УДК 004.93

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

ГЮИК. 501610.004 ПЗ

Дослідження та розробка компонентів інформаційно-аналітичних систем управління тестуванням
(тема)

Виконав:
Студент 2 курсу, групи *СПРм-19-2*
Спеціальність *122 – Комп'ютерні науки*
(код і повна назва напряму)
Тип програми *освітньо-наукова*
(освітньо-професійна або освітньо-наукова)
Освітня програма Системне проектування
(повна назва освітньої програми)

Мельников В.С.
(прізвище, ініціали)

Керівник *проф. Рябченко І.М.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ *Гребеннік І. В.*
(підпис) (прізвище, ініціали)

2021 р

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*

(повна назва)

Кафедра *Системотехніки*

(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність *122 – Комп'ютерні науки*

(код і повна назва)

Тип програми *освітньо-наукова*

(освітньо-професійна або освітньо-наукова)

Освітня програма Системне *проектування*

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«____»_____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Мельникову В'ячеславу Станіславовичу*

(прізвище, ім'я, по батькові)

1. Тема роботи *«Дослідження та розробка компонентів інформаційно-аналітичних систем управління тестуванням»*

затверджена наказом по університету від « *23* » 03 20*21* р. № 389Ст

2. Термін подання студентом роботи до екзаменаційної комісії 24 *травня 2021 р.*

3. Вихідні дані до роботи *Управління тестуванням, генерування тест-кейсів.*

4. Перелік питань, що потрібно опрацювати в роботі *Вступ. Аналіз стану проблеми. Огляд літератури за темою дослідження. Огляд аналогів. Концепція тестування. Створення тест-кейсів. Експертне опитування. Особливості використання методу генерації тест-кейсів. Використаний підхід до автоматичного генерування тест-кейсів. Опис реалізації розробленого модуля системи управління тестуванням для удосконалення підходу до управління тестуванням. Висновки.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів, комп'ютерних ілюстрацій) *Потік подій, системний фреймворк, дизайн модулю автоматичної генерації тест-кейсів, діаграма випадків використання розробленого модуля, схема послідовності для створення робочого простору, діаграма послідовності для створення сценаріїв для кожного варіанта використання, діаграма послідовності для генерації тестового випадку, інтерфейс модуля, інтерфейс модулю автоматичної генерації тест кейсів, інтерфейс модулю автоматичної генерації тест кейсів, тест-кейс реєстрація на курси, тест-кейс вибір курсу для вивчення, тест-кейс закрити реєстрацію.*

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Отримання, аналіз завдання, уточнення*	*23.03.2021*	
	плану роботи		
2	*Аналіз існуючих аналогів*	*25.03.2021*	
3	*Постановка задачі*	*05.04.2021*	
4	*Проведення дослідження концепції тестування*	*16.04.2021*	
5	*Оформлення пояснювальної записки*	*20.04.2021*	
6	*Підготовка презентації*	*12.05.2021*	
7	*Подання закінченої роботи науковому керівникові*	*16.05.2021*	
8	*Подання роботи на рецензування*	*21.05.2021*	
9	*Попередній захист*	*21.05.2021*	
10	*Подання роботи до комісії*	*24.05.2021*	

Дата видачі завдання *23 березня 2021 р.*

Студент _____ *Мельников В.С.*

(підпис)

Керівник роботи _____ *проф. Рябченко І.М.*

(підпис) (посада, прізвище, ініціа

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 72 с., 31 рис., 3 додатки, 20 джерел інформації.

ТЕСТОВИЙ АРТЕФАКТ, ТЕСТ-КЕЙСИ, СИСТЕМА УПРАВЛІННЯ ТЕСТУВАННЯМ, ФРЕЙМВОРК, ГЕНЕРУВАННЯ ТЕСТ-КЕЙСІВ, USE-CASE ДІАГРАМА, АВТОМАТИЗАЦІЯ.

Об’єкт дослідження – управління процесом тестування, розробка тест-кейсів.

Предмет дослідження – системи управління тестуванням, і розробка тест-кейсів з використанням систем управління тестуванням.

Мета кваліфікаційної роботи – розробка модуля для систем управління тестуванням який дозволяє генерувати тест-кейси з допомогою use-case діаграм для удосконалення підходу до управління тестуванням.

Методи досліджень – проведені дослідження грунтуються на методах системного аналізу, загальної теорії систем з областю застосування для аналізу існуючих систем управління тестуванням із використанням методу експертного опитування.

Галузь застосування – управління тестуванням та оптимізація процесу тестування.

В процесі виконання роботи був проведений аналіз стану проблеми тестування, був проведений огляд аналогів, а так само огляд літератури по темі дослідження.

Були виявлені основні проблеми в процесі управління тестуванням, і був розроблений модуль для удосконалення підходу до управлінні тестуванням, для вирішення цих проблем.

ABSTRACT

Master’s explanatory note: 72 pages, 31 figures, 3 applications, 20 sources.

TEST ARTIFACT, TEST CASES, TEST CONTROL SYSTEM, FRAMEWORK, TEST CASE GENERATION, USE-CASE DIAGRAM, AUTOMATION.

The subject of the research is management of the testing process, development of test cases

The subject of the study - test management systems, and the development of test cases using test management systems.

Objective of the qualification work - development of a module for test-quiz management systems which allows generating test-cases with the help of use-case games to improve the approach to test-quiz management.

Research methods - the conducted research is based on the methods of systems analysis, general systems theory with application to the analysis of existing test management systems using the method of expert interviewing. Application area - test management and optimization of the test process.

Within the master thesis the analysis of the testing problem has been done and different test management systems were investigated.

The main issues in the process of test management were identified, and a module was developed to improve the approach to test management, and partial solution of these problems.

ЗМІСТ

ВСТУП…….....7
1. АНАЛІЗ СТАНУ ПРОБЛЕМИ СТВОРЕННЯ І УПРАВЛІННЯ ТЕСТ-КЕЙСАМИ У СИСТЕМАХУПРАВЛІННЯТЕСТУВАННЯМ ТА ПОСТАНОВКА ЗАДАЧІ……...…9
1.1 Аналіз стану проблеми……..…….....9
1.2 Огляд літератури по темі дослідження…….....10
1.3 Огляд аналогів……..16
1.4 Постановка мети та завдань дослідження…….....…..27
2. ДОСЛІДЖЕННЯ КОНЦЕПЦІЇ ТЕСТУВАННЯ……..28
2.1 Концепція тестування……....…..28
2.2 Створення тест-кейсів…….37
2.3 Експертне опитування……...…..……40
3. ПІДХІД ДО АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ТЕСТОВИХ ПРИКЛАДІВ НА ОСНОВІ ВАРІАНТІВ ВИКОРИСТАННЯ НА ЄТАПІ ВИМОГ……...45
3.1 Особливості використання методу генерації тест-кейсів……..……..…..45
3.2 Використаний підхід до автоматичного генерування тест-кейсів.....49
3.3 Опис реалізації розробленого модуля системи управління тестуванням для удосконаленням підходу до управління тестуванням……...…51
ВИСНОВКИ……67
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ……...…...68
ДОДАТОК А……...…70
ДОДАТОК Б……71
ДОДАТОК В……72

ВСТУП

Тестування програмного забезпечення - це процес перевірки системи з метою виявлення будь-яких помилок, прогалин або відсутніх вимог порівняно з фактичною вимогою.

Тестування програмного забезпечення важливо, тому що, якщо в програмному забезпеченні є будь-які помилки або помилки, вони можуть бути виявлені на ранньому етапі і можуть бути усунені до поставки програмного продукту.

Правильно протестований програмний продукт забезпечує надійність, безпеку і високу продуктивність, що в подальшому призводить до економії часу, рентабельності та задоволеності клієнтів. Тестування важливо, тому що помилки в програмному забезпеченні можуть бути дорогими або навіть небезпечними.

Помилки програмного забезпечення потенційно можуть призвести до фінансових втрат і людських жертв, і історія сповнена таких прикладів.

Коли починати тестові заходи: Тестування слід розпочати якомога раніше, щоб зменшити витрати та час на переробку та виготовлення програмного забезпечення без помилок, щоб його можна було доставити клієнту. Однак у життєвому циклі розробки програмного забезпечення (SDLC) тестування можна розпочати з фази збору вимог і продовжувати доти, поки програмне забезпечення не з’явиться у виробництві. Це також залежить від моделі розвитку, яка використовується.

Однією з найважливіших складових в тестуванні є процес створення тест-кейсів. В даний час є безліч методик а також інструментів які дозволяють створювати даний артефакт більш комфортніше, швидше і ефективніше, однак як би там не було, створення даного артефакту є трудомістким процесом в плані сил і часу.

Мета роботи полягає в аналізі існуючих системах управління тестування і аналізу актуальності проблем в даному процесі методом експертного опитування. Запропонувати модуль для систем управління тестуванням який дозволяє генерувати тест-кейси з допомогою use-case діграм для удосконалення підходу до управління тестуванням.

Об'єктом дослідження є процесс управління тестуванням.

Предмет дослідження – системи управління тестуванням.

Для досягнення поставленої мети необхідно виконати такі завдання:

– провести аналіз стану проблеми управління тестуванням;

– провести аналітичний огляд літератури та аналогів за темою атестаційної роботи

– провести теоретичне дослідження концепції тестування, розробки тест-кейсів, та експертне опитування

– провести опис модулю для систем управляння тестуванням за допомогою якого можливо генерувати тест-кейсы з use-case діаграм;

Наукова новизна полягає в тому що в даній роботі запропоноване нестандартне рішення, аналогів якому не існує на даний момент, а саме концепція модульності в системах управління тестувати яка в свою чергу дає можливість інтеграції розробленого модуля з допомогою якого можливо генерувати тест-кейси з use-case диаграм , що дозволить почати тестування на ранніх етапах а так само допоможе заощадити ресурси на написання даного тестового артефакту і скоротити в рази час на тестування програмного продукту.

1 АНАЛІЗ СТАНУ ПРОБЛЕМИ СТВОРЕННЯ І УПРАВЛЯННЯ ТЕСТ-КЕЙСАМИ У СИСТЕМАХ УПРАВЛІННЯ ТЕСТУВАННЯМ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз стану проблеми

В даний момен на ринку присутня достатня кількість систем управління тестуванням які дозволяють, ефективно керувати процессом тестування, починаючи зі створення тестових артефактів закінчуючи отриманням метрик.

Не одна система управління тестуванням не дозволяє проводити автоматичну генерацію тест-кейсів. Створення тест-кейсів є вагомою частиною в процесі управління тестуванням, і процес створення даного артефакту на пряму пов'язаний з управлінням тестуванням.

Є безліч інструмент які дозволяють нстворювати тест-кейси, в іншому, це може бути навіть звичайний текстовий документ, однак, використання будь-яких інструментів які не належать до системи управління тестуванням які в свою чергу активно використовуються компаніями, є дуже не ефективним вибором, через те що - даний тестовий артефакт після його створення буде активно брати участь і бути складовою процесу управління тестуванням.

Написання тест-кейсів в даний час досить затребуване в якості обов'язкового інструменту для забезпечення якості проекту. Зокрема, використовується як одна з форм надання важливої інформації про якість продукту в результаті проведення робіт з тестування. Крім того, написання тест-кейсів також користується популярністю як окрема послуга на різних етапах робіт над проектом, незалежно від його розміру та складності.

Але на жаль написання даного тестового артефакту є дуже трудомістким процесом, і займає досить велику кількість часу.

1.2 Огляд літератури по темі дослідження

Automatic Test Case Generation for SOA Based Services

Генерування тестового випадку вручну вимагає багато часу [2]. Ручні тестові випадки через деякий час втрачають свою придатність, оскільки послуги динамічно публікуються, прив'язуються, викликаються та інтегруються.

Це робить автоматичну генерацію тестових кейсів більш важливою. Автоматичне створення тестових кейсів служб, заснованих на SOA, забезпечує фундаментальну концепцію тестування SOA та досліджує різні перспективи тестування SOA та проблеми тестування SOA. Наше обговорення включає деякі існуючі методи тестування, такі як Техніка збурення даних, Техніка тестування веб-служб (WSGT), Техніка тестування на основі тегів та деякі підходи на основі тестування служб на основі SOA.

Далі ми пояснюємо нашу нову модель тестування за допомогою нашого алгоритму генерації тестових кейсів та алгоритму вибору тестових кейсів.

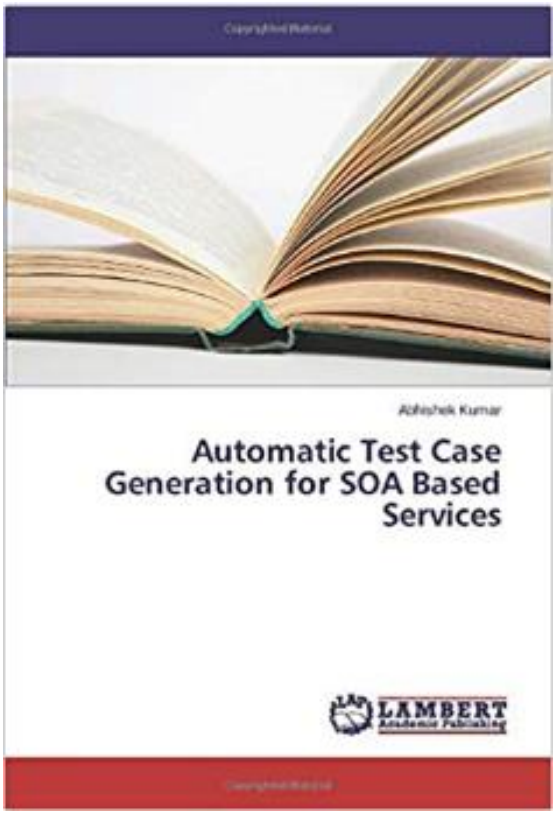

Рисунок 1.1 - Automatic Test Case Generation for SOA Based Services

A Test Manager’s Guide

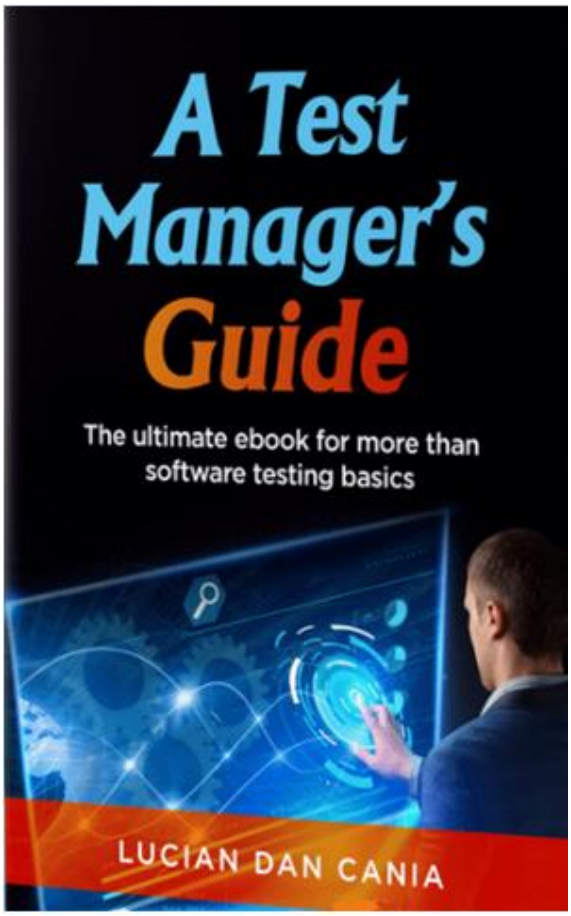

Рисунок 1.2 - A Test Manager’s Guide

Створений експертами галузі, він перенесе вас від основ тестування програмного забезпечення до управління дефектами, методів тестування та показників. Ви навчитесь життєво важливим навичкам для прискорення вашої кар'єри в управлінні тестами програмного забезпечення, включаючи динаміку команди тестування, фактори успіху та виконання стратегій управління тестами від початку до кінця [3].

Вас глибоко заглибиться у цю постійно мінливу галузь - консалтинг з управління тестами програмного забезпечення Cania Consulting, який складається з лідерів галузі, які спеціалізуються на аудиті, стратегії та управлінні тестуванням програмного забезпечення.

Вони щодня використовують зі своїми клієнтами методи та ідеї, наведені в цій книзі, і діляться своїми інсайдерськими порадами як ключовим ресурсом для читачів у галузі управління тестами.

Якщо ви хочете вдосконалити свої навички з основ тестування програмного забезпечення та отримати доступ до кращих інсайдерських знань у цій галузі, ця електронна книга для вас. Посібник менеджера випробувань внесе важливе доповнення до вашої колекції, якщо ви продовжуватимете розвиватися в цій галузі.

The Art of Software Testing, 3rd Edition

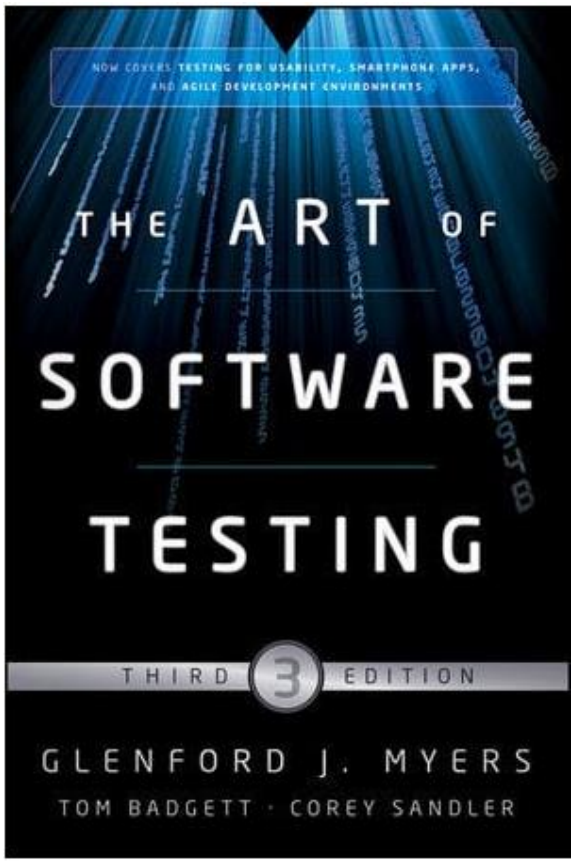

Рисунок. 1.3 - The Art of Software Testing, 3rd Edition

Третє видання «Мистецтво тестування програмного забезпечення» забезпечує коротку, але потужну та всебічну презентацію перевірених часом підходів до тестування програмного забезпечення [4]. Якщо ваш проект з

розробки програмного забезпечення є критично важливим, тоді ця книга - це інвестиція, яка окупиться першою виявленою помилкою.

Деякі найкращі теми, доступні в цій книзі, - це Психологія тестування програмного забезпечення, розробка тестових кейсів, тестування у гнучкому середовищі, тестування інтернет-додатків та тестування мобільних додатків.

Ця остання версія включає тестування мобільних додатків, що працюють на різних платформах, таких як iPhone, iPad та Android. Він також включає тестування Інтернет-додатків, різних веб-сайтів, особливо для електронної комерції, та гнучке середовище тестування.

Якщо ви студент, який прагне зробити кар'єру в тестуванні програмного забезпечення, або якщо ви співробітник, який працює в IT-галузі та хочете зростати у тестуванні, то це найкраща книга для вас.

Agile Testing: A Practical Guide for Testers and Agile Teams

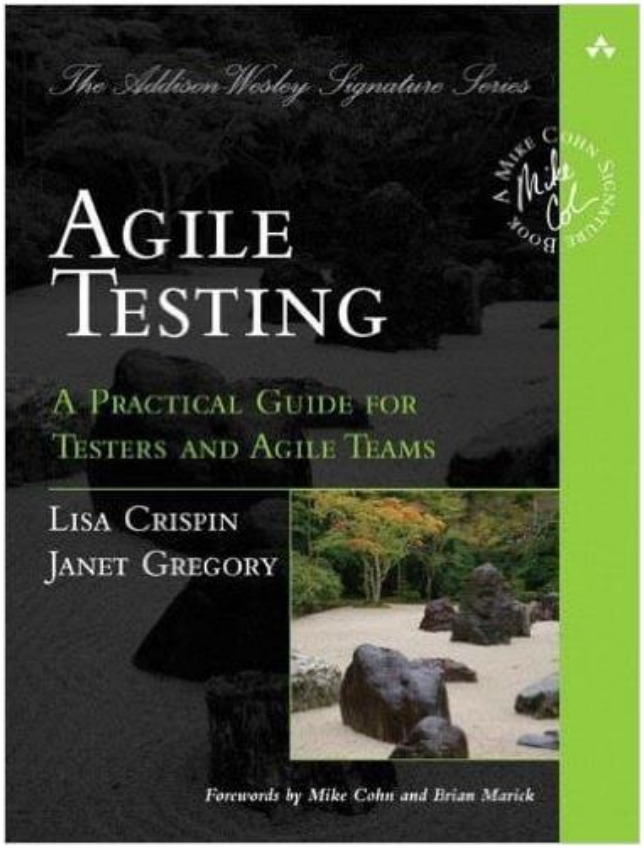

Рисунок. 1.4 - Agile Testing: A Practical Guide for Testers and Agile Teams

Він чітко визначає спритне тестування та ілюструє приклади ролі тестувальника у спритних командах [5].

Ця книга розповідає про використання Agile тестуючих квадрантів, щоб з'ясувати, яке тестування потрібно, хто може проводити тестування та які інструменти можуть йому допомогти. Він також пояснює 7 ключових факторів успішного гнучкого тестування та допомагає у виконанні тестування за короткі ітерації.

A Practitioner's Guide to Software Test Design

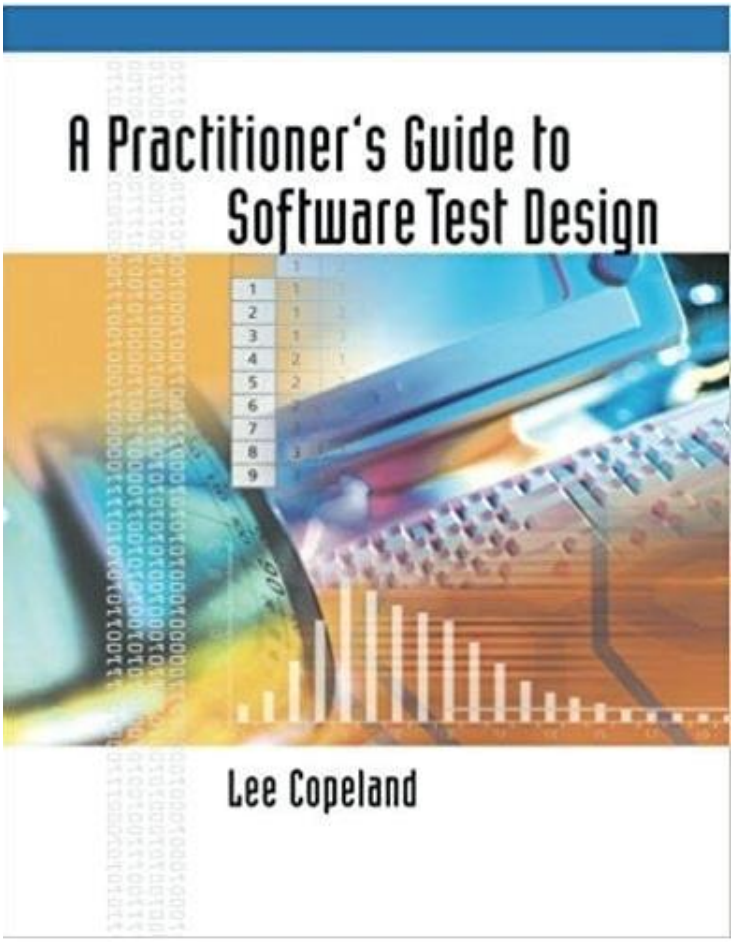

Рисунок. 1.5 - A Practitioner's Guide to Software Test Design

Всебічне, сучасне та практичне введення в розробку тестування програмного забезпечення [6]. У цій книзі представлені всі важливі прийоми проектування тестів в одному місці та у послідовному та легкозасвоюваному форматі. Відразу корисний довідник для інженерів-випробувачів, розробників,

професіоналів із забезпечення якості та вимог та системних аналітиків, він дозволяє вам: вибрати найкращий дизайн тестового кейсу; знаходити дефекти програмного забезпечення за менший час і з меншою кількістю ресурсів; і розробити оптимальні стратегії, які допомагають зменшити ймовірність дорогих помилок.

Це також допомагає вам оцінити зусилля, час і вартість хорошого тестування. включені, допомагаючи вам повністю зрозуміти практичне застосування цих методів.

Від усталених методів, таких як класи еквівалентності, аналіз граничних значень, таблиці рішень та діаграми переходу стану, до нових методів, таких як тестування випадків використання.

1.3 Огляд аналогів систем управління тестуванням

Інструменти управління тестуванням - відмінний ресурс, який допоможе вам знайти і усунути важливі помилки, щоб ви могли зосередитися на створенні кращого програмного забезпечення і більш досконалому процесі розробки програмного забезпечення.

Особливо сьогодні, коли все більше інженерів-програмістів застосовують циклічні методи, такі як розробка через тестування, наявність інструменту для управління всіма ітераціями тестування має вирішальне значення.

Які характеристики і функції ми шукаємо при виборі інструментів управління тестуванням для перевірки? Ось короткий виклад моїх критеріїв оцінки:

1. Інтерфейс користувача (UI): чистий та привабливий.
2. Юзабіліті: Чи легко навчитися та опанувати? Чи пропонує компанія хорошу технічну підтримку, підтримку користувачів, навчальні посібники та навчання.

3. Особливості та функціональність:
    - Планування тестування - чи має інструмент можливості для створення та запису планів тестування, тестових випадків та тестових запусків? Чи допомагає це членам команди контролю якості планувати тестові проекти та створювати тестові набори.
    - Управління вимогами - чи дозволяє командам розробників встановлювати та відстежувати вимоги протягом усього процесу тестування.
    - Виконання тесту - чи підтримує інструмент виконання тесту для різних видів тестування, таких як ручне тестування, автоматизація тестів, безперервне тестування, інтеграційне тестування тощо
    - Звітність та аналітика - чи можуть члени команди створювати детальні та корисні звіти про результати тестування та прогрес протягом процесу тестування
4. Інтеграція: Чи легко підключитися до інших інструментів чи доповнень? Будь-яка попередньо побудована і безперешкодна інтеграція з іншим програмним забезпеченням, яке ви вже використовуєте, наприклад, програмне забезпечення для відстеження помилок, програмне забезпечення для відстеження проблем, програмне забезпечення управління життєвим циклом додатків (ALM), програмне забезпечення для управління проектами, інструменти безперервної інтеграції або засоби автоматизації тестування.

Ці інструменти для тестування допомагають процесу забезпечення якості програмного забезпечення та зменшують зусилля з ручного тестування, завдяки чому ви витрачаєте менше часу на роботу з електронними таблицями Excel або тестами старих шкіл, а також більше часу на тестування програмного забезпечення та виправлення помилок.

Practitest - найкраще для унікальних ієрархічних дерев фільтрів

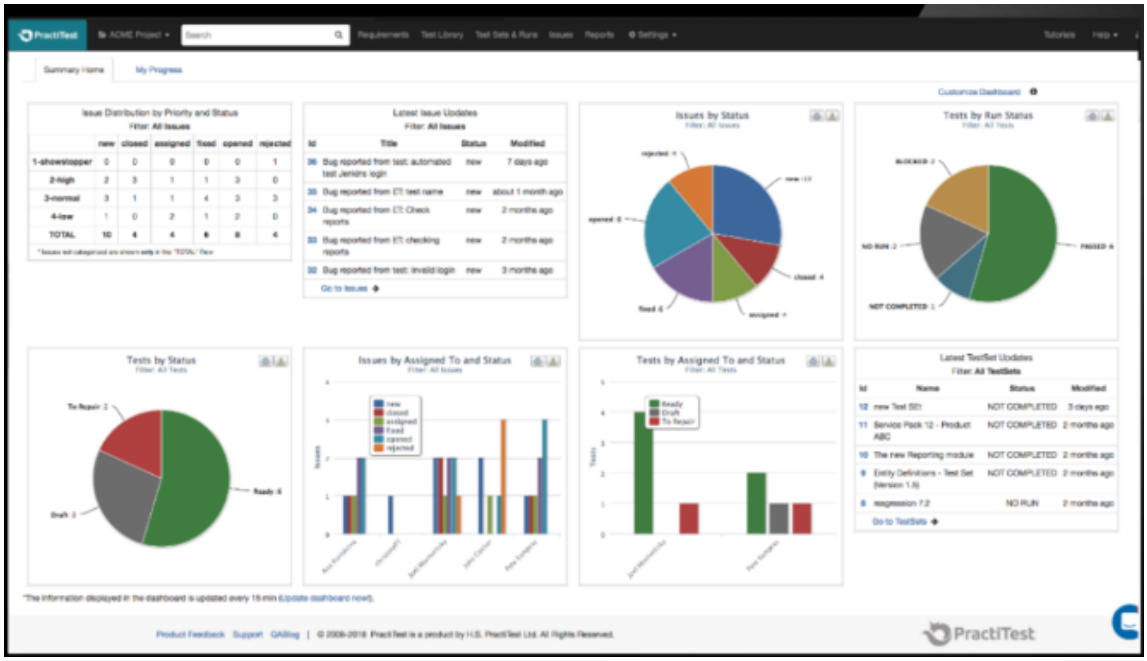

Рисунок. 1.6 – Practitest

PractiTest - це наскрізний інструмент управління тестами. Спільне місце для зустрічей усіх зацікавлених сторін з контролю якості, воно забезпечує повну видимість процесу тестування та глибше, ширше розуміння результатів тестування. Інструмент повністю настроюється та гнучкий для постійно мінливих потреб команд контролю якості, що дозволяє їм налаштовувати поля, подання, дозволи, видавати робочі процеси тощо. Програмне забезпечення також дозволяє користувачам повторно використовувати тести та співвідносити результати в різних випусках та продуктах, а також уникати повторної роботи з дублікатами проти помилок, перестановками, параметрами кроку та функцією виклику для тестування. Ще однією чудовою особливістю є унікальні ієрархічні дерева фільтрів, які чудово підходять для організації всього і швидкого пошуку будь-чого. Члени команди QA можуть візуалізувати дані за допомогою вдосконалених інформаційних панелей та звітів. PractiTest забезпечує широкий спектр інтеграцій сторонніх розробників із поширеними програмами відстеження помилок, такими як Redmine, Jira, Pivotal Tracker

тощо, а також засобами автоматизації. Інструмент також пропонує надійний API для подальшої інтеграції.

TestRail - найкраще для вбудованих та спеціальних шаблонів

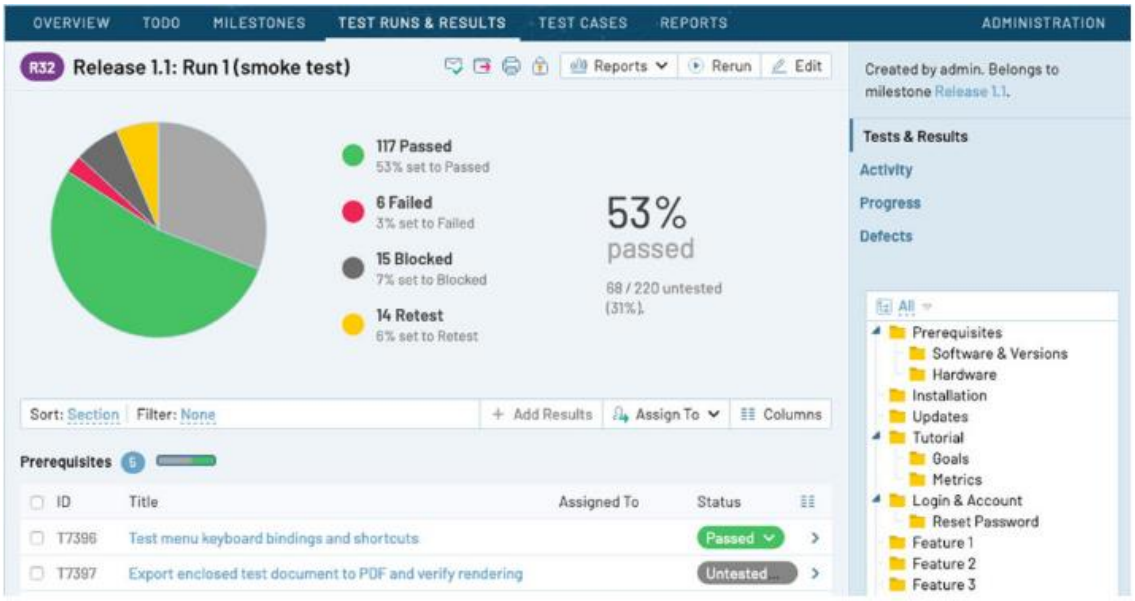

Рисунок 1.7 – TestRail

TestRail - це ваше джерело для масштабованого, настроюваного та веб-управління тестовими кейсами. Вони пропонують хмарне рішення / SaaS, або ви можете встановити TestRail на своєму власному сервері.

Ефективне керування ручними та автоматизованими тестовими кейсами, планами та прогонами та отримуйте інформацію в реальному часі про тестування за допомогою інформаційних інформаційних панелей, показників та звітів про діяльність. TestRail дозволяє користувачам документувати тестові випадки за допомогою знімків екрану та очікуваних результатів. Ви можете використовувати гнучкі вбудовані шаблони або створювати власні власні шаблони. TestRail також включає деякі чудові функції підвищення продуктивності, такі як етапи, особисті списки справ та сповіщення електронною поштою для підвищення ефективності. Інтегруйте TestRail з інструментами у ваш конвеєр CI / CD / DevOps, включаючи JIRA, Bugzilla,

Jenkins, TFS та багато іншого. TestRail також пропонує підтримку контейнерів Docker.

Kualitee - найкраще для управління дефектами

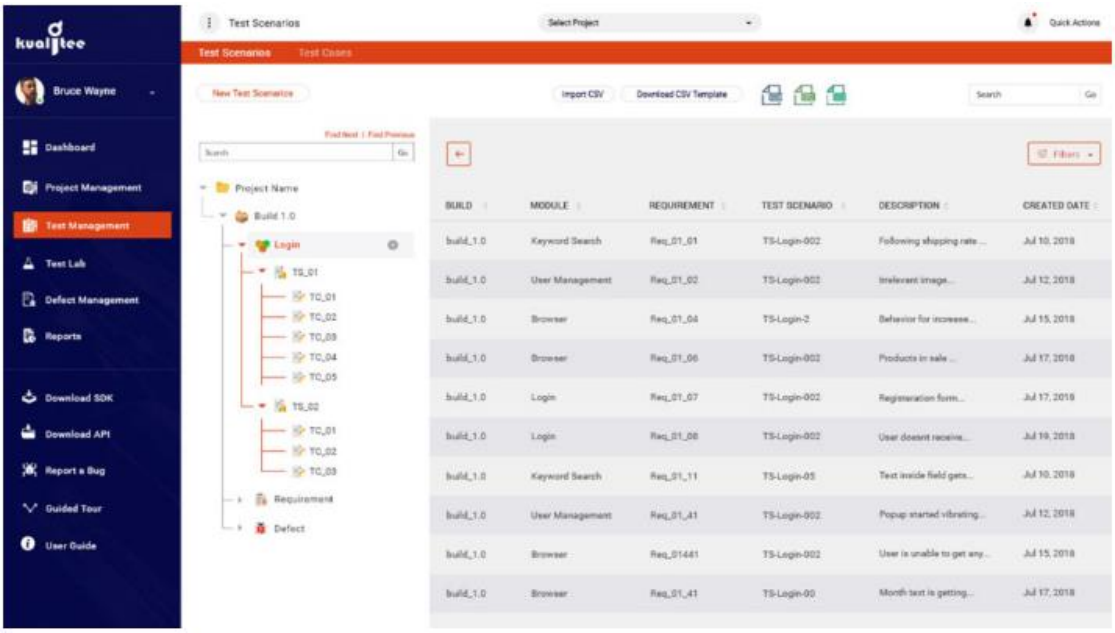

Рисунок 1.8 – Kualitee

Kualitee - це хмарний інструмент управління тестами, який підтримує як ручне, так і автоматизоване тестування. Тестери можуть створювати та імпортувати тестові кейси та асоціювати їх із збірками та вимогами, щоб отримати повну наскрізну простежуваність. Завдяки своїй функції управління дефектами, Kualitee проводить повний потік тестування від планування тесту до виконання та відстеження проблем. Це універсальний інструмент, де тестувальники, розробники та менеджери проектів можуть співпрацювати. Інструмент дозволяє користувачам запускати власні звіти про тестування, помилки та виконання тестів, а також включає функції для планування вимог. Їхні інтерактивні панелі інструментів інтуїтивно зрозумілі та зручні. Kualitee інтегрується з Jira, Selenium, Jenkins та Bitbucket. Інструмент також пропонує мобільний додаток для більшої зручності використання.

Testpad

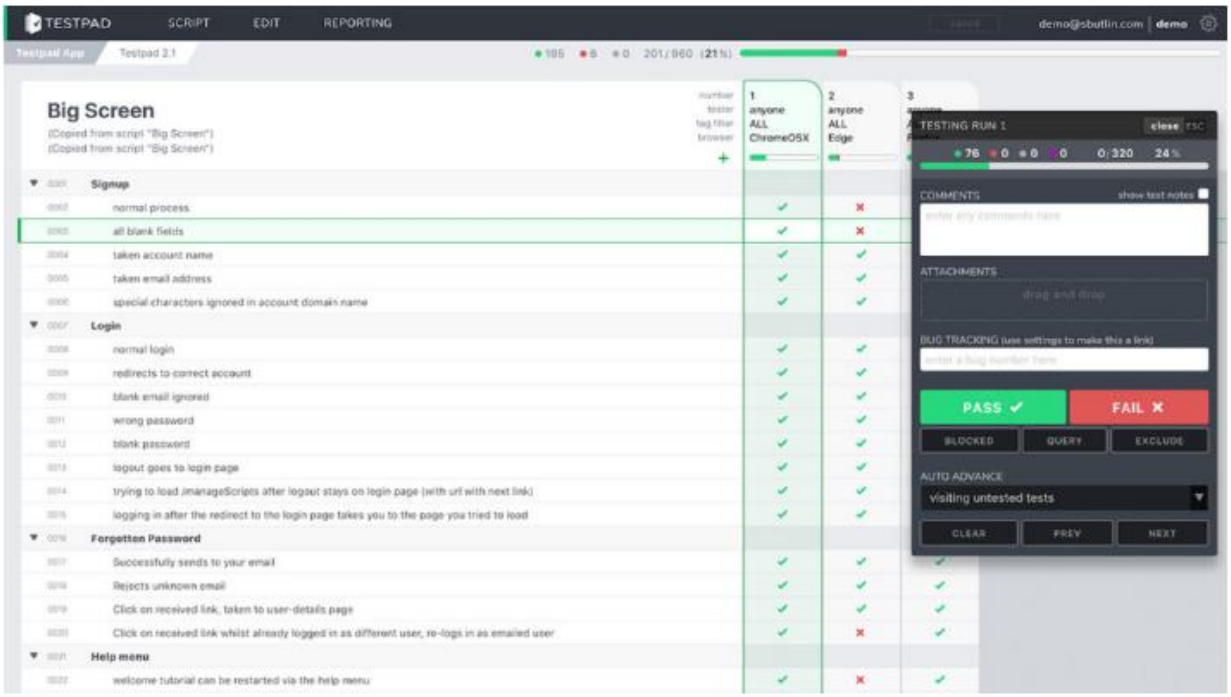

Рисунок 1.9 – Testpad

Testpad - це простіший і доступніший інструмент для ручного тестування, який надає пріоритет прагматизму над процесом. Замість того, щоб керувати справами по одному, він використовує плани тестування на основі контрольних списків, які можуть бути адаптовані до широкого кола стилів, включаючи дослідницьке тестування, ручну сторону Agile, виділений синтаксис BDD і навіть традиційне управління тестовими кейсами. За допомогою Testpad користувачі можуть запросити запрошених тестувальників електронною поштою, яким не потрібні облікові записи для використання інструменту. Інструмент також корисний та інтуїтивно зрозумілий, з можливістю перетягування та планування тестових планів. Навіть ті, хто не тестує, можуть швидко зрозуміти програмне забезпечення. Тестовий редактор керується клавіатурою з адаптивним інтерфейсом на основі JavaScript. Також легко додавати нові тести під час тестування, коли ви думаєте про нові.

TestMonitor

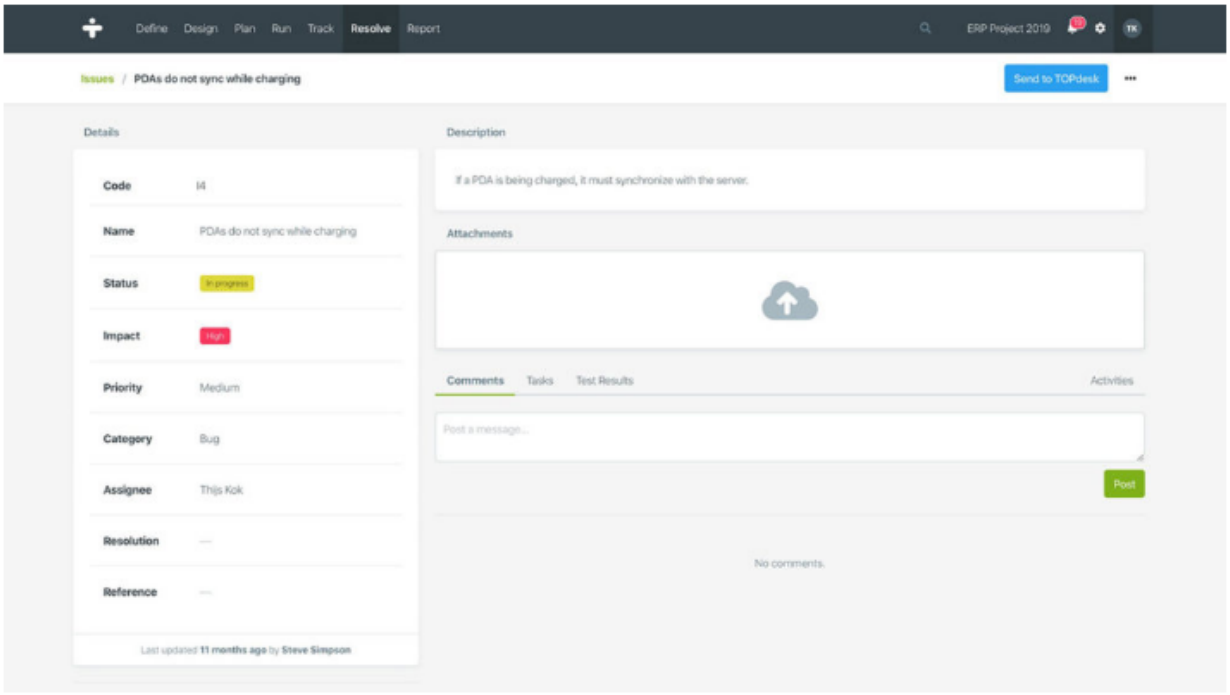

Рисунок 1.10 – TestMonitor

TestMonitor пропонує вдосконалений дизайн тестових кейсів, який може підтримувати тисячі справ. Надійні інструменти планування інструменту мають багатотестерні пробіги та клонування. Ще однією чудовою функціональністю є комплексне відстеження результатів та інтелектуальне звітування з безліччю опцій фільтрації та візуалізації. Інтегроване управління проблемами, а також тестування на основі вимог та ризику також включено. TestMonitor має простий інтерфейс та професійну підтримку з швидким часом відгуку. Доступні сторонні інтеграції, такі як Jira, DevOps та Slack, а TestMonitor включає REST API для подальшої інтеграції.

IBM Rational Quality Manager

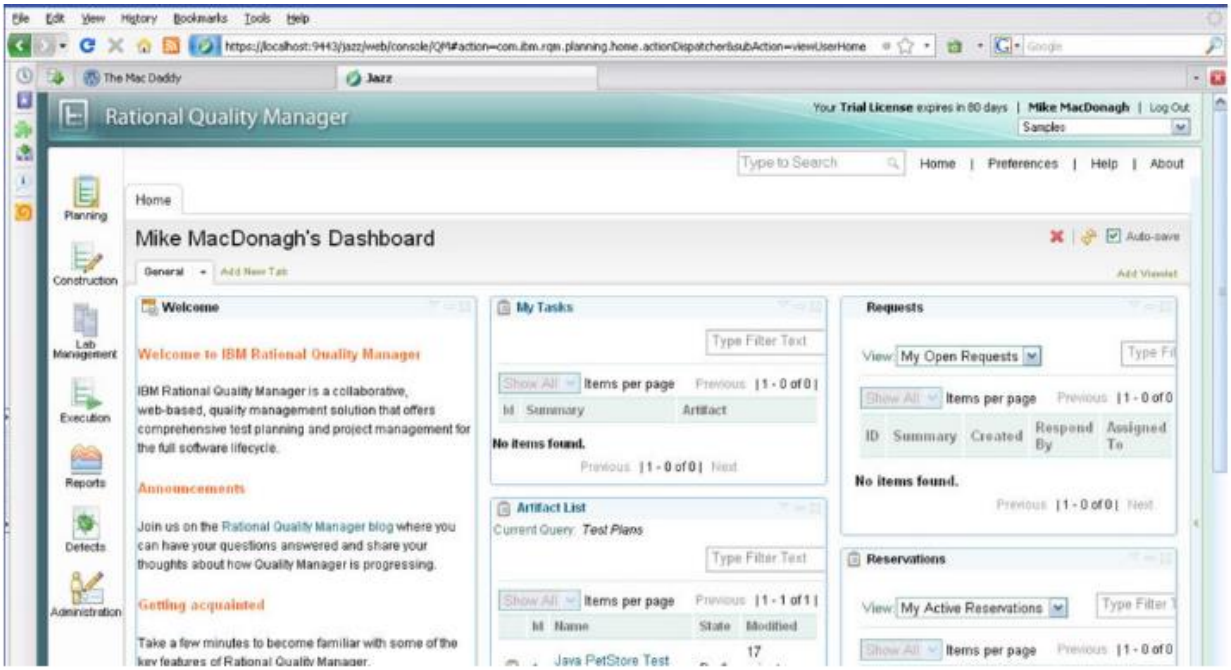

Рисунок 1.11 - IBM Rational Quality Manager

IBM Rational Quality Manager може автоматизувати та прискорити графіки проектів, а також надає звіти про показники, щоб ваша команда могла приймати обґрунтовані рішення щодо випуску. Він може використовуватися для відстеження управління тестами в Інтернеті, ведення простих онлайн-бібліотек, відстеження технічних випусків, випусків клієнтів тощо. Це не залежить від платформи, і це дозволяє командам ефективно управляти проектами контролю якості. Визначте плани тестування та відредагуйте тестові кейси безпосередньо в Rational Quality Manager. IBM Rational Quality Manager також пропонує функції для аналізу та звітування про тести та загальний проект, а також реальні перегляди статусів виконання тестів. Посильте співпрацю команди завдяки можливості розподіляти завдання іншим членам команди та переглядати стан команди Rational Quality Manager може інтегруватися з іншими продуктами IBM Rational у їх наборі спільного управління життєвим циклом, а також з іншим програмним забезпеченням IBM для таких елементів, як тестування автоматизації, управління вимогами та управління змінами.

Microfocus Silk Central

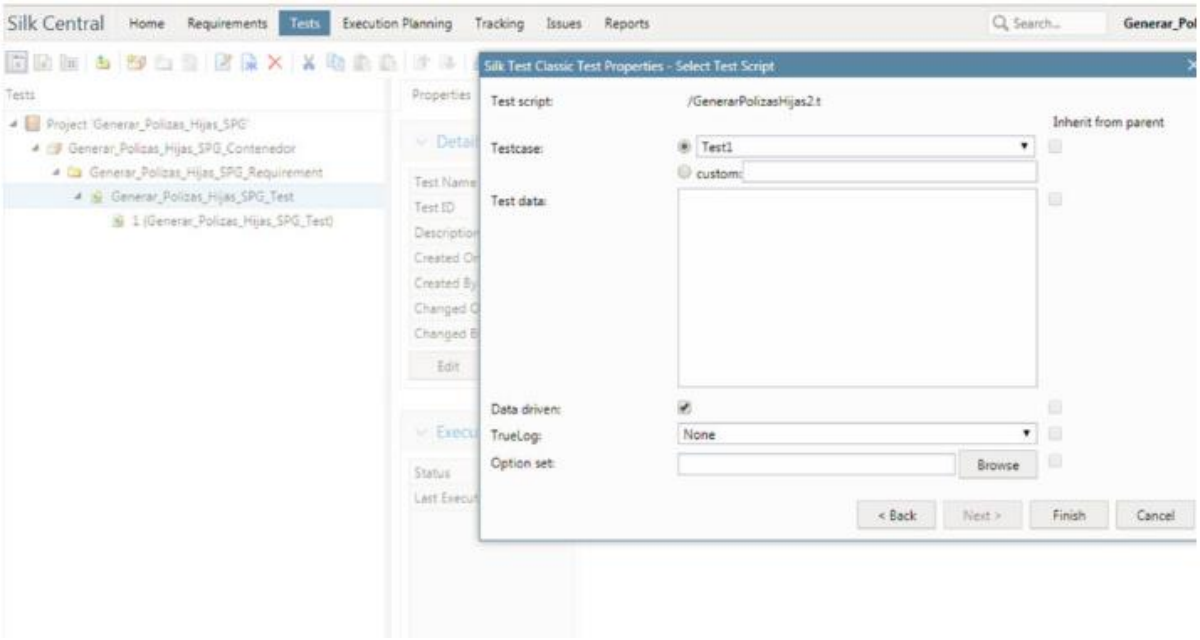

Рисунок 1.12 - Microfocus Silk Central

Silk Central дозволяє об’єднати всі тестові матеріали в один простий у використанні центр планування, відстеження, звітування та виконання. За допомогою Silk Central легко відтворити помилки та проблеми в програмному забезпеченні. Silk Central надає уявлення про тестування за допомогою панелі приладів TestBook для ручного тестування та дізнається, хто над чим працював, щоб відстежувати прогрес.

Тестувальники можуть ділитися коментарями та ставити запитання, публікуючи коментарі безпосередньо в TestBook під час тестування, що збільшує співпрацю між членами команди. Сторонні інтеграції Silk Central включають IBM DOORS, IBM Rational ClearQuest, Bugzilla, Jira та Git. Silk Central пропонує подальшу інтеграцію через їх API

TestLink

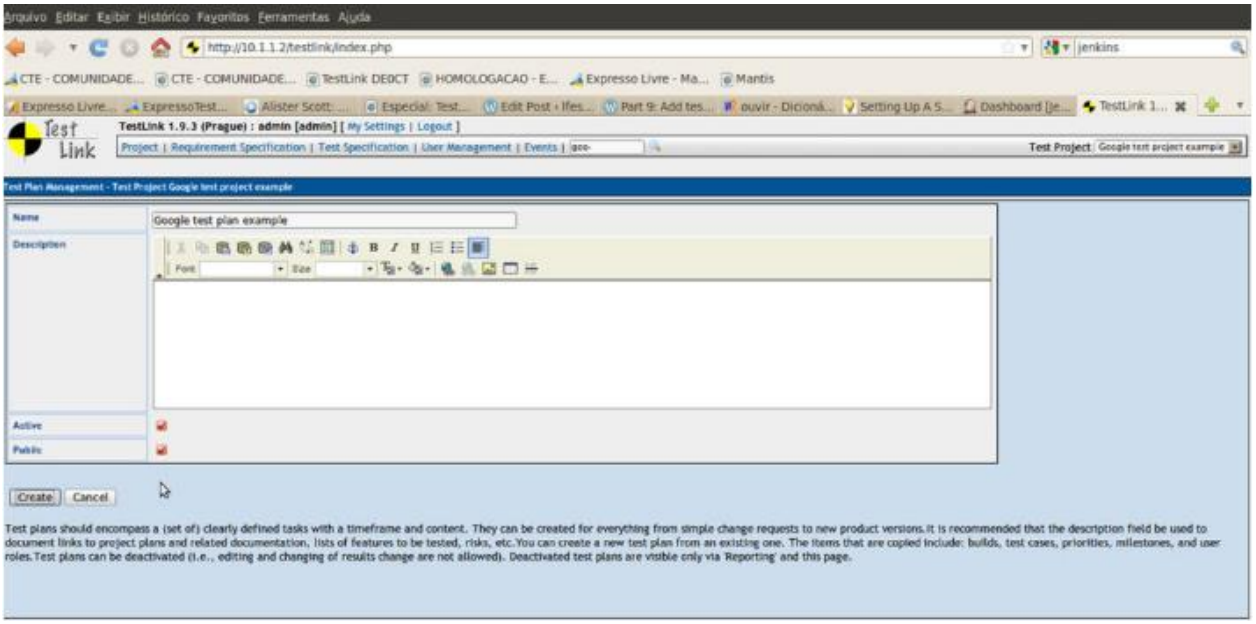

Рисунок 1.13 – TestLink

TestLink дозволяє користувачам створювати тестові кейси та плани тестів, керувати користувачами, а також візуалізувати результати тестування та прогрес за допомогою звітування та статистики. Експорт та імпорт тестових кейсів є простим та інтуїтивно зрозумілим, а TestLink також пропонує простий метод призначення тестових кейсів для кількох користувачів.

TestLink - це програмне забезпечення з відкритим кодом, тому користувачі можуть налаштувати його відповідно до своїх потреб.

TestLink може інтегруватися з багатьма інструментами управління дефектами, а також такими інструментами, як Selenium, Katalon Studio, TeamForge, Jira та іншими. Як інструмент з відкритим кодом, TestLink можна використовувати безкоштовно.

qTest Manager

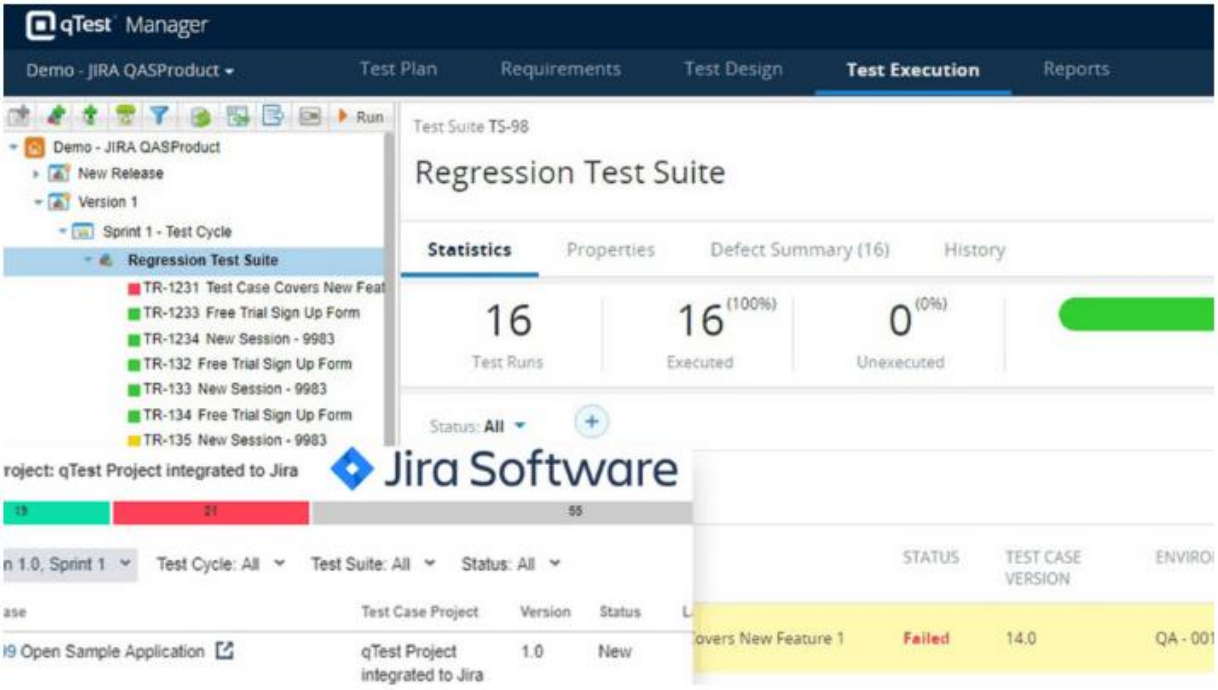

Рисунок 1.14 - qTest Manager

qTest Manager трансформує процес управління тестовими кейсами, допомагаючи командам підприємств швидше та ефективніше. Це тестова платформа, створена для корпоративних команд, що практикують процеси Agile та DevOps для управління тестами. Ця платформа має сучасний інтерфейс користувача на основі браузера, що полегшує всі дії тестування, включаючи управління тестами, автоматизацію та звітування.

Деякі інші корисні функції включають плавне планування автоматизації тестування та інтеграцію CI, надійну звітність та аналітику, а також пошукове та сесійне тестування.

Програмне забезпечення також пропонує рішення для корпоративного поведінки (BDD). qTest також має інтеграцію з інструментами розробки, такими як Jenkins та GitHub, для відстеження, а також інтеграцію в реальному часі з Jira та іншими інструментами Atlassian. Ціни на менеджер qTest доступні за запитом.

Test Collab

Рисунок 1.15 - Test Collab

Test Collab - це зручний інструмент управління тестами, який підтримує автоматизацію та ручне тестування. Користувачі можуть скористатися плануванням та розробкою тестових кейсів, вимогами до тестування документів, а також переглядати власні звіти про тестові комплекти, розподіл тестових кейсів та виконання тестів.

Інструмент також дозволяє користувачам встановлювати власні фільтри та поля для сортування та фільтрації тестових випадків та виконання для детального аналізу відповідних статистичних даних. Test Collab також має вбудований відстежувач часу, критерії управління та прийняття вимог, а також можливість встановити багаторазові кроки, які можуть бути перепрофільовані в декількох тестових випадках.

1.4 Постановка мети та завдань дослідження

Метою атестаційної роботи розробка модуля системі управління тестуванням для удосконалення процесу управління тестуванням, а також

провести дослідження концепції тестування, розробки тест-кейсів а так само провести експертне опитування для з'ясування актуальності проблеми.

Для досягнення поставленої мети необхідно виконати такі завдання:

У першому розділі роботи пропонується проаналізувати стан проблеми управління тестуванням, та розробки тест-кейсів, розглянути літературу по темі дослідження, розглянути аналоги , а так само зробити постановку целі і завдань даної роботи.

У другому розділі пропонується провести теоретичне дослідження концепції тестування, а так само буде проведено дослідження у вигляді опитування фахівців в даній сфері для з'ясування:

- чи використовується на проекті система управління тестуванням;
- актуальність використання тест-кейсів на проекті;
- чи дійсно це є поширеної проблемою, те що створення даного артефакту займає велику кількість часу, і є блокуючим фактором для проведення більш ефективного і швидкого виконання тестування.

У третьому розділі буде описана реалізація модуля для систем управління тестуванням є удосконаленням підходу до управління тестуванням, який надає можливість створення тест-кейсів із випадків використання.

Таким чином, в ході дослідження буде проводитися теоретичне дослідження та експертне опитування.

Так само буде розроблений модуль системи управління тестуванням для удосконалення процесом тестування.

2 ДОСЛІДЖЕННЯ КОНЦЕПЦІЇ ТЕСТУВАННЯ

2.1 Концепція тестування

Людські помилки можуть привести до появи дефекту або збою на будь-якому етапі життєвого циклу розробки програмного забезпечення, і, в залежності від наслідків помилки, результати можуть бути тривіальними або катастрофічними. Під час розробки та обслуговування необхідно ретельне тестування для виявлення дефектів, щоб зменшити кількість збоїв в операційному середовищі та підвищити якість. операційної системи.

Це включає в себе пошук місць в інтерфейсі, де користувач може зробити помилку при введенні даних або інтерпретації виведення, а також пошук потенційних слабких місць для навмисної і зловмисної атаки. Виконання тестів допомагає нам рухатися до підвищення якості продуктів і послуг, але це лише один з методів перевірки та валідації, що застосовуються до продукти. Процеси також перевіряються, наприклад, аудитом. Для перевірки роботи можуть використовуватися різні методи, деякі з яких виконуються автором роботи, а інші - іншими для отримання незалежної точки зору [7].

Ми можемо також вимагати проведення тестування програмного забезпечення на відповідність договірним або юридичним вимогам або галузевим стандартам. Ці стандарти можуть визначати тип техніки, яку ми повинні використовувати, або відсоток програмного коду, який потрібно застосовувати. Це може бути несподіванкою, коли ми дізнаємось, що ми не завжди перевіряємо весь код; це було б занадто дорого в порівнянні з ризиком, з яким ми намагаємось боротися.

Тестування допомагає нам виміряти якість програмного забезпечення з точки зору кількості виявлених дефектів, запущених тестів та системи, охопленої тестами. Ми можемо зробити це як для функціональних атрибутів

програмного забезпечення (наприклад, правильний друк звіту), так і для нефункціональних вимог та характеристик програмного забезпечення (наприклад, друк звіту досить швидко).

Тестування може надати впевненості у якості програмного забезпечення, якщо воно виявляє незначні дефекти або відсутні взагалі, за умови, що ми раді, що тестування є досить суворим. Звичайно, поганий тест може виявити кілька дефектів і залишити у нас помилкове почуття безпеки. Добре розроблений тест виявить дефекти, якщо вони є, і тому, якщо такий тест пройде, ми справедливо будемо більш впевнені в програмному забезпеченні та зможемо стверджувати, що загальний рівень ризику використання системи знижений [8]. Коли тестування виявляє дефекти, якість програмної системи підвищується, коли ці дефекти виправляються, за умови, що виправлення проводиться належним чином.

Коли ми виявляємо несправності, ми можемо спробувати відстежити їх до їх першопричини, справжньої причини, що вони сталися. Існує кілька способів проведення аналізу корінних причин, часто залучаючи групу мозкових штурмів і обговорюючи їх, тому ви можете бачити різні методи в різних організаціях.

Наприклад, припустімо, що у організації виникає проблема із повторним збоєм друку. Деякі спеціалісти з технічного обслуговування збираються разом, щоб вивчити проблему, і вони починають з мозкового штурму всіх можливих причин збоїв [9]. Потім вони об’єднують їх у вибрані ними категорії та перевіряють, чи є загальні основні причини чи основні причини. Деякі з очевидних причин, які вони виявляють, можуть бути:

- У принтера закінчуються витратні матеріали (чорнило або папір).
- Помилка програмного забезпечення драйвера принтера.
- У приміщенні принтера занадто жарко для принтера, і він схоплюється. Це безпосередні причини. Якщо ми подивимось на

одну з них - „У принтера закінчуються витратні матеріали (чорнило або папір)“ - це може статися тому, що:

• Ніхто не відповідає за перевірку рівня паперу та чорнила в принтері; можлива першопричина: жодного процесу перевірки рівня чорнила / паперу в принтері перед використанням.

• Деякі працівники не знають, як міняти чорнильні картриджі; можлива основна причина: персонал, який не навчений та не отримує інструкцій щодо догляду за принтерами • немає запасних картриджів або паперу; можлива першопричина: відсутній процес контролю та замовлення запасів. Якщо ваше тестування обмежується програмним забезпеченням, ви можете подивитися на них і сказати: "Це не проблеми програмного забезпечення, тому вони нас не стосуються!" Отже, як тестувальники програмного забезпечення, ми можемо обмежитися повідомленням про несправність драйвера принтера. Однак наша компетенція тестувальників може перевищувати програмне забезпечення; ми можемо мати повноваження розглядати цілу систему, включаючи апаратне забезпечення та прошивку [9]. Крім того, навіть якщо наша компетенція - програмне забезпечення, ми можемо захотіти розглянути, як програмне забезпечення може допомогти людям запобігти або вирішити проблеми; ми можемо дивитись далі цього погляду. Програмне забезпечення може забезпечити користувальницький інтерфейс, який допомагає користувачеві передбачити, коли паперу або чорнила стає мало. Він може надати прості покрокові інструкції допомогти користувачам змінити картриджі або поповнити папір. Це може забезпечити попередження про високу температуру, щоб можна було керувати навколишнім середовищем. Як тестувальники, ми хочемо не просто думати і повідомляти про дефекти, а разом із рештою команди проекту думати про будь-які потенційні причини збоїв. Ми використовуємо тестування, щоб допомогти нам знайти несправності та (потенційні) збої під час розробки, обслуговування та експлуатації програмного забезпечення. Ми робимо це, щоб допомогти

зменшити ризик збоїв, що трапляються в робочому середовищі - іншими словами, після використання системи - та сприяти підвищенню якості програмної системи. Однак, хоча нам потрібно думати і повідомляти про найрізноманітніші дефекти та несправності, не всі можуть бути виправлені. Програмісти та інші можуть виправити дефекти до того, як ми випустимо систему для експлуатації, але може бути більш розумним обійти несправність. Виправлення дефекту має певний шанс ввести інший дефект або бути зроблено неправильно або неповно. Це особливо вірно, якщо ми усуваємо дефект під тиском. З цієї причини проекти часом дотримуються думки, що вони відкладають виправлення несправності. Це не означає, що тестер, який виявив проблеми, даремно витратив час. Корисно знати, що є проблема, і ми можемо допомогти користувачам системи обійтись і уникнути її. Чим ретельніше наше тестування, тим більше дефектів ми знайдемо. Раніше ми бачили, що однією зі стратегій боротьби з помилками, дефектами та відмовами є спроба їх запобігти, і ми розглянули виявлення причин дефектів та збоїв. Коли ми починаємо новий проект, варто вчитися на проблемах, що виникали в попередніх проектах або в виробничому програмному забезпеченні. Розуміння корінні причини дефектів є важливим аспектом заходів із забезпечення якості, і тестування сприяє тому, що допомагає нам виявляти дефекти якомога раніше до використання програмного забезпечення. Процес вдосконалення повинні запобігти повторному виникненню цих дефектів і, як наслідок, поліпшити якість майбутніх систем. Організаціям слід розглядати тестування як частину розширеної стратегії забезпечення якості, яка включає інші види діяльності (наприклад, стандарти розвитку, навчання та аналіз першопричин).

Ми побачили, що тестування допомагає нам знаходити дефекти та покращувати якість програмного забезпечення. Скільки тестування ми повинні зробити? У нас є вибір: протестувати все, нічого не тестувати або протестувати деяку частину програмного забезпечення. Тепер ваша негайна

відповідь на це цілком може бути тим, що ви сказали: «Все треба перевірити». Ми не хочемо використовувати програмне забезпечення, яке не було повністю перевірено, чи не так? Це означає, що ми повинні застосовувати всі аспекти

програмної системи під час тестування [10]. Що нам потрібно врахувати, так це те, чи ми повинні, або навіть можемо перевірити повністю.

Давайте подивимося, скільки тестування нам потрібно зробити, щоб мати змогу вичерпно тестувати. Скільки тестів потрібно зробити, щоб повністю перевірити одноцифрове числове поле? Безпосереднє запитання: "Що ви розумієте під тестом повністю?"

Існує 10 можливих дійсних числових значень, а також дійсних значень, які нам потрібні для того, щоб усі недійсні значення були відхилені. Є 26 великих літерних символів, 26 нижчих регістрів, принаймні 6 спеціальних та пунктуаційних символів, а також порожнє значення. Отже, для цього прикладу а було б щонайменше 68 тестів однозначне поле.

Ця проблема просто погіршується, коли ми розглядаємо більш реалістичні приклади. На практиці системи мають більше одного поля введення, причому поля змінюються

розміри. Ці тести були б поряд з іншими, наприклад, тестування в різних середовищах. Якщо ми візьмемо приклад, коли на одному екрані є 15 полів введення, кожне з 5 можливих значень, то для тестування всіх дійсних комбінацій вхідних значень вам знадобиться 30 517 578 125 (515) тестів! Навряд чи часові шкали проекту дозволять провести таку кількість тестів.

Тестування нашого одноцифрового поля зі значеннями 2, 3 та 4 робить наші тести більш ретельними, але це не дає нам більше інформації, ніж якщо б ми щойно тестували зі значенням 3.

Тиск на проект включає час і бюджет, а також тиск на надання технічного рішення, яке відповідає потребам клієнтів. Клієнти та менеджери проектів хочуть витратити суму на тестування, що забезпечує для них рентабельність інвестицій. Ця рентабельність інвестицій включає запобігання

дорогим помилкам після випуску. Повне тестування - навіть якщо про це просять замовники та керівники проектів - це просто не те, що вони можуть собі дозволити.

Натомість нам потрібен тестовий підхід, який забезпечує необхідну кількість тестувань для цього проекту, цих клієнтів (та інших зацікавлених сторін) та цього програмного забезпечення. Ми робимо це, узгоджуючи тестування, яке ми проводимо, з ризиками для клієнтів, зацікавлених сторін, проекту та програмного забезпечення. Оцінка та управління ризиками є одним з найважливіших видів діяльності у будь-якому проекті та є ключовою діяльністю та причиною тестування. Вирішуючи, наскільки достатньо тестування, слід враховувати рівень ризику, включаючи технічні та ділові ризики, пов'язані з обмеженнями продукту та проекту, такими як час та бюджет. Ми проводимо оцінку ризику, щоб вирішити, скільки проводити тестування.

Коли ми можемо досягти наших цілей тесту?

Ми можемо використовувати як динамічне тестування, так і статичне тестування як засіб для

досягнення подібних цілей тесту. Обидва надають інформацію вдосконалити як систему, що перевіряється, так і розробку та тестування процесів. Вище ми згадали, що тестування може мати різні цілі та завдання, які часто включають:

• виявлення дефектів;

• набуття впевненості та надання інформації про рівень якості;

• запобігання дефектам.

Багато видів огляду та тестування відбуваються на різних етапах життєвого циклу, як ми побачимо у розділі 2. Вони мають різні цілі. Дочасне тестування - наприклад, раннє проектування тестів та огляд - виявляє дефекти на ранніх стадіях, коли їх дешево знайти та виправити. Після написання коду

програмісти та тестувальники часто проводять набір тестів, щоб виявити та виправити дефекти

програмне забезпечення. У цьому «тестуванні на розробку» (яке включає тестування компонентів, інтеграції та системи) основною метою може бути спричинення якомога більшої кількості відмов, щоб виявити дефекти програмного забезпечення та їх можна було виправити.

Після цього тестування користувачі програмного забезпечення можуть провести прийомне тестування, щоб підтвердити, що система працює належним чином, і отримати впевненість у тому, що вона відповідає вимогам.

Виправлення дефектів не завжди може бути метою тесту або бажаним результатом. Іноді ми просто хочемо зібрати інформацію та виміряти програмне забезпечення. Це може мати форму показників атрибутів, таких як середній час між невдалими оцінками надійності або оцінка щільності дефектів у програмному забезпеченні для оцінки та розуміння ризику його випуску.

Під час обслуговування програмного забезпечення шляхом його вдосконалення або виправлення помилок ми змінюємо програмне забезпечення, яке вже використовується. У такому випадку метою тестування може бути гарантування того, що ми не допускали помилок та не виявляли дефектів під час заміни програмного забезпечення. Це називається регресійним тестуванням - тестуванням, щоб переконатися, що нічого не змінилося, що не повинно було змінитися.

Ми можемо продовжувати тестувати систему, коли вона буде в експлуатації. У цьому випадку основною метою може бути оцінка таких характеристик системи, як надійність або доступність.

Під час впровадження та виконання тесту ми приймаємо умови тестування, перетворюємо їх на тестові кейси та тестове програмне забезпечення та налаштовуємо тестове середовище. Це означає, що, підготувавши висококласний дизайн для наших тестів, ми зараз починаємо їх будувати. Ми

перетворюємо наші умови тестування на тестові кейси та процедури, інше тестове програмне забезпечення, таке як скрипти для автоматизації. Нам також потрібно створити середовище, де ми будемо запускати тести та будувати наші тестові дані. Налаштування середовищ та даних часто вимагає значних витрат часу та зусиль, тому слід ретельно планувати та контролювати цю роботу. Впровадження та виконання тесту мають наступні основні завдання приблизно в наступному порядку:

• Впровадження:

- Розробіть і визначте пріоритети наших тестових кейсів, використовуючи методи, які ви побачите в розділі 4, та створіть дані тестів для цих тестів. Ми також напишемо інструкції для проведення тестів (процедур тестування). Для екзаменатора з водіння це може означати зміну умови тестування "переїзд" на "поїздку по трасі Мейфілд-Роуд до перехрестя з Літньою дорогою і попросити водія повернути ліворуч на Літню дорогу, а потім праворуч на Зелену дорогу, очікуючи, що водій перевірки дзеркала, сигнали та маневри правильно, залишаючись в курсі інші учасники дорожнього руху. Можливо, нам доведеться автоматизувати деякі тести за допомогою тесту джгути та автоматизовані тестові сценарії.

- Створіть набори тестів із тестових кейсів для ефективного виконання тесту. Набір тестів - це логічна сукупність тестових кейсів, які природно працюють разом.

Тестові набори часто обмінюються даними та загальним набором завдань високого рівня.

Ми також встановимо графік виконання тесту.

- Впровадити та перевірити навколишнє середовище. Ми переконуємось що тестове середовище було налаштовано правильно, можливо, навіть було проведено певні тести на ньому.

• Виконання:

- Виконайте тестові набори та окремі тестові кейси, дотримуючись наших процедур тестування. Ми можемо зробити це вручну або за допомогою інструментів виконання тесту відповідно до запланованої послідовності.
- Зафіксувати результати виконання тестування та записати дані та версії програмного забезпечення, що тестується, інструментів тестування та програмного забезпечення. Ми повинні точно знати, які тести ми використовували щодо якої версії програмного забезпечення; ми повинні повідомляти про дефекти щодо конкретних версій; а журнал тестів, який ми ведемо, надає слідку аудиту.
- Порівняйте фактичні результати (що сталося, коли ми проводили тести) з очікувані результати (те, що ми передбачали, відбудеться).
- Там, де є різниця між фактичними та очікуваними результатами, повідомляти про розбіжності як інциденти. Ми аналізуємо їх, щоб зібрати додаткові подробиці про дефект, повідомити додаткову інформацію про проблему, виявити причини дефекту та розмежувати проблеми в програмному забезпеченні та інших продуктах, що тестуються, та будь-які дефекти в даних тесту, в документах про тестування або помилки таким чином, як ми виконали тест. Ми хотіли б реєструвати останні, щоб покращити саме тестування.
- Повторіть тестові заходи в результаті вжитих заходів щодо кожної невідповідності. Нам потрібно повторно виконати тести, які раніше не проходили, щоб підтвердити виправлення (тестування на підтвердження або повторне тестування).

Ми проводимо виправлені тести та комплекти, якщо в наших тестах були дефекти [12]. Ми знову перевіряємо виправлене програмне забезпечення, щоб переконатися, що дефект справді був виправлений правильно (тест підтвердження) і що програмісти не виявляли дефектів у незмінних областях програмного забезпечення, а також, що виправлення дефекту не виявляло інших дефектів (регресійне тестування).

2.2 Створення тест-кейсів

Написання тестових прикладів є основним видом діяльності і вважається однією з найважливіших частин тестування програмного забезпечення. Він використовується командою тестування, командою розробників, а також керівництвом.

Якщо для програми немає документації, ми можемо використовувати тестовий приклад в якості базового документа. Так чому ж так важливі тестові приклади? Типове визначення тестового прикладу - це «набір умов, при яких тестувальник визначатиме, чи працює додаток, програмна система або одна з її функцій належним чином». Це означає, що тестові приклади роз'яснюють, що потрібно зробити для тестування системи. Він дає нам кроки, які ми виконуємо в системі, значення вхідних даних, які ми вводимо в систему, разом з очікуваними результатами, коли ми виконуємо конкретний тестовий приклад.

Тестові приклади об'єднують весь процес тестування. Якщо тестові приклади готові, вони дійсно допоможуть визначити, чи виправдалися очікування клієнта щодо погоди. Коли ми виконуємо тестові приклади, ми можемо отримати більше дефектів, які можуть бути пропущені при спеціальному тестуванні[15]. Ми можемо отримати відповіді на наведені нижче питання з тестових прикладів і отримати уявлення про продукт і про те, де він стоїть. Питання наступні: [контрольний список icon = "check" iconcolor = "" circle = "no"] Які модулі були протестовані? Скільки тестів було виконано? Скільки модулів або функцій стабільні? Які модулі вимагають додаткової роботи в залежності від кількості дефектів, виявлених у відповідних модулях?

Випробовуються чи достатні вхідні комбінації і розглядаються їх в тестових прикладах? Чи видає додаток правильні повідомлення про помилки, якщо користувач використовує його не так, як передбачалося? Тестувалося чи додаток в іншому браузері або на різних мобільних платформах для перевірки

сумісності? Чи відповідає призначений для користувача інтерфейс специфікаціям? Чи відповідають функції вимогам специфікації? Чи всі вони охоплені? Чи можна відстежити користувача сценарії до документа варіанти використання? Чи всі вони охоплені? Чи достатньо хороші тестові приклади? Чи знаходять вони дефекти? Чи готове програмне забезпечення до виробництва? Чи достатньо тестування?

Якщо вдасться отримати правильні відповіді на ці запитання, ми зможемо прокоментувати стабільність всього програми [11]. Ми можемо використовувати його як контрольний список, щоб переконатися, що ми нічого не пропустили. Також важливо виміряти охоплення тестуванням. Ось чому тестові приклади надзвичайно важливі!

Створення якісних сценаріїв тестових випадків вимагає знання того, що ви тестуєте і чому, а також вимог користувачів і того, для чого призначено додаток. Як тільки ці речі стануть відомі, стає можливим писати ефективні сценарії тестування. Сценарії тестування можуть бути засновані на функціональних або не функціональних вимогах. Які вимоги мають пріоритет, залежить від того, що тестується. Структура сценарію тестового прикладу заснована на списку вимог в поєднанні з взаємодією з користувачем і, при необхідності, операційним середовищем.

Це вимагає певної кількості деталей, щоб дозволити тестувальникам легко виконувати сценарії, і кожен сценарій може мати більше одного тестового прикладу [13]. Важливо, щоб опис тестового прикладу було максимально простим і зрозумілим. В ідеалі тестові приклади повинні бути простими для розуміння і не вимагають пояснень.

Тестувальник повинен бути повідомлений не тільки про кроки, які необхідно виконати, а й про будь-якої інформації або засобах автоматизації, необхідних для успішного виконання тесту.

Розробка тестових прикладів може бути складним завданням, тому що сценарії тестових прикладів часто призначені для охоплення ряду

можливостей, і тому інструкції можуть бути досить розпливчастими або складними. Ось чому важливо мати якомога більше документації, що стосується різних етапів будь-якого конкретного тесту. Це включає в себе всі необхідні попередні умови, а також умови публікації і очікувані результати.

Існує тенденція намагатися змусити даний тест робити більше, ніж потрібно. Це ускладнює сценарій і додає громіздке кількість кроків, які можуть збивати з пантелику і сповільнювати тестування.

Тести повинні бути простими і зрозумілими, і ніякі тести не повинні містити більше кроків, необхідних для правильного визначення того, що тестується [14]. Тестові приклади не повинні охоплювати більше двадцяти кроків, а в ідеалі - менше п'ятнадцяти.

Пам'ятайте, що ваша кінцева мета при написанні сценаріїв тестових випадків - створити тести, які легко адмініструвати, при цьому дотримуючись документацію. Це включає в себе забезпечення 100% покриття всіх зазначених вимог.

Тестування з тест-кейсів все ж виграє. Але це зовсім не означає, що даний підхід буде хороший для будь-якого проекту і для будь-якої компанії. При промисловому тестуванні великих проектів використання сценарного підходу принесе вам більше користі і буде доцільним. Перш ніж писати тест-кейси слід в першу чергу оцінити обсяг і складність проекту, необхідність автоматизації, яка стає все більш популярною. Якщо такий аналіз був проведений, впровадження тестових сценаріїв на проекті буде максимально ефективним для забезпечення високої якості продукту. А якщо ви працюєте з невеликими мобільними додатками, яких у вас багато - більш раціонального використання чек листи, тест-кейси в такому випадку будуть тільки навантажувати ваш процес.

2.3 Експертне опитування

Експертне опитування - різновид опитування, в ході якого респондентами є експерти - висококваліфіковані фахівці у певній галузі діяльності.

На сучасному етапі розвитку суспільства, в період найбільших науково-технічних і соціально-економічних перетворень, зростає потреба в отриманні достовірної та обґрунтованої інформації як основи прийняття управлінських рішень. У більшості випадків джерелами такої інформації виступають найбільш компетентні, що володіють глибокими знаннями про предмет дослідження, включені в проблему фахівці - експерти. Масові опитування забезпечують отримання даних про громадську думку, знаннях і установках людей щодо тих чи інших явищ соціальної дійсності, виявлення їх кількісних параметрів і поширеності. Вони розкривають відображення тієї чи іншої проблеми в суспільній свідомості, але не дають розуміння можливих причин і способів її рішення. Крім того, оцінки та самооцінки, висловлені респондентами в масовому опитуванні, часто можуть виявитися спотвореними. Завдання експертного опитування сфокусовані на отриманні обґрунтованої інформації про досліджуваної проблеми, відображеної в думках і оцінках фахівців, що володіють достатнім досвідом і знаннями в області вирішення вузлових проблемних завдань для формулювання висновків і практичних рекомендацій.

Метод застосовується в випадках, коли відсутні інші способи збору даних, що задовольняють цілям дослідження, а також коли об'єкт має специфічними характеристиками, про яких знають тільки професіонали.

Даний метод збору інформації на основі авторитетної думки дозволяє залучити більш цілеспрямовану аргументацію для обґрунтування теоретичних положень і практичних рекомендацій дослідження

Приклади досліджень: проекти, присвячені оцінці обсягу високотехнологічних і спеціалізованих секторів ринку або латентних (прихованих) процесів. Таку інформацію може дати тільки фахівець, методи масових опитувань виявляються неефективними.

Для проведення опитування інтерв'юер повинен володіти достатню компетентність в досліджуваному предметі, а також знати термінологію, яка використовується професіоналами при обговоренні питань з теми дослідження.

Опитування проводилося в соціальній мережі. Охоплення аудиторії на момент проведення опитування становить 2 270 осіб.

Було створено 3 опитування:

- чи використовуєте Ви систему управління тестуванням - для того щоб з'ясувати чи дійсно системи управління тестуванням використовуються у компаніях та є важливим іструментом в процесі тестування програмного продукту
- чи використовуєте Ви на проекті тест-кейси - дане питання було поставлене фахівцям для того щоб з'ясувати, як часто використовуються тест-кейси, і зрозуміти чи дійсно даний тестовий артефакт є важливою складовою процесу тестування
- чи багато часу займає написання тест-кейсів - і нарешті саме ключове питання, відповіді на дане питання дадуть можливість побачити, чи дійсно написання тест-кейсів є досить трудомістким процесом, який в свою чергу може вплинути на швидкість процесу тестування.

На наступному рисунку представление першие опитування.

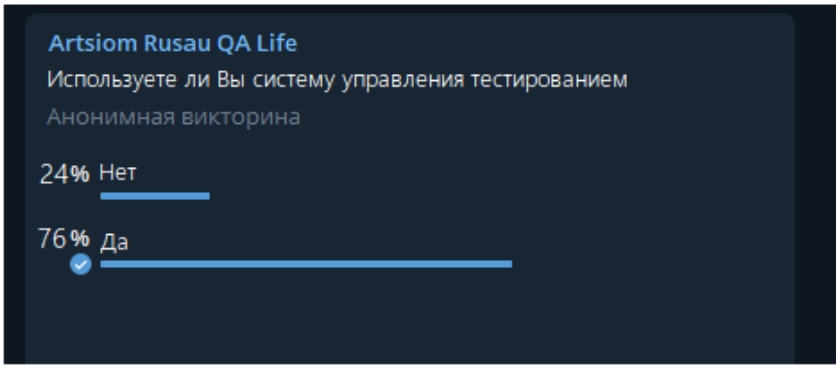

Рисунок. 2.1 - Опитування про використання систем управління тестуванням

На наступному рисунку представлені результати другого опитування.

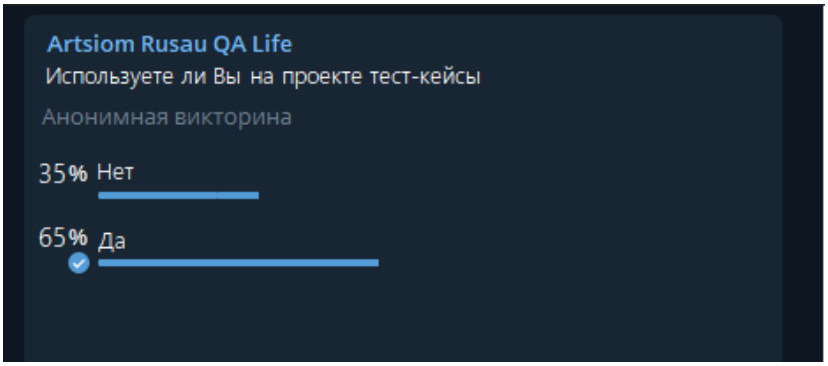

Рисунок. 2.2 - Опитування про актуальність використання тест кейсів

І наступний малюнок на якому представлений третє опитування.

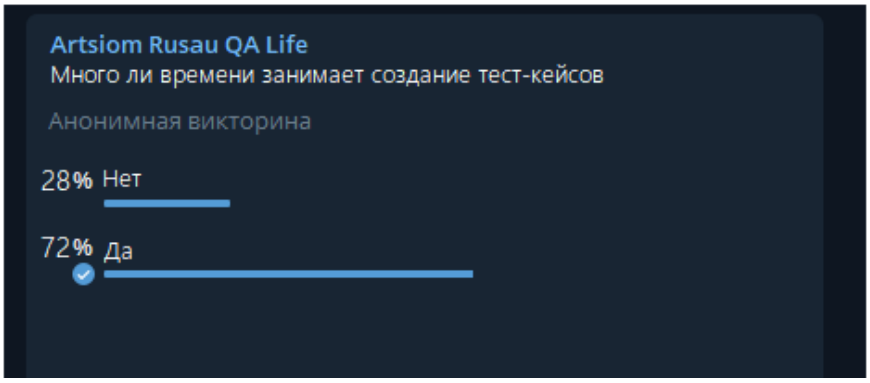

Рисунок 2.3 - Опитування про час використаний на написання тест кейсів

Я ми бачимо результати даного опитування говорять про те що:

- Більшість компаній використовують тест-кейси, це говорить що даний тестовий артефакт є практично невід'ємною частиною більшість проектів, а так само є одним з ключових документів який використовується в процесі тестування програмного продукту. Актуальність говорить про те що, проблеми з трудовитратами на створення тест-кейсів є все ще практично невирішені.
- Більшість компаній використовують системи управління тестуванням. Дані системи допомагають не тільки налагодити процес тестування а так же оптимізувати його зробивши його більш якісним. Що не менш важливо, дані системи дозволяють розробляти, зберігати, а так само використовувати тестові артефакти для отримання метрик і автоматизації, що є так само не мало важливо частиною процесу тестування.
- І останнє питання яке було задане експертам, це скільки часу займає розробка такого тестового артефакту як тест-кейс. Більшість відповіло, що даний процес є досить витратним за часом, а час в даній сфері є одним з найбільш ценніх ресурсів, заощадивши який, можливо дуже значно не тільки прискорити процес тестування, а так само використати даний час на інші важливі частини процесу тестування.

Здійснивши дане експертне опитування, можна зробити висновки.

Перше, що хочу виділити, це те, що хоч і частина компаній не користуються тест-кейсами і специфіка деяких проектів дійсно дозволяє не використовувати даний тестовий артефакт, проте, як ми бачимо, за результатами опитування, актуальність використання даного тестового артефакту все ще дуже значима, і досить багато компаній використовують

даний артефакт для проведення як ручних тестів, так і для автоматизації тестування.

Так само в опитуванні було розглянуто такий інструмент як система управління тестуванням, на дане питання так само большість людей відповіли що використовують даний інструмент, і це дійсно в даний час є дуже затребуваним інструментом, так як прогрес не стоїть на місці і з'являється все більше проектів які досить масштабні а так само володіють складною логікою, в зв'язку з цим вимагають більше співробітників, більше часу, і більше ресурсів, і подібні проекти потребують дуже якісного управління.

3 ПІДХІД ДО АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ТЕСТОВИХ ПРИКЛАДІВ НА ОСНОВІ ВАРІАНТІВ ВИКОРИСТАННЯ НА ЄТАПІ ВИМОГ

3.1 Особливості використання методу генерації тест-кейсів

Основною метою роботи є створення тестових кейсів із випадків використання. У сценарії реального часу доводиться стикатися з кількома проблемами, такими як неточність, неоднозначність та неповнота вимог, це тому, що вимоги не належним чином оновлюються після різних запитів на зміну.

Для подолання цих проблем розробляється рішення, яке генерує тестові кейси на ранніх етапах розвитку системи цикл, який враховує максимальну кількість вимог.

Оскільки вимоги найкраще враховуються випадками використання, наша увага зосереджена на формуванні тестових випадків із діаграм випадків використання.

У програмній інженерії найпоширенішим визначенням тестового випадку є набір умов або змінних, за яких тестувальник визначає, частково чи повністю задовольняються вимоги чи варіанти використання заявки.

Створення тестових кейсів додає додатковий рівень точності варіанту використання. Тестові кейси зазвичай розробляються на основі вихідного коду програми. Це ускладнює створення тестових кейсів, особливо для тестування на кластерних рівнях.

Далі цей підхід виявляється неадекватним при розробці програмного забезпечення на основі компонентів, де вихідний код може бути недоступним для розробників.

Тому бажано автоматично створювати тестові кейси з програмного забезпечення проектні документи, а не код або специфікації на основі коду. Генерація тесту з проектної документації має додаткова перевага дозволяє

тестовим кейсам бути доступними на початку циклу розробки програмного забезпечення, роблячи тим самим планування тестів більш ефективним. Крім того, в тестах на основі проекту згенеровані дані тестів не залежать від будь-якого конкретного здійснення проекту.

Перевага створення тестового кейсу полягає в тому, що він допомагає ідентифікувати важливі поняття у випадку використання. У цій роботі запропонована нова схема для створення тестових кейсів з використанням схем випадків.

Після того, як було зафіксовано варіант використання, першою частиною є перевірка випадку використання, щоб забезпечити його реалізацію. Створення тестових кейсів додає додатковий рівень точності варіанту використання. Деякі проблеми, з якими можливо зіткнутися, полягають у тому, що згенеровані тестові кейси можуть бути неповними, кожен із випадків не описує достатньо деталей використання, вони можуть не оновлюватися при зміні вимог, неточності, відсутності всієї функціональності.

Додатковою перевагою цього підходу є те, що створення тестових кейсів допомагає визначити важливі поняття у кожному випадку використання.

У розробці використовуються класи для представлення концепцій, тому така рання номінація кандидатських класів є корисною.

Тестові випадки є ключовими для процесу, оскільки вони визначають та повідомляють умови, які будуть впроваджені в тесті, і необхідні для перевірки успішного та прийнятного виконання вимог до продукту.

Усі вони спрямовані на те, щоб виріб відповідав вимогам системи.

Отже, автоматичне створення тестових кейсів із випадків використання є перевагою для тестування програмного забезпечення.

Розробка сценарію використання починається рано, тому це дає можливість охопити ключові функціональні можливості продукту, доступні на ранніх ітераціях. Приклади використання повідомляють замовнику, чого очікувати, розробнику, що кодувати, технічному автору, що задокументувати,

а тестувальнику, що тестувати. Для тестування програмного забезпечення є створення тестових кейсів

фундаментальний крок, а потім тестові процедури розроблені для тестових випадків і, нарешті, тестові сценарії.

UML - це мова для візуалізації, специфікації, конструювання та документування артефактів програмних систем [1].

Мова складається зі словника і правил, що дозволяють комбінувати входять до нього слова і отримувати осмислені конструкції. У мові моделювання словник і правила орієнтовані на концептуальне і фізичне уявлення системи. Мова моделювання, подібний UML, є стандартним засобом для складання "креслень" програмного забезпечення.

Моделювання необхідно для розуміння системи. При цьому єдиною моделі ніколи не буває достатньо. Навпаки, для розуміння будь-якої нетривіальної системи доводиться розробляти велику кількість взаємопов'язаних моделей [18]. У застосуванні до програмних систем це означає, що необхідна мова, за допомогою якого можна з різних точок зору описати уявлення архітектури системи протягом циклу її розробки

Словник і правила такого мови, як UML, пояснюють, як створювати і читати добре певні моделі, але нічого не повідомляють про те, які моделі і в яких випадках потрібно створювати. Це завдання всього процесу розробки програмного забезпечення. Добре організований процес повинен підказати, які потрібні артефакти, які ресурси необхідні для їх створення, як можна використовувати ці артефакти, щоб оцінити виконану роботу і керувати проектом в цілому. є найбільш широко використовуваною мовою для генерації випадків використання, багато дослідників використовують діаграми UML, такі як діаграми стану, діаграми випадків, діаграми послідовностей тощо, для створення тестових випадків, і це призвело до створення моделі генерація тестових кейсів.

Незважаючи на те, що було запропоновано різноманітні підходи, з появою таких інструментів моделювання, як Rational Rose, протягом десятиліття постійно проводяться дослідження щодо створення тестових кейсів на основі специфікацій та дизайнерських моделей. Для зручності розуміння ми класифікували підходи до створення тестових випадків переважно на дві категорії.

1. Генерація тестового випадку на основі специфікації.

2. Генерація тестових кейсів на основі моделі

Багато дослідників та практиків працюють над створенням оптимальних тестових кейсів на основі специфікацій;

Існує кілька дискусій щодо того, як використовувати кейси можуть допомогти здійснити процес тестування на початку життєвого циклу розробки. Якобсон та ін. пояснювані тести можна отримати з варіантів використання трьох типів: по-перше, тести очікуваного потоку подій; по-друге, тести незвичного потоку подій; і по-третє, перевірка будь-яких вимог, що додаються до випадку використання.

Наступним кроком є створення сценарію на основі потоку подій, перш ніж його можна буде використовувати для створення тестових випадків.

Однак автоматизація тестування може знизити вартість та підвищити надійність та ефективність тестування програмного забезпечення.

3.2 Використаний підхід до автоматичного генерування тест- кейсів

Запропоновано кілька підходів для генерації тестових кейсів, переважно випадковий, орієнтований на шлях, цільовий та інтелектуальний підходи.

Випадкові методи визначають тестові випадки на основі припущень щодо розподілу дефектів.

Методи, орієнтовані на шлях, зазвичай використовують інформацію про потоки управління для ідентифікації набору шляхів, які потрібно охопити, та генерування відповідних тестових випадків для цих шляхів.

Ці методи можна додатково класифікувати як статичні та динамічні.

Статичні прийоми часто базуються на символічному виконанні, тоді як динамічні прийоми отримують необхідні дані, виконуючи програму, що тестується. Була використана ця динамічна техніка при розробці цього модулю.

Припущення

Передбачається, що попередні умови для кожного сценарію отримує користувач системи.

Для ефективного створення тестових випадків сценарії використання слід виводити з кожного випадку використання.

Принаймні один сценарій може бути виведений з кожного випадку використання, і для кожного сценарію буде створено принаймні один тестовий приклад.

Діаграма варіантів використання відображає функціональні можливості і вимоги системи з використанням діючих осіб і варіантів використання. Варіанти використання моделюють служби, завдання, функції, які повинна виконувати система. Варіанти використання представляють функціональні можливості високого рівня і то, як користувач буде звертатися з системою.

Сценарій використання - це екземпляр варіанту використання або повний "шлях" через варіант використання.

Список цілей, створений в процесі написання сценарію використання, можна використовувати для визначення складності та вартості системи. Зосередивши увагу як на користувача, так і на системі, реальні потреби системи можуть бути ідентифіковані на ранніх етапах процесу проектування. Оскільки варіанти використання написані в основному на мові оповідання, вони легко зрозумілі зацікавленим сторонам, включаючи клієнтів,

користувачів і керівників, а не тільки розробникам і тестувальникам. Створення розширюються з варіантів використання і ідентифікація винятків з успішних сценаріїв використання економить час розробників, спрощуючи визначення тонких системних вимог. Визначаючи межі системи в рамках проекту варіанту використання, розробники можуть уникнути розповзання масштабів.

Передчасного проектування можна уникнути, зосередивши увагу на тому, що система повинна робити, а не на тому, як вона повинна це робити. Крім того, варіанти використання можна легко перетворити в тестові, зіставивши загальний курс і альтернативні курси і зібравши тестові дані для кожного із сценаріїв. Ці функціональні тестові приклади допоможуть групі розробників переконатися, що всі функціональні вимоги системи включені в план тестування.

3.3 Опис реалізації розробленого модуля системи управління тестуванням для удосконаленням підходу до управління тестуванням

Головною метою цієї розробленої системи є створення тестових випадків із випадків використання. Протягом багатьох років було запропоновано ряд різних методів для створення тестів. Тестові кейси також можна взяти з системних вимог.

Однією з переваг створення тестових кейсів із сценаріїв використання є те, що вони можуть бути створені раніше в життєвому циклі розробки та бути готовими до використання до того, як будуть побудовані програми.

Крім того, коли тестові кейси генеруються на початку, інженери-програмісти часто можуть виявити суперечності та неоднозначності у специфікації вимог та проектній документації. Це, безумовно, знизить вартість побудови програмних систем, оскільки помилки усуваються на початку життєвого циклу.

Найважливішою частиною випадку використання для створення тестових кейсів є потік подій. Дві основні частини потоку подій - це основний потік подій та альтернативні потоки подій.

Основний потік подій повинен охоплювати те, що "зазвичай" відбувається, коли виконується сценарій використання. Чергові потоки подій охоплюють поведінку необов'язкового або виняткового характеру щодо нормальної поведінки, а також варіації нормальної поведінки.

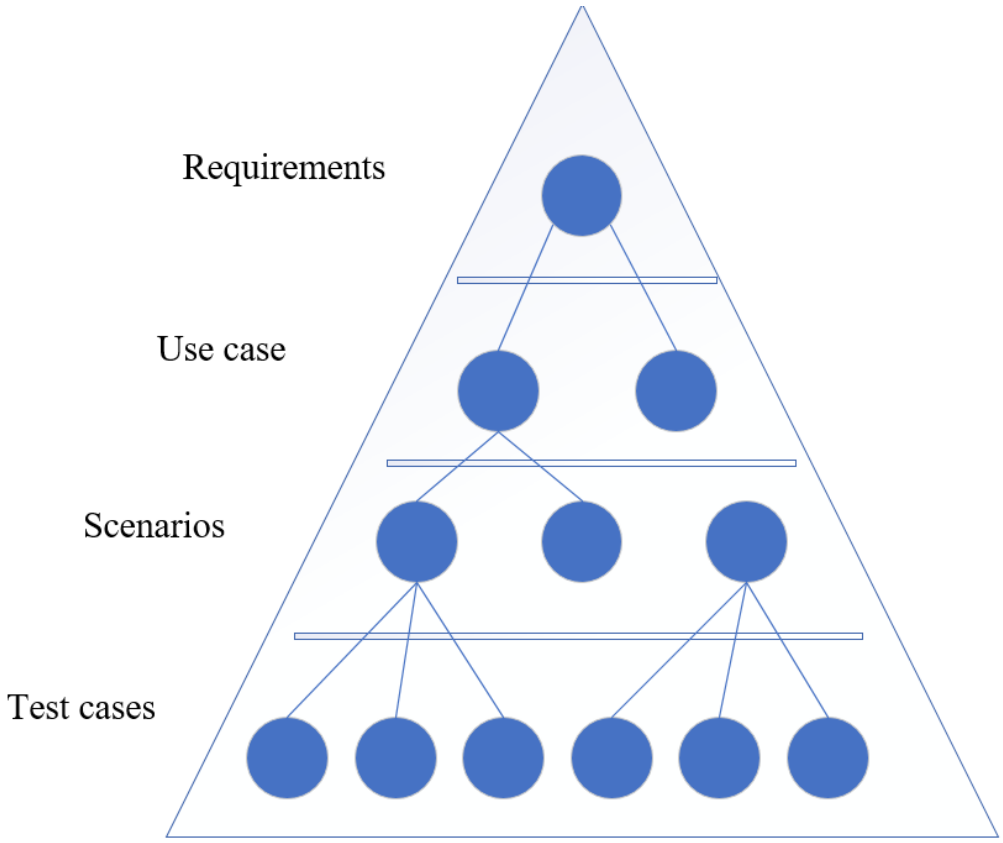

Рисунок 3.1 - Потік подій

Варіанти використання описують функціональні вимоги. Крім того, кожен випадок використання відображає багато сценаріїв.

Тоді відображення випадків використання до сценаріїв є взаємозв'язком один із багатьма. Сценарії складаються для тестування також у відносинах один до багатьох. У цій системі задіяні три основні компоненти: діаграма використання, сценарій використання та тестові приклади.

На діаграмі випадків використання овали представляють випадки використання, а фігури на паличці - "дійові особи", якими можуть бути або люди, або інші системи.

Рядки представляють спілкування між актором та варіантом використання. Кожен випадок використання представляє велику частину функціональних можливостей, яка буде реалізована, і кожен актор представляє когось або щось поза нашою системою, яке взаємодіє з ним.

Найважливішою частиною сценарію використання для генерації тестових кейсів є потік подій.

Дві основні частини потоку подій - це основний потік подій та альтернативні потоки подій.

Основний потік подій повинен охоплювати те, що "зазвичай" відбувається, коли виконується сценарій використання.

Чергові потоки подій охоплюють поведінку, необов’язковий або винятковий характер щодо нормальної поведінки, а також варіації нормальної поведінки. Ви можете думати про альтернативні потоки подій як про "від'їзди" від основного потоку подій.

Сценарій використання - це екземпляр варіанту використання або повний "шлях" через варіант використання.

Кінцеві користувачі завершеної системи можуть пройти багато шляхів, виконуючи функціональність, зазначену у випадку використання.

Наступним основним потоком буде один сценарій.

Слідом за основним потоком плюс альтернативний потік 1А буде іншим. Базовий потік плюс альтернативний потік 2А був би третім тощо.

Тестовий випадок - це сукупність вхідних даних тесту, умов виконання та очікуваних результатів, розроблених для певної мети:

наприклад, для здійснення певного програмного шляху або перевірки відповідності певній вимозі.

Метою тестового кейсу є визначення та передача умов, які будуть впроваджені в тесті.

Тестові кейси необхідні для перевірки успішного та прийнятного виконання вимог до продукту (випадки використання).

Створено робочий простір для креслення діаграми використання, в якому включено всі вимоги до креслення UCD.

Після того, як всі основні вимоги зібрані, початковим кроком є складання схеми використання.

Першим кроком є збір деталей щодо кожного випадку використання.

Значна кількість деталей йде на повне уточнення використання справа.

Всі зібрані деталі зберігаються в базі даних і отримуються з бази даних, коли це необхідно.

Друге, на чому слід сконцентруватися, перш ніж ми використовуємо кейси для генерації тестових кейсів, - це сценарії використання.

Сценарій використання - це екземпляр випадку використання або повний шлях до випадку використання.

Сценарій - це поєднання основних потоків та альтернативних потоків. Слідом за основним потоком буде один сценарій.

Один базовий потік і один альтернативний потік були б іншим, один основний потік плюс два альтернативні потоки були б іншим і продовжуються, з цих сценаріїв використання будуть сформовані тестові випадки для даного випадку використання.

У проекті розробки програмного забезпечення випадки використання визначають вимоги до системного програмного забезпечення. Розробка варіантів використання починається рано, тому реальні випадки використання ключових функціональних можливостей продукту доступні на ранніх ітераціях.

Тестові кейси також можна взяти з системних вимог. Після того, як усі вимоги зібрані, система тепер готова скласти схему використання, для якої

можна створити тестові кейси на основі їх текстових описів. Однією з переваг створення тестових кейсів із специфікацій та дизайну є те, що вони можуть бути створені раніше в життєвому циклі розробки та бути готовими до використання до того, як будуть побудовані програми.

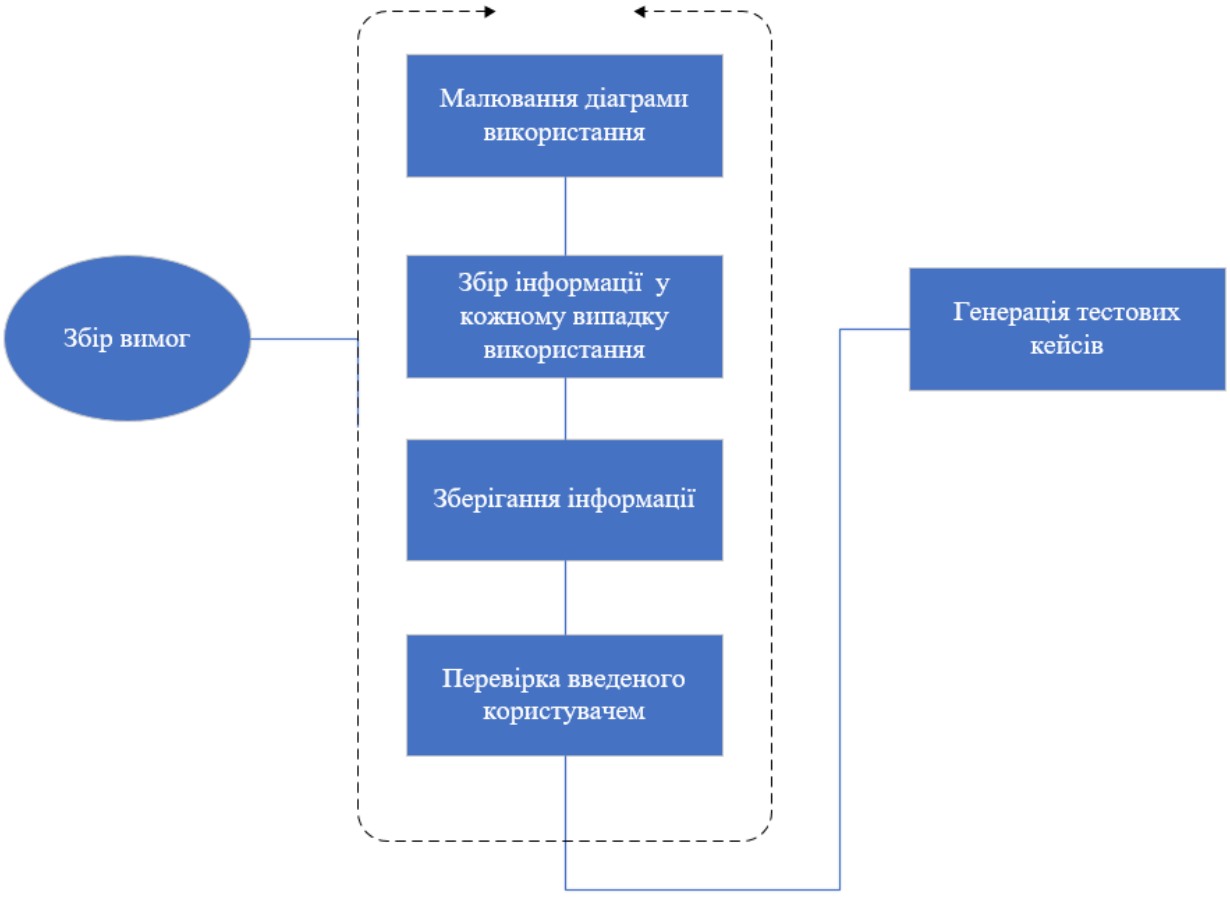

Рисунок 3.2 - Системний фреймворк

Далі пропоную розглянути методику побудови модуля.

1. Проектування робочого простору для малювання Схема випадку використання.

2. Отримання діаграми використання від користувача та внутрішнє збереження всіх деталей.

3. Створення сценаріїв на основі випадків використання.

4. Створення тестових кейсів

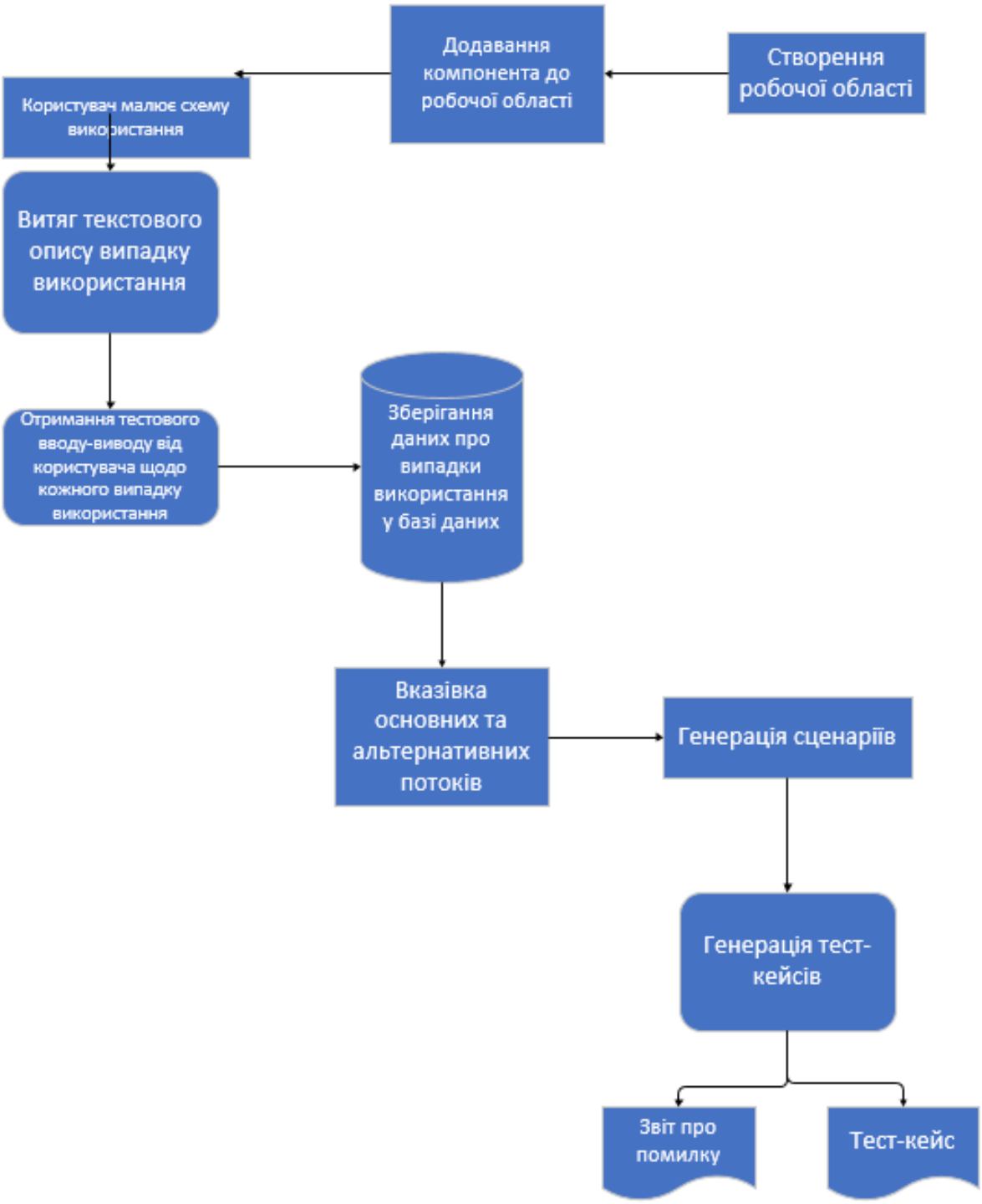

Рисунок 3.3 - Дизайн модулю автоматичної генерації тест-кейсів

Потім ці дані зберігаються у базі даних. Тестові випадки генеруються на основі основних потоків, які є звичайними явищами використання та альтернативних потоків, які є винятковими речами, пов'язаними із випадком використання.

Поєднання основного та альтернативного потоків утворює кілька сценаріїв.

Кожен варіант використання матиме принаймні один сценарій, а кожен сценарій - принаймні один тестовий приклад. На основі кількох сценаріїв будуть сформовані тестові приклади.

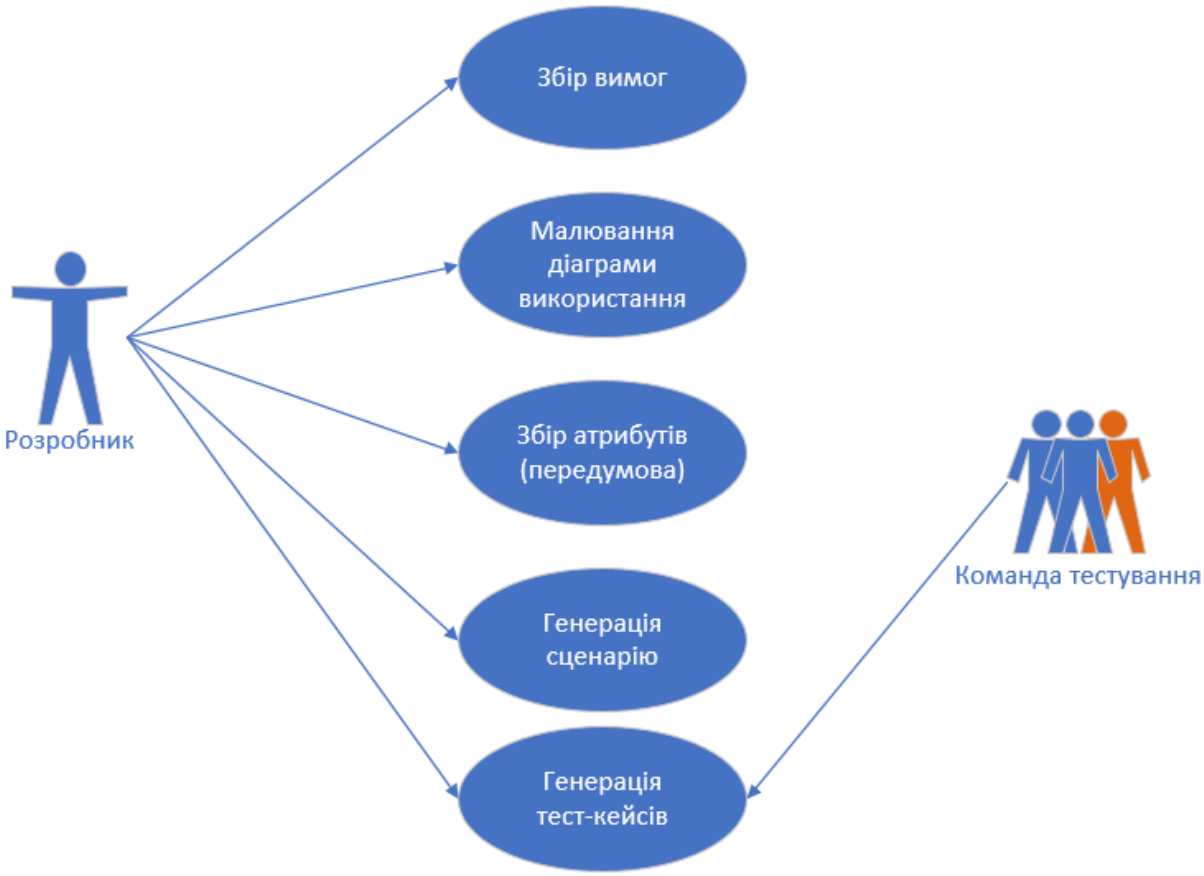

Рисунок 3.4 - Діаграма випадків використання розробленого модуля

У специфікації UML аналіз вимог та проектування зазвичай здійснюється за допомогою діаграм.

Одна конкретна діаграма (діаграма використання) використовується для визначення вимог системи. У діаграмі випадку використання для опису вимог системи використовуються два важливі фактори.

Вони діють і використовують випадки.

Актори - це зовнішні сутності, які взаємодіють із системою, а випадки використання - це поведінка (або функціональні можливості) системи.

Варіанти використання використовуються для визначення вимог системи. Ці випадки використання представляють функціональні можливості системи.

Найчастіше кожен варіант використання потім перетворюється у функцію, що представляє завдання системи.

Тому ми можемо перетворити з кожного варіанту використання в один тестовий приклад або багато тестових випадків.

Зв'язок перетворення є або один до одного, або один до багатьох. Однак якщо у нас багато випадків використання, тоді у нас буде багато тестових кейсів.

Далі пропонується розглянути класифікацію створення робочої області.

Тип компонента – це модуль.

- Визначення

Створено робочий простір, щоб полегшити користувачам малювання діаграм використання.

- Обов’язки

Користувач повинен мати можливість малювати схеми використання в цьому середовищі. Цей робочий простір забезпечений усіма вимогами, необхідними для складання діаграм випадків використання.

- Обмеження

Робочий простір повинен бути спроектований таким чином, щоб користувачі не стикалися з труднощами або проблемами під час виконання ними певних дій у робочій області, таких як малювання, редагування або збереження намальованих діаграм випадків використання. Інтерфейс між користувачем та робочою областю повинен бути хорошим.

- Склад

Робоча область складається з достатньої площі, щоб намалювати схеми використання. Там повинні бути окремі розділи, в яких розміщені різні символи, що будують схеми випадків використання.

- Використання / взаємодія

Проектування робочої області стоїть на першому рівні при формуванні тестових кейсів. Користувачі зможуть складати схеми випадків використання, а деталі кожного випадку використання також зберігаються в базі даних, яку можна отримати пізніше.

- Обробка

Коли користувач малює діаграму використання, усі деталі кожного випадку використання приймаються як вхідні дані від користувача і зберігаються в базі даних.

Виведення сценаріїв для кожного випадку використання

- Визначення

Сценарій використання - це екземпляр варіанту використання або повний "шлях" через варіант використання. Сценарій є

побудовані з використанням основного потоку та почергових потоків.

Базові потоки - це звичайні випадки використання, коли

альтернативні потоки - це виняткові випадки.

Сценарій - це не що інше, як поєднання основних потоків та альтернативних потоків.

- Обов'язки

Сценарій повинен чітко пояснювати всі можливі основні та альтернативні потоки, пов'язані з кожним випадком використання.

Кожен сценарій матиме принаймні один тестовий приклад.

- Обмеження

Кожен варіант використання повинен мати принаймні один сценарій, а кожен сценарій повинен мати принаймні один тестовий приклад.

Зазвичай для кожного випадку використання будується більше одного сценарію. Слідом за базовим потоком буде один сценарій, один основний потік плюс один альтернативний потік буде іншим, один основний та два альтернативні потоки будуть іншим і триває.

- Склад

Для будь-якого випадку використання перший сценарій починається з основного потоку, а згодом поєднується з альтернативними потоками.

З кожним сценарієм буде пов’язана назва.

- Використання / взаємодія

Виведення сценаріїв відіграє важливу роль у формуванні тестових випадків із діаграм випадків використання. На основі

різні сценарії, виведені для кожного випадку використання, буде створено кілька тестових випадків. Сценарії залежать від деталі кожного випадку використання, який подається як тестовий вхід.

- Ресурси

Сценарії будуть сформовані на основі деталей випадку використання.

Ця інформація зберігається в базі даних, яку можна отримати пізніше для створення тестових кейсів.

- Обробка

Користувач надасть усі подробиці щодо кожного випадку використання, які зберігаються в базі даних.

На основі цих деталей з’ясовуються основні та альтернативні потоки, різне поєднання яких будувало сценарії.

Створення тестових кейсів

- Визначення

Тестовий випадок - це сукупність вхідних даних тесту, умов виконання та очікуваних результатів, розроблених для конкретного

об'єктивний.

Тестові кейси необхідні для перевірки успішного та прийнятного виконання вимог до продукту (випадки використання).

- Обов'язки

Тестові кейси є ключовими для процесу, оскільки вони визначають та передають умови, які будуть впроваджені в тесті.

Усі вони спрямовані на те, щоб виріб відповідав вимогам системи.

- Обмеження

Для ефективного створення тестових випадків необхідний набір сценаріїв.

І для генерації кожного сценарію необхідно детальне пояснення кожного випадку використання.

Для кожного сценарію повинен бути принаймні один тест.

- Склад

Тестові випадки генеруються у вигляді табличної колонки, яка складається з декількох сценаріїв, основного та альтернативного потоків і, нарешті, результату, який показує, успішний конкретний тестовий випадок чи ні.

- Використання / взаємодія

Формування тестового кейсу є важливою діяльністю для будь-якої програмної системи, яку слід виконувати.

Генерування тестових кейсів із сценарію використання додає додаткову перевагу для будь-якої системи, оскільки тестування проводиться на ранніх стадіях життєвого циклу розробки програмного забезпечення.

- Ресурси

Ресурсами для створення тестових кейсів є кілька сценаріїв, які будуються з використанням деталей. Ці деталі надаються користувачем як тестові дані.

- Обробка

Матриця тестових кейсів розробляється на основі сценаріїв, які матимуть усі допустимі та недійсні умови.

Це перший крок у формуванні тестових випадків, і нарешті даються значення даних, щоб перевірити, чи дійсні вони чи ні, і генеруються відповідні результати.

Наступним етапом даного розділу є огляд етапу впровадження

Впровадження - це етап проекту, де теоретичний дизайн перетворюється на робочу систему і надає новій системі впевненості для користувачів, що вона буде працювати ефективно .

Він включає ретельне планування, дослідження поточної системи та обмеження на впровадження, розробку методів для досягнення змін, оцінку та зміну методів. На початку фази розробки складається попередній план реалізації для планування та управління багатьма різними видами діяльності, які повинні бути інтегровані в план.

План впровадження оновлюється протягом всієї фази розробки, що закінчується планом переходу на фазу експлуатації. Основними елементами плану впровадження є план випробувань, план тренувань, план встановлення обладнання.

Починаючи з визначення випадків використання, будується спрощений загальний огляд необхідних функціональних можливостей системи.

Через обмеження вираження, діаграми випадків використання самі по собі мало корисні - якщо окремі випадки використання не уточнені загальнотекстовим способом.

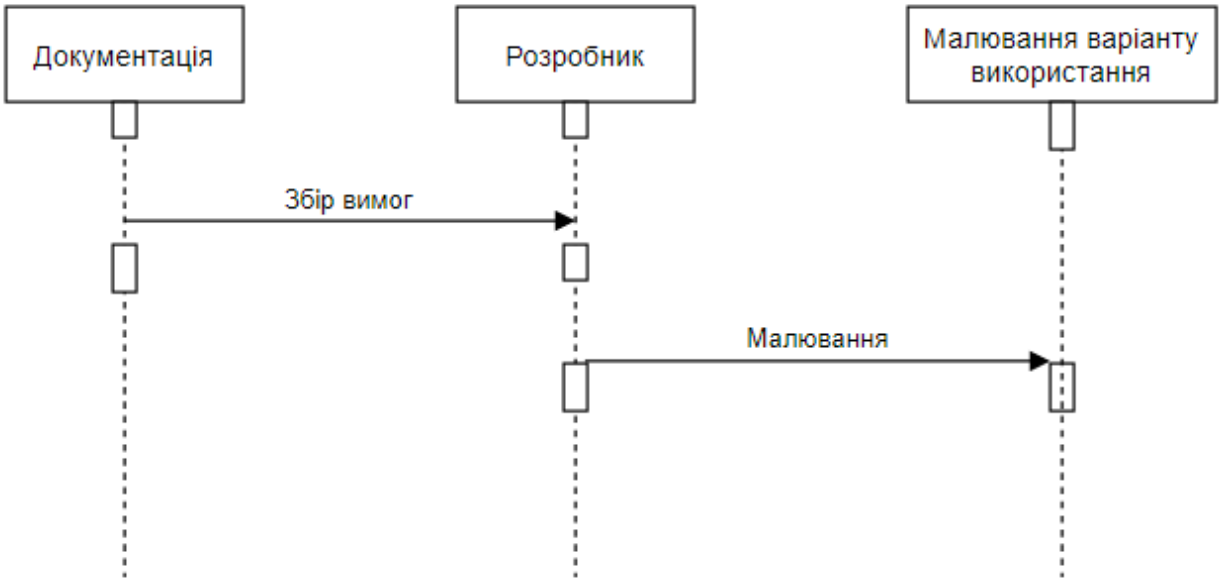

Рисунок 3.5 - Схема послідовності для створення робочого простору

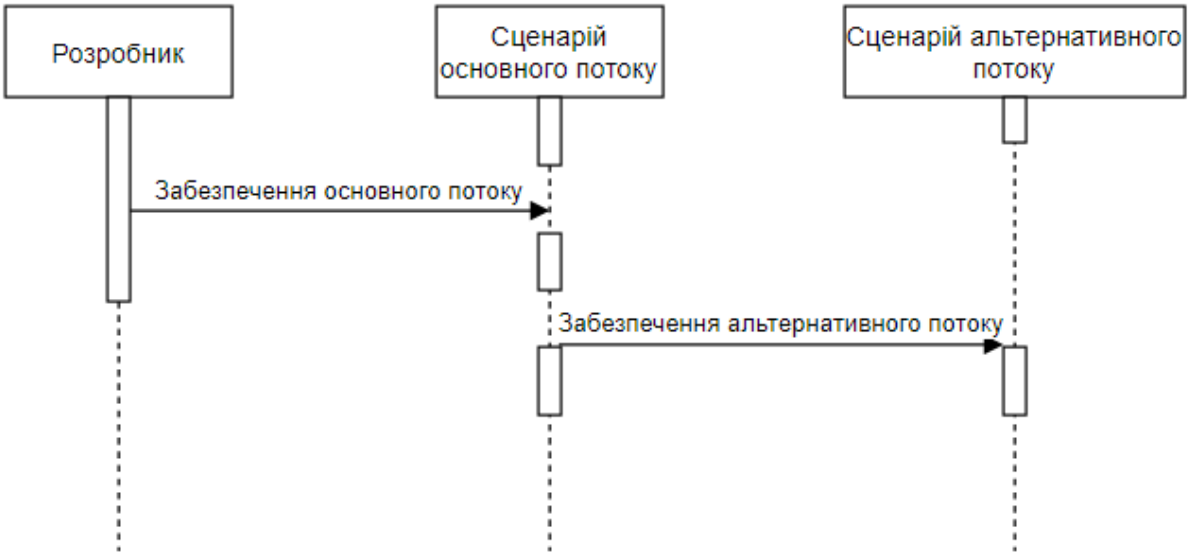

Рисунок 3.6 - Діаграма послідовності для створення сценаріїв для кожного варіанта використання

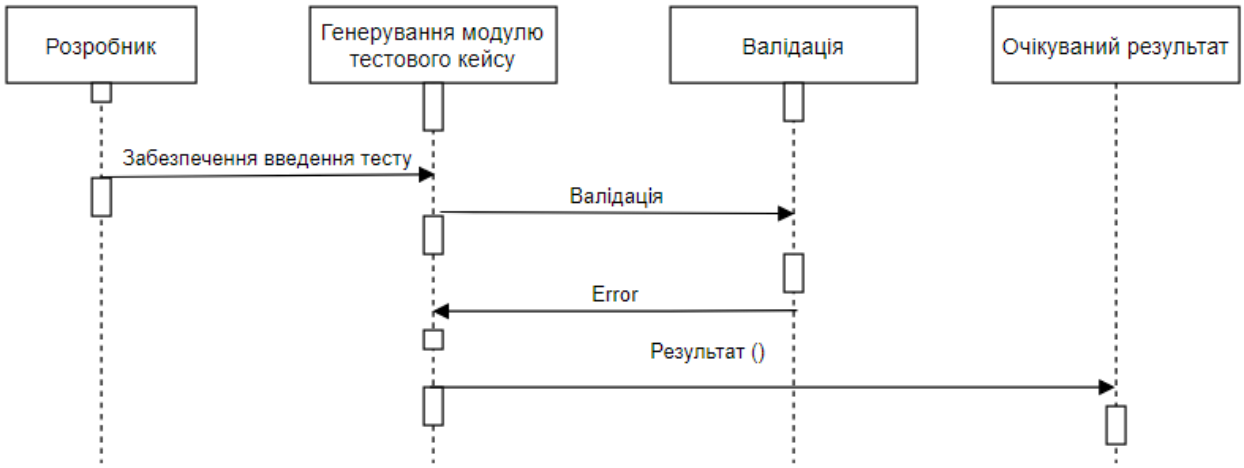

Рисунок 3.7 - Діаграма послідовності для генерації тестового випадку

На наступному зображенні представлений майбутній інтерфейс даного модуля, за допомогою якого можливо буде виробляти малювання діаграм [20].

У лівій частині розташовані фігури з використанням яких можливе комфортне побудова діаграм. Центральна область призначена для конструювання діаграми.

У правій частині предосталена можливість відключення і включення сітки, а так само зміна відображення сторінки.

Так само в правій частині інтерфейсу є виподающій список для вибору розміру сторінки. Нижче представлена Кнопка "Генерація тест кейса" натиснувши яку почнеться процес генерації тестово випадку. Нижче є дві кнопки "Редагувати дані" а так само "Очистити стиль за замовчуванням"

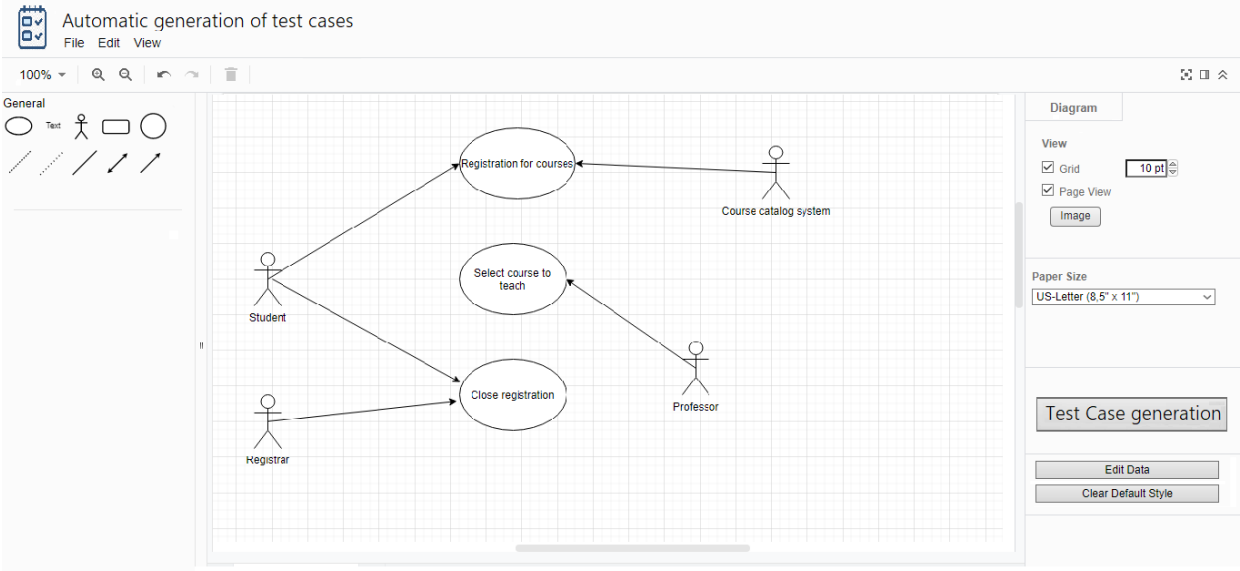

Рисунок. 3.8 - Інтерфейс модуля

У сучасній практиці випадки використання пов'язані з початковим кінцем життєвого циклу розробки програмного забезпечення, а випадки тестування зазвичай пов'язані з останньою частиною життєвого циклу.

Однак, використовуючи випадки використання для створення тестових кейсів, групи тестування можуть розпочати роботу набагато раніше в життєвому циклі, що дозволяє їм виявляти та виправляти дефекти, які було б дуже дорого виправити пізніше, доставити вчасно, і забезпечити надійну роботу системи.

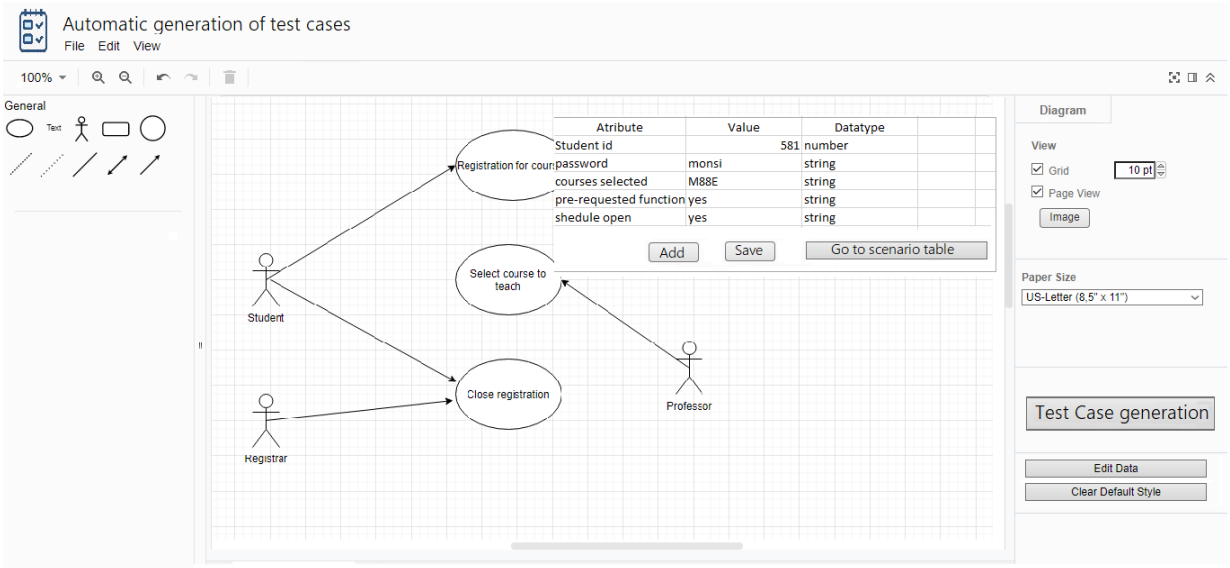

Рисунок 3.9 - Інтерфейс модулю автоматичної генерації тест кейсів

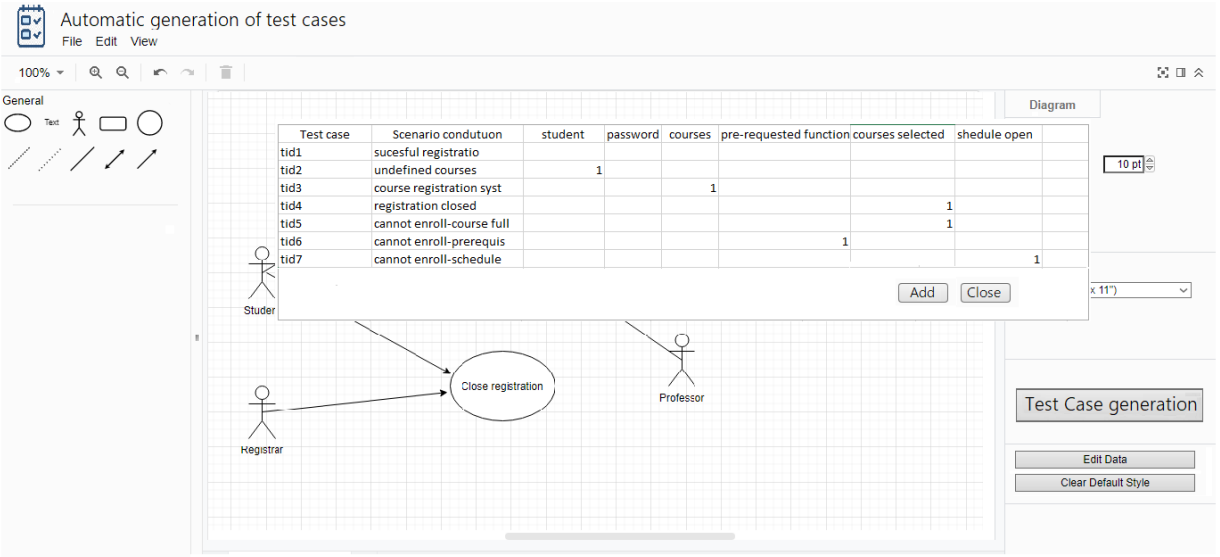

Рисунок 3.10 - Інтерфейс модулю автоматичної генерації тест кейсів

Test id	Scenario condutuon	student	password	courses selecte	pre-requested function	shedule open		Expected result
tid1	sucessful registration	581	monsi	M88E	yes	yes		Succesful
tid2	unifinded student	581	monsi	M88E	yes	yes		Error masage
		581	monsi	M88E	yes	yes		*
tid3	course registration system unavailable	581	monsi	M88E	yes	yes		Error masage
		581	monsi	M88E	yes	yes		*
		581	monsi	M88E	yes	yes		*
tid4		581	monsi	M88E	yes	yes		*
		581	monsi	M88E	yes	yes		Error masage
		581	monsi	M88E	yes	yes		*
tid5	cannot unroll - course full	581	monsi	M88E	yes	yes		Error masage
		581	monsi	M88E	yes	yes		*
		581	monsi	M88E	yes	yes		*

Рисунок 3.11 - Тест-кейс реєстрація на курси

Test id	Scenario condition	courses name	courses selected	Expected result
tid1	Succesful	M88E	yes	Succesful
tid2	Course not found	M88E	yes	Error masage
		M88E	yes	*
tid3	Course not selected	M88E	yes	Error masage
		M88E	yes	*
		M88E	yes	*

Рисунок 3.12 - Тест-кейс вибір курсу для вивчення

Test id	Scenario condition	user name	password	date	Expected result
tid1	Succesful	Viacheslav	12345	07.03.2021	Succesful
tid2	Unknown user	Viacheslav	12345	07.03.2021	Error masage
		Viacheslav	12345	07.03.2021	*
tid3	Registration closed	Viacheslav	12345	07.03.2021	Error masage
		Viacheslav	12345	07.03.2021	*
		Viacheslav	12345	07.03.2021	*

Рисунок 3.13 - Тест-кейс закрити реєстрацію

ВИСНОВКИ

В результаті виконання магістерської атестаційної роботи було проведено аналіз стану проблем тестування та управління тестуванням.

Розглянуто актуальні системи управління тестуванням та з'ясовано, які переваги і недоліки мають існуючі системи управління тестуванням, і що необхідно зробити для їх вдосконалення. З недоліків можна виділити недостатню модульність, завдяки якій можливо розширювати функціонал і робити систему більш гнучкою, так само мало які системи дають можливість автоматизувати створення тестових артефактів, а тільки дають можливість створювати їх більш ефективно порівняно з іншими інструментами які безпосередньо для цього не призначені.

Для досягнення поставленої мети було проведено дослідження, яке складалося як з теоретичної так і практичної частини у вигляді експертного опитування, за допомогою якого були отримані дані, які в свою чергу показують затребуваність не лише систем управління тестуванням, а так само необхідність у використанні тест кейсів, і відбивають той факт, що даний тестовий артефакт є необхідним в процесі тестування, але в свою чергу вимагає досить чималого часу на розробку.

Також був розроблений та описаний модуль, який дозволяє генерувати тест-кейси з use-case діаграм, що також передбачає впровадження концепції модульності в системах управління тестуванням, і є удосконаленням підходу до управління тестування, що в свою чергу дозволить не лише економити багато часу, а також генерувати тест-кейси з великою кількістю комбінацій.

ПЕРЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кендалл С. UML. Основи. / С. Кендалл, 2018. – 192 с.
2. Abhishek K. Automatic Test Case Generation for SOA Based Services / K. ABHISHEK, 2015. – 64 с.
3. Lucian D. A test manager’s guide / D. Lucian, 2017. – 230 с.
4. Glanford M. The Art of Software Testing / M. Glanford, 2017. – 240 с.
5. Lisa C. Agile Testing A Practical Guide for Testers and Agile Teams / Crispin Lisa., 2016. – 534 с.
6. Copeland L. A Practitioner's Guide to Software Test Design / Lee Copeland., 2003. – 300 с.
7. Хуан Л. Дослідження процесів тестування програмного забезпечення на основі графічного інтерфейсу. Пекін, 2014. 113 с.
8. Роберт А. Writing Software Security Test Cases - Putting security test cases into your test // QA Guidelines. 2017. С. 2-5.
9. Майерс Г., Барджетт Т., Сандер К. Мистецтво тестування програм. Лондон: Eye Book, 2011. 371 с.
10. Уіттакер Д., Арбон Д. Як тестують у Google? Вашингтон: Уайли, 2018. 254 с.
11. Test Case Selection and Adequacy /Pre-print for U. Toronto. 2007. URL: http:// www.cs.toronto.edu/~chechik/courses18/csc410/Ch9 AdequacyAndFunctional.pdf (дата звернення 20.10.2020).
12. Рекс Д. Управління процесом тестування: практичні інструменти та методи управління апаратним та програмним тестуванням. Нью-Джерсі: Уайли, 2009. 102 с.
13. Кріспін Л., Грегорі Д. Agile тестування: практичний посібник для тестувальників та Agile команд. / Л. Кріспін. Аддісон-Веслі, 2012. 192 с.
14. Бейзер Б. Тестування методом чорного ящику. Нью-Йорк: Уайли, 1995. 189 с.
15. Кеннер К. Що таке добрий тест-кейс? Нью-Йорк: STAR, 2003. 256 с.

16. Weinberg G. Perfect Software and other illusions about testing / Gerald Weinberg., 2018. – 145 с.

17. Віттакер Д. Як зламати програмне забезпечення: практичний посібник із тестування / Джеймс Віттакер., 2002. – 208 с.

18. Глухих Н. Інтелектуальні інформаційні системи / 2018. – 128 с.

19. Мельников В.С. Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Конференція «Інформаційні інтелектуальні системи» : Міжнародний молодіжний форум. Харків, 2021: Том 6 С. 68.

20. Хряпкин А.В. Мищеряков Ю.В. Евсеев В. В. Метод синтезу адаптивного інтерфейсу взаємодії користувачів з освітнім середовищем з урахуванням їх переваг : Проблеми інформаційних технологій / Херсонський національний технічний університет, 2014. 132 - 135 с.