ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально–науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи
на ступінь вищої освіти магістр

на тему: «**РОЗРОБКА МЕТОДИКИ ТЕСТ-МЕНЕДЖМЕНТУ НА ОСНОВІ ПРОДУКЦІЙНОЇ МОДЕЛІ**»

Виконав: студент 6 курсу, групи ПДм-62
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Читулян Вадим Олегович
(прізвище та ініціали)

Керівник Дібрівний О. А.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти «Магістр»

Спеціальність підготовки 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри
Інженерії програмного
забезпечення
_____Негоденко О.В.
«____» _____ 2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Читулян Вадим Олегович
(прізвище, ім'я, по-батькові)

1. Тема роботи: «Розробка методики тест-менеджменту на основі продукційної моделі»

Керівник роботи Дібрівний Олесь Андрійович, доцент кафедри, доктор філософії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від — «12» жовтня 2021 року №122.

2. Строк подання студентом роботи 31.12.2022

3. Вхідні дані до роботи: Наукова література, продукційна модель як основа методики тест-менеджменту.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Тестування програмного забезпечення;

4.2. Продукційна модель представлення знань;

4.3.Аналіз процесу та систем тест-менеджменту

4.4. Порядок організації взаємодії з trainee на проекті;

4.5. Розробка моделей для представлення даних trainee, ментора та проекту;

4.6. Реалізація методики тест-менеджменту;

4.7. Відтворення моделі та аналіз результатів.

5. Перелік графічного матеріалу (презентація)

5.1. Моделі представлення даних в базі фактів продукційної моделі

5.2. Модель визначення сумісності трійки ментор-трейні-проект

5.3. Модель обчислення сумісності пар

5.4. Екранні форми для представлення моделі

6. Дата видачі завдання «14» жовтня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Отримання завдання на магістерську роботу	14.10.2022	
2	Огляд та аналіз особливостей тест-менеджменту при розподілі членів команди тестувальників на IT-проектах та організації навчання трейні	25.10.2022	
3	Аналіз моделей та методів для організації тест-менеджменту	12.11.2022	
4	Розробка продукційної моделі представлення знань системи тест-менеджменту	20.11.2022	
5	Розробка методики тест-менеджменту для забезпечення розподілу трейні та менторів команди тестувальників IT-проекту	01.12.2022	
6	Розробка програмних засобів для апробації методики тест-менеджменту	12.12.2022	
7	Моделювання та аналіз результатів	21.12.2022	
8	Написання та оформлення пояснювальної записки	29.12.2022	
9	Розробка графічних та презентаційних матеріалів	31.12.2022	
10	Захист магістерської роботи	17.01.2023	

Студент _____ Читулян В.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Дібрівний О.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи 90 сторінок.

Ключові слова: ТЕСТУВАННЯ ПЗ, МЕТОДИКА ТЕСТ-МЕНЕДЖМЕНТУ, ПРОДУКЦІЙНА МОДЕЛЬ, ТРЕЙНІ, МЕНТОР.

Об'єкт дослідження – розподіл трейні та менторів команди тестувальників у процесі тест-менеджменту ІТ-проекту.

Предмет дослідження – методика тест-менеджменту для оптимального розподілу трейні та менторів команди тестувальників ІТ-проекту на основі продукційної моделі.

Мета роботи – оптимізація розподілу трейні та менторів команди тестувальників в процесі тест-менеджменту ІТ-проекту на основі продукційної моделі.

Методи дослідження – математичні: статистичні методи, методи представлення знань за допомогою продукційних моделей; емпірико-теоретичні: абстрагування, аналіз, синтез, методи експертних оцінок, моделювання; методи проектування та розробки програмного забезпечення.

У дипломній роботі розглянуто та проаналізовано актуальний стан тест-менеджменту. Охарактеризовано переваги та недоліки методики тест-менеджменту. Описано правила побудови продукційної моделі, та її аспекти використання у галузі тест-менеджменту. Розроблено модель представлення знань trainne та методику тест-менеджменту на основі продукційної моделі.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

ПП – програмний продукт

SQA – Software Quality Assurance

SRET − Software Reliability Engineered Testing

GUI − Graphical user interface

ІТ – інформаційні технології

СІТ – сучасні інформаційні технології

TMS – Test Management System

ALM − Application Lifecycle Management

ЕС – експертна система.

ВСТУП

Актуальність обраної теми. Тестування програмного забезпечення має важливе значення для забезпечення якості програмних засобів. У більшості програмних проектів на тестування витрачають щонайменше 30 відсотків часу відведеного на реалізацію проекту. Для додатків, що призначені для безпеки, тестування програмного забезпечення можуть витрачати від 50 до 80 відсотків часу.

Актуальність дипломної роботи полягає в тому, що на сьогоднішній день, тестування програмного забезпечення є поширеним видом діяльності для підтвердження якості програмних продуктів. Однак, існує нестача кваліфікованих фахівців у цій галузі. Це може бути викликано складністю викладання тестування ПЗ через підходи, які використовують лише теоретичні заняття та традиційні інструменти. Крім того, існує недостатня мотивація через робоче середовище, стратегії розподілу та відповідальності цих фахівців у командах розробників та тестувальників. Входження на реальний проект студента-стажера – це складний процес, який в провідних компаніях відбувається під керівництвом досвідчених фахівців-менторів. Однак, визначення трійки трейні-ментор-проект часто викликає складності, оскільки передбачає дуже велику кількість варіантів розподілу та врахування значної кількості факторів.

Мета роботи – оптимізація розподілу трейні та менторів команди тестувальників в процесі тест-менеджменту ІТ-проекту на основі продукційної моделі.

Об'єкт дослідження − розподіл трейні та менторів команди тестувальників у процесі тест-менеджменту ІТ-проекту.

Предмет дослідження – методика тест-менеджменту для оптимального розподілу трейні та менторів команди тестувальників ІТ-проекту на основі продукційної моделі.

Щоб досягнути поставленої мети, слід виконати такі завдання:

1. Провести аналіз досліджень методів та моделей, які використовуються для організації тестування ІТ-проектів;

2. Обґрунтувати обраний інструментарій для розробки практичної частини;

3. Розробити методику тест-менеджменту на основі продукційної моделі;

4. Провести аналіз і узагальнення результатів дослідження.

Під час написання роботи було використано такі *методи дослідження*: математичні: статистичні методи, методи представлення знань за допомогою продукційних моделей; емпірико-теоретичні: абстрагування, аналіз, синтез, методи експертних оцінок, моделювання; методи проектування та розробки програмного забезпечення.

.

1 ОГЛЯД ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ МЕТОДИКИ ТЕСТ-МЕНЕДЖМЕНТУ

1.1. Тестування програмного забезпечення

Регулярна і послідовна практика комплексного підходу до тестування може підвищити рівень якості відповідної галузі, який забезпечують розробники програмного забезпечення та очікують замовники. Використовуючи комплексний підхід до тестування, тестувальники програмного забезпечення можуть перетворити негативний ризик значних втрат бізнесу в позитивну конкурентну перевагу.

Тестування є:

- важливою, обов'язковою частиною розробки програмного забезпечення;
- методом оцінки якості продукту, а також шляхом непрямого поліпшення цієї якості;
- методом виявлення дефектів і багів.

Забезпечення якості продукту − це перевірка етапів розробки, тобто набагато зручніше уникнути проблеми, яка може виникнути, аніж усувати її. Тестування, в першу чергу, повинно розглядатися як засіб, для перевірки чи була вона ефективною, а також для виявлення ситуацій, коли з якихось причин вона не дала потрібного результату. Адже після успішного завершення тестування проекту, програмне забезпечення все одно може містити дефекти. Виправлення системних збоїв, які виникають після запуску програми, забезпечується (коригувальними) заходами технічного обслуговування.

Тестування більше не розглядається як діяльність, яка починається тільки після завершення етапу кодування, з обмеженою метою виявлення збоїв. Сьогодні тестування програмного забезпечення розглядається як структура, яка повинна охоплювати весь процес розробки, і є важливою частиною фактичного створення продукту. Дійсно, планування тестування повинно починатися з ранніх стадій

аналізу вимог, а плани і процедури тестування повинні систематично і безперервно вдосконалюватися по мірі того, як триває розробка. Ця діяльність з планування та проектування випробувань є корисним внеском для розробників, для виявлення потенційних слабких місць (таких як, наприклад, проектні помилки або протиріччя, а також упущення, пропуски або неясності в документації) [20].

Тестування програмного забезпечення складається з динамічної перевірки поведінки програми на кінцевому наборі тестових випадків, належним чином вибраних з зазвичай нескінченної області виконання, на відповідність заданій очікуваній поведінці.

У наведеному вище визначенні, а також у наступних, виділені слова відповідають ключовим питанням у визначенні області знань тестування програмного забезпечення.

Динамічна перевірка. Завжди передбачає виконання програми на вхідних даних. Значення вхідних даних не завжди є достатнім для визначення тесту, оскільки складна, недетермінована система може реагувати по-різному на один і той самий вхід, залежно від стану системи. Але слід розрізняти термін «вхідний сигнал» та вхідний стан. Вхідний стан тестування доповнюють його статичні методи аналізу, такі як експертна оцінка та інспектування (які іноді неправильно називають як «статичне тестування»), такі методи не розглядаються як частина цієї галузі знань (як і виконання програм на вхідних даних або символічного оцінювання).

Скінченність. Навіть для простих програм теоретично можлива така кількість тестових випадків, що для вичерпного тестування можуть знадобитися роки. Ось чому на практиці весь набір тестів можна вважати нескінченним. Але кількість виконань, які реально можна спостерігати при тестуванні, повинна бути скінченною. Для забезпечення якості продукту слід проводити «достатню» кількість випробувань. Тестування завжди передбачає умови обмеження між ресурсами і графіками, і по суті необмеженими вимогами до тестів: цей конфлікт вказує на добре відомі проблеми тестування, як технічні за своєю природою

(критерії визначення адекватності тесту), так і управлінського характеру (оцінка зусиль, які необхідно докласти до тестування) [22].

Відбір. Багато запропонованих методик тестування суттєво відрізняються за способом відбору (кінцевого) набору тестів, і тестувальники повинні усвідомлювати, що різні критерії відбору можуть давати значною мірою різну результативність. Як визначити найбільш підходящий критерій відбору за даних умов − дуже складна проблема. На практиці часто застосовують такий метод аналізу ризиків як – експертиза. Експертиза проводиться для вибору даних, які повинні бути підтверджені тестувальником на основі його знань і досвіду. Тому досвід тестувальника має велике значення.

Очікування. Один з компонентів тестування програмного продукту (ПП), що дає можливість вирішити, чи є спостережувані результати виконання програми прийнятними чи ні, інакше процес тестування був зроблений марно. Поведінка, що спостерігається може бути перевірена на відповідність очікуванням користувача (зазвичай це називається тестуванням для перевірки) або зі специфікацією (тестування для верифікації). Рішення про проходження/непроходження тесту зазвичай називають в літературі з тестування − проблемою оракула [24], яку можна вирішити за допомогою різних підходів, наприклад шляхом перевірки результатів людиною або шляхом порівняння з існуючою системою відліку. У деяких ситуаціях очікувана поведінка може бути визначена лише частково, тобто, лише деякі частини фактичної поведінки повинні бути перевірені на відповідність деяким заявленим твердженням.

1.1.1. Концепція тестування програмного забезпечення

Тестування програмного забезпечення зазвичай проводиться на різних рівнях в процесі розробки. Тобто, об'єкт тестування може бути різним: ціла система, її частини (пов'язані за призначенням, використанням, поведінкою або структурою), окремий модуль [10].

Тестування проводиться з огляду на певну мету, яка формулюється із різним ступенем точності. Формулювання мети в точному, кількісному вираженні дозволяє встановити контроль над процесом випробувань.

Однією з цілей тестування є виявлення збоїв (якомога більшої кількості), для цього було розроблено багато популярних тестових методик для досягнення цієї мети. Такі методи по-різному намагаються «зламати» програму, виконуючи один (або декілька) тестів, взятих з визначених класів (що вважаються еквівалентними) виконань. Провідним принципом, що лежить в основі таких методів є визначення репрезентативного набору поведінки програми (як правило, у вигляді підкласів вхідної предметної області). Всебічний погляд на область знань про тестування програмного забезпечення як засобу забезпечення якості має включати інші, не менш важливі цілі тестування, наприклад, вимірювання надійності, оцінка юзабіліті, прийняття підрядником, для яких будуть застосовуватися різні підходи. Слід зазначити, що мета випробування змінюється залежно від об'єкта випробування, тобто, в загальному випадку різні цілі вирішуються на різних рівнях тестування.

Мета та завдання тесту полягає в таких аспектах, як:

- як визначити набір тестів;
- скільки часу відведеного на тестування достатньо для досягнення поставленої мети;
- які тестові кейси повинні бути відібрані для виконання завдання [28].

Зазвичай частина «для досягнення поставленої мети» залишається неявною, і до уваги беруться лише перші 2 пункти, які вказані вище. Критеріями для відповіді на перші два запитання є критерії адекватності тесту, а на третє − критерії відбору тестів.

Інколи користувачі плутають цілі та методи тестування. Тестові методики слід розглядати як допоміжні засоби, що допомагають забезпечити досягнення цілей тесту. Наприклад, покриття гілок є популярною тестовою технікою. Досягнення визначеного показника для покриття не повинно розглядатися як ціль

тестування. Ціль тестування − це засіб підвищення шансів на виявлення збоїв шляхом систематичного відпрацювання кожної гілки програми, що виходить з точки прийняття рішення. Щоб уникнути таких непорозумінь, слід чітко розрізняти тестові заходи, які оцінюють ретельність набору тестів, наприклад: міри покриття, які замість цього надають оцінку програми, що тестується, на основі спостережуваних результатів тестування.

Концепції, стратегії, методи та заходи тестування мають бути інтегровані у визначений та контрольований процес, яким керують люди. Процес тестування підтримує діяльність з тестування та надає вказівки командам з тестування, починаючи з планування тестування та закінчуючи оцінкою результатів, для того, щоб забезпечити впевненість у тому, що цілі тестування досягнуті економічно ефективно.

Тестування програмного забезпечення є дуже дорогою і трудомісткою частиною розробки. З цієї причини важливу роль відіграють інструменти для автоматизованого виконання тестів, реєстрації та оцінки результатів тестування, а також в цілому для підтримки тестової діяльності. Крім того, з метою підвищення співвідношення «витрати-ефективність», ключовим питанням завжди було просування автоматизації тестування наскільки це можливо [20].

1.1.2. Огляд базових концепцій тестування ПЗ

Відповідно до представленої нижче концептуальної схеми (рис.1.1), опис предметної області «Тестування програмного забезпечення» організовано наступним чином.

Частина А присвячена основним поняттям та визначенням тестування. Вона охоплює основні визначення в галузі тестування програмного забезпечення, а також введення в термінологію. У цій же частині викладено сферу застосування цієї галузі знань, що пов’язана з іншими видами діяльності.

Частина Б розкриває поняття рівнів тестування. Вона складається з двох підрозділів:

- В.1 − перераховує рівні, на які традиційно поділяється тестування великих програмних систем;
- В.2 − розглядається тестування специфічних умов або властивостей тестування, що є поясненням поняття «Цілі тестування».

Очевидно, що не всі види тестування застосовуються до кожної системи, і не всі типи тестування використовують на практиці дуже часто.

За останні два десятиліття було розроблено декілька методик випробувань за різними критеріями, але на цьому розширення теми методики не зупинилося, з кожним роком пропонуються нові «Загальноприйняті» методи (описані в *частині C*).

Події, що пов'язані з випробуваннями, розглядаються в *частині D*.

Питання, пов'язані з управлінням процесом випробувань описуються в *Частині E*).

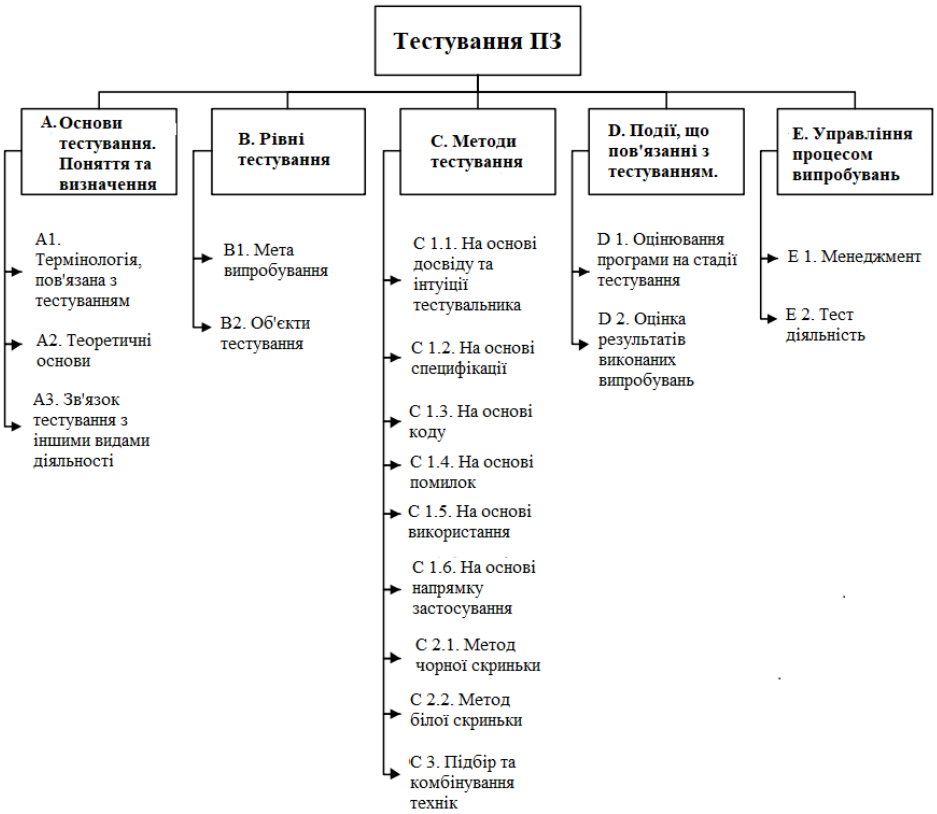


Рисунок 1.1 – Концепція тестування програмного забезпечення [4]

Отже, сфера тестування програмного забезпечення складається з об’ємного переліку розділів, кожен з яких містить відповідні теми, які потрібно опрацювати для отримання знань, умінь та навичок для роботи у відповідній галузі.

1.1.3.Розподіл тем для вивчення тестування програмного забезпечення

Для вивчення галузі «Тестування програмного забезпечення» стажеру, який підключається до IT-проекту, необхідно попередньо опанувати певний перелік тем теоретичного та практичного матеріалу (див. табл.1.1, табл.1.2, табл. 1.3).

Таблиця 1.1 – Теми базових понять та визначень [4]

Тема	**Параграф**
A. Перевірка основних понять та визначень	
A1. Термінологія, що стосується тестування.	Поняття тестування та пов'язана з ним термінологія
	Баги та дефекти
A2. Теоретичні засади	Критерії відбору тестів
	Ефективність тестування/Цілі тестування
	Випробування на усунення дефектів
	Проблема оракула
	Теоретичні та практичні обмеження тестування
	Проблема невиконаних завдань
	Можливість тестування
A3. Зв'язок тестування з іншими видами діяльності	Тестування та методи статичного аналізу
	Тестування та підтвердження виконання умов. Верифікація
	Тестування та налагодження
	Тестування та програмування
	Тестування в рамках SQA
	Випробування в умовах «чистого середовища»

Таблиця 1.2 – Рівні тестування [4]

Тема	Параграф
B. Рівні тестування	
B1. Мета випробування	Unit тестування
	Інтеграційне тестування
	Системне тестування
B2. Цілі тестування	Приймальне тестування
	Тестування встановлення
	Alpha і Beta тестування
	Тестування на відповідність. Функціональне тестування. Тестування на коректність
	Досягнення та оцінка надійності шляхом тестування
	Регресійне тестування
	Тестування продуктивності
	Стрес тестування
	Back-to-back тестування
	Відновлювальне тестування
	Тестування конфігурації
	Usability тестування

Таблиця 1.3 – Техніки тест-дизайну [4]

Підрозділ	Тема	Параграф
C. Техніки тест-дизайну		
C1. Критерій «База на якій формуються тести»	C1.1. На основі інтуїції та досвіду тестувальника	Ad hoc тестування
	C1.2. На основі специфікації	Розбиття на частини за класами еквівалентності
		Аналіз граничних значень

Продовження таблиці 1.3 – Техніки тест-дизайну [4]

Підрозділ	**Тема**	**Параграф**
		Таблиця прийняття рішень
		На основі скінченних автоматів
		Тестування на основі формальних специфікацій
		Випадкове тестування
	С1.3. На основі коду	Моделі для тестування на основі коду (flow граф, граф викликів)
		Тести на основі блок-схеми
		Тести на основі даних
	С1.4. На основі помилок	Помилкове припущення
		Мутаційне тестування
	С1.5. На основі використання	Профіль діяльності
		SRET [7]
	С1.6. Залежно від характеру застосування	Об'єктно-орієнтоване тестування
		Компонентне тестування
		Паралельне тестування
		Тестування на відповідність вимогам протоколу
		Тестування розподілених систем
		Тестування систем реального часу
		Тестування наукового програмного забезпечення

Продовження таблиці 1.3 – Техніки тест-дизайну [4]

Підрозділ	**Тема**	**Параграф**
С2. Критерій «Відомості про реалізацію»	С2.1. Метод тестування «Чорна скринька»	Розбиття на частини за класами еквівалентності
		Аналіз граничних значень
		Таблиця прийняття рішень
		На основі скінченних автоматів
		Тестування на основі формальних
		специфікацій
		Помилкове припущення
		Випадкове тестування
		Профіль діяльності
		SRET
	С2.2. Метод тестування «Біла скринька»	Моделі для тестування на основі коду (flow граф, граф викликів)
		Тести на основі блок-схеми
		Тести на основі даних
		Мутаційне тестування
С3. Підбір та комбінування технік		Функціональні та структурна
		Покриття та ефект/насичення

Отже, тестування програмного забезпечення важливе тому, що якщо в програмному забезпеченні є якісь помилки, їх можна виявити на ранній стадії та виправити до того, як програмний продукт буде випущений. Належним чином протестований програмний продукт забезпечує надійність, безпеку та високу продуктивність, що призводить до економії часу, ефективності та задоволеності клієнтів

1.2. Продукційна модель представлення знань

Продукційна модель базується на наборі правил поведінки. Ці правила є базовим представленням, яке вважається корисним в експертних системах, автоматизованому плануванні та виборі дій. Це також забезпечує певну форму штучного інтелекту.

Продукційна модель – модель представлення знань, яка зазвичай використовується для забезпечення певної форми штучного інтелекту, що складається в першу чергу з набору правил поведінки, але також включає механізм, необхідний для дотримання цих правил, оскільки модель складається відповідно до вхідних даних [13].

1.2.1. Складові продукційної моделі

Основними компонентами продукційної моделі є:

1. Глобальна база даних. Є центральною структурою даних, що використовується продукційною моделлю.

2. Набір правил. Кожне правило зазвичай має передумову, яка або задовольняється, або не задовольняється глобальною базою даних. Якщо передумова задовольняється, правило зазвичай застосовується. Застосування правила змінює базу даних.

3. Система управління. Система управління вибирає, яке правило має бути використане, і припиняє обчислення, коли виконується умова припинення роботи бази даних. Якщо кілька правил спрацьовують одночасно, система управління вирішує такі конфлікти.

1.2.2. Особливості продукційної моделі

До основних особливостей продукційної моделі можна віднести (рис.2.1):

1. *Простота*. Структура кожного елементу в продукційній моделі є унікальною та одноманітною, оскільки використовується структура «ЯКЩО-ТО».

Ця структура забезпечує простоту представлення знань. Така умова покращує читабельність продукційних правил.

2. *Модульність*. Правила продукційної моделі кодують знання, доступні в дискретних частинах. Інформація може розглядатися як набір незалежних фактів, які можуть бути додані або видалені з системи практично без шкідливих побічних ефектів.

3. *Модифікованість*. Можливість модифікації правил. Тобто є можливість спочатку розробити правила виробництва в загальній формі, а потім уточнити їх відповідно до конкретного застосування.

4. *Великий обсяг інформації*. База знань продукційної моделі зберігає «чисті» знання. Ця частина не містить жодної інформації з управління або програмування. Кожне виробниче правило зазвичай записується у вигляді англійського речення; проблема семантики вирішується самою структурою представлення [30].

1.2.3. Стратегії контролю/пошуку

Для вибору правила, що потрібно застосувати під час пошуку рішення існують певні вимоги, про які потрібно пам'ятати:

- у кожному елементі моделі повинні бути описані певні дії процесу;
- модель має бути систематичною;
- дії описані в моделі повинні бути ефективними, для того щоб знайти правильне рішення проблеми.

Правила продукційної моделі можна класифікувати як:

- правила дедуктивного виводу;
- правила абдуктивного виведення.

Знання у продукційній моделі можна представити у вигляді набору правил разом з системою управління та базою даних. Це можна записати у вигляді: *Якщо (Умова) Тоді (Умова).* Продукційні правила також відомі як пари:

- умова-дія;
- антецедент-консеквент;

- шаблон-дія;
- ситуація-реакція;
- зворотний зв'язок-результат [30].

1.2.4. Класи продукційних моделей

Продукційні моделі представлення знань можна поділити на 4 основні класи.

Послідовна продукційна модель. Модель, у якій застосування одного правила ніколи не перешкоджає подальшому застосуванню іншого правила, яке також могло бути застосоване в той момент, коли було обрано перше правило.

Частково комутативна виробнича модель. Тип виробничої моделі, у якій застосування послідовності правил перетворює стан X в стан Y, тоді будь-яка допустима перестановка цих правил також перетворює стан X в стан Y.

Непослідовна продукційна модель. Використовують для розв'язання складних проблем. Ці моделі важливі з точки зору реалізації, оскільки вони можуть бути реалізовані без можливості повернення до попередніх станів, коли виявляється, що було обрано неправильний шлях. Така модель виробництва підвищує ефективність, оскільки немає необхідності відстежувати зміни, що вносяться в процесі пошуку.

Комутативні моделі. Використовують для вирішення проблем, у яких зміни відбуваються, але можуть бути скасовані, і в яких порядок роботи не є критичним. Виробничі моделі, які зазвичай не є частково комутативними, є корисними під час вирішення багатьох проблем, де є незворотні зміни, наприклад, хімічний аналіз. При роботі з такими системами порядок виконання операцій дуже важливий і тому правильні рішення повинні прийматися з першої спроби.

1.2.5 Переваги та недоліки продукційної моделі представлення знань

Серед переваг продукційної моделі у сфері штучного інтелекту можна виділити наступні:

- надає хороший набір інструментів для структурування програм;

- модель є високомодульною, оскільки окремі правила можуть бути додані, видалені або змінені незалежно;
- відокремлення знань та циклу «контроль-визнання-акт»;
- в моделі використовується шаблонне управління, яке є більш гнучким, ніж алгоритмічне управління;
- надає можливості евристичного управління пошуком;
- корисна для використання в середовищі реального часу та додатках.

Але також є деякі недоліки:

- складно проаналізувати потік контролю всередині виробничої моделі;
- у моделі описані операції, які можна виконати в процесі пошуку вирішення проблеми;
- відсутність навчання зумовлена продукційної моделлю, що базується на правилах і не зберігає результат розв'язання задачі для подальшого використання;
- правила у продукційній моделі не повинні мати жодного типу вирішення конфліктів, оскільки, коли нове правило додається до бази даних, воно повинно гарантувати, що воно не має конфлікту з будь-яким існуючим правилом.

Отже, продукційна модель представлення знань дозволяє користувачеві оперувати знаннями (створювати, модифікувати, переглядати, видаляти) та виконувати певні операції. Ці функціональні можливості повинні бути максимально зручними, інакше використання моделі не матиме результату.

2 РОЗРОБКА МЕТОДИКИ ТЕСТ-МЕНЕДЖМЕНТУ НА ОСНОВІ ПРОДУКЦІЙНОЇ МОДЕЛІ

2.1. Аналіз процесу та систем тест-менеджменту

Оскільки програмні проекти стають все більш складними і об’ємними, і містять велику кількість різних платформ і пристроїв, які необхідно протестувати, важливо мати надійний процес управління тестуванням і переконатися, що обмежені ресурси тестування зосереджені на областях з найбільшим ризиком і важливістю. Спеціальні системи для управління тестуванням допомагають керувати цим процесом.

Кожен проєкт унікальний і в кожної команди є свої запити. Однак усіх учасників об'єднує бажання працювати з якісними інструментами, які економлять час і дають змогу QA-фахівцям тестувати якісніше та швидше, в ідеалі щоб TMS могла поєднувати ручне та автоматизоване тестування.

Було проаналізовано перевірені часом і нові системи управління тестуванням, які зараз популярні на ринку. Описано функції, які мають бути в Test Management System та їх можливості. Далі буде представлено перелік відповідних інструментів з дослідженими можливостями.

Для зручності роботи система управління тестуванням має відповідати таким критеріям:

- зручне встановлення та підтримка;
- зрозумілий інтерфейс;
- створення та управління проєктами;
- створення користувачів і проєктних ролей для них;
- зручна інтеграція з автоматичними тестами;
- робота з тестовими артефактами: тест-план, тест-кейс, чек-лист, загальні кроки;
- версіонування тест-кейса/чек-листа;

- створення користувацьких атрибутів/конфігурацій;
- зрозуміла система звітності;
- вбудована система баг-трекінгу;
- можливість оповіщення колег усередині та поза системою;
- можливість інтеграції з іншими інструментами [**Error! Reference source not found.**].

При створенні єдиної Test Management System (TMS) для роботи з усією документацією проєкту використовують декілька способів:

1) вибір open-source баг-трекеру для ведення відомостей про дефекти та Google Docs для матриці трасованості;

2) використання TMS з інтегрованими баг-трекерами відповідної компанії;

3) вибір системи управління тестуванням, виходячи з обсягів завдань, специфіки проєктів, використовуваних видів тестування та типів документації.

Під час вибору системи управління тестуванням важливо враховувати багато аспектів, адже для компанії виникнення помилки може коштувати дорого.

2.1.1 ALM Octane

ALM Octane [3] − це платформа для тестування Application Lifecycle Management (ALM), яка налаштована для високошвидкісних Lean та Agile команд. Містить безстрокову (локальну) та передплачену (SaaS) моделі з 3 версіями: Enterprise, Pro та Team.

Можливості:

- для спільного використання програм в корпоративних організаціях;
- для декількох команд, підтримує Agile, Waterfall і Hybrid;
- оптимізована для невеликих команд Agile розробників [3].

Продукт пропонує сучасний браузерний інтерфейс з великою гнучкістю, що дозволяє різним командам виконувати свою роботу так, як вони хочуть, і об'єднує всю інформацію.

ALM Octane підтримує широкий спектр сторонніх інтеграцій (30 за останніми підрахунками), що робить його ідеальним вибором для повного управління життєвим циклом з наскрізною видимістю. Інтеграції підтримки включають: Jenkins, Jira, TFS, VSTS, Azure DevOps, Selenium, Gherkin і Open REST API.

Головними завданнями лідера команди тестувальників є впровадження найкращих практик та покращення комунікації між розробниками, тестувальниками та зацікавленими сторонами.

Безперервне тестування це − прискорення процесу підготовки, підвищення якості та масштабування розгортання. А також можливість визначення ризиків, відстежуючи покриття конвеєра, аналіз і зміни коду [3].

ALM Octane ідеально підходить для тестування від командного до корпоративного рівня і підтримує налаштовані конфігурації робочого простору з видимістю на корпоративному рівні.

Octane пропонує хороші інструменти для управління і відстеження, з підтримкою механізмів бізнес-правил, налаштування форм, дозволів ролей користувачів і великих звітів.

2.1.2 TestRail

«TestRail – веб-інструмент завдяки якому можна керувати і відстежувати процес тестування програмного забезпечення та організовувати відділ контролю якості. Інтуїтивно зрозумілий веб-інтерфейс дозволяє легко створювати тестові кейси, керувати тестовими прогонами і координувати весь процес тестування» [1].

Можливості веб-інструменту:

1. Документація планів тестування та відстеження прогресу в режимі реального часу. Організація та структурування багаторазових тестових кейсів в папках, можливість створити гнучкі плани тестування і відстежити хід виконання тестів за допомогою швидкого і простого у використанні інтерфейсу, розробленого спеціально для управління тестуванням.

2. Збільшення тестового покриття та відстежуваності. Швидке створення тестових кейсів, оцінка покриття та пов'язування тестів з вимогами і дефектами в Atlassian Jira, GitHub Issues, GitLab тощо. Миттєва генерація звітів про простежуваність, для спостереження статусу тестування в режимі реального часу.

3. Забезпечення прозорості та видимості у тестуванні та контролі якості. Можливість відстежити всю тестову діяльність та показники якості на єдиній платформі, щоб покращити співпрацю, оцінити ризики та випустити більш якісне програмне забезпечення.

4. Централізація автоматизованого та ручного тестування. Інтеграція з будь-яким інструментом автоматизації або фреймворком, звітність про автоматизовані тести, що виконуються через CI/CD, та централізована звітність про автоматизовані та ручні тести для наочності та ефективного аналізу [8].

2.1.3 Allure TestOps

Платформа управління якістю програмного забезпечення нового покоління об'єднує автоматизоване та ручне тестування. Можливість підвищення рівеня контролю якості продукту та збільшення продуктивності команди QA і розробників, завдяки налаштуванням TestOps.

Можливості платформи:

1. Тестування:

- розробка як ручних, так і автоматизованих тестів;
- створення планів для тестування продукту як ручними, так і автоматизованими тестами в одному середовищі;
- керування запуском тестів з одного місця;
- створення тестової документації в реальному часі для автоматизованих тестів на основі прописаних правил;
- порівняння історичних даних тестів з поточною документацією;
- імпорт тестів з інших TMS як вручну, так і в автоматизованому режимі;

- групування і перегрупування тестів в різні деревовидні представлення на основі користувацьких полів [2].

2. Тестові запуски:

- запуск, зупинка, повторний запуск завдання збірки з Allure TestOps на серверах збірки;
- встановлення різних параметрів середовища з Allure TestOps для запуску тестів;
- вибір тестів (фільтрація непотрібних тестів) в завданні збірки для виконання;
- вибір кілька завдань збірки (навіть на різних серверах збірки) для виконання плану тестування;
- спостереження за ходом збірки і статусом в режимі реального часу.

3. Результати тестування:

- збір результатів тестів з сервера в реальному часі під час виконання завдання;
- прив'язування невдалих тести до дефектів, щоб заощадити час на аналіз;
- прив'язування збоїв до проблем у системі відстеження проблем;
- експорт результатів тестів в системи відстеження проблем;
- імпорт результатів тестів вручну, якщо це необхідно;
- імпорт результатів тестів прямо з проекту IntelliJ IDEA.

4. Аналітика:

- аналіз тенденції за допомогою вбудованих дашбордів;
- аналіз результатів тестів за допомогою вбудованої мови запитів для побудови власних дашбордів;
- створення тестових звітів у форматі PDF або CSV.

5. Інтеграція:

- готові до використання інтеграції з популярними білд-серверами: Jenkins, Bamboo, Gitlab, TeamCity;

- готові інтеграції з популярними системами відстеження проблем: Jira, Youtrack, Gitlab, GitHub, Bitbucket;
- готові інтеграції з популярними TMS для експорту результатів тестування: TM4J, TestRail, Xray, Azure [2].

2.1.4 PractiTest

«PractiTest − це наскрізна платформа управління тестуванням SaaS, яка централізує всю роботу з контролю якості, процеси, команди та інструменти на одній платформі, щоб подолати розрізненість, уніфікувати комунікацію та забезпечити єдине джерело у організації» [1].

Можливості платформи:

1. Централізація роботи з контролю якості, процеси, команди та інструменти на одній платформі. Підвищення продуктивності тестування, наочність, командна співпрацю та узгодженість бізнесу.

2. Прийняття рішень на основі наскрізної видимості. Огляд високого рівня та детальна інформація про процес контролю якості за допомогою потужних звітів PractiTest, інформаційних панелей, що налаштовуються в реальному часі, і динамічних фільтрів. Отримані дані тестування можна використовувати для прогнозування потенційних проблем, прийняття рішень на їх основі і прискорення роботи [6].

3. Менше витрат, більше користі. Використання елементів тестування та динамічних фільтрів дані відповідно до потреб. Усунення повторюваних кроків дозволить команді зосередитися на тому, що дійсно важливо, і підвищити продуктивність.

4. Виконання потреб. PractiTest має все необхідне, щоб працювати так як зручно користувачеві.

5. Поєднання аспектів контролю якості з цінністю для бізнесу. Повна відстежуваність всіх ресурсів тестування аж до історій користувачів, для узгодження роботи з тестування з бізнес-цілями.

Отже, тестування програмного забезпечення є важливою частиною життєвого циклу розробки програмного забезпечення, а інструменти управління тестуванням є необхідними для досягнення максимальної ефективності. Програмне забезпечення для управління тестуванням дозволяє зробити процес більш ефективним, надаючи єдину платформу для управління автоматизованими або ручними тестами і помилками.

2.1.5 Порівняння систем тест-менеджменту

Системи тест-менеджменту допомагають фіксувати вимоги до тестів, розробляти тестові кейси, звіти про виконання тестів, управляти ресурсами тощо. Недогляд за якістю програмного забезпечення може призвести до значних грошових втрат, втрати репутації або піддати компанію судовим ризикам. Хороший інструмент управління тестуванням − це ключ до уникнення потрапляння помилок і дефектів у виробництво.

Але кожна компанія використовує системи-тест менеджменту, що найбільш підходять за індивідуальними критеріями співробітників та проектів. Для того щоб обрати найкращий інструмент для управління процесом тестування потрібно провести аналіз переваг та недолік вищевказаного переліку систем. Результати порівняння наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння систем тест-менеджменту

Назва	**Функції**	**Переваги**	**Недоліки**
ALM Octane	• створення історій користувачів; • встановлення	• зручність використання; • простий інтерфейс;	• немає можливості спробувати

Продовження таблиці 2.1 − Порівняння систем тест-менеджменту

Назва	Функції	Переваги	Недоліки
	термінів виконання роботи; • керування тестуванням; • створення відповідної документації; • персональні інформаційні панелі з діаграмами в реальному часі; • інструменти для керування життєвим циклом додатків.	• великий набір шаблонів для звітів; • робота з проектами у закритій командній структурі; • контроль кожного етапу роботи; • підтримка гібридного ПЗ.	безкоштовну версію; • відсутні функції інтеграції; • складність з завантаженням деяких елементів через проблеми з рівнями доступу.
TestRail	• управління тестами та певними комплексами тестування; • система відстеження проблем; • керування, планування та запуск тестів;	• скрипти інтерфейсу користувача для налаштування додаткових функцій; • інтеграція з Jira; • хороша ціна; • статистика прогресу тестування в реальному часі;	• немає можливості відновлення тестових прикладів; • немає можливості позначення позиції тесту (pass/faild/blocked);

	• ручне та автоматизоване тестування.	• підвищення продуктивності тестування;	• складний для розуміння інтерфейс;

Продовження таблиці 2.1 − Порівняння систем тест-менеджменту

Назва	**Функції**	**Переваги**	**Недоліки**
		• зручна для роботи у великих командах.	• немає можливості видалення одразу великої кількості об’єктів.
Allure TestOps	• ручне та автоматизоване тестування; • створення набору тестів; • набір результатів тестування в межах одного або кількох тестів; • збереження даних про виконання тесту з параметрами, тегами та інформацією про середовище; • спеціальні категорії, які	• вибірковий та повторний тестовий запуск; • підтримка користувачів; • спрощення багатьох видів тестування; • єдиний компонент для керування багатьма функціями; • зручне сортування тестових кейсів.	• погана інтеграція з платформою iOS; • складний у використанні початківцями; • немає можливості змінювати повідомлення про результати тестування.

	використовуються для створення ієрархії дерева тестів;		

Продовження таблиці 2.1 − Порівняння систем тест-менеджменту

Назва	**Функції**	**Переваги**	**Недоліки**
	• розширенні можливості для описів статусу тестів.		

PractiTest	• організація процесу контролю якості; • наскрізна видимість; • оптимізація роботи контролю якості; • можливість налаштування полів відповідно до процесу чи проекту; • сортування тестів; • створення звітів низького рівня; • інформаційні панелі в реальному часі.	• покращення продуктивності та комунікації команди; • швидке встановлення ПЗ; • розширений API для підключення PractiTest до решти процесів та інструментів; • централізоване управління тестуванням; • історія учасників тестування;	• немає можливості створення групи; • користувачів; • малий функціонал модуля Звітність; • немає можливості копіювання та вставляння пройдених кроків тестування; • немає версії з відкритим кодом.

Отже, порівнявши функції, переваги та недоліки вищевказаних систем тест-менеджменту можна зробити висновки, що для управління процесом тестування слід обирати PractiTest та TestRail, через великий набір функцій та перелік переваг у використанні.

2.2 Порядок організації взаємодії з trainee на проекті

2.2.1 Взаємодія trainee з менторами та іншими учасниками проекту

Трейні, як правило, працює під керівництвом ментора, займаючись різними аспектами проектних ініціатив за його відсутності. Ці молодші співробітники відділу управління проектами часто проходять підготовку для роботи на більш

високих посадах керівників проектів. Багато з їхніх посадових обов'язків та відповідальності є однаковими, але вони можуть мати нижчий загальний рівень відповідальності, оскільки їхні наставники часто беруть участь у спостереженні або моніторингу проектів на вищому рівні.

Одним з важливих компонентом того, що роблять багато фахівців-стажерів під час навчання, щоб стати повноцінними менеджерами проектів, є нагляд. Ці особи можуть працювати як зі співробітниками, так і з зовнішніми підрядниками компанії, індивідуально або в групах. Вони можуть брати участь у вирішенні питань, пов'язаних із заробітною платою або дотриманням політики компанії, або ж вони можуть бути залучені до моніторингу робочих груп на відстані. Все це спостереження за робочою силою та робота з персоналом є важливою частиною того, що багато менеджерів проектів та їх стажерів надають компанії.

У процесі своєї роботи багатьом трейні на посади керівників проектів може знадобитися ознайомитися з низкою програмних додатків, які регулярно використовуються в компанії. Це може включати багато форм програмного забезпечення для підтримки прийняття рішень. Програмне забезпечення для підтримки прийняття рішень допомагає людям приймати тактичні рішення для роботодавця. Оскільки однією з основних частин роботи керівника проекту є прийняття таких рішень високого рівня, компанії можуть попросити цих осіб використовувати та звикнути до певних програмних додатків. Важливо зазначити, що програмне забезпечення для підтримки прийняття рішень − це не єдиний вид IT-ресурсів, які можуть використовуватися такими фахівцями для управління проектами; також може бути задіяне програмне забезпечення для презентацій або інші інструменти.

Трейні проекту може бути зв'язковим або посередником між різними відділами компанії. Ця людина може проводити час на зустрічах на високому рівні, щоб визначити, як рухатися далі з проектом. Він також може складати докладні звіти для керівництва вищого рівня, щоб тримати топ-менеджерів в курсі того, що відбувається на різних етапах бізнес-проекту. Стажеру з управління проектами,

можливо, доведеться подолати значну криву навчання, що включає в себе встановлення та відстеження детальних термінів для окремих фаз складних проектів.

Метою відбору трейні є визначення кандидатів, які можуть добре проявити себе як стажери, так і в подальшому як викладачі. Стажери проектів забезпечують виконання завдань від початку до кінця, допомагаючи у плануванні та виконанні. Вони часто працюють над великими, складними завданнями, допомагаючи більш досвідченим керівникам проектів, або тісно співпрацюють з іншим членом команди, який може надати рекомендації, коли це необхідно.

Trainee проекту має багато обов'язків, але вони, як правило, зосереджені на наданні допомоги в організації та реалізації проектів. Стажер допомагатимете координувати завдання, результати та аналіз даних, а також виконувати адміністративні обов'язки, такі як організація зустрічей або складання рахунків-фактур, щоб гарантувати, що все йде за планом.

Трейні проекту повинен володіти відмінними комунікативними навичками, оскільки він працює з декількома командами та відділами протягом усього проекту. Вони також повинні мати хороші організаторські здібності, щоб керувати кількома проектами одночасно і забезпечувати виконання всіх аспектів проекту вчасно і в рамках бюджету. Звісно, буде наявна робота з різними відділами, щоб забезпечити своєчасне завершення проекту, і, як правило, під наглядом керівника проекту.

Для роботи на проекті трейні має виконувати такі обов’язки:

- виконання всіх завдань, поставлених керівником, та надання допомоги, де це можливо;
- дотримання існуючих стратегій і методів та внесення пропозицій щодо їх вдосконалення;
- проведення досліджень і зіставлення даних.
- поїздки на різні об'єкти та отримання практичного досвіду в нових робочих зонах;

• тісна співпраця з персоналом з метою формування професійних цінностей та побудови добрих стосунків;

• постійне дотримання правил охорони праці та техніки безпеки;

• відвідування нарад та семінарів.

• проходження всіх форм оцінювання під час стажування;

• складання звітів та проведення презентацій для співробітників та інших зацікавлених сторін.

Для початку роботи на проекті trainee також має відповідати певним вимогам:

• відповідна вища технічна освіта;

• знання математики;

• навички аналітичного та критичного мислення;

• хороші комунікативні навички;

• вміння працювати в команді;

• хороші навички спостережливості та готовність до навчання.

2.2.2 Визначення рівня компетенцій трейні з метою коригування шляху його навчання

Тестувальники програмного забезпечення відіграють важливу роль у відділі забезпечення якості різних компаній та підприємств. Для опанування кар'єри тестувальника ПЗ, потрібно розвивати аналітичні навички та знання комп'ютерних систем, щоб гарантувати, що програми працюють ефективно і будуть відповідати очікуванням клієнтів.

Відповідно до бази знань, що складається з теоретичного та практичного матеріалу курсу «Тестування ПЗ», з якою приходить кандидат для працевлаштування, виділяють відповідні рівні знань (опанування навчального матеріалу). Досягнуті рівні знань теоретичного та практичного матеріалу дають змогу визначити, який варіант курсу потрібно опанувати кандидату.

Набори знань трейні:

1. Початковий рівень. Характеризується такими стеками знань:

- теорія тестування;
- знання поняття «баг»;
- різниця між QA та QC;
- поняття валідації та верифікації;
- типи тестування;
- рівні тестування.

2. Середній рівень. Кандидати, що опанували такі теоретичні та практичні навички:

- етапи життєвого циклу ПЗ;
- типи, види, методи тестування та застосування їх на практиці;
- знання баг-трекінгових систем;
- знання web-технологій;
- базові знання SQL, ООП;
- розуміння Agile / SCRUM методологій.

3. Високий рівень. Стажери, що опанували такий набір знань:

- досвід роботи з баг-трекінговими системами (Jira / YouTrack);
- глибокі знання з Server response codes, cookie & session, JSON, HTML, DOM, HTTP;
- досвід ведення тестової документації;
- базові знання певних мов програмування;
- досвід роботи з Agile / SCRUM методологіями.

Відповідно до рівня опанування навчального матеріалу кандидата йому буде призначено ментора з відповідними навичками. Після призначення керівника, менторові необхідно проаналізувати наявні знання трейні та відкоригувати їх до типу проекту до якого увійде трейні. Для цього було розроблено набори тем за напрямом Тестування ПЗ, що має знати стажер.

Варіанти-набори складаються з інформації про: основні обов'язки тестувальника та перелік тем для вивчення, що можна використати для початку кар'єри тестувальника програмного забезпечення. Буде представлено 5 варіантів.

Варіант 1. Базові аспекти тестування. (Матеріал, яким має володіти кожен трейні, що прийшов у компанію).

- Урок 1. Вступ. У вступі буде розміщена інформація про структуру курсу, а саме:

1. Дефекти, способи відслідковування.
2. Тест дизайн. Тестові випадки.
3. Процес розробки.
4. Типи тестування.
5. Основи тестування.

Також детально описана структура необхідних знань, які потрібно отримати для освоєння відповідної професії.

- Урок 2. Якість продукту та теорія тестування. Заняття, що містить відомості про:

1. Поняття якості продукту та її його визначення.
2. Визначення відмінностей між Quality Assurance та Quality Control.
3. Поняття «Тестування» та його характеристика.

- Урок 3. Принципи тестування. На цьому уроці буде розглянуто:

1. Опис принципів тестування.
2. Поняття верифікації у тестуванні.
3. Валідація.
4. Порівняння понять «Верифікація» та «Валідація» на прикладах.

- Підсумковий тест. Підсумкова робота має 7 тестових завдань, що побудовані на основі пройденого теоретичного матеріалу, пройшовши який можна отримати 14 балів.

• Додаткові ресурси та матеріали. Модуль, що містить посилання на додаткову літературу та інформацію, що містить більш детальний розбір пройдених тем.

Варіант 2. Процес тестування та розробки ПЗ.

• Урок 1. Розробка ПЗ, моделі. SDLC. Урок містить відомості про:

1. Опис життєвого циклу розробки програмного забезпечення.
2. Характеристика інкрементальної моделі.
3. Опис поняття «Ітеративна модель».

• Урок 2. Цикл розробки ПЗ, тестування. Матеріали уроку описують:

1. Теоретичні відомості про поняття «методологія Waterfall, Scrum, Kanban».
2. Опис підходу в розробці ПЗ Agile.
3. Основні завдання контролю та планування процесу розробки;
4. Основні завдання дизайну та аналізу процесу розробки;
5. Артефакти виконання та впровадження процесу розробки ПЗ.
6. Характеристика аспекту «Оцінка результатів».
7. Інформацію про етап «Завершення тестування».

• Підсумковий тест. Підсумкова робота має 7 тестових завдань, що побудовані на основі пройденого теоретичного матеріалу, пройшовши який можна отримати 14 балів.

• Додаткові ресурси та матеріали. Модуль, що містить посилання на додаткову літературу та інформацію, що містить більш детальний розбір пройдених тем.

Варіант 3. Аналіз вимог. Типи тестування.

• Урок 1. Опис типів тестування. Включає в себе наступні питання:

1. Які і снують типи тестування.
2. Функціональне тестування та його основні задачі.
3. Відомості про нефункціональне тестування.
4. Що таке структурне тестування.
5. Приклади структурного тестування.

6. Підтверджуюче тестування.
7. Regression testing.
8. Короткий перелік типів документів.
9. Що таке Software requirement specification.
10. Use case та його складові.
11. Ключові елементи та приклади Use case.
12. Подання Use story.
13. Нефункціональні вимоги.
14. Інформація про функціональні вимоги.
15. Як складати checklist для вимог.

• Підсумковий тест. Підсумкова робота має 7 тестових завдань, що побудовані на основі пройденого теоретичного матеріалу, пройшовши який можна отримати 14 балів.

• Додаткові ресурси та матеріали. Модуль, що містить посилання на додаткову літературу та інформацію, що містить більш детальний розбір пройдених тем.

Варіант 4. Тест-дизайн, техніки. Тестові випадки.

• Урок 1. Техніки тест-дизайну. На уроці буде висвітлено матеріал про:

1. Визначення поняття «Тест-дизайн».
2. Технічні характеристики тест-дизайну.
3. Короткий перелік технік тест-дизайну.

• Урок 2. Опис технік тест-дизайну. Урок містить відомості про:

1. Динамічна та статична техніка тест-дизайну.
2. Типи динамічного тест-дизайну.
3. Як працює розбиття за еквівалентністю.
4. Що таке аналіз граничних значень та його приклади.
5. Техніка тест-дизайн «Перехідний стан», його модель.

• Урок 3. Тестові випадки. Теоретичний матеріал заняття містить:є

1. Перелік полів тест-кейсу.

2. Наповнення поля Title.

3. Що потрібно записувати до поля Description.

4. Як описувати кроки виконання тесту.

5. Що таке очікуваний результат.

6. Інформація, яку мають містити такі поля, як: Author, Initial data, Module, submodule, Related requirement, Status, Estimate TTR, Priority.

- Підсумковий тест. Підсумкова робота має 7 тестових завдань, що побудовані на основі пройденого теоретичного матеріалу, пройшовши який можна отримати 14 балів.

- Додаткові ресурси та матеріали. Модуль, що містить посилання на додаткову літературу та інформацію, що містить більш детальний розбір пройдених тем.

Варіант 5. Відслідковування та пошук дефектів.

- Урок 1. Поняття про дефекти та робота з ними. На цьому уроці будуть розглянуті такі аспекти:

1. Типи помилок, що можна зустріти.

2. Виконавці баг-репортів.

3. Призначення баг-репорту.

4. Поля баг-репорту.

5. Поняття «Серйозність» та «Пріоритет».

6. Види Severity.

7. Види Priority.

8. Основні правила для визначення пріоритетності.

9. Перелік інструментів для роботи по виявленню помилок.

- Урок 2. Підсумок курсу. На цьому занятті буде висвітлено весь теоретичний матеріал, що було подано протягом усього курсу.

- Підсумковий тест. Підсумкова робота має 7 тестових завдань, що побудовані на основі пройденого теоретичного матеріалу, пройшовши який можна отримати 14 балів.

- Додаткові ресурси та матеріали. Модуль, що містить посилання на додаткову літературу та інформацію, що містить більш детальний розбір пройдених тем.

Після аналізу наявних знань, умінь та навичок, ментор може підібрати проект для трейні, що відповідають рівню його компетентності.

2.3 Розробка моделей для представлення даних trainee, ментора та проекту

Початківці QA (trainee/трейні, стажери, менті тощо) мають досить малу базу знань у відповідній сфері. Ментор дивиться на ситуацію «зверху», залучає підопічного до роботи і мотивує рухатися вперед особистим прикладом. Він не дає точних рекомендацій і не розв'язує проблеми за менті, а створює умови, в яких підопічний досягне успіху.

При підборі ментора для trainee потрібно акцентувати увагу на різних факторах.

1. Аналіз досвіду, навичок, знань, а також сильних та слабких сторін трейні.
2. Визначення мети, проблем та очікувань від процесу навчання.
3. Визначення мети і порядку співпраці з трейні.

Для визначення рівня досвіду, знань умінь та навичок стажера, насамперед потрібно провести інтерв’ю, де trainee необхідно розповісти про:

- виконані навчальні або особисті проекти;
- базові soft та hard skills;
- досвід роботи у відповідній сфері;
- курси та/або освіта, протягом яких трейні отримав базові знання.

Відповідно до рівня підготовки стажера йому буде призначений наставник, що допоможе розвивати уже наявні здібності trainee та давати поради у роботі з різними проектами.

Виділяють три типи менторів відповідно до знань стажера:

1. Ментори, які представляють спільні ключові навички.

2. Ментори, які навчають спеціальним навичкам.
3. Ментори, які поєднують у навчанні спільні ключові та спеціальні навички.

Для співставлення наявних даних трейні з спеціалізацією ментора та характеристиками проекту було складено відповідні моделі представлення даних, що описуються виразами (2.1), (2.2) та (2.3).

Модель трейні представляється виразом (2.1):

$$Trainee = \{Age, Spec, Edu, Courses, Lang, CV, API\} \quad (2.1),$$

де Age – вік стажера; Spec – спеціальність, яку стажер отримав під час навчання в закладі освіти; Edu – рівень освіти, який отримав трейні; Courses – множина курсів, які закінчив трейні додатково до основної освіти, та ступінь відповідності/корисності зазначених курсів для процесу тестування IT-проектів; Lang – множина мов програмування, якими володіє трейні, та рівні володіння; CV – множина інструментів контроля версій, якими володіє трейні, та рівні володіння; API – множина інструментів для тестування API, якими володіє трейні та рівні володіння.

Кожна спеціальність, яку отримав трейні в процесі здобуття освіти, співвіднесена до певної групи, яка має той чи інший рівень значущості з точки зору процесу тестування IT-проекту та описується множиною Spec = {(<назва групи галузей>,<рівень>)}.

До Групи 1 було віднесено спеціальності освітньої галузі. Освітня галузь є важливою складовою для роботі на IT-проекті, так як при прийомі на роботу важливо аби кандидат був обізнаний у сфері інформаційних технологій:

До Групи 2 входять спеціальності математичної галузі. Фахівець, який має аналітичне мислення, є великою перевагою для команди, що виконує проект, адже такі люди можуть зіставити факти, проаналізувати ситуацію та зробити відповідні висновки, що дають змогу знайти рішення певної проблеми.

До Групи 3 належать спеціальності галузі інформаційні технології. Під час роботи на проекті кандидат має бути ознайомлений з різними аспектами ІКТ, так як напрям завдань, що будуть на проекті напряму пов’язані з цією галуззю.

До Групи 4 входять фахівці, які ознайомлені з галуззю бізнес, управління і право. Якщо кандидат добре ознайомлений з вищевказаною галуззю, то його знання, уміння та навички можна використати для посади Delivery Manager, що відповідає за процес перемовин з замовником для вирішення проблем, що виникли під час виконання проекту.

Група 5 відповідає галузі соціальних наук, журналістиці та інформації. Фахівець, що прийматиме участь у проекті, повинен бути ознайомлений з такою галуззю, адже актуальні відомості, наприклад, про засоби та інструменти виконання поставлених завдань можуть скоротити час його виконання.

Рівень спеціальності може бути визначено на основі середнього балу диплому трейні та значущості групи спеціальності з точки зору проекту та галузі ІТ. Множина вибрана для представлення цього даного, оскільки трейні може мати декілька освітніх спеціальностей.

Рівень освіти трейні представляється значенням Edu, яке визначається по найвищому рівню освіти. Кожному рівню освіти відповідає свій коефіцієнт: освіті рівня «спеціаліст»/«магістр» і вище відповідає значення 1; для освіти рівня «бакалавр» визначено коефіцієнт 0,85; для освіти рівня «молодший спеціаліст»/«фаховий бакалавр» встановлено коефіцієнт 0,7; для шкільної освіти встановлено коефіцієнт 0,5.

Показник неформальної та інформальної освіти трейні визначається множиною Courses = {(<назва групи курсів>,<рівень>)}. Оскільки всю множину конкретних курсів, які проходили трейні, немає сенсу представляти окремими іменованими списками, було виділено три групи курсів:

• предметні курси – курси, що направлені конкретно по предмету вивчення;

• міжпредметні курси – розділені курси, тобто ті, що включають у себе декілька напрямків;

• надпредметні – курси, що включають у себе великий об’єм інших курсів, що відносяться до різних напрямів.

По кожній групі курсів рівень визначається середньою оцінкою трейні по курсам та цінністю групи, до якої групи вони відносяться. Найбільшу цінність з точки зору навичок трейні становлять надпредметні курси, оскільки вони дозволяють тестувальнику більш глибоко бачити можливі проблеми та помилки на проекті. Найменшу цінність мають конкретні предметні курси. Слід зазначити, що враховуються тільки ті курси, що корелюються з процесами, які мають відношення до тестування.

Для представлення навичок програмування використовується показник Lang = {(<назва мовної групи>, <рівень>)}. Параметр «рівень» визначає ступінь володіння трейні даною мовою програмування. Обрано наступні градації: 1 – професійне володіння мовою, вміння складати програми з використанням сучасних засобів мови, легке розуміння чужого коду, участь в реальних проектах в якості розробника; 0,8 – високий рівень володіння мовою, вміння складати програми з використанням засобів мови, але можливі прогалини в знанні сучасних стандартів та бібліотек мови, розуміння чужого коду на рівні, достатньому для визначення загального призначення модуля, участь в навчальних чи тренувальних проектах в рамках навчання в закладі освіти, під час практики чи проходження курсів; 0,5 – середній рівень володіння мовою, вміння складати програми з використанням базових засобів мови, розуміння чужого коду на рівні окремих операторів, але без чіткого уявлення, що саме виконує програма, можливо участь в навчальних чи тренувальних проектах в рамках навчання в закладі освіти, під час практики чи проходження курсів; 0,3 – низький рівень володіння мовою, знання окремих базових конструкцій без здатності сформувати з них працюючий модуль певного призначення; 0 – повне незнання мови.

Для представлення навичок володіння інструментами контроля версій використовується показник CV = {(<назва інструменту контроля версій>,<рівень>)}. Параметр «рівень» визначає ступінь володіння трейні даним

інструментом. Обрано наступні градації: 1 – професійне володіння інструментом, наявність досвіду використання на реальних проектах; 0,8 – високий рівень володіння інструментом, знання особливостей використання інструменту, досвід активного використання інструменту в навчальних чи тренувальних проектах в рамках навчання в закладі освіти, під час практики чи проходження курсів; 0,5 – середній рівень володіння інструментом, наявність базових навичок використання інструменту, є невеликий досвід використання інструменту в навчальних чи тренувальних проектах в рамках навчання в закладі освіти, під час практики чи проходження курсів; 0 – низький рівень володіння інструментом або повна відсутність знань про інструмент та навичок його використання.

Для представлення навичок володіння інструментами для тестування API використовується показник API = {(<назва інструменту для тестування API>,<рівень>)}. Параметр «рівень» визначає ступінь володіння трейні даним інструментом. Обрано градації, аналогічні рівню володіння інструментами контроля версій з тими ж характеристиками визначення рівня: 1 – професійне володіння інструментом, 0,8 – високий рівень володіння інструментом, 0,5 – середній рівень володіння інструментом, 0 – низький рівень володіння інструментом або повна відсутність знань про інструмент та навичок його використання

Модель ментора представляється виразом (2.2):

$$Mentor = \{Age, Lang, CV, API, Met\}, \quad (2.2)$$

де Age – вік ментора; Lang – множина мов програмування, якими володіє ментор, та рівні володіння; CV – множина інструментів контроля версій, якими володіє ментор, та рівні володіння; API – множина інструментів для тестування API, якими володіє ментор та рівні володіння, Met – методології, якими володіє ментор.

Показники Lang, CV та API визначаються аналогічно трейні. Характеристика Met представляється трійкою Met = (<scrum>, <kanban>, <waterfall>), де кожна

складова дорівнює 0 або 1, а залежності від того, чи має ментор досвід використання вказаної методології.

Модель проекту представляється виразом (2.3):

$$Project = \{Lang, CV, API, Met\}, \quad (2.3)$$

де Lang – назва мовної групи, яка використовується на проекті; CV – назва інструменту контроля версій, яка використовується на проекті; API – назва інструменту для тестування API, який використовується на проекті; Met – назва методології, яка використовується на проекті.

2.4 Розробка правил продукційної моделі системи тест-менеджменту

2.4.1 Загальні вимоги до представлення знань засобами продукційної моделі

Продукційні правила, також відомі як «ЯКЩО-ТО», є однією з найпоширеніших форм представлення знань, особливо в експертних системах (ЕС) та системах автоматизованої підтримки прийняття рішень (САПР). Вони характеризуються високою модульністю, що вносить гнучкість у процес створення та впровадження знань. Кожне правило визначає відносно невелику і незалежну частину знань; крім того, воно може бути легко додане і видалене незалежно від інших правил. Простота синтаксису та природний спосіб вираження знань дає змогу легко інтерпретувати знання. Типове продукційне правило складається з антецедента і консеквента. Антецедент відповідає передумові (умова, IF-речення), а наслідок − дії (висновок, THEN-речення). Для того, щоб увімкнути механізм міркування, правила зазвичай групуються в так звану продукційну систему, яка має свої структурно-функціональні особливості [23].

Продукційна система складається з бази знань та механізму виведення, результатом чого є два типи правил: декларативні та процедурні. Декларативні

правила констатують всі факти та зв'язки щодо проблеми, в той час як процедурні правила радять, як вирішити проблему, враховуючи певні факти. Також вони відрізняються місцем зберігання: перші зберігаються в КБ, а другі входять до механізму виведення. База знань продукційної системи має свою специфіку і поділяється на робочу пам'ять і перелік правил. До робочої пам'яті входять набори фактів, що характеризують аспекти про світ, але також можуть представляти різні структури даних. Представлення фактів і синтаксис можуть відрізнятися в різних системах так як не є строго формалізованими. База правил – це сукупність правил, які відносяться до особливого типу правил «ЯКЩО-ТО», де антецедент представлений кон'юнкцією умов, а консеквент − послідовністю дій. Структура випадкового запису в базі правил визначається за формулою: $p_1 \wedge p_2 \wedge \dots p_n \rightarrow a_1, a_2, \dots a_{1k}$. Антецеденти можуть мати форму заперечних або незаперечних предикативних висловлювань та умов щодо певного об'єкта або кількох інших. Консеквент зазвичай складається з наступних дій:

- «ДОДАТИ» факт в робочу пам'ять;
- «ВИДАЛИТИ» факт з робочої пам'яті;
- «МОДИФІКУВАТИ» атрибутивне поле;
- «ЗАПИТ» користувача на введення, якщо зобразити це у вигляді формули, то $A(x)\wedge \mathrm{B(x)}\wedge \mathrm{C(y)} \rightarrow \mathrm{REMOVE: D(x), ADD: B(y)}$.

Механізм виведення у виробничих системах, як правило, реалізується за допомогою методу прямого ланцюжка. Він починається з фактів, доступних в робочій пам'яті, і використовує набір «активних» правил з БЗ для вилучення нових, до тих пір, поки не буде досягнута мета. Правило вважається активним, якщо його антецедент виконаний; це є необхідною умовою для виконання консеквенту. Поширеною є ситуація, коли активними є декілька правил одночасно, що призводить до необхідності вказати порядок їх виконання. Насправді, порядок виконання має важливе значення, оскільки дії ADD і REMOVE можуть змінити набір активних правил так, що наступне активне правило яке в черзі може бути дезактивоване попереднім. Стратегія вибору правила для виконання з числа

можливих кандидатів називається вирішенням конфлікту. Нижче перераховані деякі з найбільш популярних стратегій вирішення конфліктів:

- відсутність дублювання − не виконувати одне і те ж правило двічі;
- новизна − пріоритет мають правила, що стосуються фактів, які нещодавно були додані до робочої пам'яті;
- специфічність − правила, які є більш специфічними, мають перевагу над більш загальними;
- рівні пріоритету − присвоєння правилам рівнів пріоритету.

Ще один не менш важливий аспект, який потрібно враховувати при огляді виробничих систем, є процес уніфікації. Адже є правила, що можуть містити великий перелік уніфікацій, особливо якщо до них входять змінні, що тестують випадки, коли правила активні. Крім того, умови багатьох правил можуть перетинатися, що призводить до багаторазового повторення одних і тих самих уніфікацій. Для вирішення проблем уніфікацій, що не є активними та конфліктних ситуацій використовується алгоритм Rete [5]. Ключовими принципами його функціонування є наступне:

- правила об'єднуються в мережу, яка об'єднує умови декількох правил між собою, що дозволяє уникнути дублювань;
- поширюються лише дійсні уніфікації;
- переоцінюються лише змінені умови [5].

Отже, правила продукційної моделі забезпечують велику модульність і гнучкість задачі представлення знань. Вони чіткі, легкі для розуміння і підтримуються потужним механізмом виведення доступним в продукційних системах. Крім того, їх корисність вже доведена успіхом величезної кількості комерційних експертних систем. Говорячи про недоліки, слід зазначити, що не всі знання можуть бути виражені у вигляді правил. Продукційні правила погано представляють структуровані знання, і в них важко дотримуватися ієрархії. Великі системи зазвичай складаються з тисячі правил, що ускладнює консолідацію знань і значно знижує продуктивність механізму виведення.

Продукційна модель складається з двох взаємодіючих структур даних, пов'язаних через простий цикл обробки:

1. Робоча пам'ять, що складається з набору символів даних, які називаються елементами робочої пам'яті.

2. Продукційна пам'ять, що складається з правил «умова-дія», які називаються продукціями, умови яких описують конфігурації елементів, що можуть з'явитися в робочій пам'яті, а дії визначають модифікації елементів робочої пам'яті.

3. Виробнича і робоча пам'ять пов'язані між собою через цикл «розпізнавання - дія». Він складається з трьох окремих етапів:

- процес зіставлення, який знаходить постановки, умови яких збігаються з поточним станом робочої пам'яті; одне і те ж правило може працювати у пам’яті по-різному, і кожне таке відображення називається конкретизацією;
- процес розв'язання конфлікту, який вибирає одну або більше з інстанційованих постановок для застосування;
- процес виконання, який застосовує інстанційовані дії вибраних правил, змінюючи таким чином вміст робочої пам'яті.

Базовий процес розпізнавання дії працює в циклах, при цьому обираються одне або декілька правил і застосовуються в них, новий вміст пам'яті призводить до нового набору правил, що застосовуються, і так далі. Цей цикл продовжується до тих пір, поки кожне правило не отримає команди зупинки. Очевидно, що цей варіант ігнорує багато деталей, а також багато варіацій, які можливі в межах базової структури, але він передає основну ідею продукційної моделі.

2.4.2 Визначення критеріїв ефективного розподілу менторів та trainee на проекти

Багато фірм, що надають послуги у сфері інформаційних технологій, щороку наймають велику кількість студентів. Ці нові працівники проходять програму навчання, щоб відточити свої навички в конкретних сферах. Паралельно менеджер

з управління персоналом збирає від керівників проектів інформацію про вимоги до проекту з розробки програмного забезпечення, такі як необхідні навички програмування, тип проекту та місце розташування. Наприкінці навчальної програми стажери надають свої побажання щодо місця розташування проекту та інтереси щодо конкретних вимог до проекту. Кожен учасник має унікальний погляд на ці вимоги проекту. Наприклад, деякі учасники можуть бути зацікавлені в проектах з нішевими технологіями, в той час як деякі учасники можуть віддати перевагу проектам, які мають робочі місця поблизу їхніх рідних міст.

Використовуючи ці дані, зібрані від стажерів та керівників проектів, менеджери з персоналу повинні розподілити стажерів за вимогами проектів з розробки програмного забезпечення. У минулому менеджери з персоналу в основному використовували жадібний підхід для такого розподілу на основі своїх знань, сприйняття та досвіду. При жадібному підході, якого дотримується одна фірма, стажери сортуються відповідно до їх навчальних досягнень, а потім кожен стажер призначається на проект, що відповідає його/її бажаному місцезнаходженню на основі проектних вимог, де є відповідні вимоги до навичок. Якщо відповідність не знайдена, стажер довільно призначається на вимогу проекту з розробки програмного забезпечення. Жадібний підхід, якщо він здійснюється вручну, займає багато часу і може не дати найкращого рішення. Нещодавно деякі фірми почали використовувати модель розподілу витрат на основі лінійного програмування. Витрати на перепідготовку та передислокацію використовуються менеджерами з персоналу для представлення витрат на розподіл наступним чином: коли стажери не розподіляються за місцем їх першого вибору, фірма оплачує витрати на логістику та проживання стажера. Крім того, якщо технологічні навички стажера не відповідають вимогам проекту, фірма несе витрати на перепідготовку залежно від тривалості навчання. Їх модель намагається мінімізувати загальні витрати на виконання завдання.

Обидва ці підходи мають ряд недоліків. Ефективний розподіл передбачає врахування індивідуальних уподобань слухачів, а також вимог програмного

проекту на додаток до вартості. При жадібному підході слухачі нижчого порядку довільно призначаються на будь-яку вимогу проекту, якщо їхні вподобання не збігаються. Хоча модель розподілу на основі вартості є покращенням порівняно з жадібною евристикою, вона має деякі обмеження. Наприклад, важливість надається мінімізації витрат на розподіл, а не вподобанням стажерів. Якщо такі стажери не отримують можливості вибору, вони можуть стати незадоволеними і покинути організацію. Це є основною проблемою для фірм, що надають IT-послуги, оскільки вони стикаються з високим рівнем плинності кадрів. Таким чином, як жадібна модель, так і модель призначення не враховують вподобання стажерів у ефективний спосіб. Аналогічно, якість або терміни виконання програмного проекту можуть значно постраждати через невідповідність працівника вимогам проекту. Зрештою, фірма може понести вищі загальні витрати.

Обидва ці підходи призводять до нестабільних пар (Firat, Hurkens, & Laugier, 2012). Іншими словами, можна знайти пару *(e, r)* «стажер - вимоги проекту», де стажер *e* віддає перевагу вимогам *r* перед призначеними вимогами, а вимога проекту *r* вважає стажера *e* більш підходящим порівняно з існуючим призначеним стажером, пара *(e, r)* називається нестабільною парою. Інтуїтивно зрозуміло, що зі збільшенням кількості таких нестабільних пар зростає і незадоволеність серед стажерів та керівників проектів. Очевидно, що мінімізація таких нестійких пар може підвищити задоволеність принаймні одного учасника, не впливаючи на задоволеність іншого учасника, оскільки обидва можуть отримати краще або, принаймні, однакове завдання.

2.4.3 Вибір методів оцінки параметрів моделі

Оскільки значна кількість параметрів об’єктів системи представлено у вигляді якісних показників, то їх необхідно привести до числового вигляду. Для цього часто використовуються методи експертних оцінок. Такі методи передбачають організацію роботи з фахівцями-експертами та обробку думок

експертів, виражених у кількісній та/або якісній формі, з метою підготовки інформації для прийняття рішень особами, які приймають рішення.

Дослідженню можливостей та особливостей застосування експертних оцінок присвячено багато робіт. У них розглядаються форми експертного опитування (різні види анкетування, інтерв'ю), підходи до оцінювання (ранжування, нормалізація, різні види упорядкування тощо), методи обробки результатів опитування, вимоги до експертів і формування експертних груп, питання підготовки експертів, оцінки їх компетентності (при обробці оцінок вводяться і враховуються коефіцієнти компетентності експертів і достовірності їх думок), методи організації експертних опитувань. Вибір форм і методів проведення експертних опитувань, підходів до обробки результатів опитування тощо залежить від конкретного завдання та умов проведення експертизи.

Експертні методи нині використовуються в ситуаціях, коли вибір, обґрунтування та оцінка наслідків рішень не можуть бути виконані на основі точних розрахунків.

Можливість використання експертних оцінок, обґрунтування їх об'єктивності зазвичай ґрунтується на тому, що невідома характеристика досліджуваного явища інтерпретується як випадкова величина, відображенням закону розподілу якої є індивідуальна оцінка фахівця-експерта про достовірність і значущість події. Передбачається, що істинне значення досліджуваної характеристики знаходиться в межах діапазону оцінок, отриманих від групи експертів, і що узагальнена колективна думка є достовірною.

Метод експертних оцінок реалізується шляхом обробки думок досвідчених фахівців про можливі величини втрат та (або) ймовірності їх настання і використовується в неформалізованих проблемних ситуаціях, коли відсутність достатнього масиву інформації або її недостовірність не дозволяє використовувати формально-математичні методи в чистому вигляді. Цей метод базується на використанні інтуїції, минулого досвіду, аналогії та логіки. Процедури методу експертних оцінок ґрунтуються на використанні людини для отримання кількісних

оцінок якісних суджень, які не піддаються безпосередньому вимірюванню. При цьому експерти проводять інтуїтивно-логічний аналіз досліджуваної ситуації з кількісними або порядковими оцінками процесів або явищ, після чого виконується формальна обробка результатів.

Для знаходження відносних ваг альтернативних рішень розглянемо 2 групи методів парних порівнянь, що базуються на основі експертних оцінок: методи типу «трикутник» та «лінія».

Вказані в [27] методи застосовують для того щоб обчислити відносний пріоритет альтернатив рішень за відповідним критерієм на підставі експертних оцінок. Основною ідеєю цього методу є те, що експерт порівнює деякі або всі пари альтернатив у відповідній шкалі порівняння.

Метод «трикутник». Підібрані альтернативи попарно порівнюють за шкалою відношень і в результаті експерт дає певну кількість оцінок (2.4):

$$\frac{n(n-1)}{2}, \tag{2.4}$$

де n - кількість альтернатив.

Ці оцінки записуються в матрицю парних порівнянь (МПП) (2.5):

$$D_{nxn} = \{d_{ij} | i, j=1,\ldots,n\} \tag{2.5}$$

де d_{ij}>0 (і-тий рядок, j-тий стовпчик). Для МПП справедлива властивість зворотної симетричності $d_{ij} = 1/d_{ij}$. МПП. Метод «трикутник» [27] обґрунтований тільки для допустимо неузгоджених МПП. Для того аби покращити аспект узгодженості МПП використовують підходи без участі експерта або зворотний зв'язок з експертом, який коригує МПП залежно від рівня її узгодженості. Оскільки кількість експертних оцінок, що дорівнює $\frac{n(n-1)}{2}$, є надлишковою, на практиці

часто застосовуються методи парних порівнянь типу «лінія», які зменшують навантаження на експерта.

У цих методах передбачається повна узгодженість знань експерта і тому від експерта вимагається тільки n-1 оцінок. У відповідних методах експерт порівнює всі альтернативи рішень з одним обраним об'єктом. Обираються провідні пари альтернатив, які порівнюються.

Вважається, що повна узгодженість (несуперечливість) оцінок − це ідеальний випадок, до якого слід прагнути експерту, виконуючи парні порівняння. Однак, якщо експерт дав узгоджені оцінки, то вони не обов'язково відображають справжні ваги порівнюваних альтернатив.

Метод експертних оцінок – це простий і найбільш часто використовуваний метод прийняття рішень за багатьма атрибутами [27]. Метод базується на середньозваженому значенні. Для кожної альтернативи розраховується оціночний бал шляхом множення значення шкали, безпосередньо призначеними особою, яка приймає рішення, з подальшим підсумовуванням отриманих добутків за всіма критеріями. Перевагою цього методу є те, що метод є пропорційно-лінійним для перетворення вихідних даних, що дає змогу відносний порядок величини залишити незмінним. Метод складається з таких етапів з 9 етапів і використовує шкалу парних порівнянь Сааті (табл. 2.2). Зазвичай використовують значення 1, 3, 5, 7, 9. Інші значення є проміжними і можуть бути використані для уточнення.

Таблиця 2.2 – Шкала парних порівнянь Сааті [27]

Важливість	**Визначення**	**Пояснення**
1	Рівноцінне значення	Два види діяльності/об’єктів в рівній мірі сприяють досягненню мети

2	Слабкі або незначні	
3	Помірні/важливі	Судження надає перевагу одному виду діяльності/об’єктів над іншим
4	Більш важливі	
5	Висока важливість	Судження на більшому рівні надає перевагу одному виду діяльності/об’єкту над іншим
6	Більш висока важливість	
7	Дуже важливі	Діяльність/об’єкт має дуже сильну перевагу над іншими
8	Понад важливі	
9	Максимально важливі	Докази на користь одного виду діяльності/об’єкту над іншою мають найвищий порядок підтвердження

1-й крок. Побудувати матрицю попарних порівнянь (n × n) для критеріїв по відношенню до мети за шкалою парних порівнянь Сааті 1-9. Використання цієї таблиці потрібне для порівняння кожного критерію з кожним іншим, і так далі.

2-й крок. Для кожного порівняння потрібно вирішити, який з двох критеріїв є найбільш важливим, далі присвоїти бал, щоб показати його важливість.

3-й крок. Обчислити кожен елемент матриці порівняння за його підсумком стовпців і обчислити вектор пріоритетів шляхом знаходження середніх значень по рядках.

4-й крок. Матриця зважених сум знаходиться шляхом перемноження матриці попарних порівнянь та вектора пріоритетів.

5-й крок. Ділення всіх елементів матриці зважених сум на відповідний елемент вектору пріоритетів.

6-й крок. Обчислити середнє арифметичне цих значень, щоб отримати максимальне значення.

7-й крок. Обрахувати Індекс узгодженості.

8-й крок. Розрахувати коефіцієнт наповнення.

9-й крок. Послідовність суджень можна перевірити, взявши індекс узгодженості та коефіцієнт наповнення. Коефіцієнт узгодженості є прийнятним, якщо він не перевищує 0.10. Якщо він більший, то матриця суджень є неузгодженою. Для отримання узгодженої матриці суджень необхідно переглянути її та вдосконалити.

2.4.4 Розробка правил для визначення сумісності менторів та trainee на проектах

Визначимо сумісність C(Trainee, Mentor, Project), як величину, що обчислюється на основі значень сумісності ментора та трейні C_{MT}, ментора та проекту C_{MP}, трейні та проекту C_{TP}, представленими в таблиці 2.3. Значення C_{MT}, C_{MP} та C_{TP} обчислюються на основі евристичних правил, які будуть описані далі.

Будемо вважати трійку (Trainee, Mentor, Project) стабільною, якщо вона має високий рівень сумісності C. Трійки (Trainee, Mentor, Project) з середнім рівнем сумісності C будемо вважати помірно стабільними, з низьким рівнем C – нестабільними. Задачу ефективного тест-менеджменту при розподілі менторів та трейні на проекти можна звести до формування набору стабільних трійок, які включають максимальну кількість поточних проектів. При неможливості формування набору для покриття всіх проектів стабільними трійками, задача зводиться до мінімізації кількості нестабільних та помірно стабільних трійок.

Таблиця 2.3 – Порядок визначення сумісності C(Trainee, Mentor, Project)

C_{MP}	C_{MT}	C_{TP}	C
високий	високий	високий	високий
високий	високий	середній	високий

високий	високий	низький	середній
середній	високий	високий	високий
середній	високий	середній	середній
середній	високий	низький	середній
низький	високий	високий	середній
низький	високий	середній	середній
низький	високий	низький	низький
високий	середній	високий	середній
високий	середній	середній	середній
високий	середній	низький	середній
середній	середній	високий	середній
середній	середній	середній	середній
середній	середній	низький	низький
низький	середній	високий	середній
низький	середній	середній	низький
низький	середній	низький	низький
високий	низький	високий	середній
високий	низький	середній	середній
високий	низький	низький	низький
середній	низький	високий	середній

Продовження таблиці 2.3 − Порядок визначення сумісності C(Trainee, Mentor, Project)

C_{MT}	C_{MP}	C_{TP}	C
низький	середній	середній	середній
низький	середній	низький	низький
низький	низький	високий	низький
низький	низький	середній	низький

низький	низький	низький	низький

Для представлення даних таблиці 2.3 будемо використовувати продукційну модель, де кожен рядок бази знань описується правилом виду (2.6):

ЯКЩО C_{MT}=<рівень сумісності>, p_{MT} І C_{MP}=<рівень сумісності>, p_{MP} І C_{TP}=<рівень сумісності>, p_{TP} ТО C=<рівень сумісності>, p_C (2.6)

Чисельна міра спільного рівня сумісності p_C визначається виразом (2.7):

$$p_C = (p_{MP} + p_{TP} + p_{MT})/3, \quad (2.7)$$

де p_{MP} визначає чисельну міру рівня сумісності ментора та проекту, $p_{MP} \in [0; 1]$; p_{TP} визначає чисельну міру рівня сумісності трейні та проекту, $p_{TP} \in [0; 1]$; p_{MT} визначає чисельну міру рівня сумісності ментора та трейні, $p_{MT} \in [0; 1]$.

Для обчислення p_{MP} використовуються показники моделі ментора Mentor = {Age, Lang, CV, API, Met} та показники моделі проекту Project= {Lang, CV, API, Met}.

Сумісність ментора з проектом за показником мови програмування $c(Lang_{Mentor}, Lang_{Project})$ будемо визначати наступним чином.

1. Якщо в стеку навичок мов програмування у ментора, що представлені множиною $Lang_{Mentor}$, зустрічається мова програмування, яка використовується на проекті $Lang_{Project}$, то сумісність будемо обчислювати, як (2.8):

$$c(Lang_{Mentor}, Lang_{Project}) = lM_i, \quad (2.8)$$

де і – номер мови програмування, яка співпадає в множині мов ментора з мовою програмування проекту, lM_i – відповідний рівень володіння ментором даною мовою.

2. Якщо в стеку навичок мов програмування у ментора, що представлені множиною $Lang_{Mentor}$, не зустрічається мова програмування, яка використовується на проекті $Lang_{Project}$, то сумісність можна обчислювати, проранжувавши мови програмування, якими володіє ментор, з використанням попарного порівняння за шкалою Сааті. При порівнянні експерт повинен враховувати мову проекту. Сумісність в такому випадку обчислюється як рівень володіння ментором мовою з максимальним рангом за результатами попарного порівняння помножений на ранг цієї мови lM_i. Приклад попарного порівняння мов програмування наведено в таблиці 2.3. Максимум в стовпчику нормованої суми визначить ранг шуканої мови.

Таблиця 2.4 – Приклад визначення рангів мов програмування ментора

	C++	**C#**	**Python**	**Java**	**PHP**	**Сума**	**Нормована сума (ранг)**
C++	1	9	7	5	3	25	1
C#	0,11	1	9	7	5	22,11	0,8844
Python	0,14	0,11	1	9	7	17,25	0,69
Java	0,2	0,14	0,11	1	9	10,45	0,418
PHP	0,33	0,2	0,14	0,11	1	1,78	0,0712

Сумісність ментора з проектом за показником системи контролю версій $c(CV_{Mentor}, CV_{Project})$ та сумісності інструменту для тестування API $c(API_{Mentor}, API_{Project})$ будемо використовувати підхід, аналогічний визначенню сумісності мов програмування.

Для обчислення сумісності методологій будемо використовувати вираз (2.9):

$$c\left(Met_{Mentor}, Met_{Project}\right) = \begin{cases} 0, \text{якщо } Met_{Project} \notin Met_{Mentor} \\ 1, \text{якщо } Met_{Project} \in Met_{Mentor} \end{cases}. \quad (2.9)$$

Результуюча величина сумісності p_{MP} обчислюється як середньозважене значення всіх складових сумісності ментора та проекту (2.10):

$$p_{MP} = (c\left(Lang_{Mentor}, Lang_{Project}\right) \cdot \alpha_1 + c\left(CV_{Mentor}, CV_{Project}\right) \cdot \alpha_2 + \\ + c\left(API_{Mentor}, API_{Project}\right) \cdot \alpha_3 + c\left(Met_{Mentor}, Met_{Project}\right) \cdot \alpha_4) / 4 \quad (2.10)$$

де $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ – вагові коефіцієнти кожної складової сумісності ментора та проекту, $\alpha_1 + \alpha_2, +\alpha_3 + \alpha_4 = 1$. Для подальших обчислень показник p_{MP} та інші показники сумісності пар нормуються до 1 по максимальному значенню сумісності.

Результуюче якісне значення рівня сумісності C_{MP} в термінах «високий», «середній», «низький» пропонується формувати на основі експертних оцінок, які наведено в таблиці 2.5. Було проведено опитування експертів на предмет визначення ними порогів рівней сумісності на основі деякого числового показника за 100 бальною шкалою. Для отримання порогу, на основі якого буде виділятись високий, середній чи низький рівень обчислюється середнє значення серед думок експертів, яке потім нормується до 1 шляхом поділу на 100.

Таблиця 2.5 – Оцінка експертів порогів значень C_{MP}

Рівень сумісності	Оцінка експертів порогів значень C_{MP}												Середнє нормоване порогове значення
	1	2	3	4	5	6	7	8	9	10	11	12	
Високий	85	75	87	72	82	90	70	75	72	89	68	82	0,79
Середній	37	39	46	60	58	37	56	59	57	37	52	35	0,48

Низький	28	12	28	29	24	23	19	22	18	26	17	24	0,23

Для обчислення p_{MT} використовуються показники моделі ментора Mentor = {Age, Lang, CV, API, Met} та показники моделі трейні Trainee = {Age, Spec, Edu, Courses, Lang, CV, API}. Важливими факторами сумісності ментора та трейні є:

- вікова сумісність – чим менше різниця в віці між тейні та ментором, тим вище ймовірність налагодження легкого контакту між співробітниками;
- сумісність мов програмування – якщо трейні та ментор мають схожість стеку мов програмування, наприклад, знаються на мовах одного сімейства, це підвищує швидкість навчання;
- сумісність інструментів контроля версій – якщо трейні та ментор мають схожість стеку систем контролю версій, це підвищує швидкість навчання;
- сумісність інструментів для тестування API – якщо трейні та ментор мають схожість стеку інструментів для тестування API, це підвищує швидкість навчання.

Додаткові показники, які можуть вплинути на сумісність ментора та трейні, визначаються початковими навичками трейні в галузі тестування, отриманими під час академічної освіти (показники трейні Spec – група спеціальностей та Edu – рівень освіти) та спеціалізованих курсів тестування (показник трейні Courses). При наявності спеціалізованої освіти у трейні ментору не доведеться витрачати час на пояснення ключової термінології та вивчення методів та технік тестування.

Для обчислення вікової сумісності $c(Age_{Trainee}, Age_{Mentor})$ будемо використовувати правила (2.11):

ЯКЩО $|Age_{Trainee} - Age_{Mentor}| \leq 5$ ТО вікова сумісність= висока;
ЯКЩО $|Age_{Trainee} - Age_{Mentor}| > 10$ ТО вікова сумісність= низька;
ЯКЩО $|Age_{Trainee} - Age_{Mentor}| > 5$ І $|Age_{Trainee} - Age_{Mentor}| \leq 5$
ТО вікова сумісність=середня. (2.11)

Для визначення сумісності мов програмування використовувати наступні правила:

1. Якщо множини мов програмування $Lang_{Trainee}$ та $Lang_{Mentor}$ мають спільні елементи, тобто $Lang_{Trainee} \cap Lang_{Mentor} \neq \emptyset$, то значення сумісності мов програмування ментора та трейні $c(Lang_{Trainee}, Lang_{Mentor})$ обчислюється як сума відношень між рівнем знань кожної мови трейні та ментора (2.12):

$$c(Lang_{Trainee}, Lang_{Mentor}) = \sum_{i=1}^{n} \frac{lT_i}{lM_i}, \quad (2.12)$$

де n – кількість мов програмування, що співпадають у ментора та трейні, lT_i – відповідний рівень знань і-тої мови програмування трейні, lM_i – відповідний рівень знань і-тої мови програмування ментора.

2. Якщо множини мов програмування $Lang_{Trainee}$ та $Lang_{Mentor}$ не мають жодного спільного елемента, тобто $Lang_{Trainee} \cap Lang_{Mentor} = \emptyset$, то спочатку визначається найважливіша мова ментора. Для цього можна використовувати ранжування методом попарного порівняння за шкалою Сааті, просте ранжування на основі думок експертів або вибрати мову з максимальним рівнем знань ментора $lM_{max} = \max_i(lM_i)$. Після цього блізкість кожної мови трейні до обраної мови ментора може бути обчислена на основі значень стовпчика таблиці Сааті, розрахованого для мови ментора. Максимальне значення в стовпчику дасть шукану мову програмування трейні, яка буде найбільш сумісною з мовою програмування ментора. Рівень сумісності визначається як відношення рівня знань трейні по найбільш сумісній мові до рівня знань ментора по найважливішій мові ментора.

Сумісність ментора та трейні за показником системи контролю версій $c(CV_{Mentor}, CV_{Trainee})$ та сумісності інструменту для тестування API $c(API_{Mentor}, API_{Trainee})$ будемо використовувати підхід, аналогічний визначенню сумісності мов програмування ментора та трейні.

Для розрахунку значень додаткових показників, які можуть вплинути на сумісність ментора та трейні, будемо використовувати ранжування параметрів на основі попарного порівняння за шкалою Сааті. Результати визначення рангів для показника спеціальності $Spec_{Trainee}$ наведено в таблиці 2.6, визначення рангів для курсів $Courses_{Trainee}$ – в таблиці 2.7. Якщо трейні має декілька освіт за різними спеціальностями, декілька закінчених курсів, то обирається максимальне значення з нормованих сум.

Таблиця 2.6 – Визначення рангу спеціальності $Spec_{Trainee}$

	Група 1	**Група 2**	**Група 3**	**Група 4**	**Група 5**	**Сума**	**Нормована сума (ранг)**
Група 1	1	9	7	5	3	25	1
Група 2	0,11	1	9	7	5	22,11	0,8844
Група 3	0,14	0,11	1	9	7	17,25	0,69
Група 4	0,2	0,14	0,11	1	9	10,45	0,418
Група 5	0,33	0,2	0,14	0,11	1	1,78	0,0712

Таблиця 2.7 – Визначення рангу курсів $Courses_{Trainee}$

	Предметні	**Між-предметні**	**Над-предметні**	**Сума**	**Нормована сума (ранг)**
Предметні	1	5	3	9	1
Міжпредметні	0.2	1	5	6,2	0,68
Надпредметні	0.33	0.2	1	1,52	0,16

Результуюча числова величина сумісності ментора та трейні p_{MT} обчислюється як середньозважене значення всіх складових сумісності (2.13):

$$p_{MT} = (c(Lang_{Mentor}, Lang_{Trainee}) \cdot \alpha_1 + c(CV_{Mentor}, CV_{Trainee}) \cdot \alpha_2 + \\ + c(API_{Mentor}, API_{Trainee}) \cdot \alpha_3 + Edu_{Trainee} \cdot \alpha_4 + \\ + Spec_{Trainee} \cdot r_{Spec} \cdot \alpha_5 + Courses_{Trainee} \cdot r_{Courses} \cdot \alpha_6)/\,6 \quad (2.13)$$

де· $\alpha_1, \dots, \alpha_6$ – вагові коефіцієнти кожної складової сумісності ментора та трейні, $\alpha_1 + \alpha_2, +\alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 = 1$; $Edu_{Trainee}$ – рівень освіти трейні; $Spec_{Trainee}$

– ранг найбільш вагомої спеціальності трейні, r_{Spec} – рівень знань по спеціальності, обчислений на основі середнього балу диплому, нормованого до 1; $Courses_{Trainee}$ – ранг найбільш вагомих курсів трейні, $r_{Courses}$ – рівень знань, отриманих на курсах, обчислений на основі балу сертифікату курсів, нормованого до 1. Якщо трейні не має освіти за певною спеціальністю та/або пройдених курсів, то відповідні показники становлять 0, але впливають на загальну суму. Таким чином, трейні, який має попередню академічну чи неформальну освіту буде мати перевагу перед трейні без жодної освіти.

Результуюче якісне значення рівня сумісності ментора та трейні C_{MT} в термінах «високий», «середній», «низький» сформовано аналогічно сумісності ментора та проекту на основі експертних оцінок та наведено в таблиці 2.8.

Таблиця 2.8 – Оцінка експертів порогів значень C_{MT}

Рівень сумісності	Оцінка експертів порогів значень C_{MT}												Середнє нормоване порогове значення
	1	2	3	4	5	6	7	8	9	10	11	12	
Високий	74	76	81	88	79	90	86	75	82	70	87	70	0,79
Середній	43	30	62	42	40	32	41	45	38	68	59	35	0,44
Низький	20	10	24	16	21	18	22	13	17	16	12	24	0,17

Для обчислення p_{TP} використовуються показники моделі трейні Trainee = {Age, Spec, Edu, Courses, Lang, CV, API} та показники моделі проекту Project= {Lang, CV, API, Met}. Основні показники, за якими можна визначити сумісність трейні та проекту – це мови програмування, системи контролю версій, інструменти для тестування API.

Сумісність трейні та проекту за показником мови програмування $c(Lang_{Trainee}, Lang_{Project})$, показником системи контролю версій $c(CV_{Trainee}, CV_{Project})$ та показником інструменту для тестування API

$c(API_{Trainee}, API_{Project})$ можна використовувати підхід, аналогічний визначенню сумісності відповідних показників сумісності для ментора та проекту. Додатковими показниками виступають спеціальність, рівень освіти та наявність сертифікованих курсів, пройдених трейні. Для їх врахування можна використовувати той самий підхід, що і при визначенні сумісності ментора та трейні. Результуюче числове значення сумісності трейні та проекту p_{TP} описується виразом (2.14):

$$p_{TP} = (c(Lang_{Trainee}, Lang_{Project}) \cdot \alpha_1 + c(CV_{Trainee}, CV_{Project}) \cdot \alpha_2 + \\ + c(API_{Trainee}, API_{Project}) \cdot \alpha_3 + Edu_{Trainee} \cdot \alpha_4 + \\ + Spec_{Trainee} \cdot r_{Spec} \cdot \alpha_5 + Courses_{Trainee} \cdot r_{Courses} \cdot \alpha_6)/\ 6, \quad (2.14)$$

де $\alpha_1, \dots, \alpha_6$ – вагові коефіцієнти кожної складової сумісності трейні та проекту, $\alpha_1 + \alpha_2, +\alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 = 1$; показники $Edu_{Trainee}$, $Spec_{Trainee}$, r_{Spec}, $Courses_{Trainee}$, $r_{Courses}$ обчислюються по аналогії з (2.13). Для

Результуюче якісне значення рівня сумісності трейні та проекту C_{TP} в термінах «високий», «середній», «низький» сформовано аналогічно сумісності ментора та проекту на основі експертних оцінок та наведено в таблиці 2.9.

Таблиця 2.9 − Оцінка експертів порогових значень C_{TP}

Рівень сумісності	Оцінка експертів порогових значень C_{TP}												Середнє нормоване порогове значення
	1	2	3	4	5	6	7	8	9	10	11	12	
Високий	65	75	83	76	66	90	72	77	90	82	67	84	0,78
Середній	60	44	60	35	47	49	50	43	41	56	37	56	0,48
Низький	22	23	11	27	26	23	27	17	11	24	15	20	0,21

2.4.5. Розробка методу тест-менеджменту для розподілу учасників команди між проектами

Компанії завжди використовували командну роботу для підвищення загальної ефективності. Дослідження показують, що команди є більш успішними, ніж окремі особи, коли завдання вимагають різних навичок, суджень та знань. Для підвищення продуктивності та конкурентоспроможності, компанії реструктуризуються таким чином, щоб більш ефективно використовувати різні таланти своїх працівників.

Для того, щоб команди виконували свою роботу якнайкраще і стабільно, вони повинні бути ретельно сформовані та керовані. Різні дослідники вивчали фактори, які впливають на ефективність роботи команди. Ці фактори можна об'єднати в чотири категорії, як показано в таблиці 2.10.

Таблиця 2.10 – Фактори впливу на ефективність роботи

Наповнення	**Проектування робіт**	**Процес**	**Склад команди**
Достатні ресурси	Автономність	Спільна мета	Розмір
Лідерство та структура	Різноманітність навичок	Конкретні цілі	Різноманітність

Продовження таблиці 2.10 − Фактори впливу на ефективність роботи

Наповнення	**Проектування робіт**	**Процес**	**Склад команди**
Наявність довіри	Ідентичність завдання	Командна самооцінка	Індивідуальність
Система результативності	Важливість завдання	Конфлікт	Гнучкість членів команди

Оцінка та винагорода			Здібності членів команди

Структурування команди рідко відбувається випадково. Формування команди має бути ретельно сплановане, з особливим акцентом на відбір членів команди. Для вирішення відповідної проблеми було розроблено метод тест-менеджменту для розподілу учасників команди між проектами, етапи якого наведені нижче. Розподіл трейні до відповідного проекту має важливе значення з точки зору їх набуття нових вмінь. Адже, важливо розвивати їх наявні навички, щоб вони були більш придатними для працевлаштування.

Етапи методу тест-менеджменту для розподілу учасників команди між проектами описуються наступним переліком.

Етап 1. Сформувати базу фактів продукційної моделі на основі показників моделей доступних для розподілу трейні, менторів та проектів.

Етап 2. Обчислити чисельні значення показників сумісності ментор-проект, ментор-трейні, трейні-проект.

Етап 3. Перевести чисельні значення показників сумісності ментор-проект, ментор-трейні, трейні-проект в якісні терми рівня сумісності виду «низький», «середній», «високий».

Етап 4. Сформувати базу продукційних правил на основі значень сумісності ментор-проект, ментор-трейні, трейні-проект, скомбінувавши всіх трейні, менторів та проекти.

Етап 5. Обчислити праві частини продукційних правил.

Етап 5. Відсортувати правила за рівнями та вагами і сформувати рекомендаційні списки стабільних, помірно стабільних та нестабільних трійок трейні-ментор-проект.

Вказані етапи тест-менеджменту дають змогу порівняти можливості трейні з вимогами до проекту, та призначити його на відповідний проект, що відповідає

його умінням. Trainee, як правило, мають різні набори навичок, тому потрібно розподіляти їх за різними компонентами моделі.

В першу чергу в рекомендаційний список попадають стабільні трійки, але остаточне рішення щодо розподілу учасників проекту між проектами виконує відповідний представник групи менеджменту проекту. При цьому можуть використовуватись певні додаткові обмеження, які лежать поза межами даної методики. Наприклад, поточна кількість трейні, закріплених за ментором, наявність попереднього досвіду сумісної роботи, участь трейні в аналогічних проектах за певною предметною галуззю, природні здібності трейні до навчання та самонавчання, комунікативні навички учасників проекту та інші показники, які можуть вплинути на подальший процес адаптації. Запропонована методика надає експерту попередній розподіл з кращими варіантами по найбільш вагомим показникам.

3 МОДЕЛЮВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Реалізація методики тест-менеджменту

Для отримання даних трейні та призначення на їх основі ментора, потрібна відповідна форма для заповнення, де стажер може залишити відомості про себе. Для того аби здійснити вхід до системи, потрібно заповнити форми авторизації користувача (див. рис.3.1)

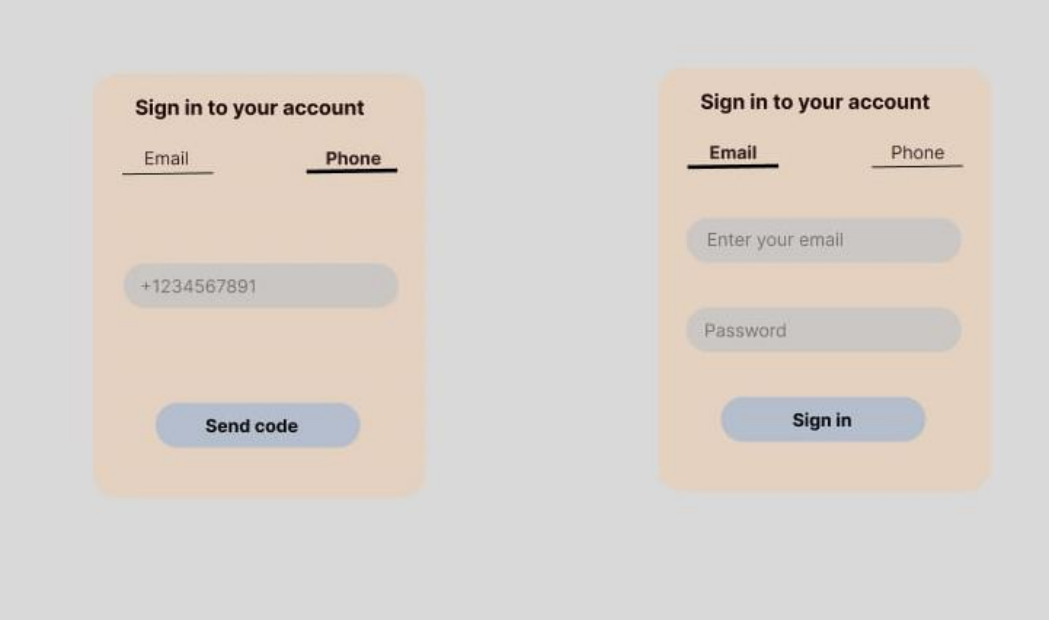


Рисунок 3.1 – Форми для авторизації користувача

Для користувача Trainee створено форму (див. рис. 3.2), де він зможе залишити такий перелік своїх даних:

- прізвище та ім’я;
- вік;
- освіта;
- спеціальність;
- пройдені курси;
- знання мов програмування;

- знання інструментів контролю версій;
- знання інструментів для тестування API.

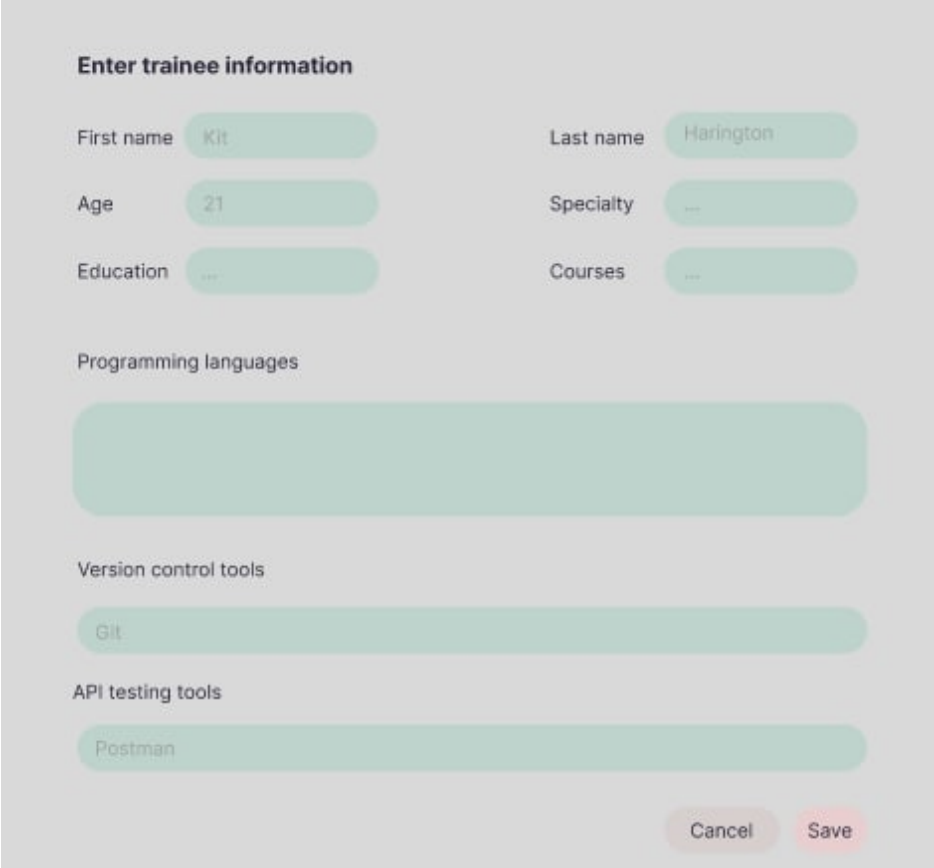


Рисунок 3.2 – Форма заповнення даних трейні

Для користувача Mentor створена інша форма заповнення (див. рис.3.3), що містить такі дані:

- прізвище та ім’я;
- поточна кількість учнів ментора;
- вік;
- сертифікати;
- знання мов програмування;
- знання інструментів контролю версій;
- знання інструментів для тестування API.

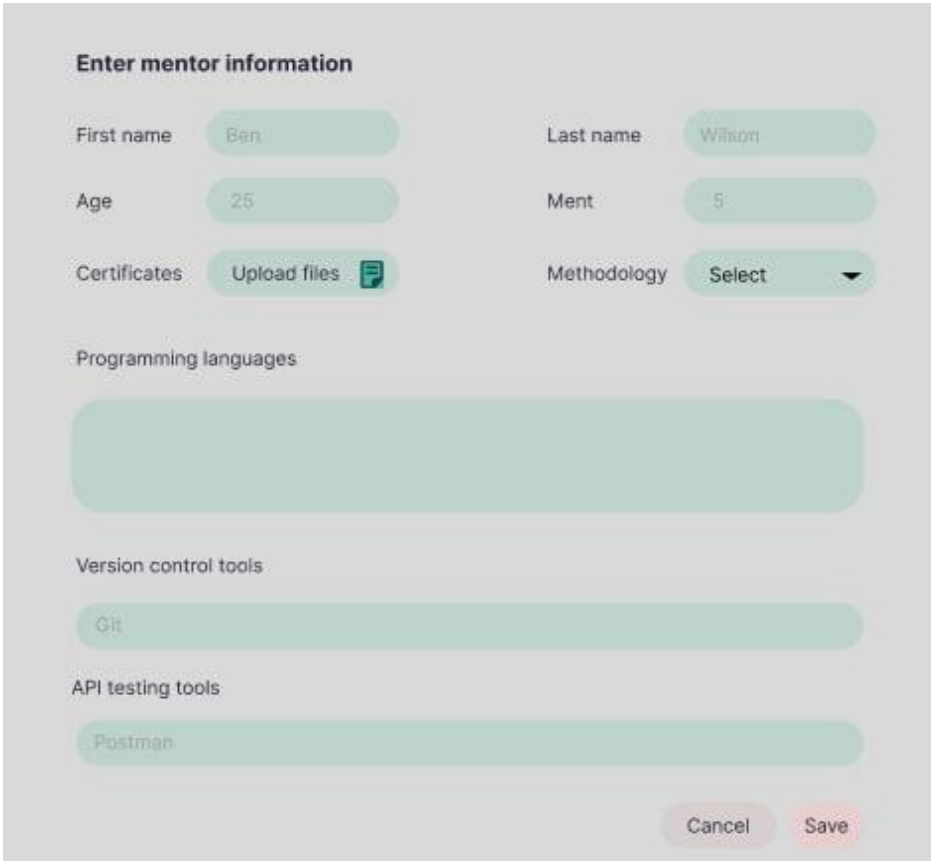


Рисунок 3.3 – Форма заповнення даних ментора

Для формування бази проектів створено відповідну форму (див. рис.3.4), що потребує таку інформацію для заповнення:

- назва проекту;
- методологія;
- мова програмування;
- версія інструментів контролю версій;
- використання інструментів для тестування API.

Після введення даних про необхідних учасників процесу формується результат сумісності трейні з ментором та проектом(див. рис. 3.5). Експерт отримує в першу чергу списки трійок, що мають найвищі показники сумісності та утворюють стабільні трійки. Повний список результатів можна зберегти у файл та використовувати при для подальшого аналізу відповідним представником групи менеджменту проекту.

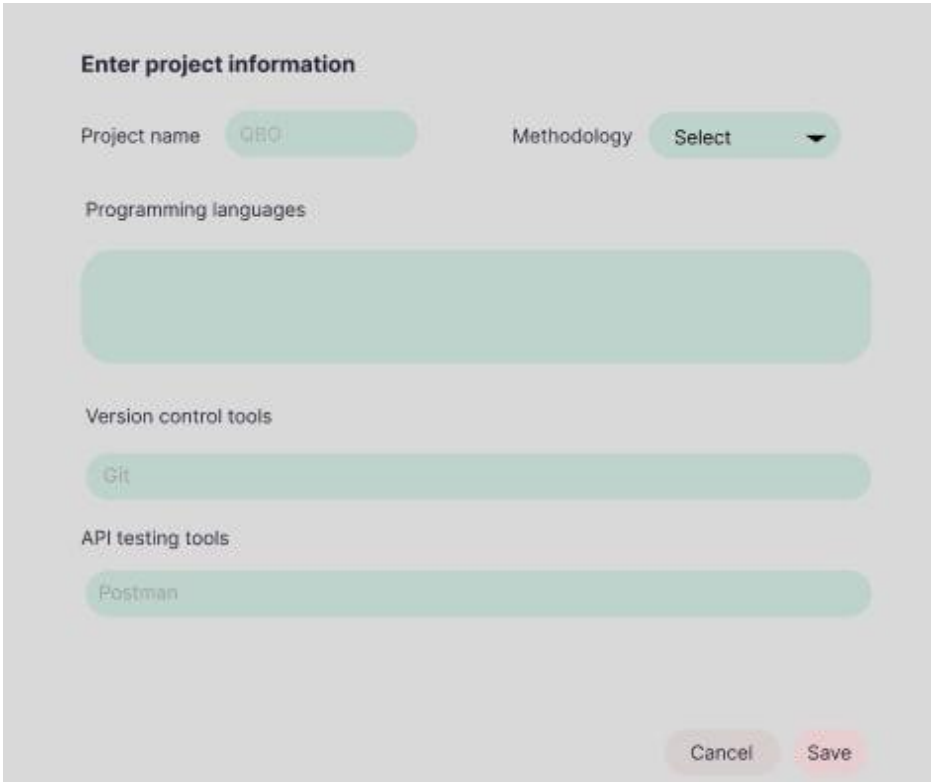


Рисунок 3.4 – Форма заповнення даних проекту

Рисунок 3.5 – Форма для виведення сформованих стабільних трійок

Для перегляду звіту з інформацією щодо результатів підбору створено відповідну форму з можливостю завантажити файли (рис.3.6).

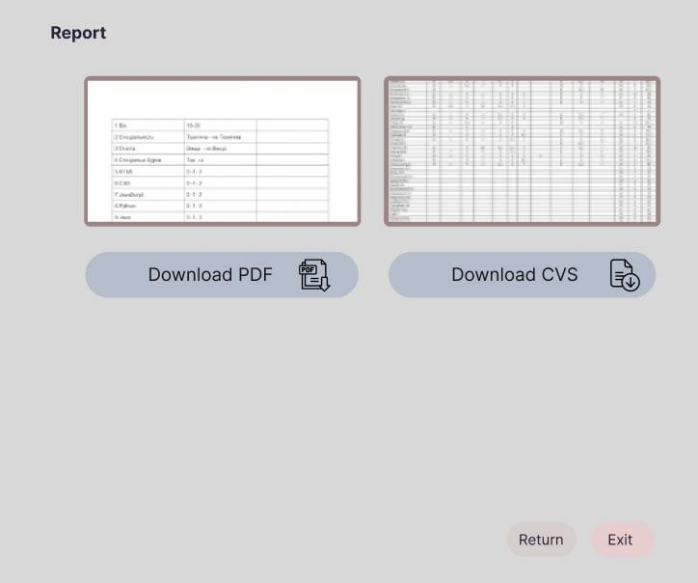


Рисунок 3.6 – Форма виведення звітів

3.2 Аналіз результатів

Для того щоб отримати дані для аналізу потрібно виконати ряд експериментів по моделюванню процесу входження трейні на проект за допомогою розробленої методики та відповідного програмного забезпечення. Моделювання будемо проводити на прикладі розподілу менторів та трейні на проектах IT-компанії, яка спеціалізується у відповідному напрямі. Для збереження комерційної таємниці та захисту персональної інформації імена учасників проекту та дані проектів змінено на умовні коди. В моделюванні використовувались дані по вже реалізованим 10 проектам для того, щоб оцінити наскільки помилковий розподіл впливає на час входження трейні в проект в залежності від того, якою була трійка розподілу: стабільною, помірно стабільною чи нестабільною. Розподілялись 15 трейні, при цьому, зважаючи, що деякі проекти виконувались в різний час, а підготовка окремих трейні була не досить вдалою, то їх закріплювали більше ніж за одним проектом. За трейні закріплювались 6 менторів. У зв’язку з обмеженою кількістю менторів, деякі з них одночасно могли вести декілька трейні як числі на різних проектах, так і на одному. Розрахунки, отримані на основі розробленої методики,

наведено в таблиці 3.1. дані в таблиці відсортовані по рівню сумісності трійок. Кольором виділено стабільні трійки.

Таблиця 3.1 – Результати моделювання

Код проекту	Код трейні	Код ментора	p_MP	C_MP	p_TP	C_TP	p_MT	C_MT	Рівень сумісності C	p_C
3	5	1	0,82	високий	0,84	високий	0,94	високий	високий	0,87
9	11	3	0,58	середній	0,96	високий	0,94	високий	високий	0,83
1	4	2	0,9	високий	0,83	високий	0,48	середній	високий	0,74
6	13	4	0,77	середній	0,73	середній	0,71	середній	середній	0,74
4	9	1	0,8	високий	0,54	середній	0,84	високий	середній	0,73
3	10	6	0,59	середній	0,62	середній	0,92	високий	середній	0,71
10	15	4	0,67	середній	0,82	високий	0,52	середній	середній	0,67
7	12	5	0,39	низький	0,83	високий	0,54	середній	середній	0,59
1	2	5	0,61	середній	0,53	середній	0,49	середній	середній	0,55
8	3	6	0,32	низький	0,76	середній	0,23	низький	середній	0,44
5	11	2	0,78	середній	0,63	середній	0,39	низький	низький	0,6
3	7	5	0,36	низький	0,64	середній	0,69	середній	низький	0,57
3	8	2	0,71	середній	0,65	середній	0,29	низький	низький	0,55
2	1	3	0,23	низький	0,71	середній	0,35	низький	низький	0,43
7	14	2	0,21	низький	0,41	низький	0,58	середній	низький	0,4
1	6	1	0,19	низький	0,66	середній	0,22	низький	низький	0,36
2	3	4	0,46	низький	0,11	низький	0,23	низький	низький	0,27

Розрахунки часу, витраченого на входження на проект наведено в таблиці 3.2.

Таблиця 3.2 – Розрахунки часу входження на проект

Рівень сумісності C	p_C	Прогнозовані витрати часу	Реальні витрати часу	Відхилення по часу, %
високий	0,87	60	42	-30
високий	0,83	74	56	-24,33
високий	0,74	62	44	-29,04
середній	0,74	64	54	-15,63
середній	0,73	68	66	-2,95
середній	0,71	54	50	-7,41
середній	0,67	62	65	4,84
середній	0,59	72	80	11,12
середній	0,55	84	95	13,1
середній	0,44	64	70	9,38
низький	0,6	62	74	19,36
низький	0,57	54	63	16,67

низький	0,55	58	65	12,07
низький	0,43	56	64	14,29
низький	0,4	60	64	6,67
низький	0,36	60	68	13,34
низький	0,27	54	74	37,04

На рисунку 3.7 наведено порівняльну діаграму, де продемонстровано прогнозований час в годинах для входження до проекту trainee та криву реальних витрати часу на входження в проект. Як видно з графіку, для проектів зі стабільними трійками спостерігається зменшення реального часу, що витрачається на входження трейні на проект, у порівнянні з прогнозованим. Для помірно стабільних трійок є відхилення як в більшу, так і в меншу сторону, а для нестабільних трійок стабільно спостерігається збільшення реального часу у порівнянні з прогнозованим.

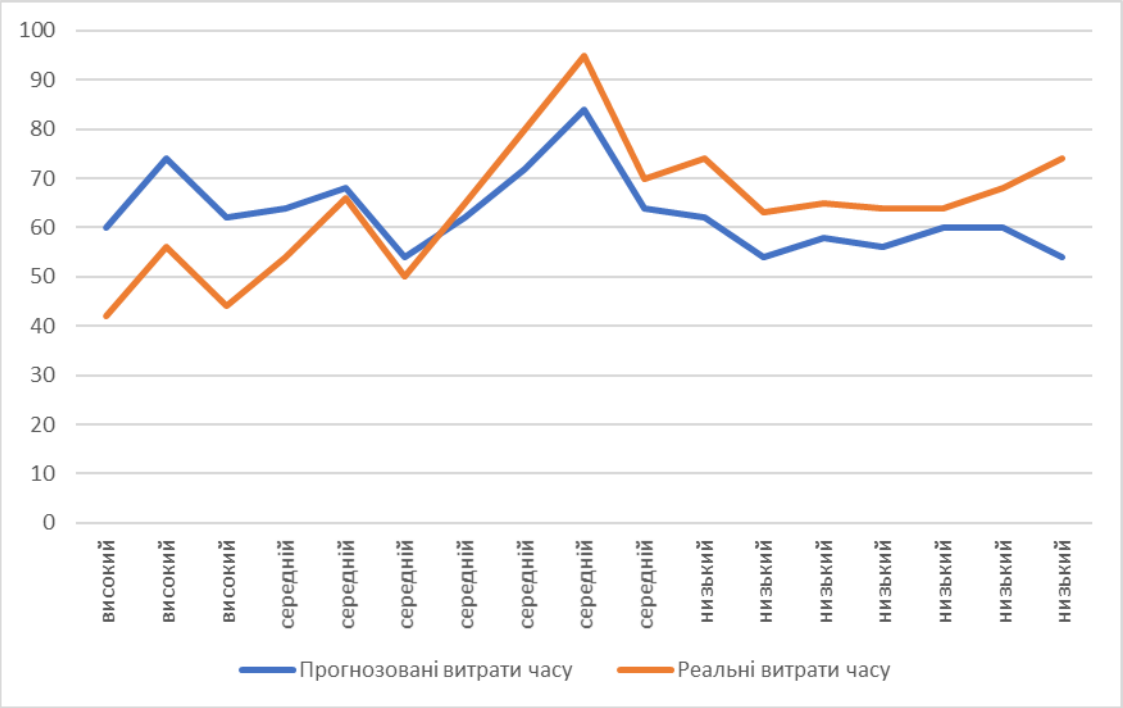


Рисунок 3.7 – Порівняльна діаграма витрат часу

Для стабільних трійок маємо зменшення часу входження в проект в середньому на 27,79%, для помірно стабільних – зменшення в середньому 1,78%, а для нестабільних трійок – збільшення часу в середньому на 17,06%.

Згідно отриманим результатам можна зробити висновок, що застосування розробленої методики тест-менеджменту на основі продукційної моделі дозволить більш раціонально розподіляти учасників проекту між проектами.

ВИСНОВКИ

1. Проаналізовано методи та моделі, що використовують для організації взаємодії учасників команди в процесі тестування ІТ-проектів.

2. Досліджено поняття продукційної моделі та її використання для представлення знань. Описано складові та особливості використання продукційної моделі для вирішення задач організації взаємодії учасників команди в процесі тестування ІТ-проектів.

3. Розроблено методику тест-менеджменту на основі продукційної моделі, яка включає модель трейні, модель ментора, модель проекту, базу продукційних правил, що описують взаємодію учасників команди в процесі тестування. Розроблено метод для визначення на основі бази продукційних правил стабільних, помірно стабільних та нестабільних трійок при закріпленні учасників команди в процесі тестування ІТ-проектів.

4. Розроблену методику було застосовано для моделювання процесу розподілу учасників команди для 10-ти вже реалізованих проектів. Результати моделювання продемонстрували достовірність отриманих даних, зокрема підвищення точності розподілу трейні та менторів по проектах з урахуванням рівня їх кваліфікації та інших характеристик у порівнянні зі традиційним експертним способом розподілу. Для визначених з використанням методики стабільних трійок спостерігається зменшення часу входження в проект в середньому на 27,79%, для помірно стабільних трійок – зменшення становить в середньому 1,78%, а для нестабільних трійок – спостерігається збільшення часу в середньому на 17,06%.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 50 найкращих засобів тестування програмного забезпечення в 2021 році. URL: https://uk.csstricks.net/8222486-top-50-software-testing-tools-in-2021. (Дата звернення: 02.10.2022р.).
2. Allure TestOps: Software Quality Management Software - Qameta.io. URL: https://qameta.io/. (Дата звернення: 03.10.2022р.).
3. ALM Octane. URL: https://www.calleosoftware.co.uk/products/test-management-tools/alm-octane#:~:text=ALM%20Octane%20is%20an%20Application,program%20sharing%20in%20enterprise%20organisations. (Дата звернення: 02.10.2022р.).
4. Guide to the software engineering body of knowledge: trial version executive editors, Alain Abran, James W. Moore; editors, Pierre Bourque, Robert Dupuis, Leonard L. Tripp.
5. Liu, Di, Tao Gu, and Jiang-Ping Xue. Rule engine based on improvement rete algorithm. *The 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding*. IEEE, 2010.
6. PractiTest Methodology and End-To-END Process – PractiTest. URL: https://www.practitest.com/help/methodology-tips/practitest-methodology-2/#:~:text=PractiTest%20is%20an%20end%2Dto,%2C%20Test%2Druns%20and%20Issues. (Дата звернення: 03.10.2022р.).
7. SRET Definition: Software Reliability Engineered Testing | Abbreviation Finder. URL: https://www.abbreviationfinder.org/acronyms/sret_software-reliability-engineered-testing.html. (Дата звернення: 19.09.2022р.).
8. TestRail: Test Management & QA Software for Agile Teams. URL: https://www.gurock.com/testrail/. (Дата звернення: 02.10.2022р.).
9. What is Figma? (And How to Use Figma for Beginners) - Theme Junkie. URL: https://www.theme-junkie.com/what-is-figma/. (Дата звернення: 14.12.2022р).

10. Авраменко А.С., Авраменко В.С., Косенюк Г.В. Тестування програмного забезпечення. Навчальний посібник. – Черкаси: ЧНУ імені Богдана Хмельницького, 2017. – 284 с.

11. Василенко І.А., Ніколаєнко Л.П. Випереджаюча освіта для сталого розвитку: навч. посібник. Дніпро: Акцент ПП, 2021. 279 с.

12. Васюк, О., Майданюк Н. Організація контролю навчання студентів. Вісник книжкової палати №5. 2009, с.27-29.

13. Відповіді на екзамен Штучний інтелект. URL: https://www.academia.edu/8531607/%D0%92%D1%96%D0%B4%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D1%96_%D0%BD%D0%B0_%D0%B5%D0%BA%D0%B7%D0%B0%D0%BC%D0%B5%D0%BD_%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82. (Дата звернення: 27.09.2022р.).

14. Галіцин В.К., Сидоренко Ю.Т., Потапенко С.Д. Технологія програмування і створення програмних продуктів: Навч. посіб. — К.: КНЕУ, 2009. — 372 с. ISBN 978–966–483–234–9.

15. Галузі знань та спеціальності вищої освіти – Освіта.UA. URL: https://osvita.ua/vnz/76723/. (Дата звернення: 16.12.2022р.)

16. Говорущенко Т.О. Підвищення достовірності тестування програмного забезпечення. 2007.

17. Дяченко М.П. Методичні матеріали щодо забезпечення самостійної роботи студентів з дисципліни. Методи та засоби тестування програмного забезпечення (для освітньо-кваліфікаційного рівня «магістр»). — К.: МАУП, 2018— 35 с.

18. Ільясова, Ф. С., Усеїнов Е. А. Методика навчання тестування програмних продуктів студентів напрямку підготовки Інформатика. *Науковий часопис НПУ імені МП Драгоманова. Серія 2: Комп'ютерно-орієнтовані системи навчання* №12. 2012, с.220-224.

19. Ліщина, Н. М., Повстяна Ю. С. Підходи до підготовки фахівців з розробки та тестування програмного забезпечення у вищих навчальних закладах. Комп'ютерно-інтегровані технології: освіта, наука, виробництво №27. 2017, с. 130-134.

20. Мельник, Д. В. Удосконалення процесу тестування програмного забезпечення. *Матеріали VI всеукраїнської студентської науково-технічної конференції „Природничі та гуманітарні науки". Актуальні питання №* 1. 2013, с.93-93.

21. Мельников В. С. Дослідження та розробка компонентів інформаційно-аналітичних систем управління тестуванням. 2021.

22. Плотницький Я.В. Методи тестування програмних систем. 2015.

23. Рамський, Ю. С. Вивчення моделей подання знань в курсі інформатики вищого педагогічного навчального закладу. 2002.

24. Теорія алгоритмів та математична логіка. Тема 15 NP - повні, складні та алгоритмічно нерозв'язні проблеми Стислий конспект. URL: https://elearning.sumdu.edu.ua/free_content/lectured:075b2e8a0bfe48bcef0ab3106c6d51679abc41f9/latest/117573/index.html. (Дата звернення: 18.09.2022р.).

25. Точ Р.В. Навантажувальне тестування системи: актуальність та доцільність. *Збірник тез доповідей підготовлено за матеріалами Міжнародної наукової інтернет-конференції №70.* 2022, с.70.

26. Усов М.А. До питання застосування методу аналізу ієрархій. 2019.

27. Циганок В. В. Метод обчислення ваг альтернатив на основі результатів парних порівнянь, проведених групою експертів. *Реєстрація, зберігання і обробка даних*. 2008.

28. Що робить тестувальник: навички QA від тест-кейса до баг-репорту. URL: https://blog.ithillel.ua/articles/ot-test-keysa-do-bag-reporta-chto-dolzhen-znat-professional-nyy-testirovshchik. (Дата звернення: 19.09.2022р.).

29. Що таке тестування програмного забезпечення? Визначення, основи & Типи. URL: https://uk.csstricks.net/8222490-what-is-software-testing-definition-basics-and-types. (Дата звернення: 19.09.2022р.).

30. Якименко, І. З., and С. О. Вербовий. *Аналіз моделей представлення знань та їх класифікація*. Diss. Тернопіль, ТНЕУ, 2013.

Додаток А.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

МАГІСТЕРСЬКА РОБОТА

«РОЗРОБКА МЕТОДИКИ ТЕСТ-МЕНЕДЖМЕНТУ НА ОСНОВІ ПРОДУКЦІЙНОЇ МОДЕЛІ»

Виконав: студент групи ПДМ-62 Читулян Вадим Олегович

Керівник: Дібрівний Олесь Андрійович, доцент кафедри, доктор філософії (PhD)

Київ - 2022

МЕТА, ОБ'ЄКТ, ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: оптимізація розподілу трейні та менторів команди тестувальників в процесі тест-менеджменту IT-проекту на основі продукційної моделі.

Об'єкт дослідження: розподіл трейні та менторів команди тестувальників в процесі тест-менеджменту IT-проекту .

Предмет дослідження: методика тест-менеджменту для оптимального розподілу трейні та менторів команди тестувальників IT-проекту на основі продукційної моделі.

АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО РОЗПОДІЛУ ТРЕЙНІ ТА МЕНТОРІВ В ПРОЦЕСІ ТЕСТ-МЕНЕДЖМЕНТУ ІТ-ПРОЕКТУ

Підхід до тест-менеджменту	Переваги	Недоліки
На основі побажань трейні: трейні надають свої побажання щодо місця розташування проекту та інтереси щодо конкретних вимог до проекту, а менеджери підбирають відповідний проект	Високий рівень зацікавленості трейні в проекті	Навички трейні не завжди відповідають вимогам проектів, таким чином, може не існувати підходящого проекту для деяких трейні
Жадібний підхід: трейні сортуються відповідно до їх навчальних досягнень, потім кожен трейні призначається на проект, де є потреба у трейні з такими навичками . Якщо відповідність не знайдена, трейні довільно призначається будь-який проект.	Врахування навичок трейні та вимог проектів	Якщо розподіл здійснюється вручну, займає багато часу. Підхід може не дати найкращого рішення. Трейні, які не отримують можливості вибору, можуть стати незадоволеними і покинути організацію.
Підхід на основі лінійного програмування: Модель намагається мінімізувати загальні витрати на виконання завдання.	Врахування дислокації трейні. Мінімізація витрат на виконання завдання.	Якщо трейні не розподіляються за місцем їх вибору, фірма оплачує витрати на логістику та проживання стажера. Якщо технологічні навички стажера не відповідають вимогам проекту, фірма несе витрати на перепідготовку залежно від тривалості навчання. Якість або терміни виконання програмного проекту можуть значно постраждати через невідповідність працівника вимогам проекту

МОДЕЛІ ПРЕДСТАВЛЕННЯ ДАНИХ В БАЗІ ФАКТІВ ПРОДУКЦІЙНОЇ МОДЕЛІ

Модель представлення даних ментора

Mentor = {Age, Lang, CV, API, Met}

Age – вік ментора;
Lang = {(<назва групи галузей>,<рівень>)}
CV = {(<назва інструменту контроля версій>,<рівень>)}
API = {(<назва інструменту для тестування API>,<рівень>)}
Met = {<scrum>, <kanban>, <waterfall>}

Модель представлення даних трейні

Trainee = {Age, Spec, Edu, Courses, Lang, CV, API}

Age – вік трейні;
Spec – спеціальність, Spec = {(<назва групи галузей>,<рівень>)}
Edu = {(<назва освітньої групи>,<рівень>)}
Courses = {(<назва групи курсів>,<рівень>)}
Lang = {(<назва мовної групи>,<рівень>)}
CV = {(<назва інструменту контроля версій>,<рівень>)}
API = {(<назва інструменту для тестування API>,<рівень>)}

Модель представлення даних проекту

Project= {Lang, CV, API, Met}

Lang = {<назва групи галузей>}
CV = {<назва інструменту контроля версій>}
API = {<назва інструменту для тестування API>}
Met = {<scrum>, <kanban>, <waterfall>}

МОДЕЛЬ ВИЗНАЧЕННЯ СУМІСНОСТІ ТРІЙКИ МЕНТОР-ТРЕЙНІ-ПРОЕКТ

C_{MP}	C_{MT}	C_{TP}	C
високий	високий	високий	високий
високий	високий	середній	високий
високий	високий	низький	середній
середній	високий	високий	високий
середній	високий	середній	середній
середній	високий	низький	середній
низький	високий	високий	середній
низький	високий	середній	середній
низький	високий	низький	низький
високий	середній	високий	середній
високий	середній	середній	середній
високий	середній	низький	середній
середній	середній	високий	середній
середній	середній	середній	середній
середній	середній	низький	низький
низький	середній	високий	середній
низький	середній	середній	низький
низький	середній	низький	низький
високий	низький	високий	середній
високий	низький	середній	середній
високий	низький	низький	низький
середній	низький	високий	середній
низький	середній	середній	середній
низький	середній	низький	низький
низький	низький	високий	низький
низький	низький	середній	низький
низький	низький	низький	низький

C(Trainee, Mentor, Project), - сумісність трійки ментор-трейні-проект

C_{MT} - сумісність ментора та трейні ,

C_{MP}, - сумісність ментора та проекту

C_{TP} - сумісність трейні та проекту

Формат продукційного правила для визначення сумісності трійки ментор-трейні-проект:

ЯКЩО C_{MT}=<рівень сумісності>, p_{MT} І C_{MP}=<рівень сумісності>, p_{MP} І C_{TP}=<рівень сумісності>, p_{TP} ТО C=<рівень сумісності>, p_C

Чисельна міра сумісності p_C:

$p_C = (p_{MP}+p_{TP} + p_{MT})/3,$

де p_{MP} - чисельна міра рівня сумісності ментора та проекту, $p_{MP} \in [0; 1]$;

p_{TP} - чисельна міра рівня сумісності трейні та проекту, $p_{TP} \in [0; 1]$;

p_{MT} - чисельна міра рівня сумісності ментора та трейні, $p_{MT} \in [0; 1]$

МОДЕЛЬ ОБЧИСЛЕННЯ СУМІСНОСТІ ПАР

Чисельна міра рівня сумісності ментора та проекту

$$p_{MP} = (c(Lang_{Mentor}, Lang_{Project}) \cdot \alpha_1 + c(CV_{Mentor}, CV_{Project}) \cdot \alpha_2 + \\ + c(API_{Mentor}, API_{Project}) \cdot \alpha_3 + c(Met_{Mentor}, Met_{Project}) \cdot \alpha_4$$

Чисельна міра рівня сумісності ментора та трейні:

$$p_{MT} = c(Lang_{Mentor}, Lang_{Trainee}) \cdot \alpha_1 + c(CV_{Mentor}, CV_{Trainee}) \cdot \alpha_2 + \\ + c(API_{Mentor}, API_{Trainee}) \cdot \alpha_3 + Edu_{Trainee} \cdot \alpha_4 + \\ + Spec_{Trainee} \cdot r_{Spec} \cdot \alpha_5 + Courses_{Trainee} \cdot r_{Courses} \cdot \alpha_6$$

Чисельна міра рівня сумісності трейні та проекту:

$$p_{TP} = c(Lang_{Trainee}, Lang_{Project}) \cdot \alpha_1 + c(CV_{Trainee}, CV_{Project}) \cdot \alpha_2 + \\ + c(API_{Trainee}, API_{Project}) \cdot \alpha_3 + Edu_{Trainee} \cdot \alpha_4 + \\ + Spec_{Trainee} \cdot r_{Spec} \cdot \alpha_5 + Courses_{Trainee} \cdot r_{Courses} \cdot \alpha_6$$

$\alpha_1, \alpha_2, \alpha_3, \alpha_4$ – вагові коефіцієнти кожної складової сумісності ментора та проекту по кожній групі пар, $\alpha_1 + \alpha_2, +\alpha_3 + \alpha_4 = 1$.

Для подальших обчислень показники сумісності пар p_{MP}, p_{MT}, p_{TP} нормуються до 1 по максимальному значенню сумісності .

ЕТАПИ МЕТОДУ ТЕСТ-МЕНЕДЖМЕНТУ ДЛЯ РОЗПОДІЛУ УЧАСНИКІВ КОМАНДИ МІЖ ПРОЕКТАМИ

Етап 1. Сформувати базу фактів продукційної моделі на основі показників моделей доступних для розподілу трейні, менторів та проектів.

Етап 2. Обчислити чисельні значення показників сумісності ментор-проект, ментор-трейні, трейні-проект.

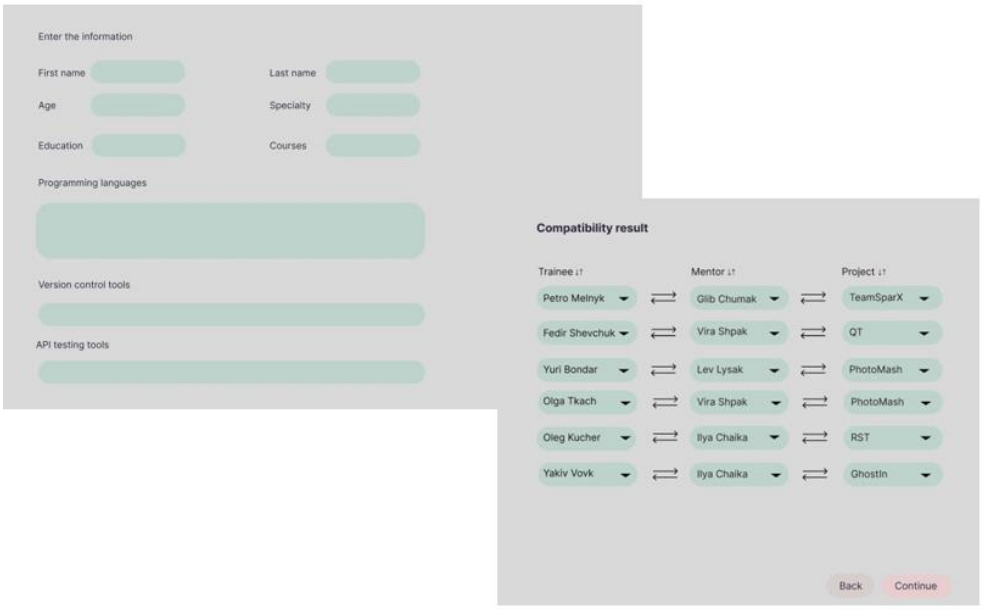
Етап 3. Перевести чисельні значення показників сумісності ментор-проект, ментор-трейні, трейні-проект в якісні терми рівня сумісності виду «низький», «середній», «високий».

Етап 4. Сформувати базу продукційних правил на основі значень сумісності ментор-проект, ментор-трейні, трейні-проект, скомбінувавши всіх трейні, менторів та проекти.

Етап 5. Обчислити праві частини продукційних правил.

Етап 5. Відсортувати правила за рівнями та вагами і сформувати рекомендаційні списки стабільних, помірно стабільних та нестабільних трійок трейні-ментор-проект.

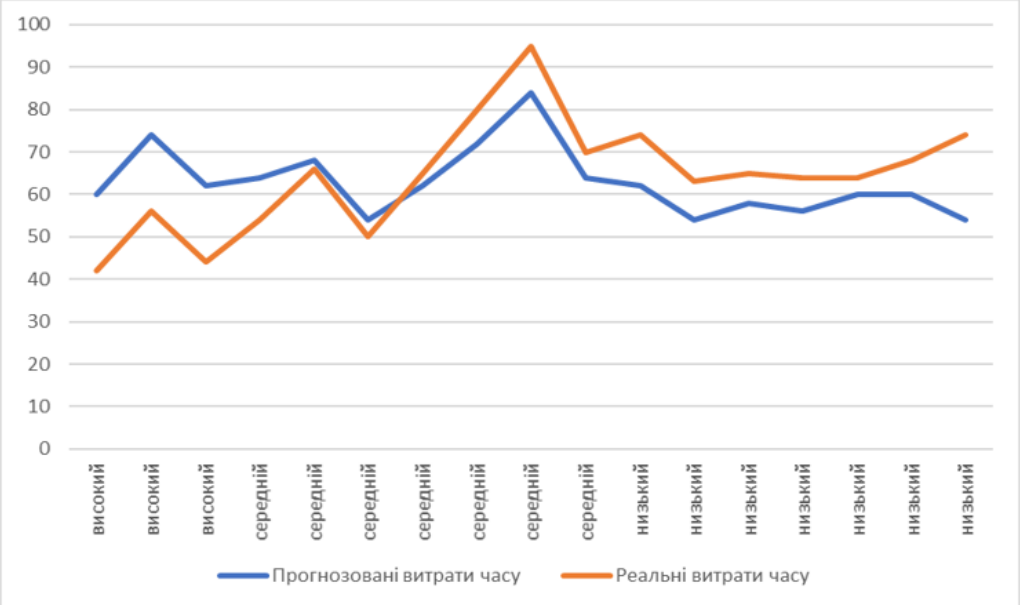
ПРАКТИЧНИЙ РЕЗУЛЬТАТ. ПРИКЛАДИ ЕКРАННИХ ФОРМ



ПРИКЛАД РЕЗУЛЬТУЮЧОГО ФАЙЛУ З РЕКОМЕНДАЦІЯМИ

Код проекту	Код трейні	Код ментора	p_MP	C_MP	p_TP	C_TP	p_MT	C_MT	Рівень сумісності C	p_C
3	5	1	0,82	високий	0,84	високий	0,94	високий	високий	0,87
9	11	3	0,58	середній	0,96	високий	0,94	високий	високий	0,83
1	4	2	0,9	високий	0,83	високий	0,48	середній	високий	0,74
6	13	4	0,77	середній	0,73	середній	0,71	середній	середній	0,74
4	9	1	0,8	високий	0,54	середній	0,84	високий	середній	0,73
3	10	6	0,59	середній	0,62	середній	0,92	високий	середній	0,71
10	15	4	0,67	середній	0,82	високий	0,52	середній	середній	0,67
7	12	5	0,39	низький	0,83	високий	0,54	середній	середній	0,59
1	2	5	0,61	середній	0,53	середній	0,49	середній	середній	0,55
8	3	6	0,32	низький	0,76	середній	0,23	низький	середній	0,44
5	11	2	0,78	середній	0,63	середній	0,39	низький	низький	0,6
3	7	5	0,36	низький	0,64	середній	0,69	середній	низький	0,57
3	8	2	0,71	середній	0,65	середній	0,29	низький	низький	0,55
2	1	3	0,23	низький	0,71	середній	0,35	низький	низький	0,43
7	14	2	0,21	низький	0,41	низький	0,58	середній	низький	0,4
1	6	1	0,19	низький	0,66	середній	0,22	низький	низький	0,36
2	3	4	0,46	низький	0,11	низький	0,23	низький	низький	0,27

ПОРІВНЯЛЬНА ДІАГРАМА ВИТРАТ ЧАСУ



ВИСНОВКИ

1. Проаналізовано методи та моделі, що використовують для організації взаємодії учасників команди в процесі тестування ІТ-проектів.
2. Досліджено поняття продукційної моделі та її використання для представлення знань. Описано складові та особливості використання продукційної моделі для вирішення задач організації взаємодії учасників команди в процесі тестування ІТ-проектів.
3. Розроблено методику тест-менеджменту на основі продукційної моделі, яка включає модель трейні, модель ментора, модель проекту, базу продукційних правил, що описують взаємодію учасників команди в процесі тестування. Розроблено метод для визначення на основі бази продукційних правил стабільних, помірно стабільних та нестабільних трійок при закріпленні учасників команди в процесі тестування ІТ-проектів.
4. Розроблену методику було застосовано для моделювання процесу розподілу учасників команди для 10-ти вже реалізованих проектів. Результати моделювання продемонстрували достовірність отриманих даних, зокрема підвищення точності розподілу трейні та менторів по проектах з урахуванням рівня їх кваліфікації та інших характеристик у порівнянні зі традиційним експертним способом розподілу. Для визначених з використанням методики стабільних трійок спостерігається зменшення часу входження в проект в середньому на 27,79%, для помірно стабільних трійок – зменшення становить в середньому 1,78%, а для нестабільних трійок – спостерігається збільшення часу в середньому на 17,06%.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Читулян В.О. Методи та засоби організації менторської підтримки процесу тестування в ІТ-проекті // І Міжнародна науково-практична конференція студентів та молодих вчених «Математика та математичне моделювання у сучасному технічному університеті» Луцьк: ДонНТУ , 2022, 11-12.

2. Читулян В.О. Система тест-менеджменту як інструмент для керування тестуванням// ІІІ Всеукраїнська науково-практична Інтернет-конференція «Сучасні комп'ютерні та інформаційні системи і технології» Запоріжжя: ТДАТУ, 2022, 45

3. Читулян В.О. Побудова моделі менторської підтримки qa trainee// XV Науково-технічна конференція «Сучасні інфокомунікаційні технології» Київ: К.ДУТ, 2022, 19-20.