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: Модель для підтримки процесу неперервної інтеграції у великих IT-проектах в умовах зміни вимог до програмного продукту

Виконав(ла): студент(ка) 6 курсу, групи СТм-61
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)

(підпис) Леськів А.І. (прізвище та ініціали)

Керівник (підпис) Млинко Б.Б. (прізвище та ініціали)

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

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

Рецензент (підпис) Кареліна О.В. (прізвище та ініціали)

Тернопіль
2020

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 2020 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 126 Інформаційні системи та технології
(шифр і назва спеціальності)

студенту Леськів Андрій Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи **Модель для підтримки процесу неперервної інтеграції у великих IT-проектах в умовах зміни вимог до програмного продукту**

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

Затверджені наказом ректора від «*06*» *листопада* 2020 року № *4/7-826*

2. Термін подання студентом завершеної роботи 22 грудня 2020 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП; РОЗДІЛ 1 ЗАГАЛЬНИЙ ОПИС ПРОЦЕСІВ НЕПЕРЕРВНОЇ ІНТЕГРАЦІЇ; 1.1 Необхідність використання процесу неперервної інтеграції; 1.2 Конвеєр CI/CD; РОЗДІЛ 2 АНАЛІЗ ПРОБЛЕМИ; 2.1 Систематичне створення цінності шляхом експериментів; 2.2 Дослідницький підхід; 2.3 Контекст проведення досліджень; РОЗДІЛ 3 РОЗРОБКА МОДЕЛІ ЕКСПЕРИМЕНТУ НА ОСНОВІ ПРОЦЕСІВ CI/CD; 3.1 Модель для безперервного експериментування з імплементацією змін вимог користувача; 3.2 Модель процесу для безперервного експериментування з використанням неперервної інтеграції; РОЗДІЛ 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ; ВИСНОВКИ; СПИСОК ЛІТЕРАТУРИ; ДОДАТКИ

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

1. Тема. 2. Наукова новизна. 3. Постановка задачі. 4. Конвеєр CI/CD. 5. Опис експериментального проекту. 6. Спрощена модель процесу випробування змін. 7. Повна модель процесу випробування змін. 8. Пропонована архітектура для інфраструктури проекту. 9. Артефакти технологічного процесу експериментальних проектів. 10. Висновки

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	Стадник І.Я., проф.		

7. Дата видачі завдання 21 вересня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	21.09.20-27.09.20	*Виконано*
2.	Підбір наукових джерел по темі роботи	28.09.20-04.10.20	*Виконано*
3.	Переклад та опрацювання наукових джерел	05.10.20-11.10.20	*Виконано*
	по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо огляду атак на	12.10.20-18.10.20	*Виконано*
	комп'ютерні системи		
5.	Оформлення першого розділу	19.10.20-25.10.20	*Виконано*
6.	Оформлення другого розділу	26.10.20-01.11.20	*Виконано*
7.	Оформлення третого розділу	02.11.20-08.11.20	*Виконано*
8.	Виконання завдання до підрозділу «Охорона праці»	09.11.20-15.11.20	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	16.11.20-22.11.20	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	23.11.20-29.11.20	*Виконано*
11.	Нормоконтроль	30.11.20-05.12.20	*Виконано*
12.	Перевірка на плагіат	05.12.20	*Виконано*
13.	Попередній захист кваліфікаційної роботи	14.12.20	*Виконано*
14.	Захист кваліфікаційної роботи	23.12.2020	

Студент _____ Леськів А.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Млинко Б.Б..
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Модель для підтримки процесу неперервної інтеграції у великих IT-проектах в умовах зміни вимог до програмного продукту" // Леськів Андрій Ігорович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СТм-61 // Тернопіль, 2020 // с. – , рис. – , табл. – , джерел – .

Ключові слова: НЕПЕРЕВНА ІНТЕГРАЦІЯ, ВИМОГИ, УПРАВЛІННЯ ПРОЕКТОМ, ЕКСПЕРИМЕНТ, АВТОМАТИЧНІ ТЕСТИ.

Розробка програмно-інтенсивних продуктів та послуг дедалі частіше відбувається шляхом постійного розгортання побільшень продуктів чи послуг, таких як нові функції та вдосконалення, серед споживачів. Розробники продуктів та послуг повинні постійно з’ясовувати, чого хочуть замовники, безпосередньо отримуючи зворотний зв’язок від споживачів та користувачів.

Таким чином в роботі досліджено процес неперервного впровадження нових вимог в програмний продукт шляхом виконання випробувань імплементованих змін з використанням системи неперервної інтеграції.

ANNOTATION

"A model for continuous integration assurance in big IT-projects under software requirements change conditions" // Diploma paper of Master degree level // Leskiv Anrii Ihorovytch // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group СТм-61 // Ternopil, 2020 // p. – , Fig. – , Tables – , Refence. – .

Key words: CONTINUOUS INTEGRATION, REQUIREMENTS, PROJECT MANAGEMENT, EXPERIMENT, AUTOMATIC TESTS.

Software products and services are increasingly being developed by constantly expanding product or service increases, such as new features and improvements, among consumers. Developers of products and services must constantly find out what customers want by receiving direct feedback from consumers and users.

Thus, the process of continuous introduction of new requirements into the software product by performing tests of implemented changes using the system of continuous integration is investigated in the work.

ЗМІСТ

ВСТУП

Актуальність теми.

Прискорення цифровізації у більшості галузей промисловості означає, що все більша кількість компаній є або незабаром буде постачальником програмних продуктів та послуг. Одночасно нові компанії вже виходять на ринок як компанії-виробники програмного забезпечення. Програмне забезпечення забезпечує більшу гнучкість у видах послуг, які можна надати, навіть після того, як початковий продукт був переданий у користування клієнтам. Багато обмежень, які існували раніше, особливо з точки зору поведінки товару чи послуги, тепер можна усунути.

Завдяки цій новій гнучкості для компаній перестає бути головним завданням те, як виявляти та вирішувати технічні проблеми, а те, як вирішувати проблеми, які є актуальними для споживачів, і тим самим забезпечувати цінність продукту. Пошук рішень цієї проблеми часто був випадковим і базувався на здогадах, але багато успішних компаній підходили до цього питання систематично. Нещодавно було запропоновано сімейство загальних підходів. Наприклад, методологія [1] пропонує триступеневий цикл: побудувати, виміряти, навчитися.

Однак детальна структура проведення систематичної розробки програмного забезпечення не розроблена. Така структура має наслідки для технічної інфраструктури продукту, процесу розробки програмного забезпечення, вимоги в відношенні навичок, що розробники програмного забезпечення повинні розробляти, виконувати, аналізувати та інтерпретувати змінювані вимоги, необхідні для роботи і управління компанією, заснованою на адаптації до вимог в дослідженнях і розробках.

Мета роботи: У цій роботі розглядаються передумови для створення системи для постійних оновлень на основі зміни вимог замовника.

Методи та підходи для постійного оновлення продукту через мінливі вимоги самі по собі повинні базуватися на емпіричних дослідженнях. У цій роботі ми представляємо найважливіші елементи для безперервного оновлення.

Зокрема, в роботі потрібно вирішити такі задачі:

1. Як можна систематично організовувати безперервну інтеграцію з програмними продуктами в умовах зміни вимог?

2. Підібрати адекватну модель для впровадження CI/CD.

3. Розробити архітектуру інфраструктури проекту для впровадження CI/CD.

Об'єкт дослідження: процес неперервної інтеграції програмного коду в робочий програмний продукт в умовах часто змінюваних вимог замовника.

Предмет дослідження: інфраструктура проекту з розробки програмного продукту.

Методи дослідження: Для вирішення поставлених задач були використані наступні методи дослідження:

– метод аналізу та узагальнення для збору інформації про стан вирішуваної задачі;

– метод імітаційного моделювання для розробки моделі впровадження процесів безперервної інтеграції.

Наукова новизна отриманих результатів. Під час виконання роботи були отримані результати:

– здійснено огляд задач, які вирішуються при впровадженні систем неперервної інтеграції;

– запропоновано модель інфраструктури компанії для ефективного використання процесів CI/CD.

Практичне значення отриманих результатів. Подальше поглиблення запропонованих рішень та їх адаптація до умов реальних проектів і компаній дозволить підвищити ефективність роботи команди розробників.

Апробація результатів та особистий внесок здобувача. Результати кваліфікаційної роботи опубліковані в тезах доповідей на студентській науковій конференції, проведена в ТНТУ у грудні 2020 року.

РОЗДІЛ 1
ЗАГАЛЬНИЙ ОПИС ПРОЦЕСІВ НЕПЕРЕРВНОЇ ІНТЕГРАЦІЇ

1.1 Необхідність використання процесу неперервної інтеграції

Безперервна інтеграція (Continuous Integration – CI) – це практика розробки програмного забезпечення, коли члени команди часто мають потребу добавляти написаний код (інтегрувати свою роботу) в робочий продукт; зазвичай кожна людина інтегрується щонайменше щодня – що призводить до кількох загальних інтеграцій на день. Кожна інтеграція перевіряється автоматизованою збіркою (включаючи тест) для виявлення помилок інтеграції якомога швидше. Багато команд виявляють, що такий підхід призводить до суттєвого зменшення проблем інтеграції та дозволяє команді швидше розробляти цілісне програмне забезпечення [2].

Найпростіше пояснити, що таке CI і як воно працює на прикладі з розробкою невеликої функції. Припустимо, що треба щось зробити з частиною програмного забезпечення, насправді неважливо, в чому полягає завдання, яке може бути виконане за кілька годин.

Почнемо з того, що беремо копію поточного інтегрованого джерела на свою локальну машину розробки. Це легко зробити за допомогою системи управління вихідним кодом (системи контролю версій). Система управління вихідним кодом зберігає весь вихідний код проекту у сховищі. Поточний стан системи зазвичай називають "master". У будь-який час розробник може зробити контрольовану копію основної лінії на власній машині, це називається "checkout". Копія на машині розробника називається "production version".

Тепер у зроблені копії можна виконувати довільні зміни для реалізації згаданої функції, тобто, щоб виконати своє завдання. Виконана робота буде містити як зміну коду, так і додавання або зміну автоматизованих тестів. Постійна інтеграція передбачає високий ступінь тестів, які автоматизовані. Часто вони виконуються за допомогою відповідного програмного інструменту, наприклад XUnit.

Після завершення виконується автоматична збірка на своїй машині розробника. Це означає взяти змінений вихідний код у робочій копії, компіляцію та зв'язування

його з виконуваним файлом та запуск автоматизованих тестів. Тільки якщо всі збірки та тестування здійснюються без помилок, загальна збірка вважається хорошою.

Маючи гарну збірку, можна подумати про внесення змін до сховища. Проблема, звичайно, полягає в тому, що інші люди могли і зазвичай вносять зміни до основної вітки раніше. Отже, спочатку треба оновити свою робочу копію з її змінами. Якщо інші зміни зіткнуться поточними змінами, це виявиться як збій або в компіляції, або в тестах. У цьому випадку розробник несе відповідальність за виправлення і буде це повторювати, поки не зможе створити робочу копію, яка належним чином синхронізується з основною віткою.

Після того, як розробник зробив власну збірку з правильно синхронізованої робочої копії, він може нарешті внести свої зміни в основну вітку, яка потім оновлює сховище.

Далі на машині інтеграції на основі основного коду треба знову виконати побудову збірки. Лише коли ця збірка вдається, ми можемо сказати, що зміни зроблені. Завжди є ймовірність того, що розробник щось пропустив на своїй машині, і репозиторій не був належним чином оновлений. Тільки тоді, коли нові цілеспрямовані зміни успішно будуються на інтеграції, робота завершена.

Якщо між двома розробниками виникає конфлікт, то це зазвичай трапляється, коли вони працюють на д ним і тим же фрагментом програмного коду. На даний момент найважливішим завданням є виправити це і змусити збірку знову працювати належним чином. У середовищі безперервної інтеграції ніколи не повинно бути невдалого побудови інтеграції надовго. Хороша команда повинна мати багато правильних збірок на день. Погані збірки трапляються час від часу, але їх слід швидко виправляти.

Результатом цього є наявність стабільного програмного забезпечення, яке працює належним чином і містить мало помилок. Кожен розробляє цю спільну стабільну базу і ніколи не відходить настільки далеко від цієї бази, що потрібно дуже багато часу, щоб інтегруватися з нею. На пошуки помилок витрачається менше часу, оскільки вони швидко виявляються.

Проекти програмного забезпечення передбачають наявність великої кількості файлів, які потрібно об’єднати для створення продукту. Відстеження всього цього вимагає зусиль, особливо коли залучено кілька людей. Тож не дивно, що протягом багатьох років команди розробників програмного забезпечення створили інструменти для управління вихідним кодом. Ці інструменти – це інструментами управління вихідним кодом, управління конфігурацією, системами контролю версій, сховищами чи різними іншими назвами – є невід'ємною частиною більшості проектів розробки. Сумно і дивно те, що вони не є частиною всіх проектів. Хоча це трапляється рідко.

Однією з особливостей систем контролю версій є те, що вони дозволяють створювати кілька гілок, обробляти різні потоки розвитку. Це корисна, дуже важлива функція – але вона часто створює людям проблеми. Зведіть кількість своїх гілок до мінімуму.

Перетворення коду на запущену систему часто може бути складним процесом, що включає компіляцію, переміщення файлів, завантаження схем у бази даних тощо. Однак, як і більшість завдань у цій частині розробки програмного забезпечення, він може бути автоматизований – і, як результат, повинен бути автоматизований. Просити людей вводити дивні команди або клацати по діалогових вікнах – це марна трата часу та ґрунт для помилок.

Традиційно збірка означає компіляцію, зв’язування та всі додаткових матеріалів, необхідні для запуску програми. Програма може працювати, але це не означає, що вона працює правильно. Сучасні статично набрані мови можуть виявити багато помилок, але набагато більше проскакують через цю мережу.

Хороший спосіб виловлювати помилки швидше та ефективніше – це включити автоматизовані тести в процес побудови. Звичайно, тестування не є ідеальним, але воно може виявити багато помилок, достатньо для того, щоб бути корисними. Зокрема, розвиток Екстремального Програмування (XP) та Тестової Розробки (TDD) зробив багато для популяризації коду самотестування, і в результаті багато людей побачили цінність цієї методики.

Для самотестування коду вам потрібен набір автоматизованих тестів, які можуть перевірити значну частину бази коду для помилок. Результат запуску

тестового набору повинен вказувати на те, що якісь тести не пройшли. Щоб збірка була самотестуючою, збій тесту повинен спричинити збій збірки.

За останні кілька років розвиток TDD популяризував сімейство інструментів OpenSource XUnit, які ідеально підходять для такого роду тестування.

Звичайно, ви не можете розраховувати на тести, щоб знайти все. Як часто кажуть: тести не доводять відсутність помилок. Однак досконалість – не єдиний момент, коли ви отримуєте окупність за самотестування. Недосконалі тести, які часто проводяться, набагато кращі за ідеальні тести, які взагалі ніколи не пишуться.

Той факт, що ви створюєте під час оновлення робочої копії, означає, що ви виявляєте конфлікти компіляції, а також текстові конфлікти. Оскільки збірка самотестується, ви також виявляєте конфлікти під час запуску коду. Останні конфлікти – це особливо незручні помилки, які можна знайти, якщо вони довгий час сидять невизначеними в коді. Оскільки між комітами є лише кілька годин змін, існує лише кілька місць, де проблема може ховатися. Крім того, оскільки мало що змінилося, ви можете використовувати diffdebugging, щоб знайти помилку.

Кожен розробник повинен привласнювати сховище щодня. На практиці часто корисно, якщо розробники здійснюють комітування частіше. Чим частіше ви виконуєте дії, тим менше місць вам доводиться шукати конфліктні помилки, і чим швидше ви виправляєте конфлікти.

Як результат, ви повинні переконатись, що регулярні побудови відбуваються на машині інтеграції, і лише якщо ця побудова інтеграції буде успішною, слід вважати, що коміт зроблено. Оскільки розробник, який здійснює комміт, відповідає за це, цьому розробнику потрібно стежити за побудовою основної вітки, щоб вони могли виправити помилки, які можуть виникнути.

Підхід до мануальної збірки є найпростішим для опису. По суті, це подібно до локальної збірки, яку розробник робить перед комітом у сховищі. Розробник переходить до машини інтеграції, перевіряє вершину основної вітки (яка тепер містить його останній коміт) і починає інтеграцію. Він стежить за його прогресом, і якщо збірка вдається, він робить зі своїм комітом.

Сервер безперервної інтеграції виконує роль монітора до сховища. Кожного разу, коли коміт проти сховища закінчує сервер, він автоматично перевіряє код на машині інтеграції, ініціює побудову та повідомляє комітеру про результат побудови.

1.2 Конвеєр CI/CD

Безперервна інтеграція (CI) – це практика розробки програмного забезпечення, при якій невеликі коригування базового коду в додатку перевіряються кожного разу, коли член команди вносить зміни. CI має на меті пришвидшити процес релізу, дозволяючи командам знаходити та виправляти помилки на початку циклу розробки, а також заохочувати посилення співпраці між розробниками – роблячи це важливою практикою для Agile-команд.

Історично склалося, що розробники працювали окремо над частинами програми, а згодом інтегрували свій код з рештою команди вручну. Залежно від того, коли відбулася наступна збірка, може знадобитися кілька днів, а то й тижнів, щоб побачити, чи не порушить новий код щось. Цей ізольований процес часто призводить до того, що розробники дублюють свої зусилля по створенню коду, розробляють різні стратегії кодування та створюють багато важких для пошуку та виправлення помилок. У середовищі IP розробники об’єднують зміни коду у спільному сховищі кілька разів на день, щоб його можна було постійно фіксувати, тестувати та перевіряти. Впровадження CI пришвидшує процес розробки та гарантує, що помилки будуть виявлені раніше в циклі.

Безперервна доставка (CD) це процес отримання нових збірок в руки користувачів якомога швидше. Це природний наступний крок за межами IT і є підходом, що використовується для мінімізації ризиків, пов’язаних із випуском програмного забезпечення та нових функцій. Випуск оновлень програмного забезпечення, як відомо, болючий та трудомісткий. Постійна доставка зменшує ризики та зусилля, пов’язані з цим процесом, гарантуючи, що кожна зміна базового коду програми є доступною – це означає, що кожне оновлення менше і може доставлятися користувачам частіше. Роблячи релізи менш драматичними подіями, які можна виконувати на вимогу, коли новий код готовий, команди можуть зробити свій

процес розробки більш ефективним, менш ризикованим та швидше отримати відгук від користувачів. Якщо проблеми виявляються у виробництві, їх можна швидко придушити, просто розгорнувши наступне оновлення.

Оскільки все більше команд розробників програмного забезпечення прагнуть задовольнити зростаючий попит на більш швидкі цикли випуску та підвищену якість програмного забезпечення, багато хто прагне впровадити конвеєр постійної розробки для впорядкування свого процесу. Застосування практики CI / CD дозволяє командам адаптувати своє програмне забезпечення на вимогу, щоб задовольнити відгуки користувачів, ринкові зміни та будь-які коригування загальної бізнес-стратегії (тобто зміни вимог).

Хоча остаточної структури конвеєра CI/CD не існує, вона зазвичай розбивається на такі етапи:

Комміт та побудова. Коли розробники закінчують вносити зміни в додаток, вони передають свій код спільному сховищу, яке потім інтегрує їх фрагмент із центральною базою коду. Потім програмний продукт або нова функція будується на основі отриманого коду та тестується модулем. CI відіграє важливу роль у впорядкуванні цього процесу шляхом автоматизації кожного кроку після написання коду.

Автоматизоване тестування. Після розробки нового програмного забезпечення його потрібно створити, а потім ретельно протестувати, щоб переконатися, що воно відповідає всім початковим вимогам. Існують різні методології тестування, які можна використовувати, щоб забезпечити вигляд та поведінку програми як слід, і охоплює все – від функціональних тестів до тестів продуктивності. На цьому етапі важливо протестувати всю систему в середовищах, подібних до виробничого, оскільки успіх програмного забезпечення залежить від того, чи працює воно в середовищах, до яких ваші кінцеві користувачі отримують до нього доступ.

Розгортання. На завершальній стадії вбудований програмний продукт впроваджується у виробництво. CD вимагає автоматизації цього процесу, що забезпечує надійну доставку користувачам.

Постійна інтеграція та доставка означає постійну якість. Після виконання тестів ви можете відстежувати зміни, об'єднувати модифікації та відновлювати попередні версії проекту або планувати, забезпечувати та часто розгортати збірки для постійної доставки. Конвеєр CI / CD вимагає команд розробників, тестів та операцій, щоб спільно створити цілісний процес випуску, а головне – автоматизація.

Проблема в тому, що ручне тестування також важливе. Це повсюдний процес, який досі використовується у всій галузі на кожному етапі циклу розвитку. Хоча воно має певну мету і завжди буде потрібно для певних видів тестування, таких як дослідницьке тестування, воно може перешкоджати ітеративному процесу доставки. Час є важливим у постійному середовищі, і ручне тестування може бути болісно млявим. Чим повільніше цикли зворотного зв'язку, тим довше буде потрібно для налаштування та випуску програмного забезпечення. Автоматизація прискорює цикл побудови та розгортання шляхом постійної перевірки коду та запуску тестів та надання зворотного зв'язку протягом декількох хвилин. Ось чому якомога більшу частину процесу слід автоматизувати, починаючи від модульних тестів і закінчуючи системними тестами і навіть забезпеченням середовища.

Наприклад, якщо ви постійно не фіксуєте свій код, у вас будуть триваліші періоди між інтеграціями, що ускладнює пошук та виправлення помилок. Дні очікування чи навіть тижні між збірками можуть легко зірвати проект з ладу, перш ніж він навіть досягне етапу тестування.

Етап тестування може бути розбитий на кілька частин, виходячи з цілей кожного з ваших тестових «наборів», кожна з яких розроблена для забезпечення відповідності програмного забезпечення початковим бізнес-вимогам. Незалежно від того, чи ви тестуєте функціональність програми, безпеку чи продуктивність, кожен пакет може і повинен бути автоматизований. За допомогою відповідного інструменту ви можете запускати регресійні тести для повторного тестування сценаріїв кожного разу, коли вносяться зміни, щоб переконатися, що наявна функціональність програми не порушена. Правильний інструмент також дозволить вам проводити паралельне тестування, тобто процес запуску декількох тестів або випадків тестів одночасно в різних браузерах або системах. Запуск тестів одночасно покращує охоплення тестом

і скорочує час тестування, що є ключовим для оптимізації ітеративного процесу тестування.

Часто упускається з уваги частина всього циклу – це те, як управляється тестовим середовищем. Більшість команд все ще витрачають години або дні на ручне створення, модернізацію та руйнування тестових середовищ. Найважливішим елементом забезпечення щасливих клієнтів є забезпечення того, щоб програмне забезпечення працювало в тому середовищі, до якого вони мають доступ, тому важливо тестувати в середовищах, що імітують виробниче середовище. Технологія сьогодні постійно змінюється, тобто нові операційні системи, версії браузерів та дозволи регулярно представляються споживачам. Якщо ви все ще керуєте тестовим середовищем вручну, ви витратите надзвичайно багато часу, намагаючись не відставати.

Автоматизоване забезпечення середовища дозволяє командам керувати тестовим середовищем лише за кілька кліків. Правильний інструмент автоматизованого тестування навіть забезпечить найновіші версії браузерів, систем та конфігурацій роздільної здатності – це означає, що команди з контролю якості можуть уникнути необхідності розкручувати, підтримувати або руйнувати середовища взагалі.

Все, що є у вашому циклі розробки програмного забезпечення, від модульних тестів до системних тестів і навіть забезпечення середовищем, має бути автоматизованим. Ви лише такі швидкі, як найповільніша фаза вашого трубопроводу, і один етап, проведений вручну, перетвориться на вузьке місце для всієї експлуатації.

Існує п’ять аспектів безперервного процесу розробки, які ви отримаєте від переходу на конвеєр CI / CD. Швидші цикли випуску: прискорення циклу складання та розгортання дозволить вам та вашій команді швидше вводити нові функції у виробництво, тобто ви можете швидше отримати ваш продукт у руках своїх споживачів.

Знижений ризик: Кінцевою метою безперервного процесу доставки є зробити кожен випуск менш драматичним і безболісним як для команд контролю якості, так і для клієнтів. Постійно випускаючи нові оновлення або функції, ви зменшуєте ризик

помилок, які закінчуються виробництвом, і можете швидше усунути виявлені недоліки.

Менші витрати. Прийняття моделі безперервної розробки зменшить ваші витрати, усунувши багато постійних витрат, пов’язаних зі створенням та тестуванням змін програми. Наприклад, автоматичне забезпечення середовищем зменшить витрати, пов’язані з підтримкою власної інфраструктури тестування. Паралельне тестування зменшить кількість машин, на яких потрібно запускати тести. Постійно фіксуючи свій код, ви витратите менше часу (а отже і грошей) на виправлення помилок.

Продукти вищої якості: Основним побоюванням впровадження трубопроводу CI / CD є попередження якості щодо швидкості, але це не так. Постійна інтеграція забезпечує посилену співпрацю між розробниками, тобто помилки виявляються та виправляються швидше раніше в процесі розробки. Запуск автоматичної регресії та паралельних тестів покращить охоплення тестом, забезпечивши, щоб ваш додаток не містив помилок і працював у більш широкому діапазоні середовищ. Постійне надходження менших оновлень до вашого програмного забезпечення змусить більшість змін (і помилок) не виявитись аж у кінцевого користувача.

Кращі переваги в бізнесі: Перехід до моделі постійного розвитку надає вашій команді гнучкість, щоб на ходу вносити зміни у ваше програмне забезпечення, щоб задовольнити нові ринкові тенденції та потреби користувачів. Ви зможете задовольнити швидко мінливі вимоги і перетворите процес релізу на конкурентну перевагу.

РОЗДІЛ 2 АНАЛІЗ ПРОБЛЕМИ

В умовах частих змін вимог до програмного продукту, коли він вже у використанні, внесення змін є важливим процесом, основна задача якого, крім власне імплементації самих змін, – це не порушити роботу програмного продукту, а отже не зменшити таку характеристику якості продукту, як рівень задоволеності клієнта (замовника) [3, 4, 5]. Саме через це таку роботу можна трактувати як виконання своєрідного експерименту. Зрозуміло, що будь-яке експериментування з робочим продуктом не сприяє його якісному функціонуванню та, відповідно, не підвищує рівень задоволеності клієнта.

Принципи роботи з замовниками [1] наголошують на важливості не тільки здійснення діяльності з розробки продуктів, але також вивчення та виявлення того, ким будуть первинні споживачі компанії та на яких ринках вони перебувають. Розробка споживачів стверджує, що для цієї діяльності необхідний окремий та чіткий процес. Робота з клієнтами – це чотириетапна модель, поділена на фазу пошуку та виконання. На етапі пошуку компанія здійснює виявлення клієнтів, перевіряючи, чи правильна бізнес-модель (відповідність продукту / ринку), та перевірку споживачів, яка розробляє повторювану модель продажів.

У світлі переваг, які може забезпечити така методологія, де контрольовані експерименти становлять основний напрямок розвитку діяльності, автори [6] описують етапи, які компанія повинна пройти, щоб досягти цієї мети, як “сходи на небо”. Цільовий етап досягається, коли організація програмного забезпечення функціонує як експериментальна система досліджень та розробок. Етапами на шляху досягнення цілі є: (i) традиційний розвиток, (ii) гнучка (Agile) науково-дослідна організація, (iii) безперервна інтеграція та (iv) постійне розгортання. Спочатку автори описують ці чотири етапи, а потім аналізують їх за допомогою багаторазового дослідження, яке вивчає бар'єри, що існують на кожному кроці на шляху до постійного розгортання. Цільовий етап лише описаний; автори не деталізують жодних засобів подолання бар'єрів. Основним висновком з тематичного дослідження є те, що перехід до гнучкого розвитку вимагає переходу до невеликих команд

розробників та зосередження на особливостях, а не на компонентах. Крім того, доречно зауважити, що перехід до безперервної інтеграції вимагає автоматизованої системи побудови та тестування (безперервна інтеграція), основної гілки контролю версій, до якої постійно надходить код, та модульної розробки.

Для переходу від постійної інтеграції до постійного розгортання такі організаційні підрозділи, як управління продуктами, повинні бути повністю залучені, і для подальшого вивчення концепції продукту необхідна тісна робота з дуже активним провідним клієнтом. Автори пропонують дві ключові дії для переходу від постійного впровадження до експериментальної системи досліджень та розробок. По-перше, продукт повинен бути інструментований таким чином, щоб дані предметної області могли збиратись у реальному використанні. По-друге, організація має мати можливості для того, щоб ефективно використовувати зібрані дані для тестування нових ідей з клієнтами.

Колектив авторів у [2] вивчали стадію безперервної інтеграції, вказуючи на те, що в галузі не існує однорідної практики безперервної інтеграції. Вони пропонують описову модель, яка дозволяє вивчати та оцінювати різні шляхи, на які можна розглядати безперервну інтеграцію. В роботі [7] представляють архітектуру, яка підтримує постійні експерименти у вбудованих системах. Вони досліджують цілі експериментальної системи, розробляють сценарії експериментів та будують архітектуру, яка підтримує цілі та сценарії. Архітектура поєднує в собі сховище експериментів, сховище даних та програмне забезпечення для розгортання на вбудованих пристроях за допомогою бездротових каналів передачі даних. Архітектура також враховує спеціальні вимоги щодо безпеки, наприклад, в автомобільній галузі. Однак основний тип експерименту зводиться до тестування A / B, і архітектура розглядається переважно з точки зору групи розробників програмного забезпечення, а не більшої організації з розробки продуктів.

Автори [8] описують модель експериментальної гіпотези, керованої даними (HYPEX). Метою цієї моделі є скорочення циклу зворотного зв'язку з клієнтами. Він складається з циклу, де потенційні об’єкти генеруються у відставання об’єктів, вибираються об’єкти та визначається відповідна очікувана поведінка. Очікуване

поведінка використовується для реалізації і розгортання мінімальної ціннісної функції (MVF).

Спостережувана та очікувана поведінка порівнюється за допомогою аналізу розривів, і якщо виявляється досить малий розрив, ознака доопрацьовується. З іншого боку, якщо виявляється значний розрив, для його пояснення розробляються гіпотези, розробляються та застосовуються альтернативні MVF, після чого аналіз розриву повторюється. Також від функції можна відмовитись, якщо очікувана вигода не буде досягнута.

2.1 Систематичне створення цінності шляхом експериментів

Моделі, описані вище, прагнуть зробити експериментування систематичним в розробці програмного забезпечення організації. Однією з важливих концептуальних проблем є визначення експерименту. Експериментування в програмній інженерії можна розглядати в широкому розумінні, включаючи як контрольовані експерименти, але також і більше дослідницьку діяльність, яка спрямована на розуміння та відкриття, а не на перевірку гіпотез.

Однак ми підкреслюємо, що всі методи вимагають суворого планування досліджень і мають захищений та прозорий спосіб міркувань та висновків на основі емпіричних даних. Це не той самий метод, який застосовується більш-менш обережно. Логіка контрольованих експериментів покладається на ретельне маніпулювання змінними, спостереження за ефектами та аналіз для перевірки причинно-наслідкових зв’язків.

Експериментування може також розглядатися з точки зору цілей і можуть існувати цілі на різних рівнях розвитку продукту. На рівні продукту експерименти можуть бути використані для вибору ознак із набору запропонованих. На технічному рівні експерименти можуть бути використані для оптимізації існуючих функцій. Однак модель, представлена в цій роботі, пов'язує експерименти на товарному та технічному рівнях із баченням продукту та стратегією на рівні бізнесу. Експериментування стає системною активністю, який керує всією організації. Це дозволяє цілеспрямовано перевіряти бізнес-гіпотези та припущення, що може

перетворитися на швидше прийняття рішень та реагування на потреби клієнтів. Залежно від конкретного використовуваного методу, результати експерименту можуть запропонувати нову інформацію, яка повинна бути включена в процес прийняття рішення.

На основі аналізу літературних джерел, що визначають етапи втілення техніки CI/CD для проведення змін в програмний код на основі оновлень специфікацій вимог, ми розробляємо та пропонуємо модель для безперервного експериментування. У цій моделі експерименти походять від бізнес-стратегій і мають на меті оцінити припущення, виведені з цих стратегій, що потенційно можуть призвести рішення щодо впровадження цих стратегій.

Представлена модель також описує інфраструктуру платформи, необхідну для створення всієї експериментальної системи. Software factory [9] може служити інфраструктурою для запропонованої моделі, оскільки це лабораторія з розробки програмного забезпечення, що добре підходить для постійних експериментів. Деякі основоположні ідеї, що лежать в основі Software factory щодо постійних експериментів, були вивчені в, наприклад, [10], а також вплив неперервної інтеграції у програмної інженерії.

2.2 Дослідницький підхід

Наші загальні рамки досліджень можна охарактеризувати як наукові дослідження з проектування ПЗ, метою яких є виведення технологічного правила, яке може бути використано на практиці для досягнення бажаного результату в певній галузі застосування. Модель безперервного експерименту, представлена в цій роботі, була вперше побудована на основі відповідних робіт, представлених у попередньому розділі.

Хоча структуру можна отримати чисто аналітичними засобами, її перевірка вимагає обґрунтування в емпіричних спостереженнях. З цієї причини було виконано аналіз даних, зібраних з проведеного раніше дослідження [11] в лабораторії згаданої вище Software Factory, в якому ми зіставили початкову модель з емпіричними спостереженнями та внесли подальші корективи для отримання остаточної моделі.

Модель все ще можна вважати попередньою, до подальшого підтвердження в інших контекстах. Важливо зазначити, що це дослідження досліджує, як безперервне експериментування можна проводити систематично незалежно від цілей проектів справи та експериментів, що проводяться в них. Ці експерименти та їх результати розглядаються як якісні висновки в контексті цього дослідження. У цьому розділі ми описуємо контекст дослідження ситуації та процес дослідження.

2.3 Контекст проведення досліджень

Software factory є освітньою платформою для досліджень та галузевої співпраці [9]. У проектах Software factory команди використовували сучасні інструменти та процеси для доставки робочих прототипів програмного забезпечення у тісній співпраці з галузевими партнерами. Метою діяльності Software factory є надання членам команди засобів для застосування своїх передових навичок розробки програмного забезпечення в середовищі, що відповідає робочому прототипу, та досягнення значущих результатів для своїх клієнтів [12].

Під час проектів, що використовувались у цьому дослідженні, двоє спостерігачів були залучені в якості учасників. Перший з них координував загальні питання проектів: розпочинав проекти, вирішував договірні та інші адміністративні питання, стежив за прогресом через безпосередню взаємодію із замовниками та командами, закінчував проекти, проводив опитування проектів та координував інтерв’ю із замовниками. Інший також брав участь у якості спостерігача у кількох зустрічах, на яких співпрацювали команди клієнтів та розробників.

Дослідники брали участь у керівництві експериментальною роботою разом із замовником, а розробники безпосередньо не брали участі в цих заходах. Тим не менше, замовник і програмісти працювали автономно і були відповідальні за управління, технічні рішення, а також інші питання, пов'язані з щоденними операціями.

2.4 Опис першої реалізації продукту

Першим проектом був невеликий стартап, який розробляє рішення для відеодзвінків для домашнього телевізора. Протягом вересня 2012 р. – грудня 2013 р. Компанія була замовником трьох проектів Software factory з метою створення інфраструктури для підтримки вимірювання та управління архітектурою їх служби відеодзвінків.

Компанія націлена на надання реального досвіду відеодзвінків. Їхня ціннісна пропозиція – «новий домашній телефон як досвід підключення та використання» – орієнтована на споживачів, які відокремлені від своїх сімей, наприклад, через міграцію, глобальні соціальні зв’язки чи закордонну роботу. Компанія робить особливий акцент на виявленні та задоволенні потреб людей похилого віку, роблячи простоту використання найважливішою нефункціональною вимогою їх продукту.

Основними засобами диференціації послуг на ринку є доступність, надійність та простота використання. Для прем’єрного комерційного запуску та встановлення основного каналу доставки свого продукту компанія націлена на співпрацю з операторами зв'язку. Компанія зробила початкову внутрішню архітектуру та часткову реалізацію на етапі перед розробкою до проектів Software Factory.

Був проведений перший проект з розширення функціональності платформи цієї реалізації. Другий проект був проведений для підтвердження проблем, пов'язаних із задоволенням вимог оператора. Після цього проекту був виконаний технічний каркас, при цьому основні частини реалізації були змінені; перші два проекти сприяли цьому рішенню. Потім був проведений третій проект з розширення нової реалізації новими функціями, пов’язаними з можливістю управління програмним забезпеченням на вже поставлених продуктах, що забезпечує постійну доставку (CD).

Стратегію запуску можна описати як запуск MVP з адаптацією після розробки. Вони також надали ранні докази щодо можливості використання продукту для конкретних зацікавлених сторін, таких як партнери-оператори, розробники та управління випуском.

2.5 Опис програмного продукту

Послуга відеодзвінків має основні функції домашнього телефону: вона дозволяє здійснювати та приймати відеодзвінки та вести список контактів. Продукт заснований на приставці ОС Android (STB), яку можна підключити до сучасного домашнього телевізора. Компанія підтримує внутрішню систему для посередницьких дзвінків своїм респондентам. Хоча сервер відповідає за маршрутизацію дзвінків, фактичний відеодзвінок виконується як одноранговий зв’язок між STB, що хостяться у будинках клієнтів.

Компанія зіграла роль власника продукту у трьох проектах Software Factory протягом вересня 2012 р. – грудня 2013 р. Метою перших двох проектів було створення нової інфраструктури для вимірювання та аналізу використання їх продукту в реальному середовищі. Ця інформація була важливою для того, щоб встановити доцільність продукту для операторів та для архітектурних рішень щодо масштабованості, продуктивності та надійності.

Для цього дослідження перші два проекти були використані для перевірки кроків, необхідних для встановлення безперервного процесу експериментів. Третій проект на Software factory забезпечив автоматизовану систему дистанційного управління та оновлення програмного забезпечення STB. Цей проект був використаний для дослідження факторів, пов'язаних з архітектурними потребами в постійних експериментах. У таблиці 2.1 детально узагальнено цілі та мотивацію проектів. Кожен проект мав команду з 4 – 8 осіб, представника компанії, доступного у будь-який час, і витрачав зусилля в межах 600 та 700 людиногодин.

2.5.1 Проект 1

Метою першого проекту на Software factory було створення засобів для вимірювання продуктивності продукту для відеодзвінків в реальному середовищі. Метою було розробити систему бізнес-аналітики на основі браузера. Команді також було доручено створити внутрішню систему для зберігання та управління даними, пов’язаними з відеодзвінками, з метою задоволення вимог моніторингу оператора. Проект Software factory був здійснений протягом семи тижнів командою з чотирьох

розробників середнього рівня кваліфікації. Компетенціями, необхідними в проекті, були дизайн бази даних, програмування програмної логіки та дизайн інтерфейсу користувача.

Таблиця 2.1 – Сфера кожного з трьох проектів на Software factory

	Мета високого рівня	Мотивація
Проект 1	Як оператор, я хочу мати можливість бачити показники дзвінків, здійснених клієнтами продукту для відеодзвінків.	... щоб я міг отримувати та аналізувати важливу для бізнесу інформацію. ... щоб я міг визначити потреби в технічному забезпеченні виробу
Проект 2	Як розробник, я хочу бути впевненим, що системна архітектура нашого продукту є масштабованою та надійною. Як розробник, я хочу знати технічні слабкі місця системи. Як розробник, я хочу отримувати пропозиції щодо альтернативних варіантів технічної архітектури.	... так що я знаю обмеження системи. ... так що я можу передбачити потреби в масштабованості платформи. ... щоб я міг розглянути можливі варіанти розвитку в майбутньому.
Проект 3	Як технічний менеджер, я хочу мати можливість оновлювати програмне забезпечення одним натисканням кнопки.	... так що я можу розгорнути оновлення програмного забезпечення на одній або декількох приставках.

Бекенд-система для збору та обробки даних була побудована на платформі Java Enterprise Edition, використовуючи фреймворк Spring Open Source. Система звітування на основі браузера була побудована з використанням фреймворків для створення яскравих та інтерактивних звітів. Для забезпечення продуктивності системи була впроваджена система кешування даних викликів.

Після завершення проекту і розробники, і замовник вважали, що товар доставлений відповідно до вимог замовника. Незважаючи на те, що деякі основоположні вимоги змінилися протягом проекту через відкриття нових

технологічних рішень, замовник зазначив задоволення кінцевим продуктом. Під час проекту спілкування між замовником та командою було частим та гнучким.

Перший проект являв собою першу спробу проведення постійних експериментів. Метою експерименту було отримання інформації про ефективність архітектури системи та її початкову реалізацію. Експеримент відбувся через необхідність оператора контролювати обсяги дзвінків та завантаження системи – це вимога, яку розробники продуктів визнали необхідною, щоб мати можливість співпрацювати з операторами. Було ясно, що існує набір потреб, що випливають із вимог оператора, але не було зрозуміло, як повинна подаватися інформація та яка функціональність потрібна для її аналізу. Однак з точки зору дослідження, точні деталі експерименту були менш важливими, ніж загальний процес початку експерименту.

2.5.2 Проект 2

Другий проект, реалізований на Software Factory, спрямований на проведення загальносистемного стрес-тесту для інфраструктури служби відеодзвінків компанії. Команда Software factory з чотирьох розробників створила інструмент тестування для моделювання дуже великих обсягів дзвінків. Інструмент був використаний для запуску декількох тестів для існуючого сервера посередника викликів.

Тестовий пакет програм включав інструмент для моделювання трафіку відеодзвінків. Інструмент був реалізований з використанням мови програмування Python. Інтерфейс візуального звітування на основі браузера також був реалізований, щоб допомогти аналізу результатів тесту. Компонент звітування був створений із використанням існуючих фреймворків JavaScript. Дані тестів зберігалися в базі даних MongoDB для використання в аналізі. Тоді як перший проект був зосереджений на потребах операторів, другий – на їх наслідках для розробників. Початкова архітектура системи та багато технічних рішень були поставлені під сумнів. Проект мав на меті надати докази для прийняття рішень при перегляді цих початкових рішень.

Команда виявила значні вузькі місця у існуючій системі перевірки концепції та проаналізувала їх походження. Були запропоновані рішення для збільшення

експлуатаційної спроможності поточної системи під навантаженням, а деякі з них також впроваджені.

Ближче до кінця проекту замовник запропонував команді Software factory запропонувати новий сервер викликів. Команда запропонувала кілька пропозицій щодо нової архітектури сервісу та створила новий сервер викликів. Для цілей цього дослідження ми вважаємо другий експеримент ще одним раундом у безперервному циклі експериментів, де результати першого циклу дали новий набір питань.

2.5.3 Проект 3

Для свого третього проекту на Software factory компанія мала на меті створити централізовану інфраструктуру для оновлення програмних компонентів продукту для відеодзвінків. Нова система дистанційного управління програмного забезпечення дозволить компанії швидко розгортати оновлення програмного забезпечення вже доставленої системи. Функціональність була важливою для компанії та її партнерів по каналу: вона дозволяла оновлювати програмне забезпечення без необхідності їздити на місця до кожного клієнта для оновлення своїх STB. Новий інструмент дозволив компанії встановити повний контроль над власними програмними та апаратними засобами.

Проект складався з команди з п’яти розробників. Команда представила робочий прототип для швидкого розгортання оновлень програмного забезпечення. У цьому проекті було розглянуто потребу в системі підтримки для надання нових функцій або програмних змін. Ми розглянули архітектурні вимоги до системи безперервної доставки, яка підтримувала б постійні експерименти.

2.6 Опис другої реалізації продукту

Цей продукт також розроблявся невеликою компанією-стартапом, яка розробляє продукт, що допомагає користувачам визначати, відстежувати та отримувати допомогу щодо життєвих цілей. Протягом травня – липня 2014 року компанія була замовником проекту «Software factory», який мав на меті розробити

механізм рекомендацій для сервісу, покращити взаємодію з інтерфейсом користувачами та перевірити основні припущення в стратегії обслуговування.

Компанія спрямована на доставку послуг у вигляді HTML5-додатку, який є оптимізованим також для планшетів, але і працює на інших пристроях з підтримкою HTML5-сумісного браузера. Послуга орієнтована на дорослих, які бажають поліпшити якість свого життя та змінити моделі поведінки для досягнення різних видів життєвих цілей.

2.7 Проект 4

Друга компанія забезпечувала інтерфейс користувача та прототип внутрішньої системи, який демонстрував загальні характеристики програми з точки зору користувача. Користувачі взаємодіють із фотографіями, які можуть бути розміщені в різних просторових шаблонах, щоб зобразити емоційні аспекти своїх цілей. Користувачі керуються програмою, щоб розташувати фотографії як карту, показуючи мету, потенційні кроки до неї та аспекти, що відповідають цілям.

Наприклад, життєвою метою може бути подорож навколо світу. Пов’язані фотографії можуть зображати місця для відвідування, настрої, які потрібно випробувати, предмети, необхідні для подорожей, такі як квитки тощо. Фотографії можуть бути розміщені, наприклад, у вигляді радіального візерунка з центральною метою посередині та пов’язаних з цим аспектів, або як часову лінію з кінцевою метою праворуч і проміжними кроками, що передують їй.

У проекті було визначено два припущення високого рівня. Клієнт припустив, що автоматична обробка на основі штучного інтелекту у серверній системі може бути використана для автоматичного спрямування користувачів до їх цілей, надання тригерів, мотивації та винагород на шляху. Крім того, замовник припускав, що мотивація для подальшого використання програми буде спричинена взаємодією з фотокартою.

Оскільки автоматична обробка залежала від мотиваційного припущення, остання стала фокусом експериментів у проекті. Клієнт використовував версії програми в тестах користувачів, під час яких спостереження та інтерв’ю

використовувались, щоб дослідити, чи справдились припущення. Цей проект використовувався в дослідженні, щоб перевірити зв'язок у нашій моделі між баченням продукту, бізнес-моделлю та стратегією та етапами експерименту.

2.8 Процес дослідження

Аналіз тематичного дослідження проводився для того, щоб обґрунтувати модель безперервного експериментування в емпіричних спостереженнях. Тому ми зібрали інформацію, яка допоможе нам зрозуміти передумови для проведення постійних експериментів, пов'язані з цим обмеження та виклики, а також логіку інтеграції результатів експерименту в бізнес-стратегію та процес розвитку. Для нашого аналізу було використано дані з відкритих джерел, які представляють собою: (i) спостереження за учасниками, (ii) аналіз артефактів проекту, (iii) сеанси групового аналізу та (iv) індивідуальні інтерв'ю.

Під час проектів ми спостерігали проблеми, з якими стикалися компанії, пов'язані з досягненням системи постійних експериментів. Наприкінці кожного проекту було проведено поглиблений брифінг, щоб отримати ретроспективне розуміння вибору, зробленого під час проекту, та аргументації за ними.

Засідання проводились у формі семінару, за участю одного наукового співробітника, команди проекту, представників замовника та будь-якого іншого спостерігача за проектом. Сесії розпочались з короткого вступу керівника, після чого присутніх попросили перерахувати події, які вони вважали важливими для проекту. Учасники записували кожну подію на окремій записці та розміщували їх на часовій лінії, яка відображала тривалість проекту. По мірі створення записок про події відбулася уточнююча дискусія щодо їх значення та місцезнаходження на часовій лінії. Коли учасники не могли подумати про будь-які інші події, їх попросили систематично розповідати про хід проекту, використовуючи часову лінію з подіями як орієнтир.

Співбесіди з представниками замовника проводились особисто у приміщенні замовника, в режимі он-лайн за допомогою відеоконференцій. Інтерв'ю являли собою напів-структуровані тематичні інтерв'ю, що мали суміш відкритих та закритих

питань. Ця техніка інтерв'ю дозволяє учасникам вільно обговорювати питання, пов'язані з основною темою.

Як мінімум двоє дослідників були присутні на інтерв’ю, щоб переконатися, що відповідна інформація була правильно отримана. Усі дослідники, які брали участь, робили нотатки під час інтерв’ю, а нотатки порівнювали після інтерв’ю для забезпечення послідовності.

В інтерв’ю представникам компаній спочатку пропонувалося розповісти про своє уявлення про свою компанію, її цілі та режим роботи до трьох проектів. Потім їм було запропоновано розглянути досягнення кожного проекту з точки зору результатів програмного забезпечення, вивченої інформації та наслідків для цілей та режиму роботи компанії. Нарешті, їх попросили роздумати про те, як компанія працювала під час співбесіди та як вони розглядали процес розробки, особливо з точки зору включення ринкових відгуків у процес прийняття рішень. Під час аналізу дані проекту перевірялись на наявність інформації, що стосується питання дослідження.

РОЗДІЛ 3 РОЗРОБКА МОДЕЛІ ЕКСПЕРИМЕНТУ НА ОСНОВІ ПРОЦЕСІВ CI/CD

У цьому розділі ми спочатку описуємо запропоновану нами модель для безперервного експериментування, а потім про представлення, отримані в результаті багаторазового тематичного дослідження, та про те, як вони інформують різні частини моделі.

Кожен з чотирьох проектів справи стосується різних аспектів постійних експериментів. Результати справи підтверджують необхідність систематичної інтеграції всіх рівнів розробки програмних продуктів та послуг, особливо коли контекст є швидким розвитком нових продуктів і послуг. Ключовим питанням є розробка продукту, який купуватимуть клієнти з урахуванням жорстких фінансових обмежень. Стартап-компанії працюють на нестабільних ринках та в умовах великої невизначеності. Можливо, їм доведеться внести кілька швидких змін, коли вони отримують зворотний зв'язок з ринком. Завдання полягає в тому, щоб досягти відповідності товарному ринку, перш ніж закінчаться гроші.

3.1 Модель для безперервного експериментування з імплементацією змін вимог користувача

Під час постійних експериментів ми маємо на увазі підхід до розробки програмного забезпечення, який базується на польових експериментах із відповідними зацікавленими сторонами, як правило, замовниками або користувачами, але потенційно також з іншими зацікавленими сторонами, такими як інвестори, сторонні розробники або партнери з екосистем програмного забезпечення. Модель складається з багаторазових блоків Build-Measure-Learn, підтримуваних інфраструктурою, як показано на рисунку 3.1. Кожен блок Build-Measure-Learn вивчає результати навчання, які використовуються як вхідні дані для наступного блоку. Концептуально можна вважати, що модель може застосовуватись не лише до розробки програмного забезпечення, а й до проектування та розробки програмно-інтенсивних продуктів та послуг.

Блоки Build-Measure-Learn вивчають структуру діяльності з проведення експериментів та поєднують бачення продукту, бізнес-стратегію та розробку технологічного продукту за допомогою експериментів. Іншими словами, вимоги, проектування, впровадження, тестування, розгортання та обслуговування етапів розробки програмного забезпечення інтегруються та узгоджуються за допомогою емпіричної інформації, отриманої в результаті експериментів.

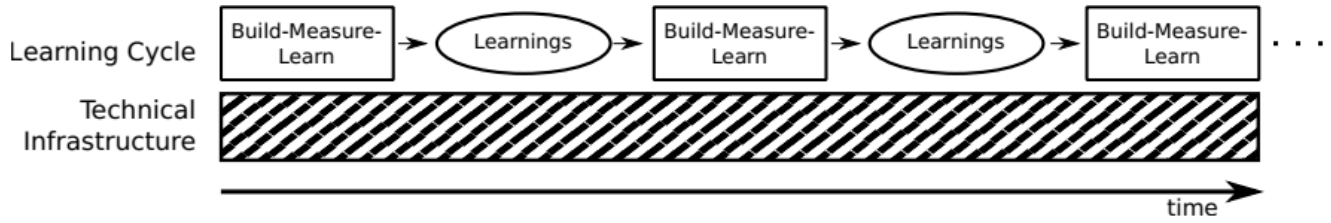

Рисунок 3.1 – Послідовність блоків діяльності при імплементації змін

Модель можна розглядати як транспортний засіб для додаткових інновацій, як це визначено в [13], але сама модель, а також перехід до безперервного експериментування в цілому, можна вважати радикальною.

3.2 Модель процесу для безперервного експериментування з використанням неперервної інтеграції

Рисунок 3.2 розширює блоки Build-Measure-Learn та описує модель процесу для безперервного експериментування. Передбачається, що існує загальне бачення товару чи послуги. Дотримуючись методології [1], це бачення є досить стабільним і ґрунтується на знаннях та переконаннях підприємців. Бачення пов'язане з бізнес-моделлю та стратегією, що є описом того, як реалізувати бачення. Бізнес-модель та стратегія є більш гнучкими, ніж бачення, і складаються з безлічі припущень щодо дій, необхідних для виведення товару чи послуги на ринок, що відповідає баченню та є стабільно вигідним.

Однак кожному припущенню притаманні невизначеності. З метою зменшення невизначеності ми пропонуємо провести експерименти. Експеримент операціоналізує припущення та формулює гіпотезу, яка може бути піддана

експериментальному тестуванню, щоб отримати знання щодо цього припущення. Спочатку обираються гіпотези з найвищим пріоритетом. Після формулювання гіпотези можуть відбуватися дві паралельні дії. Необов'язково гіпотеза може бути використана для впровадження та розгортання мінімально життєздатного продукту (MVP) або мінімально життєздатного елемента (MVF), який використовується в експерименті та має необхідні прилади.

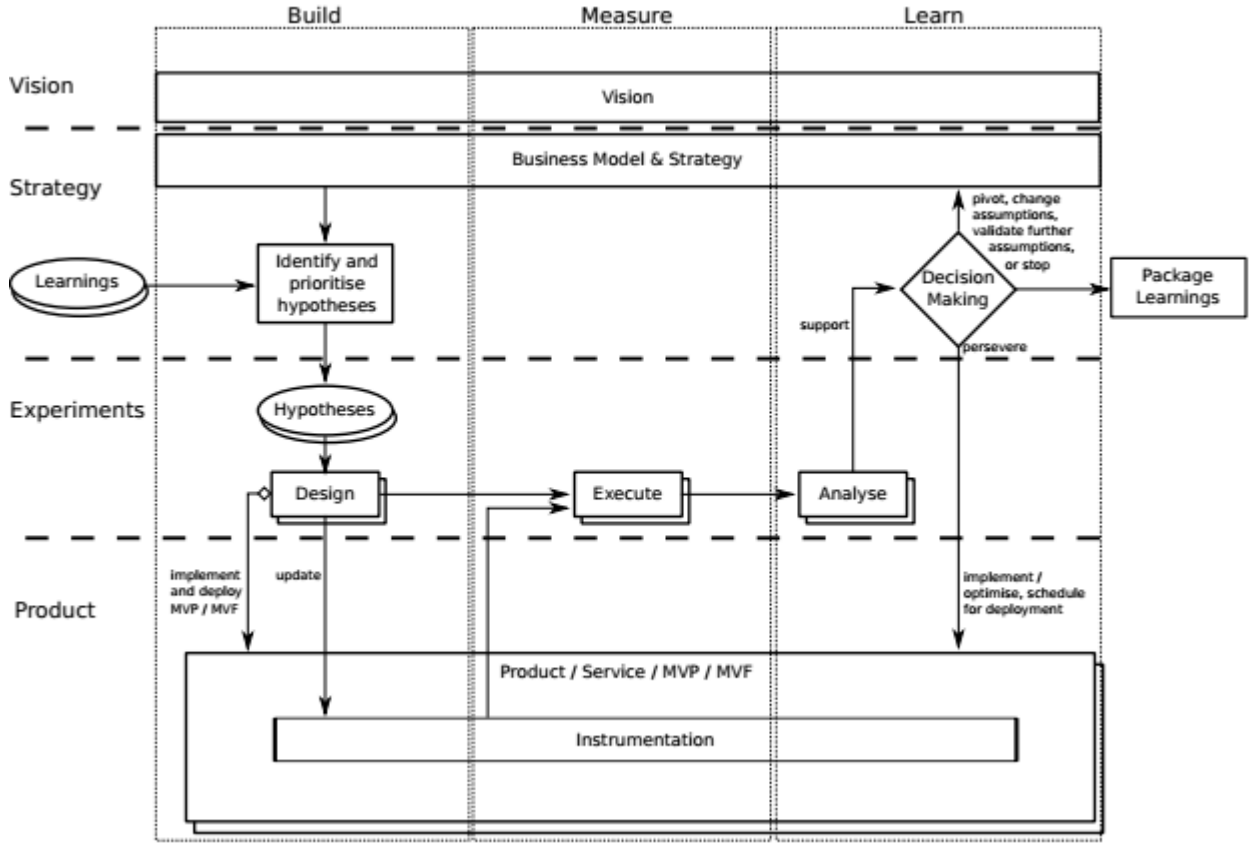

Рисунок 3.2 – Модель процесу для безперервного експериментування з використанням неперервної інтеграції

Одночасно для перевірки гіпотези виконується експеримент. Дані з MVP / MVF збираються відповідно до проекту експерименту. Отримані дані аналізують, завершуючи експериментальну діяльність. Після проведення експерименту та проведення аналізу результати аналізу використовуються на рівні стратегії для підтримки прийняття рішень. Рішення може бути або "опорним", або "наполегливим" [1], але можлива і третя альтернатива: змінити припущення у світлі нової інформації.

Якщо експеримент підтримав гіпотезу і, отже, припущення на рівні стратегії, розробляється або оптимізується повний продукт або функція та імплементуєтьяс. Стратегічне рішення в цьому випадку наполягати до обраної стратегії. Якщо, з іншого боку, гіпотеза була відкинута, що призвело до неприпустимості припущення на рівні стратегії, рішення полягає в зміні стратегії, розглядаючи наслідки припущення як хибні. В якості альтернативи перевірене припущення можна було б змінити, але не повністю відхилити, залежно від того, який експеримент був призначений для тестування та які були результати.

3.3 Архітектура пропонованої інфраструктури

Для підтримки проведення таких експериментів необхідна інфраструктура для постійних експериментів. На рисунку 3.3 наведено архітектуру інфраструктури для безперервного експериментування з ролями та відповідними завданнями, технічною інфраструктурою та артефактами інформації. Ролі, зазначені тут, будуть подані різними способами залежно від типу компанії, про яку йдеться. У невеликій компанії, такій як стартап, невелика кількість людей буде виконувати різні ролі, і одна людина може мати більше однієї ролі. У великій компанії ролі виконують кілька команд. Для роботи з чотирма класами завдань визначено сім ролей. Бізнес-аналітик та власник продукту або команда з управління продуктами спільно займаються створенням та ітеративним оновленням стратегічної дорожньої карти. Для цього вони консультуються з існуючими експериментальними планами, результатами та висновками, які розміщені у внутрішній системі. Оскільки плани та результати накопичуються та зберігаються, вони можуть бути використані повторно для подальшої розробки дорожньої карти.

Бізнес- аналітик та власник продукту співпрацюють із роллю вченого з даних, яка, як правило, є командою з різноманітними навичками, щоб повідомляти припущення дорожньої карти та складати карти невизначеності, які потрібно перевірити. Вчений-розробник проектує, виконує та аналізує експерименти. Для цього використовуються різноманітні інструменти, які отримують доступ до вихідних даних у внутрішній системі. Концептуально отримуються необроблені дані

та плани експериментів, виконується аналіз, а результати отримуються у формі навчань, які зберігаються назад у внутрішній системі.

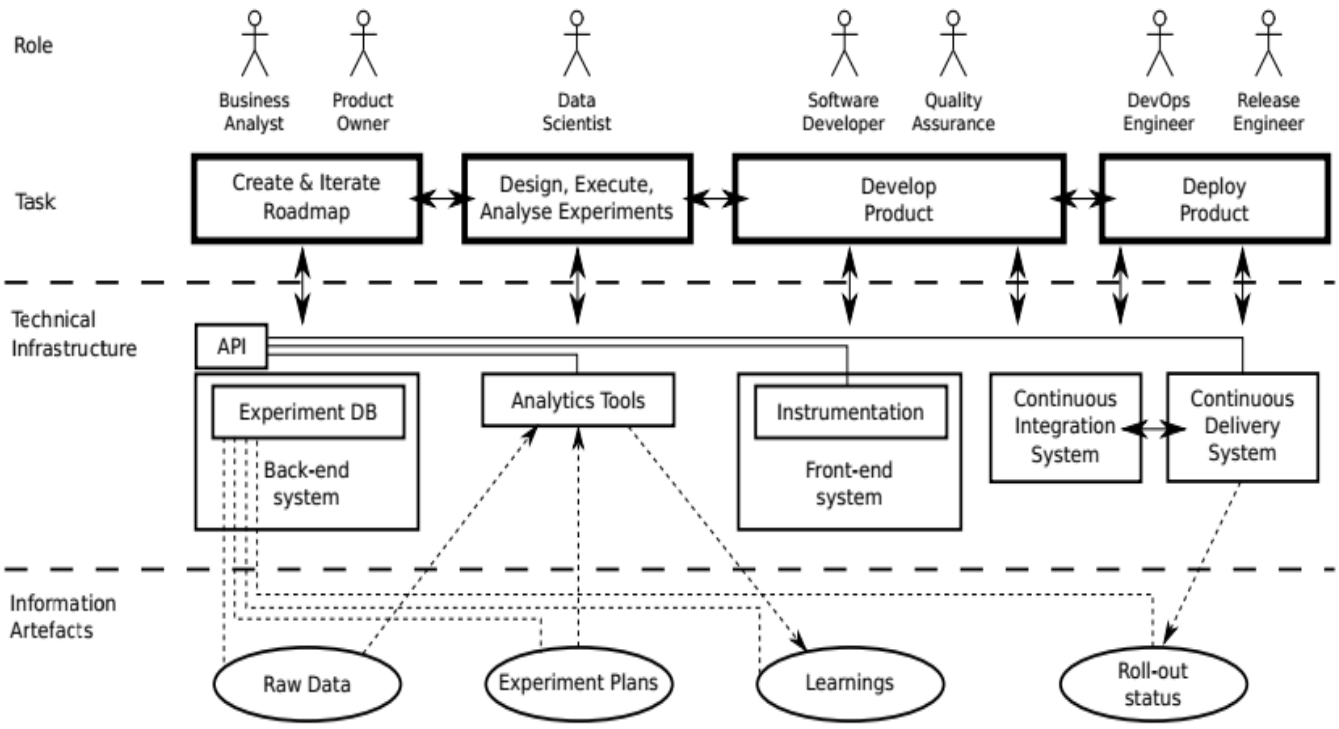

Рисунок 3.3 – Архітектура інфраструктури для пропонованої моделі

Аналітик даних також спілкується з розробником та роллю забезпечення якості. Ці ролі забезпечують розробку MVP, MVF та кінцевого продукту. Спочатку вони працюють з аналітиком даних, щоб створити належні вимірювання в інтерфейсній системі, яка є частиною програмного забезпечення, що постачається або бачиться користувачеві. У разі прийняття рішення про наполегливість вони працюють над повною розробкою або оптимізацією функції та надсиланням її для впровадження у виробництво. MVP, MVF та кінцеві продукти розгортаються для користувачів після першого проходження через системи безперервної інтеграції та безперервної доставки. Інженер DevOps виступає в якості посередника між командою розробників і операцій, а також інженер – реліз може контролювати і управляти депоями в даний час у продакшен.

Важливо, що система безперервної доставки надає інформацію про стан розгортання програмного забезпечення, дозволяючи іншим ролям контролювати

виконання експерименту і, наприклад, отримувати розуміння умов, за яких програмне забезпечення було розгорнуте для користувачів, та характеристик вибірки та швидкості відповіді експеримент. Наскрізні проблеми, такі як досвід користувачів, можуть вимагати додаткових ролей, що працюють з кількома згаданими тут ролями. Щоб спростити рисунок, ми опустили різні ролі, що стосуються операцій, такі як інженер із надійності сайтів тощо. Також ми пропустили повну розробку того, які інформаційні артефакти повинні бути видимими для яких ролей. Загалом, ми припускаємо візуалізувати стан безперервної системи експериментів для всіх ролей.

Бек-енд система складається з бази даних експериментів, яка концептуально зберігає необроблені дані, зібрані за допомогою програмного забезпечення, планів експериментів – які включають програмні особливості відбору вибірки та іншу логіку, необхідну для проведення експерименту, – та результати експерименту. Бек-енд система та база даних доступні через API.

Тут ці частини слід розуміти як концептуальні; реальна система, ймовірно, складається з декількох API – інтерфейсів, баз даних, серверів і т.д. Експериментальна база даних дозволяє архітектуру продукту, де розгорнуто програмне забезпечення налаштоване для експериментів під час виконання. Таким чином, не завжди потрібно, щоб нова версія програмного забезпечення або супровідних приладів передавалася користувачам до експерименту; експериментальна можливість може бути вбудована в ПЗ, що поставляється як конфігурується у вигляді варіаційної схеми.

Поставлене програмне забезпечення отримує параметри конфігурації для нових експериментів, переконфігурує себе та надсилає отримані дані вимірювань, усуваючи необхідність виконувати завдання Розробка продукту та Розгортання продукту. Для більших змін може знадобитися нова версія програмного забезпечення та виконати повний набір завдань.

3.4 Приклади моделювання та отримані висновки

У цьому підрозділі описано, як були створені моделі у чотирьох проектах, та описуємо отримані результати. Спочатку модель була досить простою, подібно до послідовності, описаної на рисунку 3.1, із циклом побудови-вимірювання-вивчення,

сховищем даних, інструментами аналізу та системою безперервної доставки. Не всі частини моделей були створені у всіх проектах. Припускаємо, що так буде і в інших проектах. У перших двох проектах ми зосереджувались на валідації проблем: формуванні розуміння потреб у реальних ситуаціях, на які повинна звертатись модель для постійних експериментів. У двох останніх проектах ми вже мали більшу частину моделі на місці і більше зосереджувались на валідації рішення, використовуючи докладні висновки з проектів для коригування моделі.

При внесенні змін у напрямку діяльності компанії необхідно базувати рішення на вагомих доказах, а не на здогадах. Однак виявлено, що, як правило, не бачення продукту чи послуги повинно змінюватися. Зміна повинна стосуватися стратегії, за допомогою якої реалізується бачення, включаючи особливості, що мають бути реалізовані, їх дизайн та технологічну платформу, на якій базується реалізація. Наприклад, хоча першій компанії доводилося адаптуватися кілька разів, головне бачення компанії не змінилося. Довелося робити все, що стосується технологій та встановлення пріоритетів функцій. Але основна пропозиція залишається незмінною: це новий домашній телефон і він підключається до вашого телевізора.

3.4.1 Аналіз проекту 1

У першому проекті новий інструмент бізнес-аналітики дозволив компанії дати уявлення про статистику своєї системи, надавши компанії засоби зворотного зв'язку. Вони можуть отримати уявлення про діяльність, пов’язану із дзвінками, майже в реальному часі, надаючи важливу для бізнесу інформацію для глибшого аналізу. Наявність даних дзвінків може бути використано як вхід для прийняття обґрунтованих рішень. Це також дозволило дізнатись про якість послуг та визначити моделі поведінки дзвінків клієнтів. На основі коментарів замовника така інформація буде мати вирішальне значення для прийняття рішень щодо масштабування платформи.

Таким чином можна було б уникнути надмірної завантаженості, і системі було б вигідніше працювати, зберігаючи при цьому хороший рівень обслуговування для кінцевих споживачів. Основною причиною бажання продемонструвати такі

можливості була потреба задовольнити потреби оператора. Щоб переконати операторів стати партнерами на каналі, здатність реагувати на коливання обсягів дзвінків була визнана критичною. Потенційні інвестори були б більш схильні інвестувати в компанію, яка могла б переконати операторів каналів у технічній життєздатності послуги.

Таблиця 3.1 – Екземпляри моделей у проекті 1

	Екземпляр моделі процесу
Бачення	Відеодзвінки вдома
Бізнес-модель та стратегія	Запропонувати відеодзвінки за допомогою партнерських відносин оператора (+ припущення щодо архітектури та стратегій впровадження продуктів)
Гіпотези	"Для того, щоб увійти в партнерство, операторам знадобляться інструменти аналізу управління ефективністю"
Дизайн, виконання, аналіз	Рудиментарний
MVF	Інструмент аналітики
Прийняття рішень	Почати архітектурний каркас (продовження у проекті 2) Початок основної стратегії впровадження продукту (продовження у проекті 2) Перевірити подальші припущення (щодо архітектури та реалізації продукту)
Ролі	Бізнес-аналітик, власник продукту (у ролі керівництва компанії), розробник програмного забезпечення (у ролі студентів Software Factory)
Технічна інфраструктура	Інструменти аналітики (MVF розроблений у проекті)
Інформаційні артефакти	Навчання (формально не задокументовано в проекті)

Метою першого проекту на високому рівні можна вважати визначення бізнес-гіпотези для перевірки бізнес-моделі з точки зору операторів. Проект забезпечив необхідні показники, а також інфраструктуру, яка підтримується інструментами для збору необхідних даних. Ці результати можуть бути використані для проведення експерименту з перевірки гіпотез бізнесу.

У таблиці 3.1 наведено частини нашої моделі, які були створені у Проекті 1. Проект створив кілька основних елементів моделі процесу. Обраною бізнес- моделлю та стратегією було пропонувати послугу відеодзвінка через партнерські оператори. Для того, щоб стратегія була успішною, компанії потрібно було продемонструвати доцільність послуги з точки зору потреб та вимог оператора. Ця демонстрація була присвячена самим операторам, а також іншим зацікавленим сторонам, таким як інвестори, які оцінили бізнес-модель та стратегію.

Гіпотеза для тестування не була дуже точно визначена в проекті, але її можна було б узагальнити, оскільки “операторам потрібні інструменти аналізу управління продуктивністю системи, щоб увійти до партнерства”. Експеримент, який був явно не керований один, а проводиться в рамках переговорів з інвесторами і оператора, використовується інструмент аналітики, розроблений в проекті, щоб оцінити, чи було припущення правильно, таким чином, створення екземпляра MVF і зробити елементарне виконання експерименту і аналіз. На основі цієї інформації було прийнято деякі рішення: розпочати дослідження альтернативних архітектур та стратегій впровадження продуктів.

3.4.2 Аналіз проекту 2

У другому проекті команда змогла вивчити обмеження поточної системи доказової концепції та її архітектури. Альтернативний сервер посередника викликів та альтернативна архітектура системи були дуже важливими для подальшого розвитку служби. Результати (таблиця 3.2), отримані у другому проекті, у поєднанні з результатами першого, спонукали їх до значного повороту щодо технологій, архітектурних рішень та методології розробки.

Розглядались також інші платформи. Наприклад, Samsung SDK для телевізорів Smart TV». “Вибір правильної технологічної платформи на базі Android дійсно багато чого пришвидшив. Спочатку було намагання зробити весь стек технологій від апаратного забезпечення до додатку. Фокус полягає у тому, щоб знайти свій сегмент у стеці технологій, працювати там, а решту отримати ззовні. Ми дослідили кілька варіантів на базі Android, деякі з яких були занадто дорогими.

Таблиця 3.2 – Екземпляри моделей проекту 2

	Екземпляр моделі процесу
Бачення	Відеодзвінки вдома
Бізнес-модель та стратегія	Запропонуйте відеодзвінки через партнерські оператори (+ припущення щодо архітектури та стратегій впровадження продукту)
Гіпотези	«Продукт повинен розроблятися як спеціальний програмний кодовий дизайн апаратного забезпечення» та «Архітектура повинна базуватися на технології Enterprise Java і не залежати від телевізора (який виконує лише функцію відображення) "
Дизайн, виконання, аналіз	Впровадження прототипу; оцінити поточну пропозицію рішення
MVF	Альтернативний сервер посередника викликів; альтернативна архітектура системи
Прийняття рішень	Архітектурний стрижень (апаратне забезпечення та ОС COTS на базі Android) Основна стратегія впровадження продукту (не розробляйте спеціальне обладнання)
Ролі	Бізнес-аналітик, власник продукту (у ролі керівництва компанії), розробник програмного забезпечення (у ролі студентів Software Factory)
Технічна інфраструктура	Інструменти аналітики (з попереднього проекту)
Інформаційні артефакти	Навчання (формально не задокументовано в проекті)

Цілі другого рівня на високому рівні можна розглядати як визначення та перевірку гіпотези рішення, що стосується доцільності запропонованого апаратно-програмного рішення. Проект надав оцінку технічного рішення, а також пропозиції щодо вдосконалення. Аналіз показав, що початкова архітектура та стратегія впровадження продукту були занадто трудомісткими, щоб здійснити їх повністю. Результати використовувались компанією для модифікації своєї стратегії. Замість того, щоб впроваджувати апаратне забезпечення самостійно, вони обрали стратегію, згідно з якою вони будуватимуть поверх загальних апаратних платформ і таким чином скорочуватимуть час виходу на ринок та витрат на розробку.

3.4.3 Аналіз проекту 3

У третьому проекті була розроблена можливість постійного розгортання. У STB може оновлюватися віддалено, дозволяючи нові можливості для пропонування клієнтам по дуже низькій ціні і з мінімальними зусиллями. Наслідки цієї можливості полягають у тому, що компанія здатна реагувати на зміни в просторі своїх технологічних рішень шляхом оновлення операційної системи та прикладного програмного забезпечення, а також на нові потреби клієнтів шляхом постійного впровадження нових функцій та тестування варіантів функцій.

Таблиця 3.3 –Екземпляри моделей у проекті 2.

	Екземпляр моделі процесу
Бачення	Відеодзвінки вдома
Бізнес-модель та стратегія	Запропонуйте відеодзвінки через партнерські оператори (+ припущення щодо архітектури та стратегій впровадження продукту)
Гіпотези	"Можливість автоматичного безперервного розгортання потрібна для поступової розробки та постачання продукції"
Проектувати, виконувати, аналізувати	Проект був зосереджений на створенні частин моделі архітектурної інфраструктури та не включав експеримент з продуктом
MVF	Прототип для швидкого розгортання оновлень програмного забезпечення
Прийняття рішень	Наполегливо
Ролі	Бізнес-аналітик, власник продукту (у ролі керівництва компанії), розробник програмного забезпечення (у ролі студентів Фабрики програм), інженер DevOps, інженер-реліз (у ролі технічного директора компанії та інших технічних представників; також представлений історіями користувачів із завданнями на ці ролі)
Технічна інфраструктура	Система безперервної інтеграції, система безперервної доставки (MVF розроблена в проекті)
Інформаційні артефакти	Статус розгортання

Цілі третього проекту на високому рівні можна розглядати як розвиток можливостей, що дозволяють автоматизувати безперервний процес розгортання. Передумовою цього є стабільний і контрольований темп розвитку, де основна увага

приділяється управлінню обсягом робочих предметів, які одночасно відкриваються, щоб обмежити складність.

Третій проект створив окремі частини нашої моделі архітектури інфраструктури, показані в таблиці 3.3. Зокрема, він зосередився на ролі системи безперервної доставки щодо завдань, які необхідно виконувати для постійних експериментів, тобто верхня та права частини з рис. 3, як детально викладено в таблиці.

3.4.4 Аналіз проекту 4

У четвертому проекті спочатку було важко визначити, що замовники вважали головними припущеннями. Однак, як тільки основні припущення стали ясними, можна було зосередитись на їх підтвердженні. Це підкреслює висновок, що хоча теоретично прямо вважати, що гіпотези слід виводити з бізнес-моделі та стратегії, на практиці це може бути не просто. При розробці нових продуктів і послуг бізнес-модель і стратегія не закінчені, і, особливо на ранніх циклах експериментів, може знадобитися спробувати кілька альтернатив і витратити зусилля на моделювання припущень, поки не буде отримано хороший набір гіпотез. Ми тому визнали корисним відокремити ідентифікації та пріоритизації гіпотез на рівні стратегії з докладного формулювання гіпотез і дизайну експерименту на рівні експерименту. У таблиці 3.5 наведено екземпляри деталей моделі в проекті 4.

Ми зазначаємо, що деякі з цих частин були введені в модель завдяки нашим висновкам з проекту 4.

У цьому проекті було два припущення: що взаємодія з фотокартою збереже користувачів і що здійсненний автоматизований процес спрямування користувачів до цілей. Припущення, що подальше використання програми буде спричинене взаємодією з фотокартою, виявилося неправильним. Спочатку користувачі створювали карту, але не витрачали б багато часу на взаємодію з нею – наприклад, додаванням або зміною фотографій, переставленням карти, додаванням фотографічних анотацій тощо.

Натомість користувачі повідомляли про бажання зв’язатися з іншими користувачами для спільного використання карт. і обговорити життєві цілі. Також вони висловили бажання зв’язатися з професійними або напівпрофесійними тренерами, щоб отримати допомогу у реалізації своїх життєвих цілей. Соціальний аспект послуги не враховували. Чи було це пов’язано із знайомством із існуючими додатками в соціальних мережах, так і не було досліджено.

Таблиця 3.4 – Екземпляри моделей у проекті 3

	Екземпляр моделі процесу
Бачення	Служба добробуту для визначення, відстеження та отримання допомоги у життєвих цілях
Бізнес-модель та стратегія	Рекомендації щодо продуктів та послуг, автоматизований механізм рекомендацій для мотивації прогресу до досягнення цілей
Гіпотези	"Мотивація для подальшого використання походить від взаємодії з фотокартою" та "Автоматичний механізм рекомендацій автоматично веде користувачів до досягнення цілей" (залежить від першої гіпотези)
Проект, виконання, аналіз	Тести користувачів із спостереженнями та інтерв’ю
MVF	Додаток на основі HTML5, оптимізований для планшетів
Прийняття рішень	Основна стратегія впровадження продукту (фокус на соціальній взаємодії, а не на автоматизованих рекомендаціях)
Ролі	Бізнес-аналітик, власник продукту (у ролі керівництва компанії), розробник програмного забезпечення (у ролі студентів Software Factory)
Технічна інфраструктура	Приладобудування, фронтальна система
Інформаційні артефакти	Навчання

У будь-якому випадку, припущення було недійсним, і, як наслідок, припущення щодо автоматизованих функцій для спрямування користувачів до цілей також були недійсними. Дослідження показало, що користувачів мотивувала можливість

взаємодії з іншими користувачами, і що ці взаємодії повинні включати процес мотивації їх до досягнення цілей. Важливо зазначити, що дві гіпотези можна було б визнати недійсними, оскільки вони були залежними.

Процес виявлення та встановлення пріоритетів гіпотез окремо від детального формулювання гіпотез та проекту експерименту дає можливість вибрати порядок експериментів таким чином, щоб отримати максимальну кількість інформації при мінімальній кількості експериментів. Тестування найбільш фундаментальних припущень – тих, на які покладається більшість інших припущень – по-перше, дає можливість усунути інші припущення без додаткових зусиль.

Четвертий проект також розкрив проблеми, пов'язані з інструментуванням програми для збору даних. Важко було відокремити процес постійних експериментів від технічних передумов для контрольно-вимірювальних приладів. У багатьох випадках перед проведенням експериментів можуть знадобитися значні інвестиції в технічну інфраструктуру. Ці висновки призвели до ролей, високого рівня опису технічної інфраструктури та інформаційних артефактів в архітектурі інфраструктури (див. рис .3.3).

Однак багато експериментів можливо і без вдосконалених приладів. Четвертий проект вказує на те, що експерименти, як правило, можуть бути великими або спрямовані на питання високого рівня на початку циклу розробки товару чи послуги. Вони можуть звертатися до питань та припущень, які є центральними для всієї концепції товару чи послуги. Наступні етапи експериментів можуть вирішувати більш детальні аспекти, і можуть бути розглянуті оптимізації існуючого продукту або послуг.

Модель безперервного експериментування, розроблена в попередньому розділі, можна розглядати як загальний опис. Варіантів багато. Наприклад, експерименти можуть бути розгорнуті для вибраних клієнтів у спеціальному тестовому середовищі, і кілька експериментів можуть виконуватися паралельно. Спеціальне тестове середовище може знадобитися, особливо на ринках від бізнесу до бізнесу, де наслідки змін функцій є широкими і може виникнути небажання мати нові функції взагалі. Таким чином, тривалість тестового циклу може бути довшою на ринках між бізнесом.

Пряме розгортання може бути більш підходящим для споживчих ринків, але ми зазначаємо, що ставлення до постійних експериментів, ймовірно, зміниться, оскільки як бізнес, так і споживчі споживачі звикнуть до цього.

Кожен проект міг створити правильні моделі різними способами. У першому проекті, експеримент можна було проводити з використанням макеті екрани для перевірки того, що метричні дані, visuali зация і інструменти аналізу було б достатньо, щоб переконати зацікавлені сторони. Однак це мало б шкоду, оскільки не виявило б недоліків у початковій архітектурі та стратегії впровадження.

Хоча дизайн експерименту залишав бажати кращого, бажаючи здійснити його з використанням реальної, запрограмованої системи прототипів, дозволило виявити необхідність переглянути деякі попередні варіанти стратегії. У другому проекті результати можна було б краще використовувати для визначення більш точного набору гіпотез після ретельного аналізу недоліків попередньої архітектури системи. Однак у цьому не було необхідності, оскільки метою було не точкове порівняння, а скоріше порівняння або-або між одним загальним підходом та іншим. Це висвітлює важливе поняття щодо постійних експериментів: воно прагне лише надати достатньо інформації для прийняття рішення правильно.

У третьому проекті була створена лише можливість безперервної доставки. Проект міг би також розглянуті компоненти, які необхідні для проведення реальних експериментів. Через обмеження часу проекту це не було досліджено у третьому проекті, але замість цього було розглянуто в четвертому проекті. У цьому проекті був проведений один цикл повної моделі ПРАВОГО процесу, і програмне забезпечення було пристосовано до експериментів, хоча і використовуючи готові сервіси, такі як Google Analytics. Хоча наша кінцева мета полягає в тому, щоб наші моделі охоплювали всю широту безперервних експериментів, ми припускаємо, що не у всіх реальних проектах потрібно буде створити екземпляр усіх частин.

Наприклад, експерименти можна проводити без MVP, особливо на ранній стадії розробки продукту. Також може не бути необхідним у всіх випадках мати важку інфраструктуру для експериментів – це стає актуальним, якщо експерименти проводяться у дуже великих обсягах або коли метою є підтримка безлічі

експериментів, які проводяться безперервно для збору інформації про тенденції, поки товар поступово змінюється.

На додаток до конкретних спостережень за проектом, ми розглядаємо ще кілька загальних проблем. Паралельне проведення декількох експериментів представляє особливу проблему. Складність інтерпретації онлайн-експериментів переконливо продемонстрували Кохаві та ін. [16]. Для оцінки надійності експериментів слід розглянути статистичну взаємодію між експериментами. З цієї причини важливо координувати розробку та проведення експериментів, щоб зробити правильні умовиводи. У більш загальному плані, питання про дійсність стає важливим, коли весь R & D Organi г ція є експериментом з приводом. Неправильно розроблені або реалізовані експерименти можуть призвести до критичних помилок у прийнятті рішень. Загрози обгрунтованості можуть також виникати внаслідок не врахування етичних аспектів експериментів. Неетичні експерименти можуть не тільки нашкодити репутації компанії, але вони можуть змусити респондентів свідомо чи несвідомо упереджувати результати експериментів, що призводить до помилок у прийнятті рішень.

Інші проблеми включають складність визначення пріоритетів, з чого почати: яке припущення слід перевірити першим. У проекті 4 ми виявили залежність між припущеннями щодо логіки рекомендацій бекенду та припущенням про те, що спонукає користувачів продовжувати користуватися програмою. Скасувавши останнє, ми автоматично визнали недійсним перше припущення. Це підкреслює важливість виявлення критичних припущень, оскільки їх спочатку тестування може врятувати кілька непотрібних експериментів. Ми бачимо необхідність подальших досліджень у цій галузі. Крім того, в апаратно-програмного забезпечення спільного проектування, ілюстровані перших трьох проектів, створення експериментального циклу швидко є серйозною проблемою як за рахунок більш довгою випуску циклу обладнання та потенційної Синхронізувати зації завдань між апаратним та програмним забезпеченням розробки графіків. На основі висновків, представлених у цій роботі, може бути корисно спочатку перевірити кілька стратегічних технічних припущень, таких як життєздатність певної апаратно-програмної платформи. Як

показує наш випадок, вибір правильної платформи на ранніх термінах може мати значний вплив на здатність переходити до фактичного розвитку сервісу.

Подальший набір проблем пов’язаний із моделлю збуту та мережами постачальників. По суті, всі компанії залежать від мережі постачальників та каналів збуту. Можливо, доведеться розширити представлену тут модель, щоб врахувати можливості, зокрема, постачальників обладнання, забезпечувати своєчасне постачання необхідних компонентів та необхідну гнучкість для програмного варіації параметрів поведінки цих компонентів. Крім того, коли компанія не продає свою продукцію безпосередньо кінцевим споживачам, кілька рівнів посередників можуть перешкоджати можливості збору даних безпосередньо з використання на місцях. Якщо партнер з продажу не може надати доступ кінцевим споживачам, потрібні інші засоби охоплення аудиторії. Ми передбачаємо використання програм раннього доступу та бета-тестування для цієї мети – практики, яка зазвичай використовується в галузі комп'ютерних ігор. Можливі й інші моделі, і є можливість для подальших досліджень у цій галузі.

У деяких випадках експериментальний підхід може бути зовсім непридатним. Наприклад, певні типи життєво важливого програмного забезпечення або програмного забезпечення, що використовується в середовищах, де експериментування надзвичайно дорого, можуть виключати використання експериментів як методу перевірки. Однак незрозуміло, як визначити придатність експериментального підходу в конкретних ситуаціях, і дослідження з цієї теми можуть дати цінні рекомендації щодо того, коли застосовувати модель, представлену тут. Інше питання полягає в тому, чи є безперервна доставка суворо необхідною передумовою для постійних експериментів.

На початку циклу розробки продукту експерименти повинні відбуватися до того, як буде написано багато програмного забезпечення. На цьому етапі безперервна доставка може не знадобитися. Крім того, не всі експерименти вимагають доставки нового програмного забезпечення для користувачів. Незважаючи на те, що може існувати система безперервної доставки, саме програмне забезпечення може бути спроектоване на мінливість, щоб воно могло переконфігурувати себе під час

виконання. У таких випадках для запуску нових експериментів не потрібно доставляти нову версію програмного забезпечення.

Однак не всі експерименти можливі навіть з дуже гнучкою архітектурою, яка дозволяє переконфігурувати час виконання. Безперервна доставка – це хороший інструмент для проведення експериментів серед користувачів та забезпечення якості процесу розробки. Модель, представлена тут, заснована на ітераційне, еволютивна оптимізації з особливостей продукту та інкрементної моделі інновацій. Щоб здійснити революційні інновації, процес потрібно розширити за допомогою інших засобів виявлення цінності споживача. Вони можуть глибоко призвести до непридатності бізнес-моделі чи стратегії, а можуть навіть вплинути на загальне бачення.

Нарешті, експерименти можуть проводитися з декількома типами зацікавлених сторін. Окрім споживачів та кінцевих споживачів, експерименти можуть бути спрямовані на інвесторів, постачальників, канали збуту або дистрибуторів. Компанії, продукт яких сам по собі є платформою розробки, може захотіти провести експерименти з розробниками в їх екосистемі платформи, щоб оптимізувати досвід розробників [14] своїх інструментів, методів та процесів.

Ці експерименти можуть вимагати інших видів експериментальних артефактів, крім MVP / MVF, включаючи, наприклад, процеси, API та документацію. Дослідження типів експериментальних артефактів та супутніх експериментальних конструкцій можуть призвести до плідних результатів для таких областей застосування. Крім того, відкритим питання про те, хто повинен в першу чергу вести або ведуть до експериментів, особливо коли розвиток органи зація окремо від замовника організації.

Для клієнтів може виникнути потреба у навчаннях, щоб переконатися, що вони можуть взаємодіяти з безперервним процесом експериментування працює в розвитку організації. Аналогічним чином, команда розробників може потребувати додаткової підготовки, щоб мати можливість взаємодіяти із замовником для вироблення припущень, планування експериментів та звітування про результати для подальшого

прийняття рішень. Інша можливість полягає в тому, щоб ввести роль посередника, який з'єднує клієнта та розвитку організації.

Більш загально, збільшення здатності до експериментів та безперервної інженерії програм вимагає врахування людських факторів у командах розробників програмного забезпечення [15]. Необхідні подальші дослідження, щоб визначити, як експериментальні технологічні роботи через організацію кордонів, будь то всередині або за межами однієї компанії.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Синдром професійного вигорання в ІТ

У сучасних дослідженнях наголошується, що вигорання виникає у фахівця в контексті роботи і має негативні наслідки як для нього самого, так і для організації в цілому, психічного благополуччя всіх тих, з ким він взаємодіє в процесі професійної діяльності (клієнтів, учнів, пацієнтів, колег) [17]. Подальше вивчення даного феномену важливе для дослідження закономірностей і механізмів професійного становлення механізмів професійної адаптації; виявлення зв'язку між професійним стресом і соматичними розладами, та встановлення характеру цього зв'язку тощо.

Дослідниками з часу появи перших даних про вигорання як професійного феномену неухильно наголошується його поширеність. Якщо раніше він досліджувався у представників професій «людина – людина» – вчителів, викладачів коледжів, лікарів і медичних сестер, менеджерів, торгівельних агентів, соціальних працівників, представників інших професій, пов'язаних з безперервними комунікаціями з великим потоком людей, то тепер коло професіоналів, схильних до професійного вигорання, розширилося. Дослідниками отримані дані про наявність вигорання у представників управлінських структур і у представників професії «людина – знак», «людина – машина» – бухгалтерів, програмістів, пілотів, водіїв, операторів та ін. У зв'язку з цим, дослідження феномену професійного вигорання виступає одному з пріоритетних завдань сучасної науки [18].

Якісний і кількісний склад вигорання залежить від змісту професійної діяльності: у професіях «суб'єкт-суб'єктної» сфери вигорання має трикомпонентну структуру, а в «суб'єкт-об'єктній» сфері – близьку до двофакторної структури. Психічне вигорання набуває статусу загально професійного феномену і розглядається як компонент категоріального апарату психологічної науки. Даний феномен виявляється в різних сферах особистості – емоційній, когнітивній, мотиваційній, сфері відношення людини до роботи – і найменше розроблена проблема впливу вигорання на поза професійні сфери життя людини [19].

Аналіз сучасних досліджень показав, що «психічне вигорання» ширше поняття, ніж «професійне вигорання», оскільки воно може бути викликане різними причинами – особовими, родинними, професійними. Психічне вигорання може виявлятися не лише в професійній сфері, а, наприклад,у сім'ї, в учбовій діяльності. Професійне вигорання становить один з варіантів психічного вигорання: перше частіше використовується в контексті трудової діяльності. Поняття «емоційне вигорання» вживається в тому разі, коли акцент робиться на емоційній складовій вигорання і йдеться про те, що вигоряє лише емоційна сфера психіки. Термін психічне вигорання використовується в разі, коли акцентується увага на тому, що вигорання зачіпає емоційну, інтелектуальну та мотиваційно-споживацьку сфери, а також вольовий механізм. На сучасному етапі вигорання вивчається одночасно як соціальний, професійний та особистісний феномени.

Отже, в контексті теоретичного вивчення проблем професійного вигорання виділяють основні етапи. Перший етап присвячений виявленню й опису окремих випадків вигорання. На другому етапі відбувається розуміння та узагальнення феномену вигорання. Третій період відзначаєтьсяметодологічною фазою дослідження вигорання. Зміст четвертого етапу полягає в аналітичному вивченні узагальненого концепту вигорання. Сучасний етап вивчення питання професійного вигорання характеризується появою нових напрямів досліджень, зокрема він характеризується різними сенсами, які зв'язуються з феноменом вигорання.

4.2 Створення і функціонування системи моніторингу довкілля з метою інтеграції екологічних інформаційних систем, що охоплюють певні території

Законом України "Про охорону навколишнього природного середовища" (ст.20, 22) передбачено створення державної системи моніторингу довкілля (ДСМД) та проведення спостережень за станом навколишнього природного середовища, рівнем його забруднення. Виконання цих функцій покладено на Мінприроди та інші центральні органи виконавчої влади, які є суб'єктами державної системи моніторингу довкілля, а також підприємства, установи та організації, діяльність яких призводить або може призвести до погіршення стану довкілля.

Основні принципи функціонування ДСМД визначені у постанови Кабінету Міністрів України від 30.03.1998 № 391 „Про затвердження Положення про державну систему моніторингу довкілля".

На даний час, у державній системі моніторингу довкілля (далі – ДСМД) функції і задачі спостережень та інформаційного забезпечення виконують 8 суб`єктів системи моніторингу: Мінприроди, МНС, МОЗ, Мінагрополітики, Мінжитлокомунгосп, Держводгосп, Держкомлісгосп, Держкомзем.

Кожний із суб`єктів ДСМД здійснює моніторинг тих об`єктів довкілля, що визначаються Положенням про державну систему моніторингу довкілля та порядками і положеннями про державний моніторинг окремих складових довкілля. Основні нормативні акти, що регламентують моніторинг об'єктів довкілля:

– постанова Кабінету Міністрів України від 09.03.1999 № 343 «Про затвердження Порядку організації та проведення моніторингу в галузі охорони атмосферного повітря»;

– постанова Кабінету Міністрів України від 20.07.1996 № 815 «Про затвердження Порядку здійснення державного моніторингу вод»;

– постанова Кабінету Міністрів України від 20.08.1993 № 661 «Про затвердження Положення про моніторинг земель»;

– постанова Кабінету Міністрів України від 26.02.2004 № 51 «Про затвердження Положення про моніторинг ґрунтів на землях сільськогосподарського призначення».

З метою координації діяльності міністерств та відомств, визначення основних принципів державної політики з питань розвитку системи моніторингу навколишнього середовища, забезпечення її функціонування на основі єдиного нормативно-методологічного забезпечення постановою Кабінету Міністрів України від 17.11.2001 № 1551 утворено Міжвідомчу комісію з питань моніторингу довкілля. Мінприроди здійснюється організаційно-технічне забезпечення роботи комісії та її профільних секцій.

Існуюча система моніторингу довкілля базується на виконанні розподілених функцій її суб’єктами і складається з підпорядкованих їм підсистем. Кожна

підсистема на рівні окремих суб'єктів системи моніторингу має свою структурно-організаційну, науково-методичну та технічну бази.

Функціонування ДСМД здійснюється на трьох рівнях, що розподіляються за територіальним принципом:

– загальнодержавний рівень, що охоплює пріоритетні напрямки та завдання моніторингу в масштабах всієї країни;

– регіональний рівень, що охоплює пріоритетні напрямки та завдання в масштабах територіального регіону;

– локальний рівень, що охоплює пріоритетні напрямки та завдання моніторингу в масштабах окремих територій з підвищеним антропогенним навантаженням.

Моніторинг якості повітря.

Державною гідрометеорологічною службою (МНС) здійснюються спостереження за забрудненням атмосферного повітря у 53 містах України на 162 стаціонарних, двох маршрутних постах спостережень та двох станціях транскордонного переносу. Ведуться спостереження за хімічним складом атмосферних опадів та за кислотністю опадів.

Програма обов`язкового моніторингу якості атмосферного повітря включає сім забруднюючих речовин: пил, двоокис азоту (NO2), двоокис сірки (SO2), оксид вуглецю, формальдегід (H2CO), свинець та бенз(а)пірен. Деякі станції здійснюють спостереження за додатковими забруднюючими речовинами. Проводиться аналіз наявності забруднюючих речовин в опадах та сніговому покриві.

Державна екологічна інспекція (Мінприроди) здійснює вибірковий відбір проб на джерелах викидів. Вимірюється понад 65 параметрів.

Санітарно-епідеміологічна служба (МОЗ) здійснює спостереження за якістю атмосферного повітря у житловій та рекреаційній зонах, зокрема поблизу основних доріг, санітарно-захисних зон та житлових будинків, на території шкіл, дошкільних установ та медичних закладів в містах та в робочий зоні. Крім того, здійснюється аналіз якості повітря у житловій зоні за скаргами мешканців.

Моніторинг стану вод суші.

Державна гідрометеорологічна служба (МНС) проводить моніторинг гідрохімічного стану вод на 151 водному об`єкті, а також здійснює гідробіологічні спостереження на 45 водних об`єктах. Отримуються дані по 46 параметрах, що дають можливість оцінити хімічний склад вод, біогенні параметри, наявність зважених часток та органічних речовин, основних забруднюючих речовин, важких металів та пестицидів. На 8 водних об`єктах проводяться спостереження за хронічною токсичністю води. Визначаються показники радіоактивного забруднення поверхневих вод.

Державна екологічна інспекція (Мінприроди) відбирає проби води та отримує дані по 60 вимірюваних параметрах.

Державний комітет по водному господарству проводить моніторинг річок, водосховищ, каналів, зрошувальних систем і водойм у межах водогосподарських систем комплексного призначення, систем водопостачання, транскордонних водотоків та водойм у зонах впливу атомних електростанцій. Контроль якості води за фізичними та хімічними показниками здійснюється на 72 водосховищах, 164 річках, 14 зрошувальних системах, 1 лимані та 5 каналах комплексного призначення. Крім того, у рамках радіаційного моніторингу вод водогосподарськими організаціями здійснюється контроль вмісту радіонуклідів у поверхневих водах.

Санітарно-епідеміологічна служба (МОЗ) проводить спостереження за джерелами централізованого та децентралізованого постачання питної води, а також місцями відпочинку вздовж річок та водосховищ.

Підприємствами Державної геологічної служби (Мінприроди) здійснюється моніторинг стану підземних вод. У місцях моніторингу проводиться оцінка рівня залягання підземних вод (наявність), їх природного геохімічного складу. Проводяться визначення 22 параметрів, в тому числі концентрації важких металів та пестицидів.

Санітарно-епідеміологічна служба (МОЗ) здійснює хімічний аналіз підземних вод, які призначаються для питного споживання.

Моніторинг прибережних вод.

Державна гідрометеорологічна служба (МНС) управляє мережею моніторингу стану прибережних вод, яка складається з станцій моніторингу у місцях скиду стічних

вод та науково-дослідних станцій, що розташовані на прибережних територіях Чорного та Азовського морів. На існуючих станціях проводяться вимірювання від 16 до 26 гідрохімічних параметрів вод та донних відкладів.

Державні інспекції охорони Чорного та Азовського морів (Мінприроди) мають власні системи спостережень. До їх повноважень відносяться щомісячні відбори проб та аналіз впливу джерел забруднення, які розташовані на узбережжі; моніторинг скидів з кораблів; забруднення від діяльності з пошуку та видобування нафти, газу і будівельних матеріалів на морському шельфі; нагляд за використанням живих ресурсів моря.

Державна санітарно-епідеміологічна служба (МОЗ) здійснює моніторинг якості морської води в зонах рекреаційного та оздоровчого водокористування.

Моніторинг стану ґрунтів.

Державна гідрометеорологічна служба (МНС) здійснює моніторинг забруднення ґрунтів сільськогосподарських земель пестицидами та важкими металами у населених пунктах. Проби відбираються раз у п`ять років, проби на важкі метали у містах Костянтинівка та Маріуполь відбираються щороку.

Державна екологічна інспекція (Мінприроди) здійснює відбір проб на промислових майданчиках в межах країни. Загальна кількість параметрів, що вимірюються – 27.

Установи МОЗ здійснюють моніторинг стану ґрунтів на територіях їх можливого негативного впливу на здоров`я населення. Найбільше охоплені території вирощення сільськогосподарської продукції, території в місцях застосування пестицидів, ґрунти в зоні житлових масивів, дитячих майданчиків та закладів. Досліджуються проби ґрунту в місцях зберігання токсичних відходів на території підприємств та поза територією підприємств у місцях їх складування або захоронення.

Мінагрополітики здійснює спостереження за ґрунтами сільськогосподарського використання. Здійснюються радіологічні, агрохімічні та токсикологічні визначення, залишкова кількість пестицидів, агрохімікатів і важких металів.

Моніторинг показників біологічного різноманіття.

Через обмежене бюджетне фінансування моніторинг здійснюється тільки за видами, які представляють промисловий інтерес (дерева, риба, дичина).

Підприємства Держкомлісгоспу проводять моніторинг лісової рослинності у 24 областях країни. Здійснюється оцінка біомаси, пошкодження її біотичними та абіотичними чинниками; мисливської фауни, біорізноманіття; радіологічні визначення.

Деякі дослідження здійснюються через надання міжнародної допомоги, або в рамках міжнародних програм.

Моніторинг радіаційного випромінювання.

Державна гідрометеорологічна служба (МНС) здійснює спостереження за радіоактивним забрудненням атмосфери шляхом щоденних замірів доз гамма-радіаційної експозиції (ГРЕ), осідання радіоактивних частинок з атмосфери та вмісту радіоактивного аерозолю в повітрі. Здійснюються заміри радіоактивного забруднення поверхневих вод на 8 водних об`єктах. Поблизу атомних електростанцій Державна гідрометеорологічна служба здійснює заміри радіоактивного забруднення поверхневих вод цезієм-137 у та забруднення ґрунтів.

Лабораторії моніторингу Мінагрополітики проводять контроль у місцях концентрації радіоактивних речовин у ґрунтах та харчових продуктах. МНС здійснює моніторинг доз ГРЕ на 10 автоматизованих пунктах поблизу атомних електростанцій. У межах 30-кілометрової зони навколо Чорнобильської АЕС (зони відчуження), МНС здійснює спостереження за концентрацією радіонуклідів; радіонуклідами в атмосферних опадах, а також концентрацією «гарячих» частинок у повітрі. Міжнародна радіоекологічна лабораторія Чорнобильського центру атомної безпеки, радіоактивних відходів та радіоекології у Славутичі, здійснює моніторинг впливу радіації на біоту у зоні відчуження.

Суб`єктами ДСМД створені, або розробляються відомчі бази даних моніторингової інформації. Існуюча система інформаційної взаємодії відомчих підсистем моніторингу довкілля передбачає обмін інформацією на загальнодержавному та регіональному рівнях. Організаційна інтеграція суб`єктів

моніторингу довкілля на всіх рівнях здійснюється Мінприроди та його територіальними органами.

Для упорядкування процесу обміну інформацією за показниками та термінами надання екологічної інформації між Мінприроди та суб'єктами ДСМД укладено двохсторонні угоди про співробітництво у сфері моніторингу навколишнього природного середовища, до яких розроблені відповідні регламенти обміну екологічною інформацією.

Оперативна моніторингова інформація передається територіальними органами суб'єктів ДСМД до регіональних центрів моніторингу довкілля, або державних управлінь охорони навколишнього природного середовища в регіонах.

Узагальнена аналітична інформація надається міністерствами та відомствами-суб'єктами ДСМД Мінприроди.

Отримані дані передаються до Інформаційно - аналітичного центру Мінприроди та накопичується у банках екологічних даних.

На основі отриманої щомісячної та щоквартальної інформації Мінприроди видається інформаційно – аналітичний огляд „Стан довкілля в України ", який розповсюджується серед заінтересованих користувачів.

Функціонування Інформаційно-аналітичного центру Мінприроди забезпечує інформаційний обмін з регіональними центрами моніторингу довкілля, суб'єктами державної системи моніторингу довкілля, створення уніфікованого банку екологічних даних, проведення комплексного аналізу стану довкілля, тощо.

Постановою Кабінету Міністрів України від 05.12.2007 № 1376 затверджено Державну цільову екологічну програму проведення моніторингу навколишнього природного середовища. З метою забезпечення інтеграції інформаційних ресурсів суб`єктів системи моніторингу довкілля передбачено створення та забезпечення функціонування єдиної автоматизованої підсистеми збору, оброблення, аналізу і збереження даних та інформації, отриманих в результаті здійснення моніторингу.

ВИСНОВКИ

Компанії все частіше перекладають свої традиційні функції досліджень та розробки продуктів на постійні експериментальні системи [6]. Інтеграція польових експериментів з розробкою продукції на діловому та технічному рівнях є новою проблемою. Є повідомлення про те, що багато компаній успішно проводять експерименти в Інтернеті, але бракує систематизованої рамкової моделі для опису того, як такі експерименти слід проводити та систематично використовувати при розробці продукції.

Емпіричні дослідження на тему постійних експериментів у розробці програмних продуктів є плідною основою для подальших досліджень. Програмні компанії отримають користь від чітких вказівок щодо того, коли і як застосовувати постійні експерименти при розробці та розробці програмно-інтенсивних продуктів та послуг.

У цій роботі ми підбираємо модель для безперервного експериментування, засновану на аналізі попередніх досліджень, проти багаторазового дослідження в лабораторії Software Factory. Модель описує процес експериментів, в якому припущення щодо розвитку продукту та бізнесу виводяться з бізнес-стратегії, систематично перевіряються та результати використовуються для подальшого розвитку стратегії та продукту.

Архітектура інфраструктури для підтримки моделі враховує ролі, завдання, технічну інфраструктуру та інформаційні артефакти, необхідні для проведення масштабних безперервних експериментів. Система безперервного експериментування вимагає можливості випускати мінімально життєздатні продукти або функції з відповідними приладами, розробляти та управляти планами експериментів, пов'язувати результати експериментів із дорожньою картою продукту та керувати гнучкою бізнес-стратегією. Для такої системи існує кілька критичних факторів успіху. Організація повинна бути в змозі правильно і швидко проектувати експерименти, виконувати розширений інструментарій програмного забезпечення для збору, аналізу та зберігання відповідних даних, і інтегрувати результати

експерименту в обох етапах циклу розробки продукту і розробки програмного забезпечення процесу. Петлі зворотного зв'язку повинні існувати, через які відповідна інформація подається назад з експериментів на кілька частин організації. Правильне розуміння того, що для тестування і чому повинен існувати, і організація потрібна робоча сила з можливістю збору і аналізу якісних і кількісних даних. Крім того, дуже важливо, що організація має можливість правильно визначити критерії прийняття рішень і діяти від рішень на основі надходять даних.

У майбутній роботі ми очікуємо, що модель буде розширена у міру збільшення кількості випадків використання на місцях. Також можуть знадобитися варіанти моделі для конкретного домену. Крім того, є багато конкретних питань стосовно окремих частин моделі. Деякі конкретні галузі включають (i) як визначити пріоритети припущення та вибрати, які припущення перевірити першими; (ii) як оцінити валідність та визначити, наскільки експериментальним результатам можна довіряти – особливо як забезпечити надійність експериментів, коли паралельно проводяться потенційно тисячі з них; (iii) як вибрати належні експериментальні методи для різних рівнів зрілості товару або послуги; та (iv) як створити внутрішню систему для безперервних експериментів, яка може масштабуватися до потреб дуже великих розгортань і може полегшити і навіть частково автоматизувати створення експериментальних планів.

Особливі питання щодо автоматизації включають, які частини моделі можуть бути автоматизовані або підтримуватися за допомогою автоматизації. Інше питання полягає в тому, як швидко можна виконати блок Build-Measure-Learn, і який вплив моделі на продуктивність впливає на процес розробки програмного забезпечення.

СПИСОК ЛІТЕРАТУРИ

1. Eric Ries, The Lean Startup: How Today’s Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, 2011.

2. Fawler M. Continuous Integration. [Електронний ресурс]. – Режим доступу: https://www. martinfowler. com/ articles/ continuous Integration.html

3. 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.

4. Харченко, Олександр, Ігор Боднарчук, and Ірина Галай. "Метод для отримання множини показників якості архітектури програмного забезпечення." (2013).

5. Боднарчук, І. О. "Побудова та аналіз моделі якості архітектури програмногозабезпечення." Вимірювальна та обчислювальна техніка в технологічних процесах 3 (2013): 89-98.

6. Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch, Climbing the “Stairway to Heaven” – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous 1002 Deployment of Software, 39th EUROMICRO Conference on Software Engineering and Advanced Applications (2012), 392–399.

7. U. Eklund and J. Bosch, Architecture for Large-Scale Innovation Experiment Systems, Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012, pp. 244–248.

8. Helena Holmström Olsson and Jan Bosch, The HYPEX Model: From Opinions to Data-Driven Software Development, Continuous Software Engineering, 2014, pp. 155–164.

9. F. Fagerholm, N. Oza, and J. Münch, A platform for teaching applied distributed software development: The ongoing journey of the Helsinki software factory, 3rd International Workshop on Collaborative 992 Teaching of Globally Distributed Software Development (CTGDSD), 2013, pp. 1–5.

10. Jim Nieters and Amit Pande, Rapid Design Labs: A Tool to Turbocharge Design-led Innovation, 1027 Interactions (2012), 72–77.

11. Robert Yin, Case study research: design and methods, 4th ed., SAGE Publications, Inc., 2009.

12. Jürgen Münch, Fabian Fagerholm, Patrik Johnson, Janne Pirttilahti, Juha Torkkel, and Janne Järvinen, Creating Minimum Viable Products in Industry-Academia Collaborations, Proceedings of the Lean Enterprise Software and Systems Conference (LESS 2013, Galway, Ireland, December 1-4), 2013, 1pp. 137–151.

13. Rebecca M. Henderson and Kim B. Clark, Architectural Innovation: The Reconfiguration of Existing 996 Product Technologies and the Failure of Established Firms, Administrative Science Quarterly 35 (1990), no. 1, 9–30.

14. Fabian Fagerholm and Jürgen Munch, Developer Experience: Concept and Definition, International Conference on Software and System Process, 2012, pp. 73–77.

15. Efi Papatheocharous, Marios Belk, Jaana Nyfjord, Panagiotis Germanakos, and George Samaras, Per1sonalised Continuous Software Engineering, Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, 2014, pp. 57–62.

16. Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch, Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous 1002 Deployment of Software, 39th EUROMICRO Conference on Software Engineering and Advanced Applications (2012), 392–399.

17. Вовк О. В. Особливості синдрому професійного вигорання в працівників сфери інформаційних технологій. [Електронний ресурс]. – Режим доступу: http://maup.com.ua/assets/files/psihologz/2019-1/02.pdf

18. Назарук Н. Каузально-телеологічний формат профілактики «професійного вигорання» вчителя / Н. Назарук // Психологія особистості. 2012.- No 1 (3). – С. 119–128.

19. Полякова О. Категорія та структура професійних деформацій /О. Полякова //Національний психологічний журнал.-2014. -No 1.- С. 57-64.

20. Rob J. Adams, Bradee Evans, and Joel Brandt, Creating Small Products at a Big Company: Adobe's Pipeline Innovation Process, CHI'13 Extended Abstracts on Human Factors in Computing Systems, 2013, pp. 2331–2332.

21. V Basili, J Heidrich, M Lindvall, J Münch, M Regardie, D Rombach, C Seaman, and A Trendowicz, GQM+Strategies: A comprehensive methodology for aligning business strategies with software measurement, Proceedings of the DASMA Software Metric Congress (MetriKon 2007): Magdeburger Schriften zum Empirischen Software Engineering, 2007, pp. 253–266.

22. V. Basili, R. Selby, and D. Hutchens, Experimentation in Software Engineering, IEEE Transactions on Software Engineering 12 (1986), no. 7, 733–743.

23. Steve Blank, The Four Steps to the Epiphany: Successful Strategies for Products that Win, 2nd ed., K&S Ranch, 2013.

24. Jan Bosch, Building Products as Innovation Experiment Systems, Software Business, 2012, pp. 27–39.

25. Jan Bosch, Helena Holmström Olsson, Jens Björk, and Jens Ljungblad, The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups, Lean Enterprise Software and Systems, 2013, pp. 1–15.

26. Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram, Design Science in Information Systems Research, MIS Quarterly 28 (2004), no. 1, 75–105.

27. Natalia Juristo and Ana M. Moreno, Basics of Software Engineering Experimentation, Springer, 2001.

28. Ron Kohavi, Thomas Crook, and Roger Longbotham, Online Experimentation at Microsoft, Third Workshop on Data Mining Case Studies and Practice Prize, 2009.

29. Ron Kohavi, Alex Deng, Brian Frasca, Roger Longbotham, Toby Walker, and Ya Xu, Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained, Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 786–794.

30. Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Ni Pohlmann, Online Controlled Experiments at Large Scale, Proceedings of the 19th ACM

SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’13), 2013, pp. 1168–1176.

31. Beverly May, Applying Lean Startup: An Experience Report – Lean & Lean UX by a UX Veteran: Lessons Learned in Creating & Launching a Complex Consumer App, Agile Conference (AGILE) 2012, 2012, pp. 141–147.

32. Jürgen Münch, Fabian Fagerholm, Petri Kettunen, Max Pagels, and Jari Partanen, Experiences and Insights from Applying GQM+Strategies in a Systems Product Development Organization, Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 12013), 2013.

33. Mary Poppendieck, Lean software development: an agile toolkit, Addison-Wesley Professional, 2003.

34. Mary Poppendieck and Michael A Cusumano, Lean Software Development: A Tutorial, IEEE Software (2012), 26–32.

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

М А Т Е Р І А Л И

VIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

9–10 грудня 2020 року

ТЕРНОПІЛЬ
2020

УДК 004.42
Н.М. Захарків, А.І. Леськів
(Тернопільський національний технічний університет імені Івана Пулюя)

ЗАДАЧІ НЕПЕРЕРВНИХ ПРОЦЕСІВ ІНТЕГРАЦІЇ ТА РОЗГОРТАННЯ ПРИ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

UDC 004.42
N.M. Zakharkiv, A.I. Leskiv

PROBLEMS OF CONTINUOUS INTEGRATION AND DEPLOYMENT IN SOFTWARE DEVELOPMENT

Процеси неперервної інтеграції та розгортання в інженерії процесів розробки програмного забезпечення позначаються як CI/CD. При чому «CI» завжди означає «Неперервна інтеграція», яка фактично є автоматичним процесом в розробці програмного забезпечення (ПЗ). Успішне впровадження процесу CI означає негайне долучення нового програмного коду до спільного репозиторію, що керований системою контролю версій (СКВ). Таке рішення доречно використовувати, коли при розробці програмного продукту головна вітка репозиторію має багато відгалужень (бранчів), що спричиняє конфлікти при запису змін у репозиторій та з'єднанні віток коду.

«CD» у позначенні CI/CD має значення або неперервної доставки, або неперервного розгортання. Ці терміни взаємопов'язані та часто використовуються один замість іншого. Але різниця все ж є. Неперервна доставка як правило означає, що зміни. Зроблені розробником, автоматично будуть пропущені через автотести та при успішному їх проходженні будуть завантажені у репозиторій (наприклад, GitHub, GitLab чи інша СКВ). Після цього ці зміни можуть бути розгорнуті на робоче середовище. Автоматизація цього процесу покликана вирішити задачу видимості виконаної роботи розробниками та комунікацію між ними та командою, що забезпечує бізнесову сторону проекту. Таким чином досягається мінімізація затрат людського ресурсу на розгортання нового коду.

Неперервне розгортання (інше значення «CD») означає автоматичну побудову нового програмного коду з репозиторію та, відповідно, оновлення версії робочого програмного додатка. В цьому випадку такий процес розвантажує команду, що відповідає за розгортання внесених змін.

Таким чином процес CI/CD можна зобразити, як показано на рис. 1.

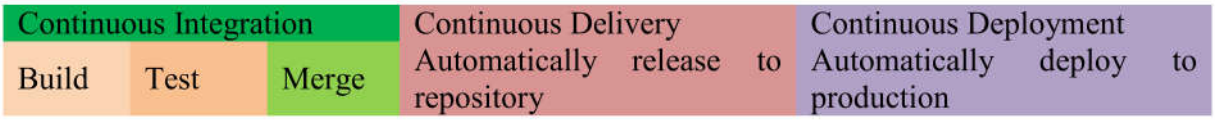

Рисунок 1. Послідовність (зліва направо) процесів CI/CD

Таким чином, не поглиблюючись в нюанси термінології, можна стверджувати, що CI/CD – це процес, який часто показують у вигляді конвеєра, та котрий на високому рівні автоматизує процес розробки ПЗ та дозволяє контролювати його. Типовим є поступове впровадження цих процесів у компаніях. Як правило, спочатку вводять процес CI для автоматизації добавлення нових змін у репозиторій. Потім виконується автоматизація процесів розгортання внесених змін. Типовим при цьому є використання хмарних сервісів.