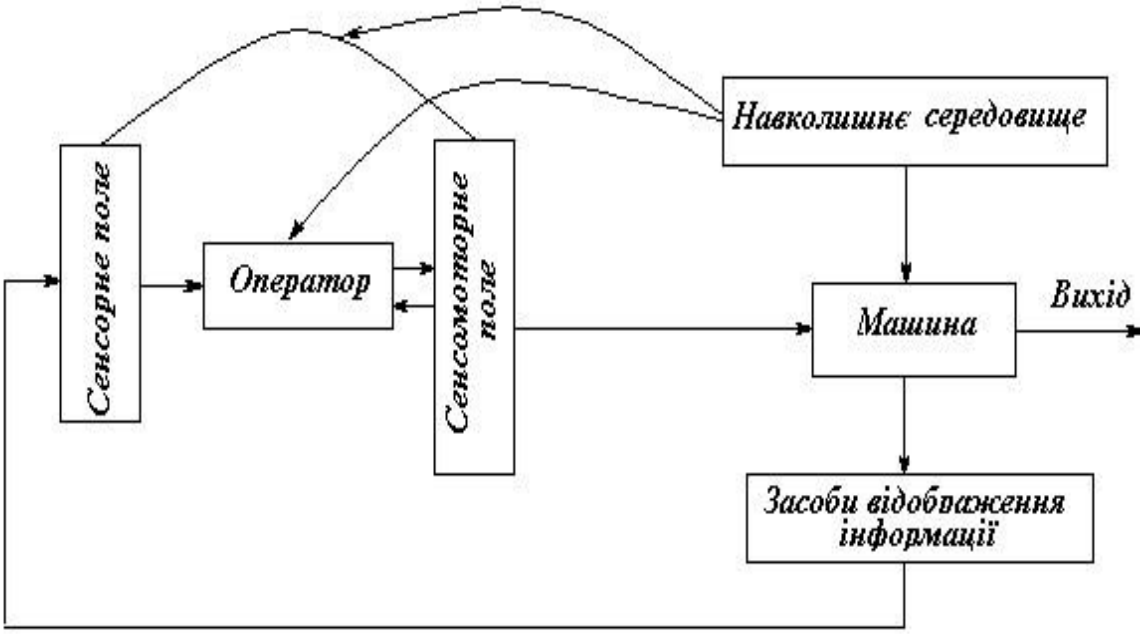

Рисунок 4.1 – Структурна схема системи „людина-машина”

Людина виконує такі функції:

– індуктивно мислить, тобто приймає рішення на базі неповної інформації, узагальнення різних факторів, доповнюючи інформацію з власного досвіду;

– розпізнає ситуацію в цілому за її окремими характеристиками, а також за не повною інформацією про неї;

– вирішує завдання, стосовно яких відсутні правила;

– вибирає шляхи вирішення завдань в умовах, що швидко змінюються. Машині доцільно передати такі функції:

– виконання громіздких математичних розрахунків та вибір відомих варіантів розв’язання;

– збереження великої кількості інформації;

– здійснення одноманітних операцій за відомим алгоритмом;

– виконання швидких дій у відповідь на певну команду.

Ці рекомендації мають узагальнений характер і у кожному конкретному випадку визначальними є експеримент з моделювання конкретної системи та умов функціонування, а також застосування певних принципів. Тому дуже

важливо оцінити умови, в яких буде працювати людина, щоб, виходячи з них, розподілити обов’язки.

Процеси приймання, переробки інформації та прийняття рішень і виконання оператором керуючих дій поєднанні в цілісну діяльність, яка полягає у гарантуванні функціонування СЛМ. Для ефективного забезпечення роботи машини людина-оператор повинна зручно себе відчувати. Таке можливо при виконанні певних вимог, які ставляться до машини і навколишнього середовища, а саме: до розміщення засобів відображення інформації та засобів керування, робочого місця, а також до освітлення, клімату, шуму, вібрації, що можуть впливати як на фізичний стан людини, так і на протікання технологічного процесу.

Згідно з ДСТУ 17025 рівень якості продукції визначається сукупністю операцій, які включають вибір номенклатури показників якості оцінюваної продукції, визначення їх величин і зіставлення з базовими. На основі ергономічної оцінки виробничого устаткування можна скласти портрет промислового підприємства, тобто описати організацію процесу виготовлення продукції, характеристику тієї частини основних фондів виробництва, яка безпосередньо впливає на якість продукції і залежить від людського фактору.

Класифікація показників ергономічної устаткування:

– антропометричні (висота, ширина, глибина пульта, висота розміщення стільниці, розміщення ЗВІ та ОК, характеристики крісла людини-оператора; досяжність ОК; показники відповідності ОК формі і розмірам частин тіла людини тощо);

– біомеханічні (зусилля, величина, напрямок переміщення ОК, частота використання ОК);

– психофізіологічні (характеристики відповідності техніки зоровому і слуховому аналізаторам людини);

– психологічні (показники відповідності техніки можливостям людини стосовно прийому, обробки інформації та прийняття рішень).

Оцінка починається із складання плану проведення досліджень. Основні етапи таких досліджень зазвичай виконуються за такою схемою:

а) ознайомлення з призначенням, метою системи, завданнями та основними вимогами до неї;

б) побудова структурної схеми, що відтворює зв’язки окремих підсистем, потоки інформації і хід регулювання. При цьому окремо виділяються ланцюги, де задіяна людина-оператор з позначенням прямих і зворотних зв’язків у СЛМ (інтенсивність зв’язків і їх відносна важливість);

в) оцінка середовища, в якому система функціонує, і його вплив на досліджувану систему СЛМ;

г) опис функцій системи і її підсистем для всіх режимів роботи (включаючи малоймовірні аварійні ситуації). При визначенні функцій системи слід зазначити, які операції найважливіші і що являє собою динамічна структура системи, тобто які зрушення виникають в окремих підсистемах при керуванні, дії перешкод і т. ін.

д) детальна ергономічна оцінка робочого місця;

е) оцінка засобів відображення інформації та органів керування;

ж) розгляд функцій операторів для нормального режиму роботи й окремо для екстремальних ситуацій.

На підставі всіх вищезазначених дій формується висновок про надійність і ефективність системи й даються рекомендації щодо модернізації або вдосконалення окремих підсистем, вузлів або всього приладу.

При вирішенні завдань, пов’язаних з пристосуванням умов праці до людини, необхідне детальне вивчення системи „людина-машина”, що являє собою складну багатофункціональну систему і включає: людину, машину, навколишнє середовище, органи керування, засоби відображення інформації, робоче місце. Основне завдання – забезпечити максимальну продуктивність при мінімальних затратах енергії. Для цього потрібно оцінити можливості людини, з’ясувати фактори, які погіршують працездатність, та забезпечити відповідне розміщення засобів відображення інформації, органів керування на робочому

місці, з метою мінімізації їх впливу як на фізичний стан людини, так і на протікання технологічного процесу.

4.2 Оцінка хімічної обстановки та розрахунок аварії на підприємстві із зберіганням аміаку

У комплексі заходів захисту населення та об'єктів господарювання від наслідків надзвичайних ситуацій (НС) значне місце займає оцінка хімічної обстановки. Оцінка обстановки в загальному плані включає визначення:

- масштабу та характеру НС;
- заходів, які необхідні для захисту населення;
- доцільних дій при ліквідації НС;
- оптимального режиму роботи об'єкта господарювання в умовах НС.

Необхідність оцінки хімічної обстановки випливає з небезпеки ураження людей сильно діючими отруйними речовинами (СДОР), що потребує швидкого втручання, враховуючи її вплив на організацію рятувальних та невідкладних аварійно-відновлюваних робіт, а також на виробничу діяльність об'єкту господарювання в умовах хімічного зараження.

Масштаби та ступінь хімічного зараження місцевості залежать від кількості СДОР, їх складу, відстані від місця аварії, метеоумов.

Оцінка хімічної обстановки включає визначення:

- розмірів зон хімічного зараження;
- часу підходу зараженого повітря до певного рубежу (об’єкту);
- часу вражаючої дії СДОР;
- вибору найбільш доцільних варіантів дій, за яких виключається ураження людей.

Основні вихідні дані при оцінці хімічної обстановки:

- тип СДОР;
- кількість СДОР;
- метеоумови та топографічні умови місцевості;

– ступінь захищеності людей, укриття, техніки та майна.

Задача оцінки хімічної обстановки.

На об'єкті зруйнована необвалована ємність з 5-ма тонами аміаку. 50% із 600 працюючих забезпечені протигазами. На відстані 2 км розміщений населений пункт з 1000 мешканцями, на 40% забезпечених протигазами. Місцевість відкрита. Інверсія, швидкість вітру 1 м/с.

Визначити:

1. Глибину, ширину площі розливу і ЗХЗ.
2. Час підходу зараженого повітря до населеного пункту.
3. Можливі страти серед працюючих і мешканців, їх структуру.

Розв'язок.

Визначення глибини зони хімічного забруднення.

При оптимальних умовах для розповсюдження хмари забрудненого повітря – місцевість відкрита, ємність необвалована, швидкість вітру 1 м/с, ступінь вертикальної стійкості повітря – інверсія, глибина розповсюдження хмари, забрудненої НХР з поправочним коефіцієнтом на інверсію 5.

Для аміаку глибина розповсюдження хмари становить 0,7 км. З врахування поправки для інверсії $Г = Г_1 \times 5 = 0,7 \times 5 = 3,5$ км.

Поправочних коефіцієнтів на обваловану ємність, швидкість вітру та закриту місцевість не використовуємо відповідно до умови задачі.

Визначення ширини зони хімічного забруднення.

Визначимо ширину зони хімічного забруднення за формулою:

$$Ш = К_ш \times Г \text{ (км)},$$

де

$К_ш$ –коефіцієнт ширини ЗХЗ, який при інверсії =0,03, ізотермії – 0,15, конвекції – 0,8

Тому, $Ш = 0,03 \times 3,5$ км $= 0,105$ км.

Площа зони можливого хімічного забруднення

Площу зони хімічного забруднення (ЗХЗ) конкретно до умов забруднення визначаємо за формулою:

$$S_{змхз}=0,5\times Г \times Ш \ (км^2)$$

Отже, $S = 0,5 \times 3,5 \times 0,105 = 0,18$ км^2

Площа розливу НХР.

У випадку зруйнування не обвалованої ємності визначається площа розливу НХР за формулою

$$S_р= G : (d \times h) \ (м^2),$$

де

G – кількість розлитої НХР (т) ;

h – висота шару розливу НХР – 0,05 м ;

d – густина НХР (т/м^2), для аміаку – 0,68 т/м^3.

Тому $S_р= 5 / (0,68 \times 0,05) = 147,1$ м^2.

Визначення часу підходу хмари забрудненого повітря.

Час підходу хмари забрудненого повітря до населеного пункту визначаємо за формулою :

$$t_{підх}=R : (60 \times W) \ (хв),$$

де

R – відстань від джерела забруднення до заданого об’єкту (м) ;

W – середня швидкість переміщення хмари забрудненого повітря (м/с), яку знаходимо за формулою W=(1,5…2)V,

де

V – швидкість вітру в приземному шарі повітря м/с; 1,5 при R<10км (в нас відстань до населеного пункту становить 2 км за умовою задачі);

Отже, $t_{підх} = 2000 /(60 \times 1,5 \times 1) = 22,2$ хв.

Отже, час, за який необхідно провести оповіщення та евакуацію населення із зони хімічного забруднення, повинен бути меншим, ніж 22,2 хв.

Визначення тривалості уражаючої дії аміаку.

Час уражаючої дії НХР залежить від часу її випаровування із забрудненої поверхні, площі розливу та швидкості вітру.

Тривалість уражаючої дії первинної хмари складає 20-30 хв., тривалість уражаючої дії вторинної хмари визначається часом повного випаровування ОР із забрудненої ділянки.

Тривалість уражаючої дії аміаку знаходимо за формулою:

$$T = t_в \times К_в \text{ (год)},$$

де $t_в$ – час випаровування НХР (табл.2)

$К_в$ – поправочний коефіцієнт на час випаровування НХР при швидкості вітру понад 1 м/с (примітка до табл.2). Згідно умови задачі $К_в = 1$.

Згідно таблиці 2 і примітки до неї знаходимо час випаровування ($t_в$) аміаку, який при швидкості вітру 1 м/с = 1,2 год. для необвалованої ємності.

Тому $T = t_в = 1{,}2$ год.

Визначення можливих втрат в осередку хімічного ураження.

Можливі втрати в ОХУ залежать від умов розташування людей, захищеності їх засобами індивідуального захисту (забезпеченості їх протигазами) і визначаються за формулою :

$$В = NB_1 \text{ (чол)},$$

де N – кількість людей (чол);

$В_1$ – можливі втрати людей в осередку ураження (%) в залежності від умов їх розташування і забезпеченості протигазами (табл.3).

У нашій задачі для відкритої місцевості при забезпеченні протигазами 50% $В_1$=50%

Тому В = 600 чол × 50% = 300 чол.

Структуру втрат населення визначаємо згідно примітки до табл.3 у [37].

Легко уражені $В_{лу} = 300 \times 25\% = 75$ чол.

Уражені середнього і тяжкого ступеня $В_{ст} = 300 \times 40\% = 120$ чол.

Смертельно уражені $В_{су} = 300 \times 35\% = 105$ чол.

Результати оцінки хімічної обстановки занесено у таблицю 4.1.

Таблиця 4.1 – Результати розрахунків

Джерела забруднення	Вид НХР	Кількість НХР, м	Глибина ЗХЗ, км	Ширина ЗХЗ, км	Площа ЗХЗ, ЗМХЗ, км^2	Площа зони розливу, м ^{2}	Час підходу забрудненого повітря, хв	Тривалість уражаючої дії НХР, год.	Можливі втрати, % (чол.)
Зруйнована необвалована ємність	Аміак	5	3,5	0,105	0,18	147,1	22,2	1,2	300

Для жителів населеного пункту при забезпеченні протигазами 40% B_1=58%.

Тому В = 1000 чол × 58% = 580 чол.

Структуру втрат населення визначаємо згідно примітки до табл.3 у [37].

Легко уражені $B_{лу}$ = 580 × 25% = 145 чол.

Уражені середнього і тяжкого ступеня $B_{ст}$ = 580 × 40% = 232 чол.

Смертельно уражені $B_{су}$ = 300 × 35% = 203 чол.

Висновок: Оскільки населений пункт потрапляє в зону хімічного ураження, тривалість уражаючої дії аміаку становить 1,2 год., час підходу хмари забрудненого повітря 22,2 хв., тому необхідно терміново провести оповіщення мешканців населеного пункту про загрозу хімічного ураження та необхідність негайно одягнути протигази і евакуюватись в напрямку, перпендикулярному до руху хмари забрудненого повітря.

ВИСНОВКИ

Мета цієї роботи полягала в тому, щоб зібрати більше доказів щодо факторів впливу на масштабування Agile-процесів розробки програмного забезпечення всередині компаній, щоб краще зрозуміти практики, які найкраще працюють у певному контексті. За допомогою огляду досліджень великомасштабної гнучкості в організацій, ми визначили практики, проблеми та фактори успіху. Початковий огляд літератури був вхідним матеріалом для проведення дослідницького дослідження в компанії, що займається розробкою програмного забезпечення, яка перебувала в процесі масштабування процесів розробки гнучкого програмного забезпечення. Ми вивчили конкретні практики масштабування, які використовувала компанія, які фактори успіху пережила компанія та з якими проблемами вона зіткнулася. Таким чином, ми внесли свій внесок у існуючий набір досліджень щодо гнучкого масштабування за допомогою досвіду, отриманого в результаті дослідження дії.

Культура гнучкості в компанії та попередній досвід гнучкості й економічності, підтримка керівництва, уніфікація поглядів і цінностей виявилися ключовими факторами успіху під час процесу дослідження дій. Опір змінам, занадто швидке розгортання, проблеми із забезпеченням якості та інтеграція з попередніми негнучкими частинами організації виявилися критичними проблемами в процесі масштабування. Загалом, позитивним результатом процесу дослідження дій стало перехресне збагачення ідей із літератури практичним контекстом компанії.

Ми згадали кілька викликів, які пов’язані з масштабним Agile. Ці виклики потребують більш детального дослідження у співпраці з великими компаніями, щоб знайти відповідні рішення. Необхідні додаткові дослідження, щоб знайти причинно-наслідкові зв’язки між факторами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 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.

2. Kharchenko, A., Raichev, I., Bodnarchuk, I., & Matsiuk, O. (2021). The Survey of Global Software Design Processes. In 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology, PIC S and T 2021-Proceedings (pp. 291-294).

3. Гузеляк, О., Шевчук, Ю., Береженко, Б. М., & Боднарчук, І. О. (2022). Програмна архітектура в розподілених командах гнучких проєктів. Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології “Тернопільського національного технічного університету імені Івана Пулюя, 110-112.

4. Волович, В., Береженко, Б. М., & Боднарчук, І. О. (2022). Задача проєктування програмної архітектури в процесах забезпечення якості. Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології “Тернопільського національного технічного університету імені Івана Пулюя, 104-106.

5. Бугай, В. П., & Боднарчук, І. О. (2020). Оцінювання програмної архітектури при гнучких методах розробки програмних систем. Матеріали Міжнародної науково-технічної конференції „Фундаментальні та прикладні проблеми сучасних технологій “до 60-річчя з дня заснування Тернопільського національного технічного університету імені Івана Пулюя та 175-річчя з дня народження Івана Пулюя, 152-153.

6. Боднарчук, І., Харченко, О., Хоміцький, Б., & Шимчук, Г. (2019). Проектування архітектури програмних систем в проектах з гнучкими методами

управління. Матеріали XXI наукової конференції Тернопільського національного технічного університету імені Івана Пулюя, 46-48.

7. Ковальчук, Р. (2022). Адаптація технологій контролю та управління проєктами в умовах невизначеності на основі Agile методології розробки програмного забезпечення. Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології “Тернопільського національного технічного університету імені Івана Пулюя, 118-118.

8. Вивюрка, А., Мариненко, Л., Нога, О., Хоміцький, Б., & Ланевич, Т. (2023). Дослідження ефективності процесів CI/CD в гнучких технологіях розробки програмного забезпечення. Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “, 402-403.

9. Eklund Ulrik, Berger Christian. Scaling agile development in mechatronic organizations: a comparative case study in Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track:173–182IEEE Press 2017.

10. Boehm, B., & Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. IEEE software, 22(5), 30-39.

11. Rolland, K. H. (2016, May). Scaling across knowledge boundaries: A case study of a large-scale agile software development project. In Proceedings of the scientific workshop proceedings of xp2016 (pp. 1-5).

12. Saddington, P. (2012, August). Scaling agile product ownership through team alignment and optimization: a story of epic proportions. In 2012 Agile conference (pp. 123-130). IEEE.

13. Paasivaara, M., & Lassenius, C. (2016, May). Challenges and success factors for large-scale agile transformations: A research proposal and a pilot study. In Proceedings of the Scientific Workshop Proceedings of XP2016 (pp. 1-5).

14. Chow, T., & Cao, D. B. (2008). A survey study of critical success factors in agile software projects. Journal of systems and software, 81(6), 961-971.

15. Rossi, B., Russo, B., & Succi, G. (2012). Adoption of free/libre open source software in public organizations: factors of impact. Information Technology & People, 25(2), 156-187.

16. Hobbs, B., & Petit, Y. (2017). Agile methods on large projects in large organizations. Project Management Journal, 48(3), 3-19.

17. Brydon-Miller, M., Greenwood, D., & Maguire, P. (2003). Why action research? Action research, 1(1), 9-28.

18. Easterbrook, S., Singer, J., Storey, M. A., & Damian, D. (2008). Selecting empirical methods for software engineering research. Guide to advanced empirical software engineering, 285-311.

19. Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. Journal of Systems and Software, 119, 87-108.

20. Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008, June). Systematic mapping studies in software engineering. In 12th international conference on evaluation and assessment in software engineering (EASE). BCS Learning & Development.

21. Dingsøyr, T., Fægri, T. E., & Itkonen, J. (2014). What is large in large-scale? A taxonomy of scale for agile software development. In Product-Focused Software Process Improvement: 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014. Proceedings 15 (pp. 273-276). Springer International Publishing.

22. Duncan, S. (2018). SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering. 2017. Richard Knaster and Dean Leffingwell. New York: Addison-Wesley Professional. 416 pages.

23. Alqudah, M., & Razali, R. (2016). A review of scaling agile methods in large software development. International Journal on Advanced Science, Engineering and Information Technology, 6(6), 828-837.

24. Larman, C., & Vodde, B. (2010). Practices for scaling lean & Agile development: large, multisite, and offshore product development with large-scale scrum. Pearson Education.

25. Ebert, C., & Paasivaara, M. (2017). Scaling agile. IEEE Software, 34(6), 98-103.

26. One V. 11th annual state of agile survey tech. rep. Technical report, Version One 2017.

27. Larman, C., & Vodde, B. (2010). Practices for scaling lean & Agile development: large, multisite, and offshore product development with large-scale scrum. Pearson Education.

28. Sutherland, J. (2001). Inventing and Reinventing SCRUM in five Companies. Cutter IT journal, 14(21), 5-11.

29. Frank, A., & Hartel, C. (2009, August). Feature teams collaboratively building products from ready to done. In 2009 Agile Conference (pp. 320-325). IEEE.

30. Ambler, S., & Lines, M. (2013). Going beyond scrum disciplined agile delivery," Disciplined Agile Consortium. Disciplined Agile Consortium.

31. Bittner, K., Kong, P., Naiburg, E., & West, D. (2017). The Nexus Framework for scaling Scrum: Continuously Delivering an integrated product with multiple Scrum teams. Addison-Wesley Professional.

32. Alqudah, M., & Razali, R. (2016). A review of scaling agile methods in large software development. International Journal on Advanced Science, Engineering and Information Technology, 6(6), 828-837.

33. Paasivaara, M., & Lassenius, C. (2016, August). Scaling scrum in a large globally distributed organization: A case study. In 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE) (pp. 74-83). IEEE.

34. Paasivaara, M., Lassenius, C., Heikkilä, V. T., Dikert, K., & Engblom, C. (2013, August). Integrating global sites into the lean and agile transformation at ericsson. In 2013 IEEE 8th International Conference on Global Software Engineering (pp. 134-143). IEEE.

35. Vallon, R., Strobl, S., Bernhart, M., & Grechenig, T. (2013). Inter-organizational co-development with scrum: experiences and lessons learned from a distributed corporate development environment. In Agile Processes in Software Engineering and Extreme Programming: 14th International Conference, XP 2013, Vienna, Austria, June 3-7, 2013. Proceedings 14 (pp. 150-164). Springer Berlin Heidelberg.

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. Методичні вказівки до практичного заняття і самостійної роботи з курсу «Техноекологія та цивільна безпека» частина «Цивільна безпека» на тему «Шляхи і способи підвищення стійкості роботи промислового об’єкта» для студентів всіх спеціальностей денної та заочної (дистанційної) форм навчання / укл. : В.С. Стручок . - Тернопіль : ТНТУ імені Івана Пулюя, 2023. - 26 с.

ДОДАТКИ

ДОСЛІДЖЕННЯ ВАРІАНТІВ ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА В ІНФОРМАЦІЙНИХ ІНТЕРАКТИВНИХ АНАЛІТИЧНИХ ПАНЕЛЯХ

23. **В. В. Никитюк, А. В. Орловська, А. К. Карнаухов, В. К. Крилов** **387**
АНАЛІЗ БІОМЕТРИЧНОЇ СИСТЕМИ СИЛУЕТА КОРИСТУВАЧІВ

24. **М. О. Слободян** **389**
КІБЕРФІЗИЧНА СИСТЕМА ДЛЯ ЕКСПРЕС-АНАЛІЗУ ПСИХОФІЗІОЛОГІЧНОГО СТАНУ НА ОСНОВІ ПУЛЬСОКСИМЕТРІЇ

25. **О. Р. Оробчук, І. М. Кивацький** **391**
АНАЛІЗ РИЗИКІВ ТА ВРАЗЛИВОСТЕЙ В СИСТЕМАХ КЕРУВАННЯ РОЗУМНИМ БУДИНКОМ

26. **О. В. Палка** **392**
ОГЛЯД КРІ РОЗУМНОГО МІСТА

27. **Т. А. Липак** **393**
ЗАСТОСУВАННЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ В ЦИФРОВОМУ ЗБЕРЕЖЕННІ КУЛЬТУРНОЇ СПАДЩИНИ

28. **Т. О. Крамар, О. М. Дуда** **395**
МЕТОДИ РЕКОНСТРУКЦІЇ РЕАЛЬНИХ ОБ'ЄКТІВ У ЦИФРОВОМУ СЕРЕДОВИЩІ

29. **М. О. Стрембіцький, О. І. Стрембіцька, І. І. Олійник, В. В Батюк, В. М. Слободян** **397**
АНАЛІЗ МЕТОДІВ РЕАЛІЗАЦІЇ ЗВ'ЯЗКУ МІЖ ВУЗЛАМИ СТОМАТОЛОГІЧНОЇ УСТАНОВКИ

30. **В. Семенюк, В. Сенківський, В. Чичук, Б. Хоміцький, О. Кучма** **399**
ОГЛЯД ІНСТРУМЕНТІВ БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІЇ В СУЧАСНИХ ПРОЄКТАХ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

31. **А. Вивюрка, Л. Мариненко, О. Нога, Б. Хоміцький, Т. Ланевич** **402**
ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСІВ CI/CD В ГНУЧКИХ ТЕХНОЛОГІЯХ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

32. **Д. С. Матюк, М. В. Деркач** **404**
ОЦІНКА СПЕКТРАЛЬНОЇ ЩІЛЬНОСТІ ПОТУЖНОСТІ ЕЕГ СИГНАЛУ

33. **М. В. Онай, А. І. Северін** **406**
КОМПЛЕКСНИЙ ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ЗБЕРЕЖЕННЯ ПРИВАТНОСТІ В МАШИННОМУ НАВЧАННІ

34. **А. С. Хом'як** **408**
ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОБРОБКИ В РЕАЛЬНОМУ ЧАСІ У СЛУЖБАХ ЧАТ-БОТІВ ЧЕРЕЗ ІНТЕГРАЦІЮ ЧЕРГИ ЗАПИТІВ ДЛЯ РОЗПОДІЛЕННЯ НАВАНТАЖЕННЯ

35. **Ю. Ю. Дзюбак, Ю. З. Лещишин** **410**
ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ СТВОРЕННЯ КОМП'ЮТЕРИЗОВАНОЇ СИСТЕМИ ОБЛІКУ УСПІШНОСТІ ТА ВІДВІДУВАННЯ ЗАНЯТЬ ЗДОБУВАЧАМИ ОСВІТИ ПІД ПОТРЕБИ ОКРЕМОГО ВИЩОГО НАВЧАЛЬНОГО ЗАКЛАДУ

36. **Ю. Ю. Дзюбак, Ю. З. Лещишин** **411**
КОМП'ЮТЕРИЗОВАНА СИСТЕМА «CLASSBOOK» ДЛЯ ОБЛІКУ УСПІШНОСТІ ТА ВІДВІДУВАННЯ ЗАНЯТЬ ЗДОБУВАЧАМИ ОСВІТИ

37. **В. І. Ковальчук** **413**
ОГЛЯД СУЧАСНИХ ТЕЛЕКОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ В МЕДИЧНІЙ СФЕРІ

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.