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: Дослідження процесів прийняття рішень стосовно архітектури програмного продукту для гнучких моделей життєвого циклу

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

(підпис) Флейтута А.А. (прізвище та ініціали)

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

Нормоконтроль (підпис) Мацюк О.В. (прізвище та ініціали)

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

Рецензент (підпис) Оробчук О.Р. (прізвище та ініціали)

Тернопіль
2021

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)
«21» вересня 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

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

студенту Флейтута Анастасія Андріївна
(прізвище, ім'я, по батькові)

1. Тема роботи **Дослідження процесів прийняття рішень стосовно архітектури програмного продукту для гнучких моделей життєвого циклу**

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

Затверджені наказом ректора від «*28*» *жовтня* 2021 року № *4/7-907*

2. Термін подання студентом завершеної роботи 20 грудня 2021 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП; 1 ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ ДОСЛІДЖЕННЯ; 1.1 Гнучка архітектура;1.2 Автономні команди розробників; 1.3 Розробка моделі менеджменту Spotify; 1.4 FinTech; 2 МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ; 2.1 Об'єкт дослідження; 2.2 Збір даних; 2.3 Аналіз даних; 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЦЕСУ МЕНЕДЖМЕНТУ; 3.1 Підготовка до впровадження змін; 3.2 Оцінка пропонованого підходу на основі введення змін; 3.3 Адаптація підходу Heterogeneous Tailoring для управління архітектурними рішеннями; 3.4 Переваги та проблеми підходу архітектурного управління; 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ; 4.1 Забезпечення психофізіологічного розвантаження працівників в ІТ-компаніях; 4.2 Попередження аварій на виробництвах із застосуванням хлору; ВИСНОВКИ; СПИСОК ВИКОРСИТАНИХ 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	Дмитроца Л.П., доц.		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викл.		

7. Дата видачі завдання 21 вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	21.09.21-27.09.21	*Виконано*
2.	Підбір наукових джерел по темі роботи	28.09.21-04.10.21	*Виконано*
3.	Переклад та опрацювання наукових джерел	05.10.21-11.10.21	*Виконано*
	по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо огляду атак на	12.10.21-18.10.21	*Виконано*
	комп'ютерні системи		
5.	Оформлення першого розділу	19.10.21-25.10.21	*Виконано*
6.	Оформлення другого розділу	26.10.21-01.11.21	*Виконано*
7.	Оформлення третього розділу	02.11.21-08.11.21	*Виконано*
8.	Виконання завдання до підрозділу «Охорона праці»	09.11.21-15.11.21	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	16.11.21-22.11.21	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	23.11.21-29.11.21	*Виконано*
11.	Нормоконтроль	30.11.21-05.12.21	*Виконано*
12.	Перевірка на плагіат	05.12.21	*Виконано*
13.	Попередній захист кваліфікаційної роботи	14.12.21	*Виконано*
14.	Захист кваліфікаційної роботи	20.12.2121	

Студент _____ Флейтута А.А.
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

"Дослідження процесів прийняття рішень стосовно архітектури програмного продукту для гнучких моделей життєвого циклу" // Флейтута Анастасія Андріївна // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНмз-61 // Тернопіль, 2021 // с. – 53, рис. – 5, джерел – 43.

Ключові слова ПРОГРАМНА АРХІТЕКТУРА, AGILE РОЗРОБКА, ВЕЛИКІ ПРОЕКТИ, УПРАВЛІННЯ ПРОЄКТОМ.

Роль програмної архітектури у Agile-розробці великих проєктів, де бере участь декілька команд, важлива, оскільки кільком командам потрібно працювати разом, щоб випустити єдиний програмний продукт, допомагаючи максимізувати автономність команд. Керування та узгодження архітектури Agile між автономними командами проблемою, оскільки в гнучких моделях Agile бракує практики управління архітектурою. Пропонований підхід до архітектурного управління включає зміну організаційної структури та процес управління змінами архітектури. Переваги, про які повідомляють фахівці-практики, включають передачу прийняття архітектурних рішень на операційний рівень, покращення обміну архітектурними знаннями між командами, мінімізацію зусиль, що витрачаються на архітектурний рефакторинг.

ANNOTATION

"Research of decision-making processes regarding software product architecture for flexible life cycle models" // Anastasiya Fleituta // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group СНмз-61 // Ternopil, 2021 // p. – 53, fig. – 5, ref. – 43.

Keywords: SOFTWARE ARCHITECTURE, AGILE DEVELOPMENT, LAGE SCALE PROJECT, TEAMS ALIGNMENT, PROJECT MANAGEMENT

The role of software architecture in Agile development of large projects involving several teams is important because several teams need to work together to release a single software product, helping to maximize team autonomy. Managing and coordinating the Agile architecture between stand-alone teams is a challenge because Agile's flexible models lack architectural management practices. The proposed approach to architectural management includes changing the organizational structure and the process of managing architectural change. The benefits reported by practitioners include transferring architectural decision-making to the operational level, improving the exchange of architectural knowledge between teams, and minimizing the effort spent on architectural refactoring.

ЗМІСТ

ВСТУП

Актуальність теми.

Проблеми розробки та зміни архітектурних рішень в гнучких методах розробки на сьогоднішній час є актуальними через розповсюдженість таких способів організації управління проєктами. Дослідженнями в цій галузі займались і представники Тернопільського національного технічного університету ім. Івана Пулюя. Тут варто згадати роботи [1, 2, 3, 4]. Одним із способів організації гнучких команд для великих проєктів є використання моделі Spotify, де дослідники виявили новий підхід до Agile створення команд під назвою Heterogeneous Tailoring [31]. Цей підхід характеризують дві ключові особливості. По-перше, кожна команда має право вибирати та адаптувати свій метод розробки. Ця ключова функція підтримується набором визначених рекомендацій, які полегшують автономію команд [33]. По-друге, кожна команда узгоджується з іншими командами та загальними цілями розробки продукту. Ця ключова функція підтримується набором визначених факторів впливу на формування автономних команд. Кожен ідентифікований фактор підтримується набором практик і процесів, які полегшують формування автономних команд. Однак наше емпіричне дослідження виявило проблему в управлінні та узгодженні архітектури Agile між автономними командами в даній моделі.

Мета роботи. Дана робота стосується вирішення задачі: як можна керувати архітектурою програмного забезпечення та узгоджувати архітектурні рішення в гнучких технологіях розробки. Було досліджено проєкти в розподіленій організації з області FinTech, щоб зрозуміти, як використовується модель Spotify, як приклад Agile. У цьому прикладі ми провели безпосереднє спостереження за практикою Agile. Аналіз виявив проблеми в управлінні та узгодженні архітектури Agile між автономними командами. Тому ми вирішили втрутитися, розробивши підхід до архітектурного управління. Зібрані дані проаналізовано за допомогою тематичного аналізу [9].

Наш підхід до архітектурного управління передбачає узгодження роботи автономних команд по вертикалі на рівні продукту та горизонтально за наборами навичок індивідів. Підхід включає структурні зміни та процес управління змінами архітектури. За словами практиків дослідження, запропонований підхід до архітектурного управління зсунув межі та полегшив узгодження та керування архітектурою Agile шляхом адаптації моделі Spotify. Ця трансформація, у свою чергу, покращила автономію команд за рахунок узгодження архітектурних рішень між автономними командами.

Об’єкт дослідження: процеси управління змінами до програмної архітектури в великих проєктах.

Предмет дослідження: літературні джерела з даними про роботу команд, залучених до великомасштабного проєкту.

Основні результати:

1) розробка та оцінка підходу до архітектурного управління при використанні моделі Spotify;

2) адаптація підходу Heterogeneous Tailoring, щоб включати узгодження та керування архітектурними рішеннями в автономних командах, запровадивши новий підхід до архітектурного управління в моделі Spotify.

Практична цінність роботи.

У цій роботі представлено характеристики, переваги та проблеми запропонованого підходу до архітектурного управління, а також представлено вплив розглянутого підходу до архітектурного управління на підхід інтегрування гетерогенних команд, залучених до великомасштабного програмного проєкту.

Апробація результатів та особистий внесок здобувача. Основні положення роботи доповідались, розглядались та обговорювались на науковій конференції Тернопільського національного технічного університету. Результати кваліфікаційної роботи опубліковані у тезах студентської наукової конференції, яка проводились у ТНТУ.

1 ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ ДОСЛІДЖЕННЯ

Ми провели це дослідження, щоб продовжити вивчення того, як приймаються дизайнерські рішення при розробці програмного забезпечення. Це відносно невивчена тема, незважаючи на визнання її важливості. Ми також провели це дослідження, щоб визначити, чи були Agile-методи корисними чи шкідливими для прийняття рішень, питання, яке, наскільки нам відомо, не розглядалося емпірично в літературі про Agile.

1.1 Гнучка архітектура

Роль архітектури програмного забезпечення в розробці програмного забезпечення Agile була суперечливою в попередніх дослідженнях [19]. Багато прихильників надають архітектурі програмного забезпечення життєво важливу роль у Agile-розробці, як і в інших підходах до розробки, інші опоненти проти цього. Це зіткнення між двома різними культурами пов’язане з різними переконаннями обох сторін. Наприклад, деякі прихильники сперечаються про важливість масштабування будь-якого підходу до розробки, який не приділяє достатньої уваги архітектурі програмного забезпечення у великому масштабі [39]. Одним із принципів, що лежать в основі маніфесту Agile, є «постійна увага до технічної досконалості, а хороший дизайн покращує маневреність». Отже, Agile-розробка програмного забезпечення повинна приділяти увагу архітектурі програмного забезпечення.

Agile архітектура – це ітеративний і поступальний спосіб еволюції архітектури. Поняття «Agile архітектура» викликає дві концепції [7]:

1) архітектуру програмного забезпечення, яку легко розвивати та модифікувати, і

2) спосіб Agile, який визначає архітектуру за допомогою ітераційного життєвого циклу.

Хоча перший досить стійкий, щоб не погіршуватися після кількох змін, другий з часом розвивається, оскільки проблема та обмеження краще розуміються. Ці два поняття не є однаковими; Не-Agile процес розробки програмного забезпечення може привести до гнучкої та адаптивної архітектури. Крім того, Agile процес може призвести до жорсткої архітектури програмного забезпечення. Однак те, що будують команди, залежить від того, як вони це будують. Крім того, на те, як команди створюють продукт, впливає сам процес розробки [10]. Таким чином, спеціалісти-практики Agile повинні зосередитися на архітектурних питаннях, які блокують рухливість команд для досягнення технічної досконалості, гарного дизайну та покращення маневреності розробки програмного забезпечення. Нехтування конкретними архітектурними міркуваннями навіть на початку процесу розробки може зробити архітектурний рефакторинг дорогим на подальших етапах розробки.

Agile Architecture можна розуміти як шаблони та тактики, які дозволяють одночасно зосередитися на архітектурі та Agile-розробці. Ці шаблони та тактики включають архітектуру шарів, окремі інтерфейси, обмеження залежностей та окремі задачі. Крім того, визначені деякі закономірності та тактики, які покращують масштабованість продукту, яка включає кластерну архітектуру з балансуванням навантаження та реплікованими копіями, інкапсуляцію алгоритмів та кешування даних. Крім того, в [7] автори визначили деякі закономірності та тактики, які зосереджуються на гнучкості розгортання та контролю витрат і часу на тестування. Ці шаблони та тактики включають віртуалізацію (розшарування як інфраструктури, так і програми), стандартизовану й конфігуровану архітектуру (параметризація та статичне та динамічне прив’язування) та виконуваний файл (структура коду, керована інтерфейсом).

Автори [29] наводять приклад використання архітектурної тактики та вирівнювання архітектури, команд Agile розробників та виробничої інфраструктури. Вони досліджували архітектурні тактики, які підтримують

масштабований розвиток Agile і покращують узгодженість архітектури та розвиток організації. Запропоноване вирівнювання в включає:

1) вертикальну та горизонтальну декомпозицію архітектури програмного забезпечення, щоб забезпечити відповідне вирівнювання команд,

2) матричну структуру команди з розширеними ролями за допомогою Scrum і

3) каталог тактик, зіставлених для Agile-розробки. Цей каталог тактик можна зібрати з успішних організацій та літератури.

Автори визначили декілька архітектурних видів діяльності та Agile-архітектурні практики. Вони виявили, що процес архітектури, який охоплює весь архітектурний життєвий цикл, складається з 11 архітектурних заходів. У розробці програмного забезпечення Agile цій архітектурній діяльності приділялися різні ступені уваги.

Це наступні процеси:

- опис архітектури,
- оцінка архітектури,
- розуміння архітектури,
- архітектурне обслуговування та еволюція,
- архітектурний аналіз,
- архітектурний рефакторинг,
- аналіз архітектурного впливу,
- архітектурна реалізація,
- архітектурний синтез,
- повторне використання архітектури та
- архітектурне відновлення.

Більшість попередніх дослідницьких зусиль було спрямовано на опис архітектури на основі побажань практиків Agile [37]. Однак розуміння архітектури, аналіз і рефакторинг були визначені як найбільш корисні види діяльності, які можна використовувати з Agile-розробкою. Крім того в [37] автор

визначив 41 метод Agile, який використовується в архітектурі. Однак лише деякі з цих практик широко використовуються на практиці та обговорюються в літературі – наприклад, Backlog, Sprint, Ітеративна та інкрементальна розробка та безперервна інтеграція. Відсутні вказівки щодо того, коли використовувати такі методи Agile архітектури. Тому практики Agile використовують такі практики на основі свого досвіду та знань.

Автори [25] визначили деякі ролі архітекторів та архітектурні практики, які потім використовуються для розробки та оцінки фреймворку для Agile-архітектури у великих організаціях із проектами вбудованого програмного забезпечення. Визначені ролі архітектора включають головного архітектора, архітектора управління та архітектора команди. Аналізуючи стосунки між архітекторами, автори виявили, що більшість практик потребують ролей для координації та співпраці та, таким чином, для пом’якшення проблем. Окрім ролей архітекторів,вони визначили різні типи команд: команду дизайну, команду розробників (runway teams), команду архітектури та команду управління. Функціональні команди керуються менеджерами продуктів і складаються з міжфункціональних команд. У кожній команді є архітектор команди, який відповідає за еталонну архітектуру та керує архітектурною діяльністю. Runway Teams – це спеціальні команди для одного або кількох спринтів, щоб зосередитися на «функції архітектури», а не на функціях, пов’язаних з клієнтами, щоб подолати архітектурну проблему, яка може призвести до кризи. Команда архітектури, яка включає вищезазначені ролі архітектора, працює разом для координації та співпраці, оскільки жоден архітектор не може мати всю інформацію, необхідну для підтримки численних команд у великих організаціях. Команда з управління залучає архітекторів управління та власників продуктів, які відповідають за стратегічну оцінку ризиків змін архітектури та визначення пріоритетів відставання команд між функціями та покращеннями архітектури, щоб збалансувати короткострокові та довгострокові цілі.

1.2 Автономні команди розробників

Масштабні проекти є складними, оскільки кільком командам потрібно працювати разом, щоб випустити єдиний програмний продукт [12]. Деякі приклади виявлених проблем у широкомасштабній розробці Agile – це створення та підтримка автономії команд та узгодження самоорганізаційних команд [26, 36].

Концепція «автономних команд» має різне походження та визначення в літератур. Цю концепцію вивчали й описували з різних точок зору в минулому; соціально-технічна, організаційна теорія та складні адаптивні системи. Найбільш близьке визначення автономних команд у застосуванні з зовнішньої інженерії програмного забезпечення в Agile розробці походить з точки зору управління знаннями. Самоорганізація визнана одним із принципів Agile з моменту введення Agile Manifesto. Agile-команди – це команди, що самоорганізуються, які можуть керувати робочим навантаженням і приймати рішення на основі команди, маючи взаємну довіру та повагу [12]. Автономія безпосередньо впливає на ефективність команди, оскільки повноваження прийняття рішень переміщуються на операційний рівень. Таке децентралізоване прийняття рішень прискорює процес розробки та підвищує точність вирішення проблем. Такі команди мають почуття відповідальності за віддану роботу. Крім того, вони працюють над досягненням сумісної мети, об’єднуючи та вирішуючи суперечливі пріоритети.

Однак автономні команди, що самоорганізуються, не є неконтрольованими (без лідера) командами. Лідерство в командах, що самоорганізуються, вважається легким і адаптивним. Лідери відповідають за розбудову стратегії, встановлення напрямків, об’єднання людей, мотивацію команд і надання зворотного зв’язку. Ці керівники мають різні посади, наприклад Scrum Master у Scrum, Coach у XP та Leader Squad або Agile Coach у Spotify.

Попередні дослідження [12, 31] виявили перешкоди для автономних команд у великомасштабних Agile проєктах. Наприклад, відсутні рекомендації щодо організації команд, як у Scrum. Крім того, висока індивідуальна автономія

може збільшити перевагу індивідів до власних цілей перед цілями команди. Додатково, узгодження автономних команд є складним завданням через різний ступінь очікувань зацікавлених сторін щодо вирівнювання команд. Крім того, існує напруженість між автономією та узгодженням команд. Занадто сильне узгодження може перешкодити автономії команд, але без вирівнювання команди автономні, але неефективні.

1.3 Розробка моделі менеджменту Spotify

Spotify – це сервіс потокової передачі музики, який у жовтні 2019 року мав загалом 248 мільйонів активних користувачів у всьому світі на місяць. Організація Spotify отримала вигоду від значного зростання за останнє десятиліття завдяки своїм інноваціям. Організація Spotify розробила власну культуру Agile і адаптувала методи Agile, щоб відповідати дуже широкомасштабній програмі (більше 300 осіб), розподіленій у чотирьох містах [31].

Модель Spotify керується створенням автономних, але вирівняних команд [21]. Щоб ініціювати створення автономних, але вирівняних команд, Spotify використовує адаптивну структуру і створює спільноти навколо цієї структури. Ця структура заснована на матриці двох вимірів (вертикального та горизонтального). Розглянемо артефакти такого підходу та їх значення.

«Плем’я» складається з сукупності розташованих разом команд і розраховано на менш ніж 100 осіб. Плем’я має на меті сприяти співпраці та пом’якшити залежність між командами. У кожному племені є невеликі групи людей, які мають подібний набір навичок і працюють у межах однієї сфери компетенції, яка називається капітулами. Члени кожного відділу регулярно зустрічаються, щоб вирішити проблеми в межах своєї компетенції. Ці зустрічі вважаються клеєм, який склеює всю організацію разом, не жертвуючи занадто великою автономією. Хоча відділи розташовані в межах одного племені, у всій

організації існують широкі групи людей, які називаються гільдіями, з бажанням поділитися знаннями та практикою з усією організацією [22].

Командам Spotify рекомендується використовувати Lean Startup для просування інновацій. Однак командам дозволяється адаптувати свої практики, маючи підтримку Agile-тренерів. Модель Spotify стала впливовою серед прихильників Agile і, отже, лягла в основу методів Agile, які використовуються в кількох інших організаціях. Попередні дослідження Spotify Tailoring виявили:

1) спеціальні методи, які сприяють ефективності автономних команд;

2) впливові фактори на вирівнювання автономних команд ;

3) впливові фактори Spotify Tailoring для Розробка продуктів B2B ;

4) моделі обміну знаннями шляхом вирощування культури участі та створення Spotify Guilds як спільнот практик ;

5) підхід до Agile адаптації під час використання моделі Spotify (тобто Heterogeneous Tailoring.

Рисунок 1.1 ілюструє наше сприйняття підходу Heterogeneous Tailoring, який характеризується двома ключовими ознаками. По-перше, кожна команда має право вибирати та адаптувати свій метод розробки, який зображений у нижній частині рисунка. Наприклад, одна команда використовує Scrum, тоді як інша використовує інший метод, наприклад Kanban або Lean. Таким чином, кожна команда має право вибирати та адаптувати свої методи Agile-розробки відповідно до своєї місії.

По-друге, кожна команда узгоджується з іншими командами та загальними цілями розробки продукту, що зображено у верхній частині рис. 1.1. Попередні дослідження виявили фактори, які впливають на формування автономних команд. Такими впливовими факторами є:

1) адаптивна структура,

2) колективне володіння кодом,

3) колективне прийняття рішень,

4) обмін знаннями,

5) координація між командами,

6) планування на основі місії та

7) стратегія доставки продукту.

Кожен ідентифікований фактор підтримується набором практик і атрибутів, які зміцнюють згуртованість автономних команд.

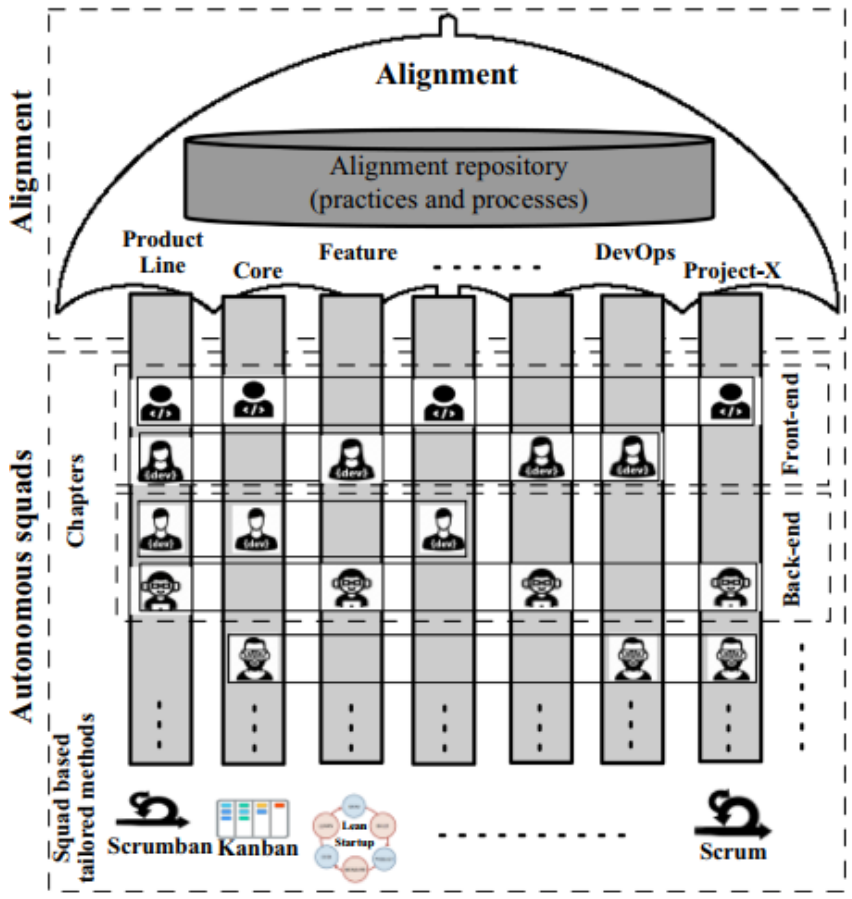

Рисунок 1.1 – Неоднорідний підхід до формування команд

Виявлені впливові фактори та пов’язані з ними методи можуть допомогти практикам Agile у вирівнюванні команд.

1.4 FinTech

FinTech – це абревіатура від «Financial Technology», що є сумішшю «фінансових послуг» та «інформаційних технологій». FinTech визначається як технологія, яка використовується для забезпечення ринків фінансовим

програмним продуктом або фінансовим програмним забезпеченням як сервісом (SaaS), що має складні технології. SaaS дозволяє компаніям використовувати програмне забезпечення, засноване на хмарі, через Інтернет, замість того, щоб купувати або створювати власний продукт.

Фінтех-організації використовують такі переваги, як безпека даних, наскрізна економія витрат, масштабованість і гнучкість. Такі організації, як правило, характеризуються фінансовими послугами, технологічними інноваціями та гнучкістю [20]. Вони мають різні бізнес-моделі, які діють шляхом витіснення або доповнення поточних фінансових послуг у галузі або створення нових фінансових послуг. Крім того, фінансові установи визнають важливість використання перевірених технологій для впровадження технологічних інновацій. Крім того, фінтех-організації повинні бути Agile, щоб швидко адаптуватися до нових можливостей ринку. Інфраструктура складається з фінансових API, які працюють як інтерфейс між фінансовими компаніями та зовнішніми сторонами. Наприклад, фінансовий API забезпечує інфраструктуру між банком та інвестиційною програмою, щоб надати клієнтам зручний банківський досвід. Послуги FinTech керуються національними та міжнародними правилами, що стосуються фінансів, для контролю збору, зберігання та звітності даних. Надаються такі протоколи, як SaaS, стандарт безпеки даних індустрії платіжних карток (тобто PCI), протидія відмиванню грошей (тобто AML) і «Знай свого клієнта» (тобто KYC), для яких Фінтех-організаціям слід автоматизувати відстеження нормативно-правових актів та їх дотримання.

2 МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

Спочатку було проведено дослідження, щоб отримати глибоке розуміння того, як модель Spotify використовується в індустрії FinTech. Дані були зібрані шляхом спостереження за командами розробників. Зібрані дані були проаналізовані за допомогою Grounded Theory [16], і результати були опубліковані, звідки і взяті для цієї роботи. Зокрема, ми виявили проблему в управлінні та узгодженні архітектури Agile між автономними командами під час використання моделі Spotify. У моделі Spotify відсутні методи управління Agile архітектурою. Тому ми вирішили провести вбудований кейс із розробкою рекомендацій щодо організації роботи командс. Ми розробили підхід до архітектурного управління, адаптувавши модель Spotify, щоб оцінити її в тій самій організації з вивчення прикладів.

2.1 Об'єкт дослідження

Для цього дослідження було відібрано багатонаціональну фінтех-організацію з масштабним проектом з використанням моделі Spotify. Огляд фінансових технологій та його характеристик наведено далі.

Це дослідження зосереджено на проекті SaaS, який керує автономними фінансовими послугами. Ці автономні програмні послуги працюють під контролем одного адміністративного проекту, який представляє для служби загальновизначену політику управління. Тисячі клієнтів (тобто організацій) використовують цей продукт для керування платіжними операціями своїх кінцевих користувачів. Цими платіжними операціями керує проект із прикладного дослідження та проходять через багато постачальників платежів у всьому світі. Проект дозволяє клієнтам легко та швидко налаштовувати постачальників платежів, оскільки він має унікальний механізм правил та інтелектуальні можливості маршрутизації, які підвищують швидкість прийняття

платежів. Також проект допомагає клієнтам виявляти шахрайство, аналізувати всі платіжні операції та зосередитися на розвитку свого бізнесу.

Програма розробки програмного забезпечення розташована в штаб-квартирі і складається з 37 учасників. Розробники розподілені між шістьма командами з приблизно п’ятьма членами в кожній команді. Розробники також розподілені на сім підрозділів. Крім того, є один архітектор, п’ять власників продуктів (PO), три менеджери по роботі з ключовими клієнтами (KAM), два Agile тренери, один керівник підтримки та один тестовий керівник.

2.2 Збір даних

Після представлення розробленого нами підходу до організації тематичних досліджень організація, що займається вивченням кейсів, погодилася спробувати його. Перший автор, який працює в організації тематичних досліджень як старший інженер-програміст, провів безпосереднє спостереження за повним життєвим циклом розробки протягом 3 місяців. Спостережувані церемонії, яких налічується 32, включають щоденні стенд-апи, ретроспективи, сесії планування, зустрічі на основі розділів, зустрічі власників продуктів та обговорення на основі архітектури на вимогу. Застосування безпосереднього спостереження дало дослідникам глибоке розуміння досліджуваного явища та пом’якшило можливість відхилення між поглядом на питання «інтерв’ю» та «реальним» випадком.

Напівструктуровані відкриті інтерв’ю були орієнтовані на учасників з різних сфер розробки програмного забезпечення. Таким чином, звернулися до практиків у кількох різних організаційних ролях.

2.3 Аналіз даних

Зібрані дані були проаналізовані за допомогою тематичного аналізу. Цей аналіз проводився шляхом виконання шести кроків:

1) ознайомлення з даними,

2) кодування,

3) пошук тем,

4) перегляд і уточнення тем,

5) визначення та іменування тем,

6) написання підсумкового звіту.

Використання цих методів теорії дало нам можливість використовувати суворий метод систематичного аналізу зібраних даних.

Аналіз даних починається зі зіставлення ключових моментів із кожної стенограми інтерв’ю. Потім кожній ключовій точці було присвоєно код, який представляє фразу, яка підсумовує ключовий момент у 2 або 3 слова. Після проведення кожного інтерв'ю використовувався метод неперервного порівняння [16]. Цей метод передбачає постійне порівняння кодів, що виникли з кожного інтерв'ю, з іншими кодами з того самого інтерв'ю, а також з кодами інших інтерв'ю. Використання цього методу постійного порівняння полегшило процес групування виниклих кодів у більш високий рівень абстракції, званий темами.

Протягом усього аналізу використовувався постійний процес створення таз званих пам'яток (Memoing) [16]. Пам’ятки представляють ідеї про нові коди та їх взаємозв’язки. Memoing вважається потужним способом вилити змінні, що з’являються (коди, підтеми чи теми). Крім того, Memoing сприяє виникненню зв’язків (подібності чи відмінності) між різними змінними. Постійний збір та аналіз даних відбилися на ідеях меморандумів і спричинив деякі зміни. Сортування теоретичних нотаток було розпочато, коли збір даних був майже завершений, а кодування було майже насиченим. Сортування зібраних нотаток дало теоретичний план, який знову об’єднав розрізнені дані. На останньому етапі нашого аналізу дані спостереження (тобто пам’ятки) були проаналізовані та порівняні з темами, отриманими з аналізованих інтерв’ю. В результаті були виявлені незначні протиріччя, які були досліджені та враховані в результатах.

В результаті, в результаті нашого аналізу виникли дві основні теми «вигоди» та «завдання», які зображені на рис. 2.1. На цьому малюнку теми виділені жирним текстом і розташовані всередині прямокутника. Кожна тема пов’язана з

кількома категоріями чи поняттями, які позначено курсивом. Кожна категорія підтримується кількома кодами, які позначені простим текстом.

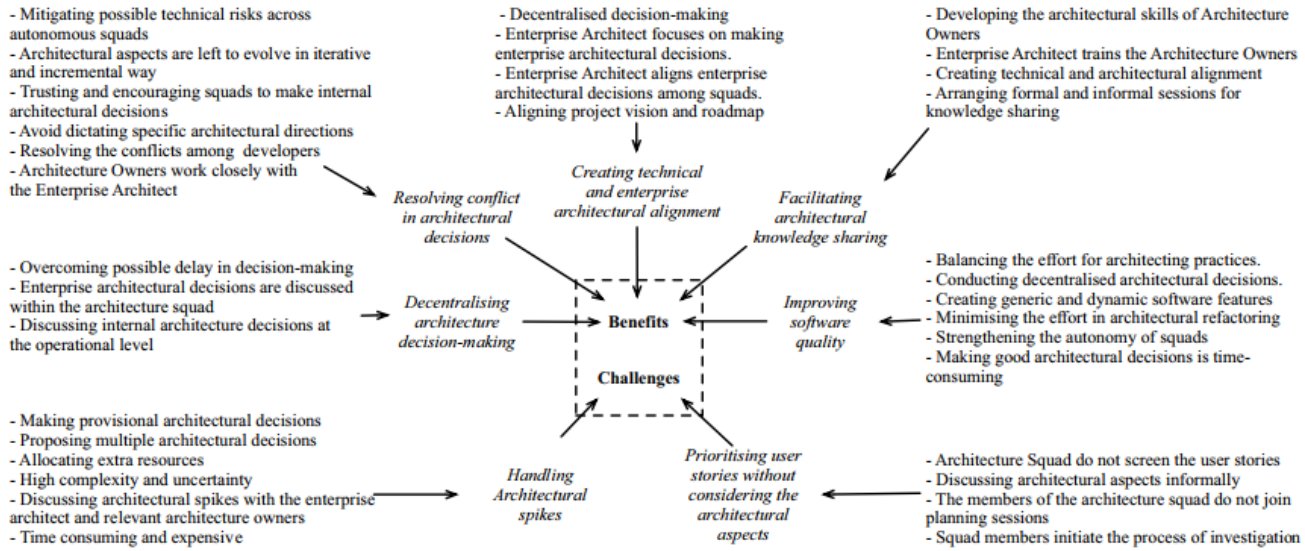

Рисунок 2.1 – Виниклі теми, категорії та коди

Виниклі теми не створили модель, яка складається з тем вищого порядку між ними, а натомість представили сильні та слабкі сторони нашого підходу, спираючись на точки зору учасників. У наступному розділі представлені ці дві теми, а також наш підхід до архітектурного управління, який включає організаційні структурні зміни та процес управління змінами архітектури за допомогою адаптації моделі Spotify.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЦЕСУ МЕНЕДЖМЕНТУ

3.1 Підготовка до впровадження змін

Організація, яка вивчає приклади, використовувала організаційну структуру Spotify, як показано на рис. 3.1, одночасно здійснюючи централізоване прийняття архітектурних рішень. Команди отримали повноваження вибирати та адаптувати свої методи розробки Agile відповідно до своїх місій. Ті люди, які мають подібний набір навичок і працюють у межах однієї сфери компетенції, були розподілені між відділами, перебуваючи в різних загонах. Отже, організація використовувала двовимірну структуру. Незважаючи на узгодження команд на рівні продукту, автономні команди мали свободу виконувати необхідну розробку різних пов’язаних частин програмного продукту завдяки реалізації колективного права власності на код. Отже, усі команди посилалися на архітектора через складність проекту.

Архітектор відповідав за розробку програмного продукту для вирішення виниклих бізнес-проблем шляхом розробки архітектурних проектів для створення передового програмного рішення високої якості.

Проте групам команд слід надати повноваження з архітектурної точки зору трансформувати рішення на оперативний рівень. На думку власника продукту (product owner), «повинен бути хтось або команда, відповідальна за загальну картину… Трохи більше структури навколо власності на основі місій, вертикалей, архітектури та з урахуванням довгострокової дорожньої карти».

Організація з вивчення прикладів мала проблеми з узгодженням архітектурних рішень між автономними командами через відсутність визначеного процесу для Agile-архітектури та використання централізованого процесу прийняття рішень. За словами архітектора обсяг програми розробки зараз набагато більший, ніж той, що був раніше. Архітектор перевантажений багатьма обов’язками, що, у свою чергу, спричиняє затримку у прийнятті архітектурних рішень та впливає на автономність команди. З іншого боку,

старший розробник підкреслює важливість належного процесу архітектури, щоб забезпечити автономію команди. Роблячи цей коментар, цей розробник повідомляє, що «нам бракувало регламентованого процесу зміни Agile архітектури, який би покращив автономію наших команд».

3.2 Оцінка пропонованого підходу на основі введення змін

Щоб подолати проблему узгодження та керування архітектурними рішеннями між автономними командами, ми провели власне дослідження. Під час цього втручання було розроблено та оцінено підхід до архітектурного управління, який включає структурні зміни та процес управління змінами архітектури. У цьому розділі описано характеристики такого підходу до архітектурного управління за допомогою моделі Spotify.

3.2.1 Структурні зміни організації

Наш підхід передбачає зміни в організаційній структурі. Ці зміни мають на меті полегшити управління та узгодження архітектурних рішень між автономними командами і в кінцевому підсумку посилити автономію команд. Структурні зміни полягають у тому, що:

1) надається роль власників архітектури керівникам відділів і досвідченим розробникам,

2) змінюють обов’язки архітектора на орієнтацію на архітектуру підприємства,

3) розміщують власників архітектури у віртуальній команді на чолі з архітектором підприємства. Рисунок 3.1 ілюструє введені структурні зміни коричневим кольором.

Роль та обов'язки власника архітектури:

Роль власника архітектури покладається на керівників розділів та інших досвідчених розробників. Оскільки розділи формуються на основі областей

компетенції, а команди вирівнюються на рівні продукту, ми відповідно вирівняли власників архітектури.

Від осіб, зайнятих на таких ролях було отримано твердження, надання їм ролі власника архітектури полегшує прийняття архітектурних рішень у своєму підрозділі. Однак ця роль збільшує накладні витрати на нього, оскільки він також працює розробником.

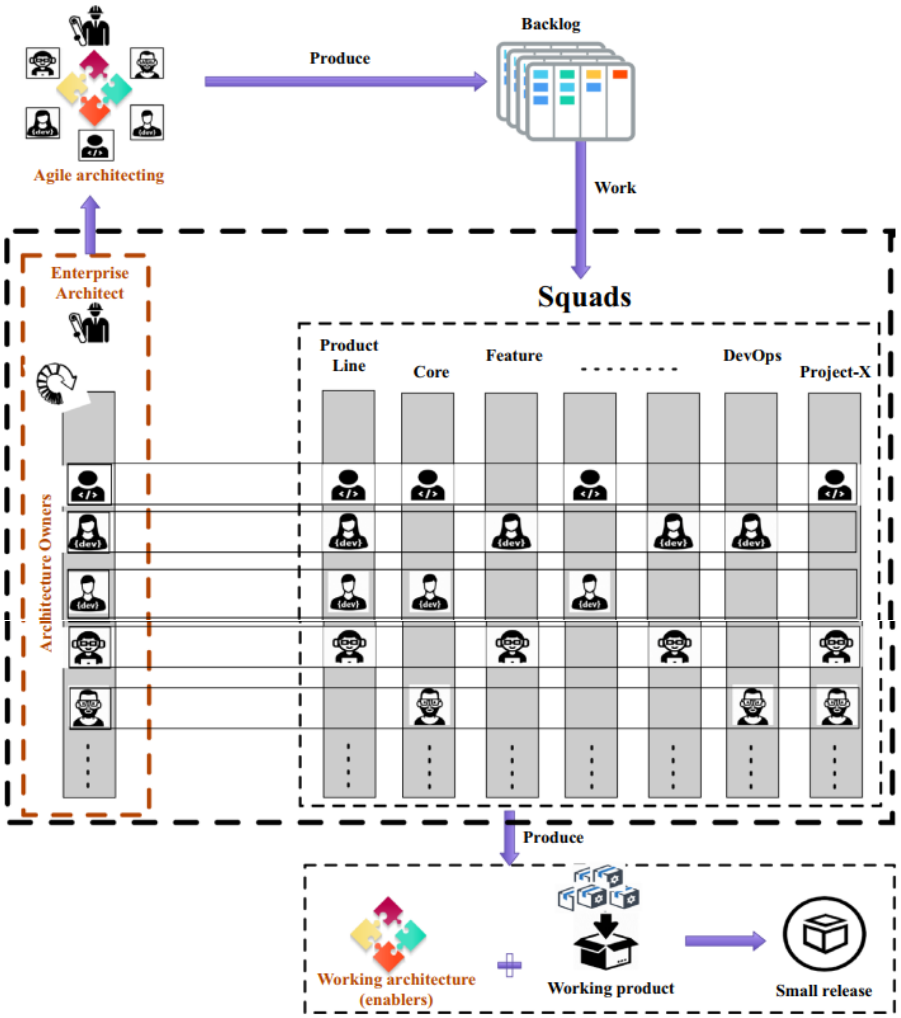

Рисунок 3.1 – Організаційні структурні зміни

Поділ ролі архітектора на ролі власників архітектури та розподіл її між керівниками підрозділів та іншими розробниками на основі їх сфер компетенції, трансформує рішення на операційний рівень, що корисно для узгодження рішень, заснованих на архітектурі.

Більшість часу власники архітектури витрачають на розробку програмного забезпечення, оскільки вони працюють у різних командах, а решту часу витрачають на виконання архітектурних завдань. Обізнаність власників архітектури щодо технічних і бізнес-дорожніх карт має вирішальне значення для узгодження архітектурних рішень у громадах, що базуються на відділах, а отже, і в автономних командах. Власники архітектури повинні знати про ділові та технічні проблеми команди, щоб працювати над ними протягом усього процесу розробки.

Обговорення архітектурних аспектів у рамках глави створює вирівнювання між командами. Обговорення архітектурних рішень у відділі полегшує досягнення домовленості між командами про те, як і коли виконувати певні роботи.

Власники архітектури відповідають за обмін архітектурними знаннями між автономними командами. Ці власники архітектури відповідають за створення технічних рекомендацій, таких як інструкції з кодування, баз даних та безпеки. Крім того, вони відповідають за наставництво та навчання членів своїх відділень щодо архітектурних та дизайнерських навичок. Поширення технічних та архітектурних знань у межах відділу є одним із обов’язків власника архітектури.

Розв’язання конфліктів в архітектурних рішеннях та пом’якшення ключових технічних ризиків між командами є іншими обов’язками власників архітектури. Розробники можуть зіткнутися з конфліктами в архітектурних рішеннях і не завжди погоджуються з ними.

Власники архітектури можуть зіткнутися зі складними архітектурними завданнями, які потребують дослідження. Отже, вони можуть або досліджувати такі архітектурні роботи самостійно, або можуть попросити члена команди, який зіткнувся з цією проблемою, дослідити її. Тоді обидві сторони могли б обговорити це далі.

Власники архітектури співпрацюють з Enterprise Architect, а також іншими власниками архітектури в рамках команд віртуальної архітектури. Ця співпраця

має вирішальне значення, щоб отримати максимальну віддачу від команди архітектури та використовувати кращу злагодженість у всій організації. Основна причина створення віртуальної архітектурної команди, яка складається з керівників відділів, які мають ролі власника архітектури, полягає в тому, щоб мати належне технічне та архітектурне узгодження через організацію Зустріч, коли це необхідно, важлива для вирішення виниклих технічних чи архітектурних проблем.

Роль та обов'язки архітектора підприємства.

Роль Enterprise Architect привласнюється Architect. Пропонується адаптувати обов'язки архітектора, щоб не втратити фокусу в проєкті на архітектурі продукту. Ентерпрайз архітектор має великі знання про технічні та бізнес предметної області. Він повинен продовжувати фокусуватися на корпоративних архітектурних завданнях. Оскільки Enterprise Architect очолює команду архітекторів, мати високий рівень довіри для забезпечення реалізації архітектури.

Архітектор підприємства працює з командою управління (тобто, власниками продуктів і менеджерами ключових клієнтів) і близький до власників архітектури. Ця тісна співпраця узгоджує архітектурні рішення з дорожньою картою проєкту. Архітектор підприємства витрачає багато часу на співпрацю зі старшими зацікавленими сторонами в організації, щоб створити належне технічне та архітектурне узгодження між командами. Така співпраця, у свою чергу, сприяє застосуванню правильних архітектурних рішень підприємства в потрібний час відповідно до цінностей бізнесу.

Створення технічного та архітектурного узгодження для повного програмного продукту доручено Архітектору підприємства. На відміну від Enterprise Architect, Власники архітектури відповідають за конкретні компоненти продукту та відповідальні за конкретні області технічної компетенції.

В обов’язки архітектора підприємства також входить просування інженерної та архітектурної практики підприємства та стимулювання архітектурних ініціатив. Замість того, щоб форсувати конкретні архітектурні

рішення, Enterprise Architect полегшує прийняття архітектурних рішень підприємства та узгодження їх між автономними командами. Таким чином, Enterprise Architect полегшує повторне використання ідей, компонентів і узгодження перевірених шаблонів між командами, одночасно працюючи з командами для розробки та розвитку архітектури.

3.2.2 Процес управління змінами архітектури

Пропонований змінений процес управління змінами архітектури був адаптований в до проєкту. Цей процес управління змінами орієнтований на керівництво залучених зацікавлених сторін (тобто, розробників, власників архітектури, корпоративних архітекторів і власники продукту) до управління і поєднання архітектурних рішень в продукті. Цей процес складається з видів діяльності, показаних на рисунку 3.2. Реалізація процесу управління змінами архітектури полягає в наступному:

Дія 1: Виявити можливі архітектурні зміни: коли розробник стикається з можливими архітектурними змінами, він визначить їх вплив на проєкт. Для того, щоб почати процес архітектурного аналізу, залучений розробник повинен створити картку Канбан (тобто, тікет), яка описує запит на зміну з додатковими технічними характеристиками та візуалізує його на етапі аналізу. Такі картки Kanban сприймаються, як активні елементи для інших робочих елементів.

Дія 2: Розуміння виявленої зміни та її вплив на проект з розробки програмного забезпечення та архітектуру продукту. Власник архітектури та залучені розробники повинні зрозуміти природу змін та визначити їх потенційний вплив на архітектуру. Власник архітектури оновлює карту Kanban більш точними технічними характеристиками. Результати цієї діяльності повинні надати правдоподібні дані про частини продукту та його архітектуру, на які вплинула зміна.

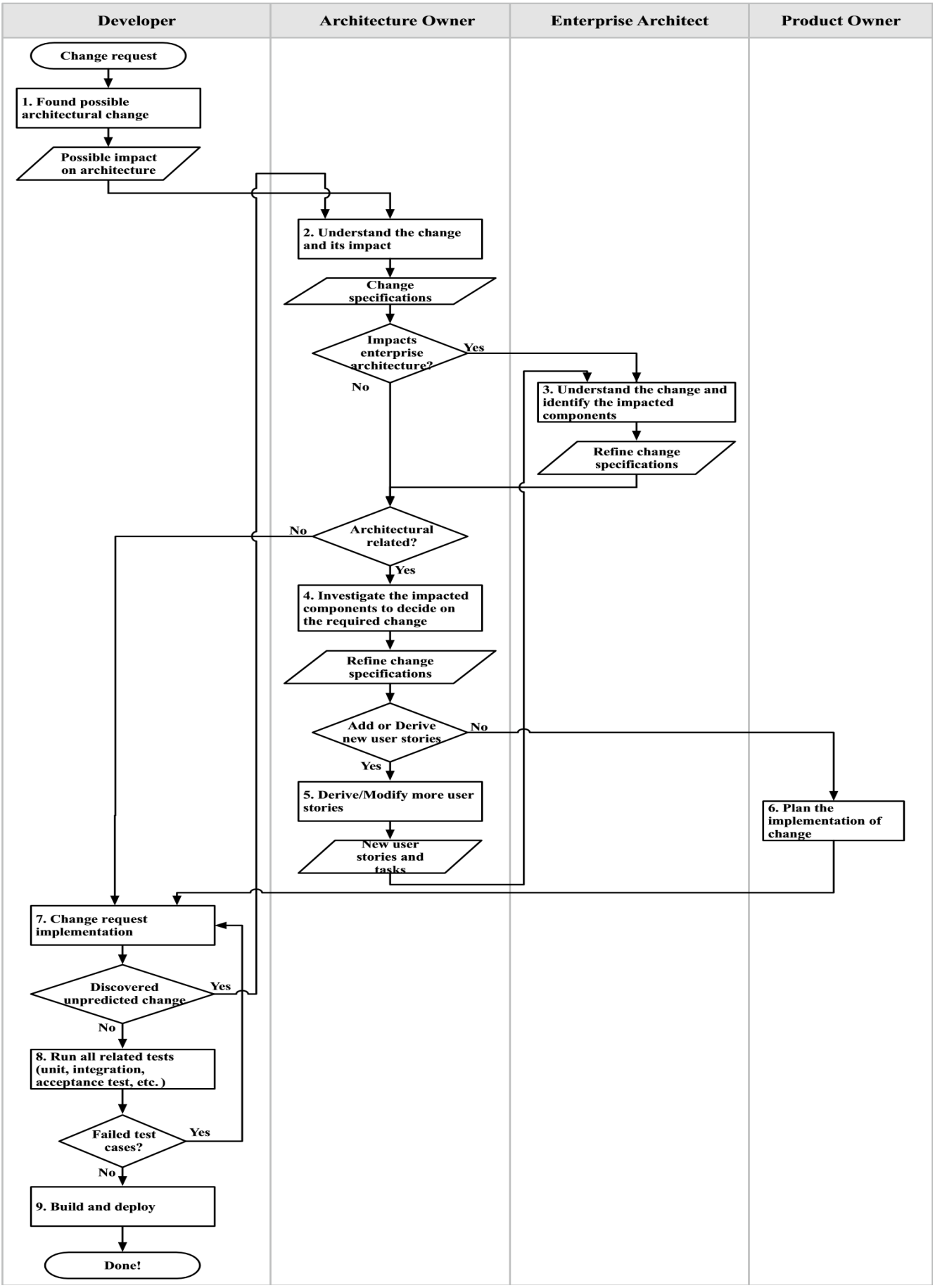

Рисунок 3.2 – Процес менеджменту змін архітектури

Якщо виявлені зміни можуть вплинути на архітектуру, слід виконати одну з двох можливих дій:

Якщо робота вимагає зміни архітектури підприємства, власник архітектури повинен обговорити необхідні зміни з архітектором підприємства і, якщо це необхідно, в команді архітектури – слід виконати дію 3. Таким чином, картка переміщається на дошку Kanban.

Визначення того, чи може запит на зміну вплинути на архітектуру чи ні:

а) якщо запит на зміну впливає на архітектуру, власник архітектури та залучені розробники повинні дослідити компоненти, на які вони вплинули, і прийняти рішення про необхідні зміни, дотримуйтесь дії 4;

б) в іншому випадку завдання можна або переслати для планування – дотримуючись дії 6, або перенаправити відповідній команді для виконання – дотримуючись дії 7).Дія 3: у цій діяльності залучені зацікавлені сторони (тобто розробники, власник архітектури та архітектор підприємства) мають обговорити отримані дані про частини програмного продукту та його архітектуру, на які вплинули ці зміни. Якщо запит на зміну архітектури впливає на інші частини проекту, власники архітектури, які працюють над відповідною частиною, запрошуються на цей сеанс. Під час цієї дії власник архітектури представляє надані дані в картці Kanban, яка є результатом дії 2. Потім починається аналіз впливу. Однак ітеративний процес аналізу впливу можна провести на основі виниклих проблем.

Результатом цієї діяльності є поглиблені дані про вплив архітектури. Цей результат включає визначення компонентів, на які впливає, і специфікації нових сценаріїв і вимог. Після цього зацікавлені сторони вирішують, чи перейти до розслідування низького рівня, дотримуючись дії 4 або продовжуючи реалізацію запиту на зміну – коли архітектурну зміну відхилено, а нове рішення вводиться і замінює запит на зміну архітектури – дотримуючись дії 7.

Дія 4: дослідити компоненти, на які є вплив, для прийняття рішення про необхідні зміни: Під час цієї дії проводиться зустріч між власником архітектури та залученими розробниками для глибокого вивчення специфікацій запиту на

архітектурні зміни. На цій зустрічі вони розбивають специфікації та вимоги на сценарії, а потім документують деталі в завдання, які можна запланувати. Ця діяльність може виявити нові компоненти, на які впливає. Таким чином, власник архітектури може створювати нові історії користувачів для непередбачуваних змін.

Якщо виявити непередбачувані зміни було можливим, перейти до кроку 5. В іншому випадку картку Kanban, яка містить історію користувача та його завдання, слід перевести в стан To-Do і переслати до власника продукту для планування, дотримуючись кроку (дії) 5.

Дія 5: отримати/змінити більше історій користувачів: коли власник архітектури та розробники виявлять компоненти, на які нещодавно вплинули, власник архітектури створить нову історію користувача для непередбачених змін. Ці історії користувачів потім переміщуються до WIP на етапі аналізу підприємства, за яким слідує Дія 3. Таким чином, починається нова ітерація аналізу впливу. У цьому випадку для обговорення таких змін проводиться неформальна зустріч.

Дія 6: Планування впровадження змін. Якщо запит на зміну було схвалено командою архітекторів та власником архітектури, картка Kanban має бути доступною для планування та розробки. Таким чином, менеджери проєкту можуть планувати виконання цього запиту на зміну та пересилати історію користувача та його завдання відповідним командам для впровадження, дотримуючись дії 7.

Дія 7: Реалізація запиту на зміну: У цій діяльності здійснюється впровадження змін. Команди використовують гібридний процес Behavior Driven Development та Test Last Development. Оскільки історії користувачів описують поведінку введених сценаріїв і вимог, очікується, що розробники реалізують описану поведінку в необхідному тестовому покритті. Ця реалізація може негативно вплинути на інші існуючі тести. Ці тестові приклади підлягають модифікації, щоб відповідати новим вимогам. Тим не менш, під час виконання запиту на зміну команда розробників може зіткнутися у двох випадках:

Якщо рішення полягає у зміні або додаванні нових вимог, які потребують архітектурних змін, розробники повинні повідомити власника продукту та власника архітектури про запропоновані зміни до вимог. У цьому випадку зміни, запропоновані розробниками, слід оголосити як неочікувані зміни. Після цього власник архітектури та розробники мають провести неформальну зустріч, щоб дослідити несподівані зміни, дотримуючись дії 2.

В іншому випадку виконати дію 8.

Дія 8: провести тестування: розробники повинні використовувати безперервну інтеграцію, щоб уникнути затримок, спричинених проблемами інтеграції. Згодом слід розпочати безперервний процес тестування для отримання негайного зворотного зв’язку щодо можливості порушення архітектурних контрзаходів для запобігання необґрунтованих ризиків, пов’язаних із випуском програмного забезпечення. Обсяг тестування має бути розширений від тестових випадків до вимог поведінки для перевірки архітектурних цілей і поведінки продукту. У разі будь-якого порушення вимог після виконання тестів, розробники повинні перевірити реалізацію та невдале тестові випадки, дотримуючись дії 7. В іншому випадку дотримуватись дії 9.

Дія 9: Створення та розгортання: Цю дію слід виконати після успішного завершення безперервного тестування. Новий випуск можна запланувати для розгортання на виробництві.

3.3 Адаптація підходу Heterogeneous Tailoring для управління архітектурними рішеннями

У цьому розділі представлено, як підхід Heterogeneous Tailoring (узгодження гетерогенних команд) був адаптований для врахування розробленого підходу до архітектурного управління, який складається із структурних змін та процесу управління змінами архітектури. Рисунок 3.3 ілюструє вплив пропонованого підходу на підхід Heterogeneous Tailoring шляхом виділення внесених змін коричневим кольором. Отже, підхід

Heterogeneous Tailoring був проілюстрований за допомогою чотирьох ключових функцій: розробка продукту, узгодження, автономні команди та стратегія випуску.

На розробку продукту, процес якої зображено у верхній частині рис. 3.3, впливає впровадження нашого підходу до архітектурного управління до організації тематичного дослідження.

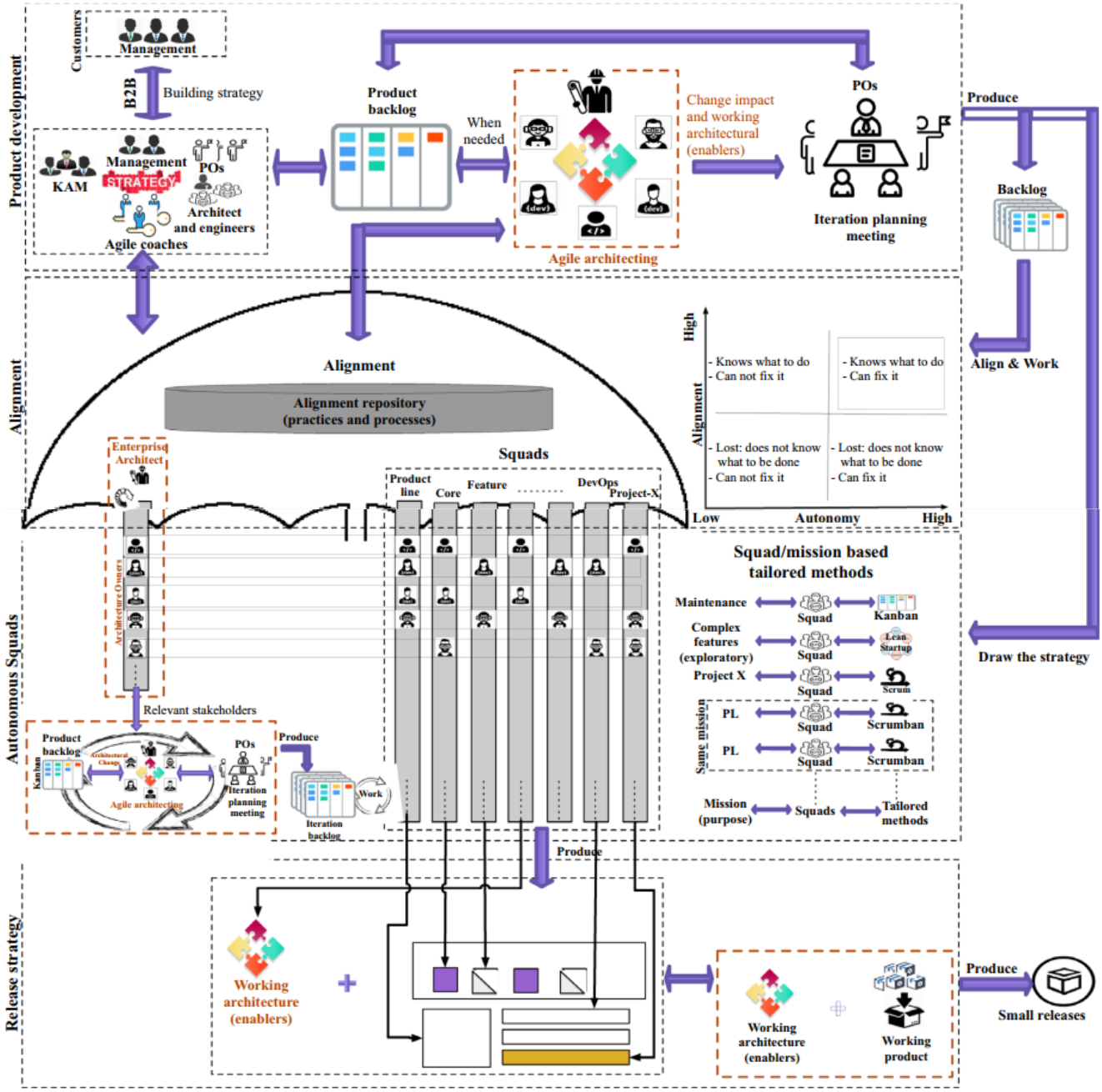

Рисунок 3.3 – Адаптація гетерогенного підходу до координації роботи команд

Цей вплив усвідомлюється в результаті використовуваного процесу управління змінами архітектури, який створює архітектурно-базовані історії користувачів (тобто інструменти) у беклозі проєкту. Ці стимули повинні бути пріоритетними та спланованими. Це процес управління змінами іноді призводить до створення нових архітектурних тікетів (тобто додатків), які потребують визначення пріоритетів і планування.

Фактори впливу на формування автономних загонів представлені в попередніх дослідженнях. Ці ідентифіковані фактори та пов’язані з ними методи можуть керувати практиками Agile у формуванні автономних команд.

Запропонована модифікація управлінням архітектурних змін покращує узгодження автономних команд, враховуючи архітектурні аспекти, як показано у другій частині рис. 3.3. Узгодження та управління архітектурою програмного забезпечення зображено у внесених змінах до організаційної структури через ролі власників архітектури та архітектора підприємства – а також у процесі управління змінами архітектури. Наша структурна модифікація змінила межі та перетворила рішення, засновані на архітектурі, на оперативний рівень, який узгоджується через спільноти команд. Обговорення архітектурних аспектів з власником архітектури (керівником спільноти команд) прискорює процес і ставить всіх членів нашого на один рівень. Запроваджений процес управління змінами вирівняв процес Agile-архітектури між усіма зацікавленими сторонами (розробниками, власниками архітектури, архітекторами підприємства та власниками продуктів), щоб підвищити автономію команд. Виконання цього стандартизованого процесу в командах допомагає всім сторонам організувати свою архітектурну роботу замість того, щоб залежати від деяких членів інших команд або навіть покладатися лише на архітектора.

Природа історій користувачів вимагає швидкого реагування на запити клієнтів. Тепер процес більш дисциплінований та організований, що, отже, збільшило швидкість прийняття архітектурних рішень.

Автономність команд, яка зображена в третій частині рис. 3.3 , була покращена завдяки використанню нашого підходу до архітектурного управління,

як описано в наступному розділі. Запроваджений підхід посилив автономію команди шляхом децентралізації прийняття рішень на основі архітектури, що зображено у введеній структурній зміні. Крім того, був запроваджений процес управління змінами архітектури, щоб збалансувати Agile та архітектуру, а отже, створити цілісний підхід, який забезпечує автономію команди. Правила, які використовуються в Agile-архітектурному процесі, який керує взаємодією між усіма зацікавленими сторонами, балансує Agile та архітектурну діяльніст разом зі структурними змінами, пом’якшили залежності та, в свою чергу, покращили автономію команд.

Використана стратегія випуску, яка зображена в останній частині рис. 3.3, залежить від вирівнювання та керування архітектурними рішеннями. Цей вплив сприймається в безперервній доставці робочої архітектури (тобто, активаторів) разом із робочим продуктом.

3.4 Переваги та проблеми підходу архітектурного управління

У цьому розділі представлені виявлені переваги та проблеми запропонованого підходу до архітектурного управління, який включає структурні зміни та процес управління змінами архітектури.

3.4.1 Переваги

Підхід до архітектурного управління перетворив прийняття архітектурних рішень із централізованого процесу у децентралізований процес. Ця трансформація принесла користь досліджуваній організації у різних аспектах. Таким чином більше не потрібно чекати на архітектора. Натомість розробник може безпосередньо зв’язатися з власником архітектури в команді. Однак архітектурні рішення підприємства необхідно обговорювати в команді архітекторів. Прийняття рішень про те, як інтегрувати різні компоненти або API, може вимагати глибокого дослідження кількома власниками архітектури та архітектором підприємства.

Запропонований підхід дозволив вирішити конфлікт в архітектурних рішеннях і зменшив технічні ризики між автономними командами. У запропонованому підході до управління архітектурою багато складних технічних та архітектурних аспектів залишаються для розвитку шляхом ітеративної та поступової розробки та навчання. Таким чином, складні технічні та архітектурні рішення завершуються пізніше в життєвому циклі, як це показано в процесі управління змінами. Крім того, командам довіряють самостійно приймати локальні архітектурні рішення, не чекаючи архітектора підприємства. Розробників заохочують приймати архітектурні рішення самостійно та за підтримки власника архітектури, не чекаючи архітектора, оскільки технічні деталі розвиваються постійно. Власники архітектури зазвичай намагаються уникати диктування конкретних архітектурних напрямків на користь підходу, заснованого на спільній роботі.

Однак у розробників можуть виникнути конфлікти в архітектурних рішеннях і не завжди прийти до згоди. Власники архітектури співпрацюють з Enterprise Architect для вирішення корпоративних архітектурних рішень, які впливають на два або більше компонентів у проекті програмного забезпечення.

Крім того, децентралізація рішень на основі архітектури надала Архітектору підприємства можливість зосередитися на створенні технічного та корпоративного архітектурного узгодження для повного програмного продукту.

Узгодження архітектури підприємства передбачає забезпечення того, що бачення проекту та дорожня карта реалізуються між робочими елементами, заснованими на архітектурі. Архітектор підприємства гарантує, що всі власники архітектури підтримують бажані архітектурні можливості та напрямки загального рішення. Це досягається шляхом тісної співпраці в архітектурному відділі шляхом використання процесу управління змінами архітектури. Запроваджений процес управління змінами архітектури полегшує узгодження архітектурних рішень у всій організації.

Більше того, наш підхід до архітектурного управління сприяв обміну архітектурними знаннями між автономними командами. Архітектор

підприємства працює над переходом архітектурних навичок та навчанням власників архітектури. Архітектор підприємства може організовувати та проводити семінари, щоб навчати та навчати наші команди аспектам, пов’язаним з архітектурою. Крім того, власники архітектури зосереджуються на створенні технічного узгодження та поширенні технічних та архітектурних знань. Власники архітектури влаштовують офіційні та неформальні сесії для обміну знаннями з архітектури.

Крім того, наш підхід покращив якість програмного забезпечення та пом’якшив перешкоди для узгодження архітектурних рішень між автономними командами. Наша організація кейсів намагається збалансувати зусилля для архітектури, використовуючи введений процес управління змінами, маючи при цьому децентралізоване прийняття рішень на основі архітектури. Цей баланс, у свою чергу, полегшує створення загальних функцій програмного забезпечення, мінімізує витрати зусиль на архітектурний рефакторинг і керує архітектурою, одночасно зміцнюючи автономію команд. Проведення належного архітектурного аналізу в команді, а потім оцінка та обговорення результатів, якщо необхідно, з архітектором підприємства покращили якість виконаної роботи. Однак цей процес може зайняти багато часу. Прийняття гарного архітектурного рішення, яке можна буде легко підтримувати в майбутньому, без великого рефакторингу, може зайняти багато часу, не звертаючи уваги на деякі архітектурні аспекти, які можна розглянути в даний момент. спричиняють багато відходів через потреби в рефакторингу.

3.4.2 Проблеми

Виконане дослідження виявило дві проблеми запровадженого підходу. По-перше, визначення пріоритетів історій користувачів без урахування архітектурних аспектів може негативно вплинути на планування. Запроваджений процес управління змінами архітектури не сприяє перегляду історій користувачів командою архітектури до або під час планування. Ми не розглядаємо історії користувачів у команді архітектури, перш ніж планувати

роботу команди. Проте наш процес управління змінами справляється з такою ситуацією, коли стикається з непередбачуваними архітектурними змінами, переходячи від дії 6 до дії 1. Члени архітектурної групи не приєднуються до нашої сесії планування, тому що розробники повинні мати можливість виконувати свою роботу автономно. Ми очікуємо, що члени команди повинні розпочати процес розслідування та надати належну інформацію власнику архітектури або архітектору підприємства.

По-друге, обробка архітектурного "збурення" може вимагати прийняття попередніх архітектурних рішень або навіть кількох можливих рішень для проведення необхідного дослідження. Наша організація з вивчення прикладів розглядає таке збурення, як інвестицію, щоб з’ясувати, що потрібно побудувати і як це побудувати. Виділяються певні ресурси на складні робочі елементи, перед цільовим терміном релізу, щоб з’ясувати, що потрібно зробити. Такі інвестиції вважаються необхідністю для вирішення архітектурних проблем, які працюють як стимул для наступного спринту. Складність і невизначеність змін вимог можуть призвести до прийняття кількох попередніх архітектурних рішень.

Іноді при обговоренні історії користувача з архітектором підприємства та власником архітектури, результат може бути неоднозначним, оскільки немає конкретного рішення, яке можна прийняти, тоді доводиться досліджувати декілька рішень. Отже, власники архітектури або навіть старші розробники можуть написати достатньо коду, щоб вивчити архітектурні зміни, перш ніж продовжити розробку. Цей процес дослідження може бути дорогим для складних архітектурних змін. Власники архітектури можуть об’єднатися з іншим розробником, щоб досліджувати складні архітектурні зміни. Таким чином, практикуючим розробникам іноді може знадобитися використовувати ітеративний і поетапний шлях еволюції архітектури, використовуючи введений процес управління змінами. Цей ітеративний і поетапний спосіб архітектури може бути трудомістким і водночас потужним засобом для зниження ризику невдачі проєкту.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

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

Для галузі ІТ інформаційна культура є необхідною умовою виживання, тому що зміна технологій в розробці програмного забезпечення відбувається кожні 6-8 місяців, а інвестиції на підготовку персоналу і освоєння нової технології величезні і великих компаніях варіюються від 1,5 до 2 млрд. доларів на рік [41].

Аналіз свідчить, що інформатизація та інтеграція комунікаційного простору України сприяє різкому підвищенню інформаційної та професійної компетентності, ділової активності, стимулюванню конкуренції, створенню інноваційних підприємств та організацій, нових робочих місць, зниженню витрат на утримання управлінського апарату [41]. Поряд із задачами і здобутками окреслилися негативи використання інформаційних технологій:

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

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

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

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

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

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

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

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

У [42] наведено перелік протипоказань з боку органів зору та загальних (соматичних) протипоказань, які забороняють роботу на ЕОМ, а також комплекс вправ для поліпшення здоров’я і підвищення працездатності.

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

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

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

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

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

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

Третій – активізація – відповідає фазі підвищеної збудженості.

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

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

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

4.2 Попередження аварій на виробництвах із застосуванням хлору

Хлор є частиною таблиці хімічних елементів і розташовується в ній під номером 17. У природі він зустрічається виключно у формі газу. Найчастіше він має специфічний зелений з жовтим переливом колір. Цей елемент важчий за повітря в 2,5 рази, тому накопичується в підвалах будинків, а на пересіченій місцевості в ярах і низинах. У воді ж хлор розчиняється без сліду і його наявність помітно тільки при великій концентрації (за рахунок специфічного запаху) [43].

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

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

Отруєння хлором можливе в разі:

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

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

– відбілювання і прання в закритому не провітрюваному приміщенні;

– аварії на підприємстві;

– використання хлору в якості зброї масового ураження.

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

Ознаки отруєння хлором. До перших ознаках отруєння хлором відносяться:

– дискомфорт і подразнення слизової дихальних шляхів;

– підвищене слиновиділення і спазм голосових зв’язок;

– кашель і утруднене дихання;

– відчуття різі та печіння в очах, сльозотеча;

– нудота і гіркота у роті;

– головні болі і можливі судоми.

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

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

До прибуття медиків слід надати домедичну допомогу потерпілому:

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

– забезпечити доступ чистого повітря;

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

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

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

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

До профілактичних заходів отруєння хлором належать:

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

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

– регулярні перевірки концентрацій хлору в повітрі робочої зони;

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

– дотримання вимог безпеки у використанні хлорвмісних рідин в побуті.

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

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

б) для захисту очей – захисними окулярами;

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

При проведенні попереднього (під час прийняття на роботу) та періодичних (протягом трудової діяльності) медичних оглядів працівники підлягають огляду оториноларинголога, дерматолога, офтальмолога. При виявленні медичних протипоказів працівники не допускаються до роботи з даним шкідливим фактором.

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

ВИСНОВКИ

Координація та узгодження архітектури програмного забезпечення між автономними командами визначено як виклик для Agile-розробки. Модель Spotify є прикладом підходу Agile, який керується створенням автономних, але узгоджених команд. Однак модель Spotify не надає вказівок щодо узгодження та керування архітектурними рішеннями в автономній команді.

Щоб дослідити, як архітектуру програмного забезпечення можна керувати та узгоджувати за допомогою масштабування моделі Spotify, у умовній компанії з розробки програних продуктів для FinTech було проведено дослідження, щоб отримати глибоке розуміння того, як використовується модель Spotify.

У цьому дослідженні ми провели пряме спостереження за практикою Agile шляхом проведення напівструктурованих відкритих інтерв'ю. Ми виявили, що компанія використовує централізований архітектурний процес, найнявши архітектора через складність програмного продукту. Це централізоване прийняття архітектурних рішень негативно вплинуло на автономію команд. Повноваження щодо прийняття рішень повинні існувати на оперативному рівні, щоб забезпечити автономію команд. Однак у моделі Spotify бракує практики управління Agile-архітектурою в автономних командах.

У цьому дослідженні ми висвітлюємо поточну проблему для архітектурного управління та узгодження у масштабній програмі Agile-розробки (2–9 команд з менш ніж 100 людьми) під час реалізації моделі Spotify. Ця прогалина вказує на те, що ми можемо зробити свій внесок шляхом:

1) розробки та оцінки підходу до архітектурного управління та

2) адаптації підходу Heterogeneous Tailoring для врахування запропонованого нами підходу.

Наш підхід до архітектурного управління вводить конкретні ролі архітектора (тобто власників архітектури та архітектора підприємства), щоб перетворити прийняття архітектурних рішень на операційний рівень. Роль власника архітектури покладається на керівників відділів або досвідчених

розробників з різними наборами навичок, які відповідають за координацію та керування Agile-архітектурою. Оскільки розділи формуються на основі областей компетенції – для горизонтального (на рівні виконавців однієї ролі) узгодження людей – і команди узгоджуються на рівні продукту під час спільного створення продукту, власники архітектури узгоджуються, відповідно, також. Роль Enterprise Architect є ключовою у вирішенні Agile архітектури підприємства та масштабуванні Agile архітектури у великих організаціях. Enterprise Architect працює з управлінськими рішеннями (і з власниками продуктів) і близько до власників архітектури, щоб вирішувати архітектурні рішення на рівні узгодження з політикою і баченням продукту на рівні компанії. У цьому підході власники архітектури знаходяться у віртуальній команді, яку очолює Enterprise Architect.

Ми запровадили процес управління змінами архітектури, щоб керувати зацікавленими сторонами (тобто розробниками, власниками архітектури, архітекторами підприємства та власниками продуктів) у керуванні та узгодженні рішень на основі архітектури в автономних групах моделі Spotify. Процес управління змінами архітектури включає в себе набір заходів і практик. Це архітектурний аналіз і синтез (дія 1 і 2), архітектурна оцінка та аналіз впливу (дія 3), архітектурний рефакторинг (дія 6), архітектурний супровід та еволюція (перехід від дії 6 назад до дії 1). Однак діяльність з опису та розуміння архітектури певною мірою використовується на рівні архітектури підприємства. Крім того, за нашими спостереженнями, архітектурне повторне використання практикується в командах і заохочується лідерами відділу.

Ми адаптували підхід узгодження гетерогенних команд для застосування нашого підходу до архітектурного управління з використанням моделі Spotify. Отже, підхід узгодження гетерогенних команд включає в даний час чотири основних функції: розробка продукту, узгодження, автономні команди і стратегія випуску релізів. Цей аналіз змін вимог на архітектурні рішення виробляє нові робочі елементи, які дозволяють архітектору програмного

забезпечення приймати рішення. Ця зміна в розробці продукту полегшує узгодження автономних команді, отже, підвищує їх автономію.

По-друге, узгодження архітектурних рішень спричинило введення структурних змін в колективи, зайняті у розробці для реалізації процесу управління змінами архітектури.

По-третє, введена зміна організаційної структури поліпшила автономію команд в плані вироблення та впровадження архітектурних рішень. Крім того, процес управління змінами збалансував Agile і архітектурybq процеси з метою зміцнення автономії команд. Нарешті, ці автономні команди працюють незалежно один від одного і співпрацюють, щоб створити робочу архітектуру, яка дозволяє в майбутньому планувати традиційні scrum-спринти, або / та робочий продукт.

Зокрема, ми виявили покращення, які пом’якшують недоліки, висвітлені до впровадження нашого підходу до архітектурного управління, пов’язані з такими аспектами/

– Перетворення архітектурних рішень на децентралізоване прийняття рішень.

– Створення технічної та корпоративної архітектурної узгодженості для повного програмного продукту, що, у свою чергу, вирішує конфлікти в архітектурних рішеннях та пом’якшує ключові технічні ризики в командах.

– Обмін архітектурними знаннями між автономними командами.

– Мінімізація даремних зусиль на архітектурний рефакторинг.

– Підвищення якості програмного забезпечення.

Незважаючи на те, що є покращення щодо управління архітектурними ризиками в автономних підрозділах, все ж очевидно, що управління архітектурою залишається проблемою, яка потребує чітко визначених методів. Такий негативний момент виникає через відсутність практики, яка полегшує дослідження архітектурних аспектів перед визначенням пріоритету нових історій користувачів, що може негативно вплинути на планування. Однак

складні архітектурні рішення остаточно опрацьовуються пізніше в життєвому циклі, як показано в процесі управління змінами архітектури. Такий феномен пов'язаний з рішеннями архітектури щодо того, які зміни архітектури необхідно провести, щоб мати прийнятне співвідношення затрачених ресурсів та впливу змін вимог.

Запроваджений підхід до архітектурного управління був оцінений в рамках однієї організації, яка розробляє проект FinTech за допомогою моделі Spotify. Опитані-учасники виконували різні ролі: розробники, власники архітектури, архітектор підприємства, власник продукту та Agile Coach. Спостережувані церемонії включають планування беклогу проєкту, сесії планування спринтів, ретроспективи, щоденні стендапи, зустрічі з синхронізації вимог та збори команди архітекторів.

Ці великі організації зосередити увагу на технологічних інноваціях та Agile, щоб швидко адаптуватися до нових ринкових можливостей при дотриманні галузевих нормативних обмежень.

Попередня оцінка дала можливість дослідникам і практикам отримати уявлення про те, як розроблений підхід до архітектурного управління може бути використаний на практиці. Насправді, наш аналіз не виявив жодних доказів того, що наш підхід міг би бути обмежений конкретними контекстуальними чи проектними факторами.

Архітектура програмного забезпечення є одним із ключових технічних досягнень у сфері програмної інженерії за останні десятиліття. Роль архітектури програмного забезпечення важлива у широкомасштабній Agile-розробці, оскільки кільком командам потрібно працювати разом, щоб випустити один продукт без погіршення автономності команд. Ми виявили проблему в управлінні та узгодженні архітектури Agile між автономними командами під час використання моделі Spotify. У моделі Spotify відсутні методи управління Agile архітектурою.

Є два ключові внески цієї роботи. Спочатку ми розробили та оцінили підхід до архітектурного управління при використанні моделі Spotify.

Представлені характеристики запропонованого підходу до архітектурного управління, його переваги та проблеми. По-друге, ми адаптували нову модель Heterogeneous Tailoring, щоб пристосувати наш підхід до архітектурного управління.

Наш підхід до архітектурного управління спрямований на покращення вирівнювання команд без шкоди для їхньої автономії. Цей підхід включає структурні зміни та процес управління змінами архітектури. Структурна зміна включає:

1) надання лідерам відділів і досвідченим розробникам нової ролі власників архітектури,

2) зміну обов'язків архітектора, щоб зосередитися на архітектурі підприємства, і

3) розміщення нових власників архітектури у віртуальній команді, яку очолює архітектор підприємства.

Щоб спростити процес архітектури, також запровадили процес управління змінами архітектури, метою якого є керівництво зацікавленими сторонами (тобто розробниками, власниками архітектури, архітекторами підприємства та власниками продуктів) в управлінні та узгодженні архітектурних рішень між автономними командами.

Розглянутий приклад говорить про декілька переваг впровадження підходу архітектурного управління на основі Heterogeneous Tailoring. Ці переваги включають:

1) децентралізацію процесу прийняття архітектурних рішень,

2) створення технічної та корпоративної архітектурної узгодженості для повного програмного продукту,

3) вирішення конфліктів в архітектурних рішеннях та зменшення ключових технічних ризиків у автономних підрозділах,

4) обмін архітектурними знаннями між командами,

5) мінімізація витрат на архітектурний рефакторинг,

6) покращення якості продукції та пом’якшення перешкод для узгодження архітектурних рішень між автономними командами ,

7) балансування зусиль для архітектурної якості полегшує створення загальних функцій програмного забезпечення . Отже, узгодження та управління архітектурними рішеннями покращили автономію команд.

СПИСОК ВИКОРСИТАНИХ ДЖЕРЕЛ

1. Bodnarchuk, Ihor, et al. "Adaptive Method for Assessment and Selection of Software Architecture in Flexible Techniques of Design." 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT). Vol. 1. IEEE, 2018.

2. Ihor, Bodnarchuk, et al. "Multicriteria choice of software architecture using dynamic correction of quality attributes." International Conference on Computer Science, Engineering and Education Applications. Springer, Cham, 2019. Abrahamsson P, Babar MA, Kruchten P (2010) Agility and architecture: Can they coexist? IEEE Softw 27(2):16–22.

3. Kharchenko, Olexandr, et al. "Optimization of software architecture selection for the system under design and reengineering." 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET). IEEE, 2018.

4. Kharchenko, A., et al. "Trade-off optimization in the problem of software system architecture choice." 2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). IEEE, 2016.

5. Anderson C, McMillan E (2003) Of ants and men: self-organized teams in human and insect organizations. Emergence 5(2):29–41.

6. Bass J, Salameh A (2020) Architectural Governance Interview Guide. University of Salford, UK.

7. Bellomo S, Kruchten P, Nord RL, Ozkaya I (2014) How to Agilely architect an Agile architecture. Cut IT J 27(2):12–17.

8. Booch G (2009) The defenestration of superfluous architectural accoutrements. IEEE Softw 26(4):7–8.

9. Braun V, Clarke V (2006) Using thematic analysis in psychology. Qual Res Psychol 3(2):77–101.

10. Buschmann F, Henney K (2013) Architecture and agility: married, divorced, or just good friends? IEEE Softw 30(2):80–82.

11. Cockburn A, Highsmith J (2001) Agile software development, the people factor. Computer 34(11):131–133.

12. Conboy K, Carroll N (2019) Implementing large-scale Agile frameworks: challenges and recommendations. IEEE Softw 36(2):44–50.

13. Dingsøyr T, Dybå T, Moe NB (2010) Agile software development: current research and future directions, 1st edn. Springer Publishing Company (Incorporated)

14. Erdogmus H (2009) Architecture meets agility. IEEE Softw 26(5):2–4.

15. Gimpel H, Rau D, Röglinger M (2018) Understanding FinTech startups – a taxonomy of consumer-oriented service offerings. Electron Mark 28(3):245–264.

16. Glaser BG (1998) Doing grounded theory: issues and discussions. Sociology Press.

17. Hammond S, Umphress D (2012) Test driven development: the state of the practice. In: Proceedings of the 50th Annual Southeast Regional Conference, Tuscaloosa, Alabama.

18. Hoda R, Noble J, Marshall S (2013) Self-organizing roles on Agile software development teams. IEEE Trans Software Eng 39(3):422–444.

19. Kilu E, Milani F, Scott E, Pfahl D (2019) Agile software process improvement by learning from financial and fintech companies: LHV Bank Case Study. Software Quality: the complexity and challenges of software engineering and software quality in the cloud.

20. Knewtson HS, Rosenbaum ZA (2020) Toward understanding FinTech and its industry. Managerial Fin 46(8):1043–1060.

21. Kniberg H (2014) Spotify Squad framework. Retrieved July, 2020 from https:// medium.com/ project-management-learnings/.

22. Kniberg H, Ivarsson A (2012). Scaling Agile @ Spotify with tribes, squads, chapters & guilds. Retrieved June, 2020 from https://blog.crisp.se/wp-content/uploads/ 2012/11/ SpotifyScaling.pdf.

23. Lincoln YS, Guba EG (1985) Naturalistic inquiry. Sage Publications, Berlin

24. Linders B (2016) Don’t copy the Spotify model. https://www.infoq.com/news/ 2016/10/no-Spotify-model.

25. Martini A, Bosch J (2016) A multiple case study of continuous architecting in large Agile companies: current gaps and the CAFFEA framework. In: 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA).

26. Moe NB, Dingsøyr T (2008) Scrum and team effectiveness: theory and practice. Agile Processes in Software Engineering and Extreme Programming, Berlin.

27. Moe NB, Dingsøyr T, Dybå T (2008) Understanding self-organizing teams in Agile software development. In: Proceedings of the 19th Australian Conference on Software Engineering (aswec 2008).

28. Moe NB, Olsson HH, Dingsøyr T (2016) Trends in large-scale Agile development: a summary of the 4th workshop at XP2016. In: Proceedings of the Scientific Workshop Proceedings of XP2016.

29. Nord RL, Ozkaya I, Kruchten P (2014) Agile in distress: architecture to the rescue. Agile methods. Large-scale development, refactoring, testing, and estimation.

30. Robinson H, Segal J, Sharp H (2007) Ethnographically-informed empirical studies of software practice. Inf Softw Technol 49(6):540–551.

31. Salameh A, Bass JM (2018) Influential factors of aligning Spotify squads in mission-critical and offshore projects – a longitudinal embedded case study. Product-Focused Software Process Improvement.

32. Salameh A, Bass JM (2019a) Spotify tailoring for B2B product development. In: Proceedings of the 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) Salameh A, Bass JM (2019b) Spotify tailoring for promoting effectiveness in cross-functional autonomous squads. Agile processes in software engineering and extreme programming – Workshops.

33. Salameh A, Bass JM (2020) Heterogeneous tailoring approach using the Spotify model. In: EASE ’20 proceedings of the evaluation and assessment on software engineering, Trondheim, Norway.

34. Šmite D, Moe NB, Levinta G, Floryan M (2019) Spotify guilds: how to succeed with knowledge sharing in large-scale Agile organizations. IEEE Softw 36(2):51–57.

35. Šmite D, Moe NB, Floryan M, Levinta G, Chatzipetrou P (2020) Spotify guilds. Commun ACM 63(3):56–61.

36. Stray V, Moe NB, Hoda R (2018) Autonomous Agile teams: challenges and future directions for research. In: Proceedings of the 19th international conference on Agile software development: companion, Porto, Portugal.

37. Yang C, Liang P, Avgeriou P (2016) A systematic mapping study on the combination of software architecture and Agile development. J Syst Softw 111:157–184.

38. Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

39. Abrahamsson P, Babar MA, Kruchten P (2010) Agility and architecture: Can they coexist? IEEE Softw 27(2):16–22

40. Информационная культура предприятий, виды информационной культуры, информационное поведение [Електронний ресурс]. – Електронні дані (Лекції). –Режим доступу: https://lektsii.com/1-78900.html

41. Пивоваров М.Г., Медко Д.А. Перспективы создания и развития информационно-коммуникационной системы Украины // Економіка: проблеми теорії та практики: Зб. наук. праць. – Вип. 49. –Дніпропетровськ: Дніпропетр. Нац. Ун-т, 2000. – С. 56-61.

42. 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

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

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

IX НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ
2021

УДК 004.42
Н. Базюк, А. Флейтута
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ІНЖЕНЕРІЯ ВИМОГ ДО ПРОГРАМНОГО ПРОДУКТУ В ГНУЧКИХ ТЕХНОЛОГІЯХ РОЗРОБКИ

UDC 004.42
N. Baziuk, A. Fleituta

SOFTWARE REQUIREMENTS ENGINEERING IN AGILE DEVELOPMENT

Управління ризиками визнано однією з важливих галузей знань в управлінні проєктами. Підхід до управління ризиками відрізняється як для традиційної, так і для гнучкої методології управління проєктами. Попередні дослідження з управління ризиками підкреслюють, що традиційне управління ризиками є процесом, який керується важким планом, і передбачає, що більшість вимог проєкту наявні до початку проєкту. З іншого боку, Agile-менеджмент проєктів використовує методологію, за якою результати поставляються ітераційно. Таким чином, гнучка методологія спрямована на мінімізацію тривалості проєкту, визначення пріоритетності результатів з високою вартістю, а також покращення якості результатів проєкту та зниження ризику.

Цілі застосування традиційного та гнучкого підходу в управлінні проєктами різні і дають найкращий результат лише за певних обставин. Традиційний підхід до управління ризиками проєкту, управління ризиками виконується під час ініціації проєкту, і це підходить, оскільки проєкти з високими термінами вимагають належного планування через поєднання ризиків, розтягнутих вздовж терміну.

У гнучкому підході до управління ризиками проєкту, через ітераційний характер управління, очевидно, що процеси управління ризиками виконуються через часті проміжки часу, оскільки проєкт розділений на кілька фаз. Це допомагає детально визначити ризики, пов'язані з етапами проєкту, їх серйозність та вплив. Крім того, оскільки фази розділені, терміни, протягом яких необхідно виконувати управління ризиками, скорочуються, що допомагає краще зрозуміти природу ризику та зменшує його невизначеність. Також ймовірність пропуску будь-якого ризику мінімізується за допомогою гнучкого підходу до управління ризиками проєкту.

В Agile проєктах найкритичнішими є проблеми, пов'язані з людьми. Справді, одним з найважливіших факторів успіху в Agile проєкті є індивідуальна компетентність. Крім того, оцінка зусиль є постійною задачею для команди розробників Agile, особливо коли вона виконується вперше, і є проблеми з навичками Agile і плинністю кадрів. У Scrum індивідуальна мотивація дуже важлива і впливає на те, наскільки старанні члени команди.

Визнання недотримання усталеної практики може дати ранні ознаки ризиків, наприклад, низький моральний дух, виражений під час щоденної зустрічі, або уникнення обговорення проблем у разі відставання від графіка. Через виявлені проблеми існує сильна мотивація покращити управління ризиками в Agile проєктах, не зменшуючи їх гнучкість. Сучасне управління ризиками має бути в змозі інтегруватися в гнучкий процес для підтримки прийняття рішень.

Процес управління ризиками застосовується відповідно до типу, розміру та складності проєкту. Agile Risk Management здійснюється більше за допомогою практик, ніж уявлення. Багато практик Agile спрямовані на виявлення та пом'якшення ризиків протягом усього проєкту. Однак у реальному житті більшість проєктів вимагають поєднання обох підходів, і можна зробити висновок, що як гнучкі, так і традиційні методології управління ризиками доповнюють одна одну. Крім того, розвиток моделі ARM зменшує зусилля в управлінні ризиком.