ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Вдосконалення процесу створення тест-плану для тестування Web-додатку з використанням штучного інтелекту»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Станіслав ЛОЗІНСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти група ПДМ-61

Станіслав ЛОЗІНСЬКИЙ

Керівник: Вікторія КОРЕЦЬКА
к.пед.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лозінському Станіславу Вадимовичу

1. Тема кваліфікаційної роботи: «Вдосконалення процесу створення тест-плану для тестування Web-додатку з використанням штучного інтелекту»

керівник кваліфікаційної роботи Вікторія КОРЕЦЬКА к.пед.н, доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023 р. №145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, вимоги до написання тест-плану.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1.Створення схеми компонентів тест-плану.
2.Аналіз існуючих інструментів штучного інтелекту.
3.Розробка схеми та додатку для автоматизованої генерації тест-плану.

5. Перелік графічного матеріалу: *презентація*

1. Схема тест-плану.
2. Аналіз існуючих інструментів штучного інтелекту.
3. Блок-схема роботи додатку.
4. Таблиця порівняння витрат часу.
5. Практичний результат.

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Вивчення матеріалів для аналізу процесу написання тест-плану	06.11-12.11.23	
3	Дослідження технологій штучного інтелекту	13.11-19.11.23	
4	Розробка схеми написання тест-плану	20.11-26.11.23	
5	Розробка блок-схеми роботи додатку	27.11-03.12.23	
6	Аналіз застосування штучного інтелекту при написанні тест-плану	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Станіслав ЛОЗІНСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Вікторія КОРЕЦЬКА
(підпис)

РЕФЕРАТ

Текстова частина магістерської роботи: 71с., 6 рис., 2 схеми, 9 джерел.

Об`єкт дослідження – автоматизована генерація тест-плану для тестування web-додатку.

Предмет дослідження – метод автоматизованої генерації тест-плану для тестування web-додатку.

Мета роботи – покращення процесу автоматизованої генерації тест-плану для тестування web-додатку.

Методи дослідження – статистичні, методи автоматизованої генерації тест-плану, аналіз, методи проектування та розробки програмного забезпечення.

У роботі розглянуто підходи для реалізації автоматизованої генерації тест-плану для тестування web-додатку. Розроблено метод автоматизованої генерації тест-плану для тестування web-додатку.

Виконано огляд використаних програмних засобів та середовищ розробки. Проведено аналіз існуючих інструментів штучного інтелекту для автоматизованої генерації тест-плану.

ABSTRACT

Text part of the master's qualification work: 71 pages, 6 pictures, 2 table, 9 sources.

The purpose of the work – improved process of automated generation of a test plan for testing a web application.

Object of research – automated generation of a test plan for testing a web-application.

Subject of research – method of automated generation of a test plan for testing a web-application.

Summary of the work: The paper discusses approaches to implement automated generation of a test plan for testing a web application. A method of automated generation of a test plan for testing a web application has been developed.
The software and development environments used were reviewed.
An analysis of existing artificial intelligence tools for automated generation of a test plan has been carried out.

ЗМІСТ

ВСТУП

Недосконале програмне забезпечення може мати величезний вплив на генерацію доходів, надійність і репутацію в довгостроковій перспективі. Так, перш ніж доставити програмне забезпечення замовнику, кожна компанія повинна переконатися, що він працює ідеально, що продукт відповідає всім специфікаціям або вимогам. Є багато випадків, коли незначні недоліки програмного забезпечення призводять до грошових проблем і не тільки. Тому тестування вже стає невід'ємною і значною частиною життєвого циклу розробки програмного забезпечення (SDLC)

Розробка та тестування програмного забезпечення не розглядаються як окремі процеси. Гнучка методологія підкреслює, що програмісти та тестувальники працюють як команда для підвищення якості програмного забезпечення. Процес тестування як невід'ємна частина має на меті оцінити повноту, правильність та якість програмного забезпечення до його поставки. У той же час результати тестування допомагають підприємствам переконатися, що програмне забезпечення відповідає всім визначеним критеріям та умовам.

Ряд досліджень показав, що вартість виправлення програмних помилок зростає, якщо вони не виявлені і зафіксовані на ранній стадії. Коли дефекти виявляються рано, програмістам легше їх виправити. Тому більшість компаній впроваджують тестування на ранніх стадіях процесу SDLC. Потім вони залучають незалежних фахівців для проведення тестування програмного забезпечення на різних етапах розробки.

Незначна помилка в додатку може призвести як до фінансових, так і до людських втрат. Наприклад, невеликий недолік в програмному забезпеченні, використовуваному для створення літака, може викликати непоправні неприємності. IT-компанія повинна виконати ряд тестів для виявлення та усунення будь-яких дефектів, помилок або недоліків у продукті перед доставкою замовнику. Або доручити цю задачу високоспеціалізованій кваліфікованій фірмі.

Щоб зберегти інтерес клієнтів, кожне підприємство повинно забезпечити високу якість програмного забезпечення. Коли продукт перевіряється повністю і багаторазово, його якість можна оцінити найбільш ефективно. За результатами тесту компанія може запустити продукт високої якості, який залишиться на ринку в довгостроковій перспективі.

В роботі розглянуто основні правила написання тест-плану та інструменти штучного інтелекту, які можна використати для автоматизації процесу написання тест-плану. Розроблено систему для автоматизованої генерації тест-плану для тестування web-додатку.

Об'єкт дослідження – автоматизована генерація тест-плану для тестування web-додатку.

Предмет дослідження – метод автоматизованої генерації тест-плану для тестування web-додатку.

Мета роботи – вдосконалення процесу автоматизованого створення тест-плану для тестування web-додатку.

Методи дослідження – математичні: статистичні.

Практична значущість результатів полягає в використанні розробленої системи для автоматизованої генерації тест-плану для тестування web-додатку.

Для досягнення мети вирішено наступні завдання:

1. Аналіз процесу написання тест-плану.
2. Аналіз існуючих інструментів штучного інтелекту для автоматизації процесу написання тест-плану.
3. Розробка системи для автоматизованої генерації тест-плану для тестування web-додатку.

1 АНАЛІЗ ПРОЦЕСУ НАПИСАННЯ ТЕСТ-ПЛАНУ

1.1 Актуальність написання тест-плану

Тестовий план (Testplan) - це документ, що описує весь обсяг тестових робіт, починаючи від опису об'єктів, що підлягають тестуванню, стратегії, розкладу, початку і закінчення критеріїв тестування, до необхідних в процесі тестування, експертизи, а також оцінки ризиків.

Як ми бачимо, план тестування є важливою складовою будь-якого добре організованого процесу тестування, оскільки містить всю необхідну інформацію, що описує процес. Але в більшості випадків тест-план буде грати більш формальну роль, але все ж його наявність має багато переваг. Наприклад:

- Можливість розставити пріоритети тестових завдань.
- Побудова тестової стратегії.
- Можливість стежити за всіма необхідними ресурсами, як технічними, так і людськими.
- Планування ресурсів для тестування.
- Аналіз ризиків.

Залежно від специфікації описаних завдань план тестування може мати два рівні деталізації: майстер тест-план тестування і детальний тест-план.

1.2 Схема тест-плану

На малюнку нижче зображена схема тест-плану відповідно до міжнародного стандарту IEEE 829 із детальним поясненням до неї.

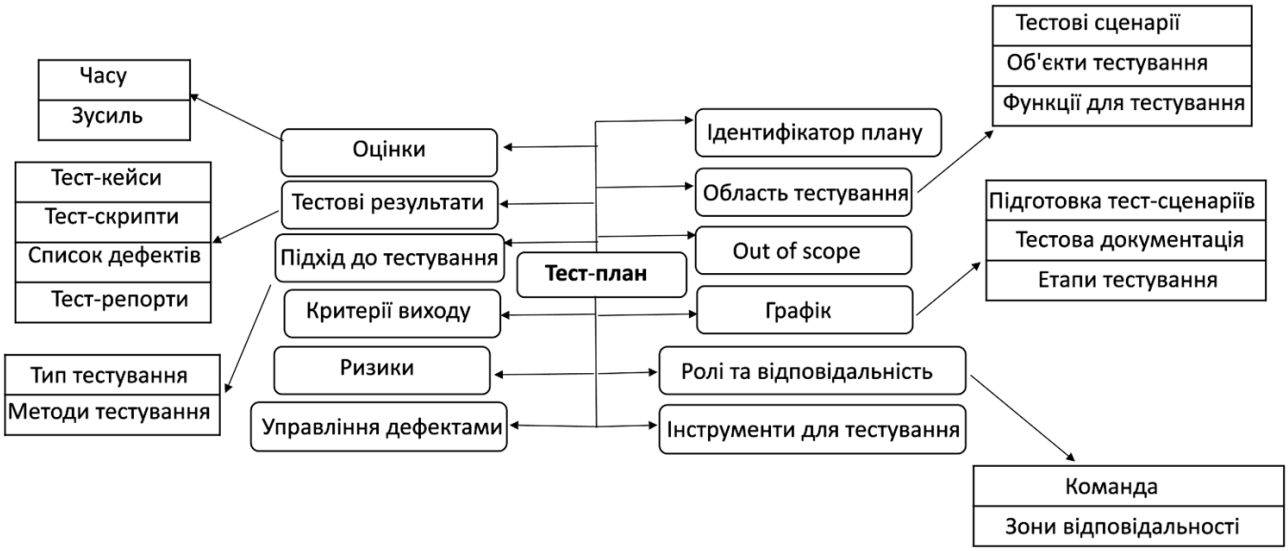


Рисунок 1: Схема тест-плану за міжнародним стандартом IEEE 829

За міжнародним стандартом IEEE 829, тест-план складається із:

- Оцінки часу та зусиль.
- Тестових результатів.
- Підходу до тестування.
- Критеріїв виходу.
- Ризиків.
- Управління дефектами.
- Ідентифікатору плану(на великих проектах).
- Області тестування.
- Out of scope.
- Графіку проведення тестування.
- Ролей та відповідальностей.
- Інструментів для тестування.

Розглянемо кожний пункт більш детально.

1.2.1 Оцінка часу та зусиль

Оцінка витрат на тестування (test assessment): Розраховане наближення результатів, пов'язаних з різними аспектами тестування (наприклад, витрачені зусилля, дата завершення, супутні витрати, кількість сценаріїв тестування тощо), результати яких можуть бути використані навіть тоді, коли вхідні дані є неповними, невизначеними або неточними. (ISTQB)

Test Estimation - це управлінська діяльність, яка приблизно показує, скільки часу та грошей буде потрібно для виконання завдання. Оцінка зусиль для тесту (Оцінка зусиль) є одним з основних і важливих завдань в управлінні тестами (Test Management).

Ми поговоримо про планування в прогнозних методологіях, тому що вони мають час, щоб подати до початку проекту. У гнучких методологіях все відбувається «тут і зараз»: не тільки регулярне планування, але і початкове планування, як правило, займається менеджментом. Це окрема тема.

Якщо ми говоримо про прогнозні методології, на слайді можуть бути цілі етапи. Ось, наприклад, перший етап - Incention - може зайняти до півроку. І це той період, коли вони не програмують і не дуже займаються вимогами. Хто може погодитися, коли нам платять, і ми нічого не робимо належним чином? Це може бути військова, медична промисловість. Це можуть бути глобальні проекти, від яких залежить життя людей, а термін розвитку становить приблизно 2-5-7 років. Зазвичай на планування йде близько 2 тижнів, але часто це просто «завтра».

Що таке тестове планування? Для тест-ліда це не тільки час, але і що, як і де перевірити.

Планування тесту:

- Визначення вимог до тесту;
- оцінка ризиків;
- розробка стратегії тестування;
- Визначення ресурсів;
- Розробка плану випробувань;
- Планування.

Кожне з цих питань варто розглянути. Але ми поговоримо про невеликий пункт «Створення графіка робіт». Звичайно, можна просто сказати про тестування, але краще показати і оформити свій аргумент. Зараз поговоримо про це. Звичайно, планування неможливе без вимірювання результатів - ось наша головна проблема.

«Ми не вміємо оцінювати!» - команда може розповісти, що таке Жуніор, той досвідчений фахівець. Причина вбачається у відсутності певних методів. Це дуже велике непорозуміння.

Наш улюблений - метод проб і помилок. Ми всі використовуємо його часто. Потрібно розуміти, чим він відрізняється від методу експертної оцінки. Якщо ви вже працювали з проектом, знаєте його і щось тестували, то вже робите як експерт. Якщо ви не робили цього для тестування, не працювали з проектом і ніколи не стикалися з цією сферою або замовником, то ви забиваєте повністю в темряві. Є велика різниця.

Мій улюблений метод - це структура роботи розкладання. Це найменш трудомісткий, але формалізований метод оцінки. Що мається на увазі? Потрібно просто розчавити свою роботу до мінімуму, що дуже легко оцінити і навіть вивчити не варто.

Наприклад, тестова авторизація. Буква «о» не проходить - можна просто сказати 5 хвилин за себе. Це мінімальна одиниця. Це може визначити навіть людина, яка працює всього пару місяців. Плюс в цьому методі - вам важче щось забути.

Ви приймаєте ваші вимоги, і велике завдання, ви розділяєте їх на три частини, наприклад: польове тестування, тестування кнопок, тестування локалізації, тестування продуктивності та тестування за видами, методами, областями. Тиснява далі: шляхом введення цифр, букв, символів і т.д.

1.2.2 Тестові результати

Тестовий випадок - це професійна документація тестувальника, послідовність дій, спрямованих на перевірку будь-якого функціоналу, що описує, як прийти до фактичного результату.

Набір тестових випадків називається тестовим набором. Іноді тестовий набір плутають з тестовим планом. План випробувань описує, яку роботу потрібно виконати, як і коли, і що потрібно для завершення продукту.

Тестові випадки повинні допомогти нам перевірити продукт, не пройшовши всю документацію. Написаний один раз, зручний у підтримці тест-кейс заощадить багато часу і сил тестувальникам.

Будь-який тестовий випадок обов'язково включає:

- Унікальний ідентифікатор тестового випадку необхідний для зручного зберігання та навігації наших тестових наборів.
- Назва - це головна тема, або ідея тестового випадку. Короткий опис її суті.
- Передумови - опис умов, які безпосередньо не пов'язані з функціоналом, що тестується, але повинні виконуватися.

Наприклад, тільки зареєстрований користувач може залишити коментар на своєму порталі. Так для тестового випадку «Створення коментаря» необхідно буде виконати попередню умову «користувач зареєстрований» і «користувач авторизований»

- Кроки - опис послідовності дій, які повинні привести нас до очікуваного результату
- Очікуваний результат - результат: те, що ми очікуємо побачити після здійснення кроків.

Не обов'язково, але бажано додати атрибут до історії редагування тесту - це значно полегшить ваше життя. Лаконічний журнал змін, який відображається ким, як і коли був змінений тестовий випадок.

Що ще потрібно знати, перш ніж створити тест-кейс?

По-перше, кожен виконаний тест дає нам один з трьох результатів:

- Позитивний результат, якщо фактичний результат дорівнює очікуваному результату,
- Негативний результат, якщо фактичний результат не дорівнює очікуваному результату. При цьому була виявлена помилка.

Тест блокується, якщо неможливо продовжити тест після одного з кроків. У цьому випадку також була виявлена помилка.

По-друге, один тестовий випадок перевіряє одну конкретну річ, і для цього повинен бути тільки один очікуваний результат.

Чого не повинно бути в тестовому випадку:

- Залежності від інших тестових випадків;
- Неясні кроки або очікуваний результат;
- Відсутність інформації, необхідної для проходження тесту;
- Занадто багато деталей.

Першого слід уникати, оскільки: пов'язаний тестовий випадок завжди можна видалити через невідповідність або його можна змінити, в цьому випадку стає незрозумілим, як виконати тестовий випадок, в якому є посилання.

Також через залежність тестових випадків може виявитися, що тестований продукт вже призведе до правильного стану завдяки виконанню відповідних тестових випадків.

З другим думати все зрозуміло. Якщо опис кроків або очікуваний результат не зрозумілий, він блокує проходження тестового випадку.

Тестовий випадок повинен містити всю інформацію, необхідну для його проходження. Наприклад, якщо ми перевіримо вікно входу на сайт, нам знадобиться логін і пароль, інакше пройти цей скрипт буде неможливо.

Крім того, ви не повинні занадто детально описувати справу. Наприклад, якщо ми перевіримо можливість створення коментаря, не варто писати, в якому кутку екрану має бути вікно входу. Надлишкова інформація лише ускладнює проходження тесту.

Оскільки всі продуктові компанії можуть і, швидше за все, використовують абсолютно різні інструменти для відстеження та виправлення помилок та

пов'язаних з ними процесів, таке програмне забезпечення з часом стало загальною системою моніторингу між різними рівнями управління та технічного персоналу.

Кожен дефект містить набір певних атрибутів, наприклад «тяжкість» і «пріоритет». Іноді нелегко зрозуміти різницю між цими термінами, але кожен поважаючий себе QA-інженер повинен це розуміти.

Строгість (severity) - це особливий атрибут, за допомогою якого можна швидко охарактеризувати рівень технічного впливу дефекту на загальний функціонал тестованого програмного забезпечення.

Кожен ступінь тяжкості помилки більше відповідає функціональності програмного забезпечення, а значить, його дефект призначається виключно тестеру.

Іноді програмісти можуть брати безпосередню участь у визначенні тяжкості дефекту, але в цілому процес є виключно відповідальністю QA інженерів. Це залежить від тестера, щоб оцінити, як конкретна функція продукту може вплинути на загальну продуктивність системи.

Тяжкість жучка можна класифікувати за такими критеріями:

- Blocker - це помилка, яка блокує роботу. Якщо така помилка з'являється в тестованому програмному забезпеченні, подальша перевірка роботи стає неможливою або складною. Для продовження тестування необхідно усунути цю помилку;
- Critical - критичний дефект. Може порушити правильність системи тестованого продукту. Це група помилок, які відтворюються на постійній основі і можуть блокувати перевірку програмного забезпечення;
- Major - значна помилка. ускладнює перевірку і застосування деяких додаткових параметрів;
- Minor - незначна помилка. Може впливати на роботу програмного забезпечення в відносно невеликій мірі або ускладнювати роботу з деякими системами;
- Trivial - банальна помилка. Не впливає на роботу програмного забезпечення, але погіршує загальну думку про проект.

Пріоритет (priority) - атрибут, який може визначати швидкість вирішення проблеми (виправлення помилок).

Коли справа доходить до встановлення пріоритету помилки, мається на увазі, що керівник проекту несе повну відповідальність за роботу.

Існує кілька класифікацій типів пріоритетів:

Top - максимальний пріоритет. Слід призначати найбільш екстремальні ситуації, які негативно впливають на загальне функціонування програмного забезпечення. Такі дефекти повинні бути налагоджені в пріоритетному порядку;

High - висока пріоритетність. Використовується для помилок, які повинні бути виправлені в першу чергу;

Normal є простим (нормальним) пріоритетом. Він призначений для дефектів, які можуть бути вирішені після закриття пріоритетних завдань;

Low - низький пріоритет, який може містити помилки, які не впливають на правильну роботу перевіреної функції. Такі помилки виправляються після виконання всіх важливих завдань і наявності часу і ресурсів.

Звіт - це документ, що містить інформацію про виконані дії, результати виконаної роботи. Зазвичай вона включає таблиці, графіки, списки, які просто описують інформацію як текст. Їх пропорції і зміст визначають корисність і зрозумілість доповіді.

Нам важливо зрозуміти, для кого, для чого і за яких умов ми це робимо і наскільки це покращить сприйняття інформації, яку ми представляємо. Ми повинні пам'ятати, що кожна дія має певну мету. У разі звіту нам важливо зрозуміти, для кого, для чого і за яких умов ми це робимо.

Нижче наведені найбільш відомі версії звітів про випробування.

Звіт про інцидент (incident report)

Звіт про інцидент (incident report): документ, що описує подію, яка сталася, наприклад, під час тесту, і яку необхідно дослідити. (IEEE 829)

Звіт про інцидент може бути визначений як письмовий опис інциденту, що спостерігається під час тестування. Для кращого розуміння почнемо з того, що таке «інцидент». Тестовий інцидент програмного забезпечення може бути визначений

як спостережувана зміна або відхилення поведінки системи від очікуваної. Це може бути відхилення від функціональних вимог або параметрів середовища. Дуже часто інцидент називають дефектом або помилкою, але це не завжди так. Інцидент - це в основному будь-яка несподівана поведінка або програмна реакція, яка повинна бути досліджена.

Інцидент повинен бути розслідуваний і інцидент може бути перетворений в дефект на підставі розслідування. Найчастіше це дефект, але іноді це може статися через різних факторів, наприклад:

- Людський фактор;
- Ніяких вимог або незрозумілих;
- Проблема оточення, наприклад, відсутність відповіді від внутрішнього сервера, що викликає періодичну несподівану поведінку або помилку. або неправильна конфігурація середовища;
- Помилкові дані тесту;
- Неправильний очікуваний результат

Звіт про інцидент покликаний зафіксувати і повідомити про подію зацікавленим особам, провести розслідування. Він схожий на звіт про помилку, можливо, з акцентом на розслідування, обговорення, вплив і може бути призначений не розробникам для уточнення деталей.

Звіт про випробування - це періодичний звіт, який документує детальну інформацію про результати та результати тестування. Він також містить умови, припущення, обмеження тесту, який елемент тесту перевіряється ким. Крім того, вносяться деталі решти робіт, щоб показати, скільки роботи залишилося виконати в проекті.

1.2.3 Підхід до тестування

Тестування - це те, як ви можете бути впевнені у функціональності, продуктивності та користувацькому досвіді. Незалежно від того, чи ви запускаєте свої тести вручну або через автоматизацію, чим раніше і частіше ви можете запускати тести, тим більше шансів виявити помилки та помилки, не тільки

позбавляючи вас та вашу команду можливих вогневих тренувань пізніше, але також гарантуючи, що ваше програмне забезпечення було ретельно протестовано та протестовано, перш ніж воно досягне вас користувачів. Якщо проблеми будуть перенесені в виробниче середовище, то дорожче і дорожче їх буде виправити.

Тестування програмного забезпечення можна розділити на два різних типи: функціональне та нефункціональне тестування. Різні аспекти програмних додатків вимагають різних типів тестування, таких як тестування продуктивності, тестування масштабованості, інтеграційне тестування, унітарне тестування тощо. Кожен з цих типів тестування програмного забезпечення пропонує відмінну видимість у вашому додатку, від коду до досвіду користувача. Давайте докладніше розглянемо деякі з найбільш поширених типів тестування програмного забезпечення.

Функціональне тестування виконується для перевірки бізнес-критичних функцій, функціональності та зручності використання. Функціональне тестування гарантує, що функції та функції програмного забезпечення поводяться так, як очікувалося, без збоїв. В основному, вся програма перевіряється на специфікації, згадані в документі специфікації вимог до програмного забезпечення (SRS). Функціональні типи випробувань включають модульне тестування, тестування інтерфейсу, регресійне тестування та багато інших.

Одноразове тестування фокусується на тестуванні окремих частин/одиниць програмного забезпечення на початку SDLC. Будь-яка функція, процедура, метод або модуль можуть бути одиницею для проходження модульного тестування для визначення його правильності та очікуваної поведінки. Унітарне тестування - перший тест, який розробники виконують на етапі розробки.

Інтеграційне тестування включає тестування різних модулів програмного забезпечення в групі. Програма складається з різних підмодулів, які працюють разом для різних функцій. Метою інтеграційного тестування є перевірка інтеграції різних модулів разом і виявлення помилок і проблем, пов'язаних з ними.

Нефункціональне тестування схоже на функціональне тестування; однак основна відмінність полягає в тому, що ці функції перевіряються під напругою на продуктивність спостерігача, надійність, зручність використання, масштабованість тощо. Нефункціональне тестування, таке як стрес-тестування та стрес-тестування, зазвичай виконується за допомогою інструментів автоматизації та рішень, таких як LoadView. На додаток до тестування продуктивності, типи нефункціональних випробувань включають в себе тестування установки, тестування надійності та тестування безпеки.

Тестування продуктивності - це тип нефункціонального тестування, проведеного для визначення швидкості, стабільності та масштабованості програмного забезпечення. Як випливає з назви, загальна мета цього тесту - перевірити продуктивність програми на різних системних та мережевих тестах, таких як завантаження ЦП, швидкість завантаження сторінки, максимальна обробка трафіку, використання ресурсів сервера тощо. Є кілька інших типів тестування в рамках тестування продуктивності, таких як тестування навантаження і стрес-тестування.

Можливо, у вас є деяке уявлення про вищезгадані різні типи тестування. Все тестування фокусується на надійності і готовності програмних додатків, але давайте краще розберемося в відмінностях між ними на деяких прикладах. Припустимо, у вас є веб-сайт/додаток електронної комерції зі стандартними функціями. Ось кілька прикладів тестування продуктивності, функціонального тестування, інтеграційного тестування та специфічного тестування:

Якщо ви хочете перевірити, як буде працювати ваш сайт, коли на ваш сайт прийде велика кількість користувачів, наприклад під час сезону продажів, потрібно виконати тести навантаження, які підпадають під категорію performance test. Це допоможе вам виявити проблеми швидкості та стабільності та усунути потенційні вузькі місця продуктивності.

Припустимо, ви хочете перевірити вхідні та вихідні дані для кожної функції, такі як реєстрація, логін, доповнення, оформлення замовлення, обробка платежів,

записи в базі даних тощо відповідно до прикладів тестів, записаних у документі SRS. При цьому необхідне функціональне тестування.

Якщо ви хочете перевірити функціональність кошика з інтеграцією касира та модуля оплати, щоб побачити, чи кількість доданих до кошика товарів успішно придбано з правильною оплатою, вам потрібно зробити тест інтеграції.

Якщо ви написали модуль для завантаження продукту і хочете перевірити, чи успішно він і продукти додаються без будь-яких помилок або дефектів, вам потрібно зробити модуль завантаження продукту.

Підсумовуючи, тестування продуктивності використовується для перевірки продуктивності веб-сайту. Функціональне тестування проводиться для перевірки всіх функціональних можливостей. Інтеграційне тестування проводиться для перевірки взаємодії між різними модулями, а модульне тестування проводиться для перевірки правильності окремих частин коду.

Тестування продуктивності:

- Оцінює швидкість і масштабованість сайту/програми.
- Визначає вузькі місця для підвищення продуктивності.
- Виявляє помилки.
- Оптимізація системи і поліпшення функцій
- Забезпечує надійність сайту при великих навантаженнях.

Функціональне тестування:

- Переконайтеся, що веб-сайт/додаток не містить дефектів.
- Забезпечує очікувану поведінку всіх функціональних можливостей.
- Він гарантує правильну архітектуру з необхідною безпекою.
- Покращує загальну якість і функціональність.
- Мінімізує ризики бізнесу, пов'язані з веб-сайтом/додатком.

Інтеграційне тестування:

- Гарантує, що всі прикладні модулі добре інтегровані і працюють разом.
- Виявляє взаємопов'язані проблеми і конфлікти, щоб вирішити їх на ранній стадії, перш ніж вони створюють велику проблему.
- Перевіряє функціональність, надійність і стабільність між різними модулями.

- Виявляє пропущені винятки для покращення якості коду.
- Підтримує конвеєр CI/CD.

Модульне тестування:

- Раннє виявлення помилок у нещодавно розроблених функціях.
- Мінімізує витрати на тестування, коли проблеми виявляються рано.
- Покращує якість коду завдяки кращому рефакторингу.
- Підтримує гнучкий процес розробки.
- Полегшує інтеграцію і дозволяє хорошу документацію.

Оскільки всі ці типи тестів розширюють функціональність і покращують взаємодію з користувачем, недоліків немає. Єдине, що ви можете розглянути, це недолік, в цілому, це час і вартість, пов'язані з тестуванням. Тестування вимагає зусиль і ресурсів, і є ризик, пов'язаний з неточними результатами тестування. Однак, не роблячи веб-сайт/тестування додатків поставить вас в компрометуючу позицію, яка може перешкодити вашому бізнесу і репутації значно.

Тестування продуктивності є обов'язковим у всіх середовищах розробки та виробництва, щоб забезпечити швидкість вашого веб-сайту/програми та витримати очікуване навантаження користувача. Функціональне тестування повинно проводитися з кожною конструкцією, щоб перевірити всі зміни та функції відповідно до специфікацій та вимог. Інтеграційне тестування повинно проводитися при інтеграції нового фрагмента коду з іншим модулем, щоб переконатися, що немає конфліктів і працювати разом правильно. Одноразове тестування повинно проводитися розробниками кожного разу, коли вони пишуть будь-який код для перевірки правильного введення і виведення.

Хоча кожен тип тестування здається окремим завданням, ви можете об'єднати їх жваво, щоб досягти більшої якості продукції. Візьмемо приклад.

Припустимо, ви створили нову веб-сторінку, запустивши тест завантаження (тест продуктивності) за допомогою LoadView Для цієї веб-сторінки як модульного тесту переконайтеся, що коли ви робите остаточну збірку з усіма сторінками, веб-сайт вже оптимізований для обробки високого навантаження користувача в сценаріях пікового трафіку. Це ефективно означає, що тестування продуктивності

є частиною конкретного тестування. Такий підхід допоможе вам зменшити кількість проблем на ранній стадії і заощадити багато витрат і часу в довгостроковій перспективі.

LoadView - це хмарний інструмент для тестування навантаження, який контролює продуктивність веб-сайту в умовах високого трафіку. Він імітує тисячі віртуальних користувачів з різних географічних місць на декількох браузерах і пристроях, щоб створити найбільш реалістичні середовища тестування продуктивності. Він також пропонує функцію створення тестового сценарію за допомогою EveryStep Web Recorder, яка не вимагає жодних навичок кодування, щоб будь-хто у вашій команді міг створювати сценарії та виконувати тестування навантаження. Ви можете протестувати свій веб-сайт, додаток, веб-сторінки та сторонні API за допомогою LoadView, щоб виявити вузькі місця продуктивності та швидко їх виправити. LoadView швидко звикає, і його звіти про продуктивність легко зрозуміти з діями ідеї.

Тестування продуктивності є обов'язковим у всіх середовищах розробки та виробництва, щоб забезпечити швидкість вашого веб-сайту/програми та витримати очікуване навантаження користувача. Функціональне тестування повинно проводитися з кожною конструкцією, щоб перевірити всі зміни та функції відповідно до специфікацій та вимог. Інтеграційне тестування повинно проводитися при інтеграції нового фрагмента коду з іншим модулем, щоб переконатися, що немає конфліктів і працювати разом правильно. Одноразове тестування повинно проводитися розробниками кожного разу, коли вони пишуть будь-який код для перевірки правильного введення і виведення.

1.2.4 Ризики

Отже, в першу чергу, ризики і проблеми часто об'єднуються. Ризик, за визначенням, є існуючим або розвивається фактором процесу, який потенційно негативно впливає на процес і, отже, результат. Ви можете, звичайно, отримати будь-яку проблему до концепції ризику, але чому? У середньому нормальне навчання з управління ризиками складається лише з 20-25% матеріалів про процес

управління ризиками та описи типових ризиків, а в інших 75% часу тренери намагаються втиснути під виглядом ризиків опису проблем процесу під соусом «і ви також можете мати це».... Знову ж таки, чому це так?

Ризик - це існуючий або розвивається фактор процесу, який потенційно негативно впливає на процес.

Простіше кажучи, чітко відрізняти ризик від проблеми: ризик - це те, що може статися і призвести до негативних наслідків, а проблема - це те, що вже сталося або є і заважає роботі. І ризик, і проблема заважають або можуть заважати роботі, але способи роботи з ризиками і проблемами дещо відрізняються: перший повинен спробувати зрозуміти, знайти і, можливо, мінімізувати їх наслідки, перш ніж вони «стріляють», а з проблемами повинен працювати насправді - виправити або «погасити». Поділ ризиків і проблем робить сферу управління ризиками набагато простішою і зрозумілішою.

Простий приклад, який не є ризиком тестування програмного забезпечення, але часто включає його, - це використання єдиного середовища для тестувальників та розробників. Незручна ситуація, яка викликає або може викликати багато проблем, але це джерело проблеми, а не ризик.

Алгоритм роботи з ризиками:

- Визначення
- Аналіз та визначення пріоритетів
- Планування
- Моніторинг та звітність
- Виправлення
- Висновки

Активність ризику є циклічною, як і будь-яка інша діяльність проекту, якщо ви працюєте в ітерації. При цьому, якщо у вас досить довга ітерація, то може бути кілька циклів роботи, пов'язаних з управлінням ризиками, такі внутрішні цикли можна назвати «петлями» для простоти.

Що зазвичай викликає труднощі на початкових етапах роботи з ризиками: власне, почати (перевести ці заходи на робочі місця); розуміти, що робота з

ризиками - це не ракетобудування і що ця діяльність, як і будь-яка інша, повинна бути спланована, забезпечена ресурсами (виконавцями), а результати повинні бути проаналізовані (те, що «звільнено», тобто те, чим ми успішно впоралися тощо).

Цікаво те, що іноді ми не можемо нічого зробити з ризиком або нашого впливу недостатньо, щоб взяти ризик зі списку, і це трапляється. Системні ризики для цього і системні, які не можуть бути повністю виключені, як часто є особливістю процесу, в якому ми працюємо. Розмінування - це ризикований процес, але ми повинні працювати. У цьому випадку ми намагаємося застрахуватися від вогневих наслідків і складаємо інструкції на випадок військових дій.

Я не зупинюся детально, але стадія аналізу отриманих результатів і уроків малювання часто ігнорується, що призводить до повторення невдалого результату на наступних ітераціях - що, по суті, характерно для будь-якого процесу: якщо не проаналізувати, «куди» черговий постріл потрапив знову «десь там»... замість того, щоб йти «в потрібне місце».

Також не хотілося б відволікати увагу від основної теми цієї статті ідеями про правильне постановці цілей, але якщо в плані управління ризиками буде написано «поговорити з начальником про тестер С-С» замість «вирішити питання збільшення С-С на 20%» результат такого завдання в плані швидше за все буде не «піднятий ЗП на 20%», а «поговорив з начальником про»....

Які я хотів би зафіксувати, перш ніж ми почнемо розглядати типові ризики, пов'язані з тестуванням програмного забезпечення. Для того, щоб правильно справлятися з ризиками і ця робота дала результат, потрібно чітко розуміти рівень ризику - рівень своєї відповідальності як керівника тестування або рівень ризиків проекту, працювати з керівником проекту і провідним розробником. Ризики системного або бізнес-рівня, з якими ви працюєте, зазвичай знаходяться поза зоною впливу проектної команди, але команда проекту може бути залучена до підготовки деяких рішень та аналізу поточної ситуації, щоб забезпечити осіб, які приймають рішення, відповідною та зрозумілою інформацією

Що таке проект? Проект з точки зору менеджера - це час, гроші і тепло замовника. Проект тестування - це той самий проект, з тією різницею, що гроші управляються безпосередньо керівниками тестів рідко, але їх ресурси у вигляді людино-годин в ці гроші можна конвертувати або працювати з трудомісткими безпосередньо.

Фредерік Брукс у своїй знаменитій книзі «Міфічний людино-місяць» зазначив, що цей ризик часто є основною причиною нереалізованих або невдалих проектів.

В цілому цей ризик, звичайно, відноситься до рівня проектних ризиків, точніше до ризиків управління проектами. Але, оскільки оцінка витрат на оплату праці проекту включає оцінку витрат на тестування праці, а тестова робота знаходиться на критичному шляху ітераційного плану, ризик часто пов'язаний з неправильною оцінкою тестової праці, яку ми розглянемо наступним індивідуальним ризиком.

Ризик полягає в тому, що тестувальники не беруть участі ні в огляді трудових ресурсів проекту, ні в самих оцінках. Ситуація, в якій оцінки для тестування просто зводяться керівником проекту, клієнтом або кимось іншим, часто клінічна і суперечить основним принципам управління проектом: завдання оцінюється підрядником, інакше виконавець може не взяти на себе завдання або не несе відповідальності за його результат.

Знову ж таки, ризик на рівні проекту, якщо ви говорите про оцінку проектної праці, але він може бути частково керований та мінімізований тестовою групою або її менеджером, включивши тестувальників до процесу отримання оцінок щодо надходження праці та перегляду отриманих оцінок та планів проекту.

Аналогічний попередній ризик, заснований в основному на порушенні принципу «оцінки витрат на оплату праці підрядником», але вже на рівні проектних завдань з тестування.

Крім основної причини «пострілу» цього ризику, також істотно ризикованими факторами можуть бути відсутність неявних вимог, неправильне визначення типів тестів і конфігурацій, в яких буде проводитися тестування. - ці

завдання є найбільш впливовими на обсяг тестування і як наслідок допущеної помилки у виконанні цих завдань, змінюють обсяг тестової роботи та суттєво впливають на план тестування.

Як боротися: огляд та аудити, формальні та «бігові». У цьому місці одна голова хороша, а дві - краще.

Ситуація може бути викликана або ускладнена поєднанням ролей менеджера тестів і дизайнера тестів. При поділі цих ролей проекту на різних членів команди тестування, тестовий дизайнер повинен обґрунтувати і захистити запропоновану стратегію тестування і його оцінку витрат на оплату праці. Такий захист часто працює краще, ніж формальний огляд.

Строго кажучи, це проблема процесу тестування, яка тим часом настільки поширена, що я б рекомендував орієнтуватися на неї як на серйозний ризик.

Тестування та розробка відбуваються на одному ресурсі проекту - вчасно. Якщо плани двох напрямків строго не пов'язані (найкраще на рівні одного загального плану роботи проекту, буквально зв'язок між завданнями в проекті MS або будь-якою іншою подібною системою) або не синхронізовані на постійній основі, є ймовірність або ризик, що зрушення в планах розвитку (що впливає на дату доставки версії при тестуванні) не буде включено до плану тестових робіт, що призведе до нестачі часу на тестування та, як наслідок, до завершення фази тестування.

Чому плани тестування і розробки все одно повинні бути прив'язані строго на рівні єдиного плану проекту: в зоні відповідальності керівника проекту, з фіксованим періодом ітерації, серед іншого, ітерація управління обсягом (обсягом), для чого йому необхідно побачити тестову роботу. Грубо кажучи, під час обмеженої за часом ітерації завдання PM-а полягає в тому, щоб вибрати обсяг функціональності, який команда встигає зробити і перевірити.

Якщо керівнику проекту не потрібно оцінювати витрати на оплату праці при тестуванні (пор. «клініка»), то завдання керівника тесту внести свою роботу в план проекту і пов'язати їх з кореляцією. завдання розвитку. У такій схемі вкрай складно

перенести дати випробувань - якусь частину тестової роботи буде просто видно за терміном виконання в плані або схемах.

Формально це не ризик, а проблема, яка створює ризик того, що стратегія тестування не буде виконана в тій частині завдань, де завдання розвитку перекриваються або не будуть забезпечені ресурсами (часто саме проектний час) і в результаті все одно не виконується.

Як боротися: формальний огляд стратегії або плану тестування зазвичай не допомагає. Формальна апробація в цьому місці часто означає: «Я бачив, що у вас є документ з назвою Стратегія або план, і ви оновили його в цій ітерації». Насправді це слова «добре, головне, щоб ти добре вчився», які не дають тобі, як тестовому менеджеру, можливості отримати необхідне розуміння в команді розробників і співвіднести ресурси для реалізації цієї стратегії і твоїх планів.

Ризик звільнення ключа або не дуже ключового співробітника завжди є.

Проблема не в тому, що люди йдуть, а в тому, що вони йдуть, коли їм це потрібно, не проект, а введення нового співробітника, його навчання і виведення на «проектну потужність» вимагає часу - відповідно. плани сповзають, швидкість падає, всі нервують.

Що робити. Тримати «лавку» не завжди можливо з економічних причин. «Краще годувати» допомагає не завжди, але іноді (у випадку не ключових товаришів) це теж просто не вигідно. Що можна зробити? Найочевидніше, що я бачу, це «домовитися з сусідами» - просто поговорити з сусідніми відомствами або проектами (які теж мають цей ризик) і погодитися, що в разі такого заходу вони можуть хоч якось (якщо дозволяє специфіка продукту і поточне навантаження) допомогти вам людям. Також будьте готові допомогти собі. Так, так, порятунок потопаючих - справа рук самих потопаючих.

Зміна навіть фіксованих вимог або їх пріоритетів часто вважається фактором ризику, який вплине на обсяг ітерації та кореляції. призведе до перегляду планів і може затримати доставку версії. Я б не назвав цю частину дизайнерської роботи ризиком - це реальність, з якою ви повинні працювати як з обмеженням дизайну і намагатися навіть не досягти стану проблеми. Ефективний спосіб полягає в тому,

щоб точно обмежити кількість ітерації за часом, коли будь-яка зміна вимог призводить до того, що якась інша частина роботи (а також розробка та тестування) переходить у наступну ітерацію. Нікому ніколи не вдавалося змусити або адміністративно «примусити» вимоги до змін - зміни бізнесу, зміни вимог і якщо Замовник готовий платити не тільки за природні зміни вимог, але і за свої «фантазії» або «неорганізовані» Ви повинні прийняти це і жити з ним.

Труднощів у роботі тестової групи, пов'язаних з тестуванням, насправді вкрай мало. Я зустрів особливості, описані у вигляді ризиків реалізації програмного забезпечення рівня продукту «відсутність графічного інтерфейсу». Насправді, не будучи ризиком, така особливість проекту або продукту може бути значним обмеженням у стратегії тестування і накладати жорсткі вимоги до кваліфікації персоналу, що бере участь у тестуванні. Знову ж таки, це не ризик, це особливість вашого продукту або проекту. Ви не скаржитеся, що інтерфейс вашого продукту написаний англійською мовою, оскільки він призначений для західного ринку, хоча російську мову може бути простіше перевірити.

На закінчення хотілося б підкреслити досить очевидний, але ігнорований ризик, який полягає в самій ідеї ігнорування ризиків.

Один ризик, який застосовується до всіх рівнів управління ризиками.

Небажання враховувати той факт, що є ризики, що процес (навіть (що зазвичай призводить до прорахунків і ще більше порушує нормальний ритм роботи) і як наслідок до невдач.

Що робити: почати працювати з ризиками (ніби побитий і банальний цей висновок не прозвучав, але іншої рекомендації немає). У тестуванні мало конкретних ризиків. Більшість ризиків на рівні проекту можуть бути вирішені спільно командами тестування, розробки та управління проектами.

1.2.5 Критерії виходу

Метою визначення критеріїв тестування є встановлення на місці набору правил, що регулюють тестовий потік. Дуже важливо визначити критерії випробувань перед початком випробувань. Введення в експлуатацію та

відповідність критеріям випробувань допомагає проекту відповідати цільовим датам, усуваючи втрати часу, які будуть неминучі в тому випадку, якщо випробувальний стенд отримає програмний продукт, не готовий до тестування, або в ситуації, коли тестування триває, продукт буде відправлений на модифікацію.

Існує п'ять типів критеріїв, які можна визначити перед початком тестування системи.

Критерій введення описує, що потрібно зробити перед початком тестування. Наприклад, може знадобитися наявність документів технічного завдання в остаточній формі. Можна поставити завдання, щоб програмний продукт був упакований в тому ж вигляді, в якому він буде доставлений замовнику. Під час тестування може знадобитися мати сервісні програми, конфігураційні файли або дані, які буде використовувати клієнт. Якщо ви несете відповідальність за перевірку документації, що супроводжує програмне забезпечення, вона також повинна бути доступна під час тестування. Одним із способів відповідності критерію вступу є перевірка готовності до тестування. Ця перевірка використовує контрольний список елементів дії, які інші групи погодилися передати як вхідні дані для тестування.

Вихідний критерій описує те, що ви вважаєте за необхідне для завершення тесту. Незважаючи на те, що групи тестування часто намагаються зробити критерій виведення умовою доставки програмного забезпечення, це нереально. Рішення про постачання приймається і повинно прийматися вищим керівництвом розробки. Критерій виходу з тесту повинен звучати так: "Всі заплановані тести були завершені, всі виправлені дефекти були перевірені, всі нові виявлені дефекти були повідомлені. Документально підтверджені елементи плану, такі як невиконання деяких тестів через відмову обладнання ". Як і у випадку з вступним тестом, перевірка готовності допускається для забезпечення виконання всіх тестів і готовності програмного продукту до здачі за результатами тестування.

Критерій призупинення/поновлення описує, що станеться, якщо через дефекти не вдасться продовжити тестування. Іншими словами, якщо випадки

настільки погані, що планові тести не можуть бути проведені, їх необхідно зупинити до виправлення дефектів.

Критерій успішного/невдалого проходження тесту. Проходження кожного тесту повинно давати відомі результати. Якщо передбачуваний результат отримано, вважається, що продукт пройшов тест, інакше тест зазнає невдачі. У той же час може статися, що деякі тести не виконуються, оскільки вони або спотворені, або заблоковані дефектами, або ресурси, необхідні для їх запуску, недоступні. Бажано заздалегідь визначити, що робити з тестами, які неможливо виконати. Може бути заплановано відзначити кожен невдалий тест буквою «N» в підсумковому звіті і пояснити в полі коментарів, що сталося і що було зроблено в контексті вирішення проблеми. Можливо, у ваші плани входить заміна спотворених тестів спеціальним видом тестування та запис результатів у спеціальний акт тестування.

Інші критерії, визначені процесом або стандартами. Якщо програмний продукт повинен відповідати певним стандартам або ваша компанія має певні вимоги до процесу, ймовірно, буде розглянуто ряд додаткових критеріїв. Наприклад, у вашому локальному процесі можна визначити конкретні інструменти, видаючи звіти про дефекти або звіти про тести.

1.2.6 Область тестування

Тестовий сценарій (Test scenario) - це послідовність дій над продуктом, які пов'язані з одним бізнес-процесом обмеженого використання та відповідними перевірками правильності поведінки продукту під час цих дій. Іншими словами, це послідовність кроків, які користувач може зробити для використання вашого програмного забезпечення. Тестові скрипти повинні враховувати всі можливі способи виконання завдання (функції) і охоплювати як позитивні, так і негативні приклади тестів, оскільки кінцеві користувачі можуть не обов'язково робити кроки, які ви очікуєте від них. Використовуючи тестові сценарії, ми оцінюємо додаток з точки зору кінцевого користувача. Насправді, якщо весь тестовий сценарій успішно

пройдений, можна зробити висновок, що продукт може виконувати покладену на нього функцію.

Як писати тестові сценарії:

- Уважно ознайомтеся з вимогами (специфікація вимог до бізнесу (BRS), специфікація вимог до програмного забезпечення (SRS), специфікація функціональних вимог (FRS)) перевіреної системи (SUT), випадки використання книг, посібників тощо;
- Для кожної вимоги дізнайтеся, як користувач може використовувати програмне забезпечення всіма можливими способами;
- Створити список тестових скриптів для кожної перевіреної функції програми (AUT);
- Створіть матрицю простежуваності і прив'яжіть всі скрипти до вимог. Це дозволить вам визначити, чи відповідають всі вимоги тестовим сценаріям чи ні;
- Надішліть тестові сценарії керівнику для їх розгляду та оцінки. Навіть тестові сценарії додатково перевіряються всіма зацікавленими сторонами.

Тестовий сценарій не слід плутати з тестовим набором.

Ми можемо бачити тестові об'єкти як компоненти, які система повинна тестувати. Ми також називаємо їх тестовими.

Простіше кажучи, це автоматизована система. Але як далеко? Ми хотіли б знати, що ми робимо і що ми не повинні перевіряти, якщо тільки зробити хороший план.

Важливо заздалегідь визначити тестові об'єкти. Ми робимо це в тестовому плані, але в цьому звіті ми повідомляємо про цю відмінність. Таким чином, всі учасники знають, де вони знаходяться. Це єдиний спосіб дізнатися, чи працює система добре. Для визначення необхідно знати наступне:

Розмір, обладнання або програмне забезпечення

Це програмна система? (SaaS) або апаратна система? Також можливе поєднання обох.

Як правило, системи дуже великі, і ми визначаємо тестовий об'єкт як тестування певного підмодуля, такого як рішення SaaS. У цьому випадку ми повинні знати, який суб-под є тестовим об'єктом. Ми також повинні знати, де суб-подуле знаходиться в системі.

Важливо знати, перш ніж ми почнемо помилки, деякі помилки не обмежуються межами тестованого модуля, але також можуть вплинути на інші модулі. Більш того, можливі наслідки для бізнес-процесів поза компанією.

Тестовий план Het - це документ, що описує обсяг, підхід, ресурси та планування запланованих дій тестування. Серед іншого, він визначає

- Тестові об'єкти.
- Функції для тестування.
- Тестові елементи.
- Хто виконає кожне завдання.
- Ступінь незалежності тестера.
- Тестове середовище.
- Використовуються методи проектування тестів.
- Критерії входу та виходу.

План також описує причини вибору випробувальних об'єктів і будь-які ризики, які вимагають аварійного планування. Ми вказуємо критерії, які ми будемо використовувати, щоб визначити, чи пройшов тестовий предмет тест. План тестування є результатом процесу планування тестування.

Постачальники SaaS, які тестують своє програмне забезпечення, особливо хочуть задовольнити своїх клієнтів. Вони завжди шукають правильний чоловік або жінка для правильного тесту. Зрештою, ми не можемо. Глибокі знання бізнес-процесів також часто бажані. Щоб знайти тестувальників, які точно відповідають індивідуальним вимогам тесту, ми часто виділяємо широкий спектр тестових об'єктів. Сайти, SaaS, програми, ігри, обладнання для Інтернету тощо. Є всі види програмного забезпечення та цифрових продуктів, кожен зі своїми професіоналами.

Однак методи випробувань можуть сильно відрізнятися залежно від типу об'єкта, що перевіряється. Наприклад, додатки та SaaS можна легко розширити на кілька користувацьких пристроїв і зробити доступними на хмарних платформах. Однак деякі ігри вимагають додаткового обладнання, наприклад консолі. Тому розробники програмного забезпечення повинні також протестувати пристрої перед випуском, а потім відправити їх тестерам.

Можна виділити наступні типи тестових об'єктів:

- Сайти як тестові об'єкти

Якщо у нас виникнуть проблеми з зручністю використання і функціональністю, це може негативно позначитися на іміджі нашого бренду. Це може призвести до втрати плинності після цього. Наприклад, широке тестування в інтернет-магазині може запобігти багатьом проблемам.

- Мобільні додатки як тестові об'єкти

Нативні, гібридні або веб-додатки - iOS, Android, Windows Phone! Розробники мобільних додатків регулярно стикаються з новими проблемами, коли справа доходить до тестування.

- Інтернет речей і носимих речей

Інтелектуальні пристрої в ньому Інтернет речей просять інтелектуальні гнучкі тестові рішення, які відповідають всім нашим потребам тестування. Від розумних годинників до прив'язаного одягу, постачальники все частіше інтегрують технології в носимі пристрої. Розумні домашні рішення, підключені автомобілі, всім їм потрібен власний підхід.

- Ігри як тестові об'єкти

Будь то в Інтернеті, на комп'ютерах, у віртуальній реальності або як додаток, ігри бувають різних форм. Наприклад, для перевірки функціональності цього програмного забезпечення необхідний досвід користувача.

- Чат-боти та віртуальні помічники

Мовні помічники, такі як Siri або Alexa і чат-боти, були незамінними в повсякденному житті протягом тривалого часу. Однак ці програми вимагають постійної оптимізації. Ми краще перевіримо їх перед випуском.

Припустимо, нам потрібні корективи в нашому модулі кошика покупок інтернет-магазину. Нові розробки впливають на те, як ми приймаємо платежі кредитною карткою за товари та послуги. Однак змін у платіжному модулі обробки операцій на картці немає. Тому ми розглядаємо інтернет-магазин тільки як тестовий об'єкт. Модуль оплати повинен працювати в звичайному режимі. Щоб класифікувати, ми призначаємо тестовий об'єкт для веб-додатків. Однак ми просимо звернути особливу увагу на різні пристрої, з якими працюють клієнти.

Однак ця відмінність не означає, що якщо ми виявимо помилку поза об'єктом тестування, ми не повідомимо про неї. Продовжуємо про це повідомляти. Але це випадковий улов. Однак це не той випадок, коли ми повністю тестуємо всі інші модулі.

1.2.7 Інструменти тестування

Інструменти тестування потрібні для будь-якої галузі та програми, будь то розробка програмного забезпечення, продукти IoT (IoT), веб-або мобільні додатки або навіть платформи без коду, такі як AppMaster. Ці програмні додатки призначені для поліпшення, автоматизації та спрощення процесу тестування, гарантуючи, що ваш продукт або додаток відповідає галузевим стандартам і відмінно працює в різних середовищах.

У сучасній індустрії швидких технологій існує широкий спектр інструментів тестування для задоволення різних потреб. Деякі інструменти спрямовані на тестування програмного забезпечення, інші призначені для тестування IoT-систем або веб-додатків і мобільних пристроїв. Незалежно від галузі та застосування, важливо вибрати правильний інструмент тестування для максимального використання ваших ресурсів та отримання високоякісного продукту.

Світ розробки програмного забезпечення величезний і різноманітний, як і інструменти тестування, створені для його підтримки. Ось огляд деяких популярних інструментів тестування програмного забезпечення, які можуть допомогти поліпшити якість і продуктивність ваших програмних проектів:

- JUnit: Він підтримує методи затвердження, тестові анотації та тестові запуски, пропонуючи міцну основу для тестування додатків Java.
- TestNG: Альтернативою JUnit, TestNG (Next Generation) також є система тестування на основі Java з потужними функціями як для модульного, так і для функціонального тестування. TestNG є популярним вибором для складних проектів Java завдяки підтримці паралельного виконання тестів, гнучкості конфігурації та індивідуальної конфігурації.
- Selenium: широко використовуваний інструмент з відкритим кодом для тестування веб-додатків, Selenium автоматизує операції браузера для тестування веб-додатків на різних платформах, браузерах і конфігураціях. Підтримуючи кілька мов програмування, таких як Java, C # і Python, Selenium дозволяє користувачам писати тестові скрипти на бажаній мові.
- JIRA - це інструмент для відстеження проектів, розроблений, щоб допомогти командам розробників програмного забезпечення керувати своєю роботою та відстежувати прогрес. Завдяки потужним можливостям запитів, настроюваним робочим процесам та інтеграції з іншими інструментами Atlassian, JIRA є популярним вибором для тестування управління протягом усього життєвого циклу розробки програмного забезпечення.
- TestRail: призначений для централізації та оптимізації управління тестами, TestRail - це комплексний інструмент веб-тестування, який допомагає командам організовувати, виконувати та оцінювати свою тестову діяльність. Завдяки таким функціям, як настроювані набори тестів, інтегроване відстеження проблем та детальні звіти, TestRail добре підходить для управління великими та складними проектами тестування.

З швидким поширенням IoT-пристроїв дуже важливо мати надійні інструменти тестування, які можуть забезпечити їх продуктивність, безпеку та функціональність. Ось деякі першокласні інструменти тестування IoT, які допоможуть зробити ваші проекти IoT успішними:

- Wireshark, широко використовуваний аналізатор пакетів з відкритим кодом, дозволяє користувачам перевіряти, аналізувати та усувати несправності зв'язку між пристроями та мережами IoT. Завдяки потужним можливостям фільтрації та пошуку та підтримці численних протоколів, Wireshark є безцінним інструментом для виявлення та вирішення проблем IoT зв'язку.
- MQTT.fx: Завдяки таким функціям, як динамічна фільтрація, шаблон корисного навантаження та підтримка JSON, MQTT.fx допоможе вам перевірити ефективність та надійність ваших IoT-систем на основі MQTT.
- IBM Watson: Завдяки вбудованій підтримці захищеного зв'язку, потужної аналітики та зберігання даних IBM Watson IoT Platform дозволяє розробникам тестувати функціональність, безпеку та масштабованість своїх IoT-систем.
- IOSTUDIO: Орієнтований на процес розробки пристроїв IoT, IOSTUDIO надає можливості тестування та прототипування для апаратних та мікропрограмних пристроїв. Завдяки моделюванню віртуальних пристроїв, налагодженню обладнання та підтримці сценаріїв IOSTUDIO може заощадити час та зменшити витрати на розробку та тестування пристроїв IoT.
- Postman: Хоча Postman не спеціально розроблений для тестування IoT, це популярний інструмент розробки API, який допоможе вам протестувати API REST та GraphQL ваших систем IoT. Завдяки потужній обробці запитів/відповідей, управлінню змінними середовища та можливостям автоматизованого тестування Postman може стати чудовим доповненням до інструментарію тестування IoT.

Вибір правильних інструментів тестування для розробки програмного забезпечення, IoT або інших пов'язаних з технологією проектів може значно поліпшити якість, продуктивність і безпеку ваших продуктів, забезпечуючи кращі можливості для кінцевих користувачів. Вивчаючи різні інструменти тестування, враховуйте свої конкретні вимоги та пріоритети, а також оцінюйте особливості та можливості, які найкращим чином відповідають потребам вашого проекту.

Інструменти тестування веб-додатків відіграють важливу роль у забезпеченні належної функціональності, зручності та продуктивності веб-додатків у браузерах, пристроях та операційних системах. Ці інструменти допомагають розробникам та команді QA визначити та вирішити проблеми, які можуть вплинути на досвід кінцевого користувача, надійність додатків та безпеку. Ось деякі з кращих інструментів тестування веб-додатків:

- Selenium: Selenium - широко використовуваний фреймворк з відкритим кодом для автоматизації браузерів, що робить його популярним вибором для тестування веб-додатків. Він підтримує багато мов програмування, таких як Java, C # і Python. Завдяки WebDriver API, Selenium дозволяє міжбраузерне тестування та тестування сумісності для широкого кола браузерів, включаючи Chrome, Firefox, Safari та Edge.
- Katalon Studio: Katalon Studio - комплексне рішення для автоматизованого тестування веб-додатків, мобільних додатків і API-тестування. Завдяки зручному інтерфейсу та підтримці декількох мов сценаріїв, Katalon Studio дозволяє легко створювати, запускати та керувати тестами. Його інтеграція з такими інструментами CI/CD, як Jenkins і Bamboo, дозволяє оптимізувати робочий процес тестування і ефективно створювати високоякісні додатки.
- Cypress: Cypress - це потужна наскрізна система тестування веб-додатків. Це дозволяє розробникам писати і виконувати тести безпосередньо в браузері, забезпечуючи кращу налагодження і зворотний зв'язок в режимі реального часу. Cypress підтримує сучасні бібліотеки та фреймворки JavaScript, включаючи React, Angular і Vue, що робить його популярним вибором для сучасних команд веб-додатків.
- Jest: Jest - популярний фреймворк для тестування JavaScript, який особливо добре підходить для тестування React-додатків. Він має простий API, потужні схвалення і вбудований генератор звітів покриття коду, що робить його відмінним вибором для тестування складних веб-додатків. Jest також підтримує паралельні тести, що робить їх більш ефективними.

Тестування мобільних додатків може бути більш складним, ніж веб-додатків, через величезну кількість пристроїв, дозволів на екран і операційних систем. Інструменти тестування мобільних додатків допомагають тестувальникам і розробникам переконатися, що їх програми функціонують правильно і забезпечують безперебійну роботу на різних пристроях. Деякі з кращих інструментів тестування для мобільних додатків включають:

- Appium: Appium - це система порівняльного аналізу з відкритим кодом для мобільних додатків, яка працює як з платформами Android, так і з iOS. Він підтримує кілька мов програмування та інтеграцію з CI/CD конвеєрами, що робить його популярним вибором для автоматизації тестування мобільних додатків. Appium допомагає розробникам і QA-командам ефективно тестувати функціональність, продуктивність і зручність використання додатків на різних пристроях і платформах.
- Espresso - це середовище тестування інтерфейсу користувача для Android додатків, розроблених Google. Він пропонує потужний і високонадійний API для моделювання взаємодії з користувачем, що робить його відмінним інструментом для тестування користувацьких інтерфейсів додатків Android. Espresso також добре інтегрується з Android Studio, забезпечуючи безперебійний процес тестування в середовищі розробки.
- XCUITest: XCUITest - це система тестування інтерфейсу користувача для додатків iOS, що надаються Apple. Він заснований на XCTest і легко інтегрується з Xcode, дозволяючи розробникам писати і виконувати тести користувацького інтерфейсу безпосередньо з середовища розробки. Вбудована підтримка iOS в XCUITest забезпечує точність і надійність при тестуванні на різних пристроях і версіях ОС.
- Detox: Detox - це середовище тестування додатків React Native. Він забезпечує швидке і надійне середовище тестування для мобільних додатків, виконуючи тести в ізоляції. Detox дозволяє розробникам писати та запускати тести за допомогою JavaScript, що робить його відмінним

вибором для тестування функціональності, продуктивності та користувацького інтерфейсу React Native додатків.

Оскільки ми розглянули різні типи інструментів тестування та їх важливість у різних галузях та додатках, очевидно, що наявність правильного набору інструментів тестування є важливим для створення високоякісних програмних та апаратних продуктів. При виборі тестового інструменту важливо враховувати його зручність використання, можливості автоматизації, інтеграцію з іншими системами і підтримку новітніх технологій і стандартів.

Організації повинні оцінити свої унікальні потреби і вимоги, щоб знайти найбільш підходящі інструменти тестування, відповідні їх цілям і завданням. Забезпечення хорошого дизайну, ефективності та ретельності в процесах тестування в кінцевому підсумку призведе до більш надійних, безпечних і найбільш ефективних продуктів.

Крім того, всі платформи розробки можуть отримати вигоду від інтеграції потужних інструментів тестування та методологій у свої бізнес-процеси. Наприклад, платформа AppMaster, потужне рішення без коду, пропонує безліч інструментів тестування та функцій, призначених для оптимізації процесу розробки в своєму унікальному середовищі. На закінчення, інвестування в правильні інструменти тестування є важливим кроком на шляху до досягнення успіху у ваших проектах розвитку.

Зосереджуючись на вдосконаленні та вдосконаленні стратегій тестування, ви зможете забезпечити відмінні продукти та забезпечити безперебійну роботу користувачів. Тому знайдіть час, щоб оцінити, порівняти та експериментувати з різними інструментами, щоб знайти найбільш підходящі для потреб вашої організації та залишатися попереду в постійно розвивається світі технологій.

1.3 Критерії оцінки якості тест-плану

Для оцінки якості тест-плану можна виділити декілька ключових пунктів:

- Інтеграція з іншими системами. На великих проектах, де існує безліч інтеграцій зі сторонніми системами, часто виникають труднощі. Важливо визначити свою зону відповідальності і зрозуміти її межі. Бувають ситуації, коли ми не можемо протестувати новий функціонал. Ще гірше, коли щось на виробництві перестає працювати і стурбований клієнт дзвонить посеред ночі з проханням виправити якомога швидше. І при детальному аналізі виявляється, що це не ваша система, а труднощі інтеграції або проблеми на боці іншого постачальника.
- Нефункціональні вимоги до додатку. Уточнення цього пункту суттєво впливає на тестову стратегію. Клієнти часто пропускають нефункціональні вимоги. Швидкість, безпека і доступність програми, вимоги до локалізації та інтернаціоналізації, особливі вимоги (наприклад, пошук по каталогу інтернет-магазину не більше двох секунд). Ці та інші вимоги важливо уточнити на етапі планування, щоб зрозуміти бюджет, склад команди і частини заявки, до яких висуватимуться найсуворіші вимоги. Це допоможе захистити колектив після виходу продукту: на питання і претензії замовника ви відповісте конкретними документами. Як правило, нефункціональні дефекти дуже дорого виправити. Зверніть увагу на необхідну продуктивність клієнта, доступність, безпеку.
- Тестові дані. Це питання, на жаль, часто задають занадто пізно, безпосередньо перед тестуванням. Але що робити, якщо клієнт не може надати необхідну інформацію так швидко? Наприклад, великі обсяги даних повинні бути спеціально згенеровані для вас, зробити їх максимально схожими на ті, які будуть використовуватися на виробництві, і при цьому маскувати особисту інформацію користувачів (номери паспортів, страховки, водійські права і т.д.).
- Кінцеві користувачі. Наприклад, у вас є конкретне рішення, призначене для людей, які знайомі з його ранньою версією. Для них підказки щодо використання можуть бути зайвими, вони вже знають, яка функція і що вам

потрібно. Але якщо рівень користувачів різний, то порадами та інструкціями нехтувати не варто. І це також додаткова робота для команди тестування. Крім того, якщо додаток буде доступний користувачам з різних країн, потрібно перевірити, які вимоги є на законодавчому рівні (наприклад, додаток повинен бути доступним для користувачів з порушенням зору або слуху). Також важливо розуміти, на яких пристроях люди будуть використовувати ваш продукт. Деякі служби, такі як StatCounter або Google Analytics, надають цю статистику залежно від географії. Це допоможе зосередити зусилля команди і не розпорошуватися на небажаних версіях. Скажімо, 90% наших користувачів використовують 10-ту версію Android і тільки 5% версію 6. У цьому випадку на старті ми будемо тримати на радарі Android 10. Також бувають ситуації, коли основна маса кінцевих користувачів завдяки робочим функціям використовують тільки пристрої з Android 4.1. Для того, щоб уточнити такі деталі, треба задати питання.

- Прилади для тестування. Це питання стає краєм в середині процесу розробки. Часто, якщо у вас немає потрібного пристрою під рукою, ви можете купити його або просто написати лист в інші відділи і позичити пристрій на деякий час. Але бувають виняткові ситуації.
- Обмеження зі сторони клієнта. Дізнайтеся, які процеси в компанії клієнта відрізняються від ваших. Можливо, ви дізнаєтеся про обмеження в інструментах тестування. У моїй практиці був випадок, коли замовник мав певний набір програмного забезпечення та бібліотеки для автоматизації. Це внесло корективи в графік, тому що наші тестувальники повинні були дізнатися деякі з нуля. Через те, що ми не обговорювали цей момент на старті, ситуація спричинила овертайм. Тепер ми завжди ставимо питання про обмеження.
- UAT/Beta-тестування. На одному з завершальних етапів важливо показати продукт невеликій групі користувачів або тестувальникам клієнта. Іноді для цього потрібна спеціальна документація.

- Аудит зі сторони замовника. Деякі клієнти мають власний відділ контролю якості, і одного разу він може прийти до вас з аудитом. Перевірте на старті проекту, які вимоги до документації пред'являє клієнт, чи є у нього шаблони. Це допоможе вам не тільки задовольнити ваші потреби в документації, але і уникнути інцидентів невідповідності.

1.4 Огляд методів створення тест-плану

Розберемо підходи до створення тест-плану.

Навіщо застосовувати стратегію тестування. Стратегія тестування - це необов'язковий документ, який за своєю логікою передує загальному плану тестування при роботі з розроблюваним веб-продуктом. Перш ніж написати детальний план роботи з тестом здоров'я програмного забезпечення, варто найбільш ефективно оформити деякі фундаментальні підходи до тестування і особисто переконатися, що всі члени вашої групи розуміють одне і те ж, що і як воно буде перевірятися.

Які види перевірок плануються;

- Буде протестований інтерфейс, продуктивність системних компонентів, робота з конфігурацією та багато іншого;
- Чи збирається команда QA впроваджувати інші статичні типи тестів, наприклад, переглянути вимоги;
- Чи будете ви проводити дослідження, тестування, і в якому процентному співвідношенні.

Дуже корисно описати всі види тестових дій, які так чи інакше будуть відтворені у вашій діаграмі.

Наприклад, дизайн тестів, підготовка необхідного тестового веб-середовища і багато іншого.

Ви запитаєте, навіщо це робити? Щоб нічого не забути при розробці плану випробувань.

Яка буде основа майбутніх випробувань? З якого значення ви збираєтеся отримати значення для тестування встановлених тестів?

Наприклад, це можуть бути особливі вимоги, сценарії відтворення, загальні класифікації та стандарти.

При роботі зі стратегією можна відзначити перелік важливих критеріїв, які відповідають за серйозність і пріоритетність виявлених дефектів або системних помилок.

Як правило, оцінка тяжкості помилок завжди викликає жваві суперечки, якщо звичайно існує група критеріїв, за якими їх можна оцінити. Найбільш підходящим варіантом є формалізація важливості помилок і стратегії їх виправлення в рамках стратегії письмового тестування.

Логічна і продумана стратегія тестування є дуже хорошою основою для подальшої розробки залишкової технічної документації по проекту (в нашому випадку плану випробувань), що допомагає уникнути якомога більшого непорозуміння в майбутньому.

Кожна існуюча методологія, яка використовується для тестування веб-продуктів, має свій оригінальний формат тестового плану. Пропонуємо подальше обговорення питань щодо написання планів розгляду на основі міжнародного стандарту IEEE 829, а саме:

- TestPlanTemplate RUP;
- TestPlanTemplate IEEE 829.

Уважно вивчивши ці документи, відразу стає зрозуміло, що їх зміст описує одні й ті ж деталі, але в різних формах.

У ситуації, коли ви не хочете використовувати класичний шаблон або вирішили розробити і реалізувати власний в більш підходящому для вашого проекту вигляді, слід створити документ, який би відповідав хоча б на такі питання:

Що саме потрібно для тестування (об'єктні тести, використовуване веб-середовище, додаткові програми, яке обладнання використовується);

Що буде протестовано (повний перелік системних функцій для тестування на максимальну продуктивність);

Як саме ви будете тестувати (горезвісна стратегія тестування - види тестів, а також їх застосування до предмета тестів);

Коли саме ви будете перевіряти (логічно описана структура проведеної перевірки: підготовчі роботи, фактичний процес тестування, аналіз отриманої інформації щодо запланованих етапів веб-розробки на проекті).

Якщо ви зможете максимально відповісти на всі перераховані вище питання під час підготовки тестового плану, то можете впевнено повірити в те, що у вас є хороший і головне правильно розроблений тестовий план проектів.

Щоб зробити документ більш повним, варто вставити наступні пункти:

- Поточне середовище випробуваної системи;
- Використовується для тестування обладнання та віртуальних систем;
- Потенційні ризики та рішення.

Типи планів тестування. На практиці взагалі можна зустріти:

- Master Test Plan;
- Test Plan (детальний план випробувань).

Аналізуючи обидва ці документи можна зробити висновок: головна відмінність між ними полягає в тому, що генеральний план випробувань вважається більш статичним, оскільки містить у своїй структурі так звані дані «високого рівня», які не підлягають постійному використанню в процесі контролю якості навколишнього середовища.

Що стосується детального плану тестування, то тут можна знайти більш конкретну інформацію про стратегію оптимального тестування, повний графік виконаних робіт. Тобто такий план можна вважати більш «живим» джерелом маніпуляцій з тестами, який постійно піддається редагуванню і можна знайти більш реальну ситуацію при тестуванні продукту.

Коли тільки одна людина пише документи плану тестування, вони дуже односторонні, і тому настійно рекомендується проводити періодичні огляди інших QA, а також планувати процедуру затвердження документа в команді проекту.

Як приклад, така команда проекту може включати:

- Директор відділу забезпечення якості;

- Менеджер випробувань (головний менеджер якості);
- Начальник відділу розробки;
- Менеджер проекту.

Кожен з перерахованих вище учасників процесу створення програмного забезпечення або іншого web-продукту перед затвердженням повинен переглянути і внести свої побажання і зауваження до проекту, щоб в результаті вийшов продукт найвищої якості.

Що стосується часу, то головною особливістю односторінкового плану випробувань буде те, що його створення займе набагато менше часу, ніж робота над об'ємним документом. Це дозволить перерозподілити вільний час тестувальника для роботи над іншими проектами або підзадачами.

Обмежена кількість красномовно показує, що все найважливіше і актуальне повинно розміщуватися на папері. Це є основним фактором для тих людей, які потім повинні будуть виконати перевірку здоров'я програмного забезпечення на цьому документі.

Не варто перевантажувати документ другорядними деталями, про які всі будуть, м'яко кажучи, все одно піклуватися.

Якщо ви як тестувальник давно написали довгі тестові плани, і хочете звести його до обсягу 1 сторінки, поговоріть з тими людьми, які будуть його читати: що вони хотіли б бачити в ньому. Як варіант, грайте з різними форматами компіляції (про що ми говорили вище) і намагайтеся містити всю поточну інформацію.

Дуже важливо усвідомлювати, що є проекти, де не зійде з рук односторінковий формат.

Якщо тільки з ділових причин, ви повинні додати щось до плану - зробіть це!

Якщо ваш майбутній план повинен відображати тільки всі тестові набори, просто збережіть місце на сторінці і прикріпіть файли або утиліти для роботи скомпільованих тестових випадків між собою. Кінцевою метою плану тестування на одній сторінці є узагальнення всіх запланованих стратегій тестування.

Якщо фігуранти проекту захочуть отримати додаткові дані, то посилання, залишені всередині тестового плану, допоможуть їм це зробити, не особливо засмічуючи структуру документа.

Наприклад, можна звернутися до наступних об'єктів і компонентів:

- Використовувані інструменти для випробувань;
- Спеціальний тестовий статут;
- Перелік потенційних ризиків;
- Прототипи, що використовуються і поточні дослідження;
- Документи про веб-інфраструктуру;
- Оформлення документа на весь проект.

Навіть якщо ви розумієте, що односторінковий формат взагалі не працює, він все одно допоможе вам поглянути на тестовий план з іншої точки зору і розібратися, що можна зробити інакше, щоб тестовий документ набув максимального значення та ефективності.

2 РОЗРОБКА МЕТОДУ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ТЕСТ-ПЛАНУ ДЛЯ ТЕСТУВАННЯ WEB-ДОДАТКУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Актуальність використання інструментів штучного інтелекту у процесі тестування і написання тест-плану

Забезпечення якості (QA) є одним з найважливіших процесів розробки програмного забезпечення. QA передбачає системний та комплексний підхід до забезпечення відповідності програмного продукту необхідним стандартам якості. Автоматизація вже призвела до значних змін у процесах тестування та значно підвищила його ефективність та ефективність. Однак розвиток штучного інтелекту знову змінює підхід до тестування. У цій статті ми обговоримо переваги, проблеми та обмеження штучного інтелекту в QA на даний момент, а також його вплив на майбутнє тестувальників.

Процес забезпечення якості існує для тестування програмного продукту відповідно до необхідних стандартів якості та включає тестування, налагодження та відстеження несправностей. В останні роки розвиток штучного інтелекту торкнувся цієї сфери, підвищивши ефективність і ефективність процесу. Частина ручного тестування була замінена автоматизованим тестуванням, що значно покращило процес контролю якості. Тестувальники мають у своєму арсеналі різноманітні інструменти для автоматизації тестових завдань на різних мовах програмування, включаючи такі об'єктно-орієнтовані мови програмування, як JavaScript, Java та Python.

Включення штучного інтелекту в автоматизацію ще більше розширило можливості тестування. Здатність штучного інтелекту вчитися на даних приносить безпрецедентну ефективність і точність, і тепер тестувальники можуть швидше і точніше виявляти дефекти і потенційні проблеми. Це, в свою чергу, прискорює створення тестових випадків за допомогою фреймворків, таких як Behavior Driven

Development (BDD) з використанням Gherkin, наприклад, і Test Driven Development (TDD).

Навіть у сценаріях тестування «білої скриньки» (яка базується на аналізі внутрішньої структури компонента або системи) алгоритми штучного інтелекту можуть перевірити, чи правильний шлях - це код для виробництва, що також допомагає в контролі якості. AI допомагає автоматизувати різні завдання ручного тестування, які раніше були трудомісткими і трудомісткими. Ця автоматизація призводить до більш швидких циклів виробництва та скорочення часу випуску.

Інтеграція штучного інтелекту в систему забезпечення якості приносить багато переваг. AI робить тестування більш стабільним, роблячи тести так само кожен раз. Помилки, обумовлені людським фактором, виключаються, що призводить до більш надійних і точних результатів. Завдяки здатності ШІ виконувати тести з високою точністю і дотримуючись заздалегідь визначених рекомендацій, інженери QA можуть бути впевнені в послідовності процесів тестування.

AI дозволяє тестувальникам більш ефективно відстежувати та виявляти дефекти - аналізуючи величезну кількість даних, виявляючи закономірності, кореляції та потенційні проблемні зони; і виконати багаторазові тести за короткий час, значно скоротивши цикл тестування. Це прискорення тестування допомагає організаціям дотримуватися стислих термінів, підвищує загальну ефективність процесу розробки та дозволяє інженерам QA швидко виявляти та вирішувати проблеми, забезпечуючи якісні програмні продукти.

Автоматизація також допомагає тестувати програмні продукти в різних середовищах і ситуаціях. Однією з найважливіших характеристик є швидкість тестового моніторингу, оскільки, наприклад, моніторинг і оповіщення в реальному часі дозволяють оперативно контролювати виконання тестів і поведінку системи, своєчасно виявляючи аномалії або проблеми з продуктивністю. За допомогою автоматичних сповіщень QA інженери здатні швидко реагувати на критичні проблеми, тим самим мінімізуючи час простою та скорочуючи час, необхідний для усунення несправностей.

Технології продовжують розвиватися безпрецедентними темпами. Ми розглядаємо штучний інтелект як фактор, що змінює гру, тому що здатність штучного інтелекту аналізувати величезну кількість даних, вивчати моделі та приймати розумні рішення відкриває нові можливості для ефективного створення та виконання тестових випадків.

Основним внеском є машинне навчання з його алгоритмами на основі ШІ, які можуть вчитися на минулих тестових пробах, аналізувати результати тестів, визначати важливі моделі - і в результаті автономно виявляти надлишкові або менш ефективні тестові випадки. Це в свою чергу допомагає оптимізувати весь набір тестів. Цінність для компанії полягає в тому, щоб зменшити часові витрати при збереженні ефективного тестового покриття.

Нейронні мережі можуть бути використані для вирішення таких проблем, як генерація тестових випадків, автоматичне виявлення помилок або навіть обробка природної мови (NLP) для аналізу вимог.

Наприклад, рекурентні нейронні мережі (рекурентні нейронні мережі, РНМ) можуть аналізувати вимоги в текстовому форматі і генерувати відповідні тестові випадки. Згорткові нейронні мережі (CNN) можуть бути використані для тестування на основі зображень або візуальної ідентифікації дефектів.

Інший відомий алгоритм - наївний алгоритм класифікації Баєса, який є алгоритмом ймовірності задач класифікації (Примітка: Microsoft Naive Bayes - це алгоритм класифікації, заснований на теоремах Баєса, який може бути використаний як для дослідження, так і для прогнозного моделювання. Слово «спрощений» у своїй назві вказує, що алгоритм використовує методи Баєса, але не розглядає можливі залежності). У нашому контексті він може бути використаний для автоматичної категоризації та пріоритизації тестових випадків. Це дозволяє класифікувати тестові випадки на основі їх відповідності певним модулям, функціональним можливостям або програмним доменам, а також покращити управління тестовими випадками та допомогти інженерам з контролю якості ефективно розподіляти ресурси.

Якщо ви хочете відстежити перебіг життєвого циклу помилок та отримати огляд результатів тестування, ви можете скористатися методом випадкового лісу (Random Forest), алгоритмом машинного навчання, який використовує ансамбль дерева рішень. Використовуючи раніше отримані дані, метрики складності коду та шаблони несправностей, цей алгоритм передбачає ймовірність виявлення дефекту або вразливості в тестовому випадку. Крім того, впровадження алгоритмів ML при тестуванні часто вимагає співпраці QA інженерів, аналітиків даних і розробників програмного забезпечення. Ця співпраця забезпечує ефективну інтеграцію алгоритмів у процес контролю якості та значні результати для покращення якості продукції. Вибір алгоритму залежить від конкретного використання, наявних даних і поставленої проблеми. Ефективність алгоритмів залежить від якості та репрезентативності навчальних даних, а також від процесів розробки.

Розробка штучного інтелекту також дозволила тестувати програмні продукти в різних налаштуваннях і ситуаціях, дозволяючи тестувальникам більш ефективно виявляти і виправляти помилки. Майбутнє QA здається сприятливим, оскільки AI відіграє значну роль у трансформації процесу тестування, але при впровадженні штучного інтелекту в наш технологічний світ ми можемо зіткнутися з деякими проблемами.

Позитивною стороною використання ШІ є те, що:

- Автоматизація та ефективність тестування: інструменти автоматизації на основі штучного інтелекту можуть значно підвищити ефективність та швидкість тестування програмного забезпечення шляхом адаптивного визначення пріоритетів тестових випадків. Це дозволяє інженерам QA оптимізувати свої робочі процеси та досягти більш високого рівня покриття тестами. AI також може бути підключений для створення тестів самовідновлення, які динамічно адаптуються до змін програмного забезпечення, скорочуючи час і зусилля, витрачені на технічне обслуговування.
- Підвищення точності та надійності: алгоритми штучного інтелекту здатні аналізувати величезну кількість даних, включаючи зміни коду, вимоги,

виявлені раніше дефекти. Це допомагає виявити закономірності і передбачити потенційні проблеми. Це дозволяє інженерам QA зосередитися на критичних областях, підвищити точність та надійність тестування програмного забезпечення, а також допомогти виявити та вирішити потенційні ризики на початку циклу розробки (SDLC), тим самим мінімізуючи виникнення критичних проблем у виробництві.

- Інтелектуальне управління тестами: системи управління тестами на основі штучного інтелекту автоматизують різні аспекти управління тестовими випадками, включаючи проектування тестових наборів, планування тестів та аналіз результатів. Використовуючи можливості штучного інтелекту, QA інженери можуть оптимізувати свою роботу з тестування, покращити взаємодію та отримати цінні висновки з проведених тестів. Це дозволяє приймати обґрунтовані рішення та покращувати загальну якість програмного забезпечення.

Приклади використання штучного інтелекту в роботі QA-тестувальника:

Перший приклад: інженер хоче створити автоматизацію для взаємодії зі спадними меню та отримати певні значення з DOM-локаторів на сайті, побудованому за допомогою HTML, CSS та JavaScript. Так штучний інтелект отримав просту «карту» того, як знайти це випадаюче меню.

За допомогою цієї вправи ми хотіли вивчити потенціал штучного інтелекту в QA. Ми хотіли визначити, чи може ШІ бути корисним, створивши код, який може взаємодіяти з необхідним елементом. Звичайно, цей код може бути виконаний інженерами QA початківців, але в експерименті ми хотіли отримати результати для запитуваної дії від ChatGPT.

Як ви можете бачити, зображення показує приклад взаємодії з випадаючим списком і використання проміжних команд Cypress для захоплення елементів у

випадаючому списку, таких як команди «wrap» і «find», які можуть відкрити кілька можливостей програми Cypress.

Ось ще один приклад того, як AI допомагає написати код автоматизації для взаємодії з двома інтерактивними елементами. У більшості випадків ChatGPT створює шаблон, який можна використовувати та змінювати залежно від завдань. Процес кодування є частиною QA, щоб застосувати саме те, що потрібно в процесі.

ChatGPT дає докладні поради та коментарі щодо кожного кроку автоматизації, якщо стає важко виконати чіткий крок у коді автоматизації.

Про проблеми, пов'язані з тим, як штучний інтелект може вплинути на майбутнє QA:

- Набір навичок та адаптивність: інтеграція AI у забезпечення якості вимагає нового набору навичок та адаптивності від професіоналів QA. Вони повинні набувати знань в області AI-методів, алгоритмів та інструментів, а також добре розуміти основні процеси розробки програмного забезпечення. Професійний розвиток та постійне знайомство з досягненнями AI матимуть вирішальне значення для ефективного використання QA інженерів.
- Етичні аспекти: Оскільки ШІ стає більш поширеним в QA, етичні аспекти виходять на перший план. Тестувальники повинні гарантувати, що алгоритми штучного інтелекту є неупередженими, справедливими і не посилюють існуючі упередження або дискримінацію. Крім того, при використанні AI в процесах забезпечення якості необхідно ретельно вирішувати питання, пов'язані з конфіденційністю, безпекою даних і дотриманням нормативних вимог.
- Баланс між автоматизацією та людським підходом: хоча штучний інтелект може автоматизувати різні завдання тестування, важливо знайти баланс між автоматизацією та людськими сильними сторонами. Критичне мислення, великі знання та інтуїція QA інженерів не можуть бути повністю замінені

штучним інтелектом. Тестувальники можуть використовувати AI як допомогу, використовуючи свої навички для перевірки результатів, інтерпретації результатів та прийняття обґрунтованих рішень.

Оскільки розробка програмного забезпечення продовжує розвиватися з розвитком автоматизації та штучного інтелекту, важливість людського досвіду не можна недооцінювати. Хоча автоматизація та штучний інтелект, безсумнівно, принесли значну користь у цій сфері, роль QA-фахівців залишається незамінною.

Критичне мислення і вирішення проблем - одна з сильних сторін QA-фахівців. Людина має когнітивні навички, необхідні для аналізу складних сценаріїв, розуміння складної поведінки системи, виявлення потенційних ризиків і проблем, які не можуть бути ідентифіковані тільки за допомогою автоматизованих тестів. Людський досвід дозволяє інженерам вийти за рамки сценаріїв і застосувати свій особистий досвід, інтуїцію, розуміння і знання предметної області для виявлення тонких дефектів і вразливостей.

Іншою людською силою є здатність адаптуватися до динамічних вимог, які стосуються зміни очікувань користувачів і частих оновлень. Фахівці QA ідеально адаптуються до динамічного середовища та мають гнучкість, щоб швидко зрозуміти нові функції, охопити перспективи користувачів та налаштувати стратегії тестування, одночасно змінюючи пріоритети.

Нарешті, робота зі складними сценаріями тестування та постійного вдосконалення вимагає творчого мислення та навичок вирішення проблем, які можуть включати багатоступеневі робочі процеси, інтеграційне тестування або складні системні взаємодії. Тестувальники здатні розробляти і виконувати такі тести, забезпечуючи охоплення різних сценаріїв, які не легко автоматизувати, а їх досвід дозволяє виявити прикордонні випадки і неочевидні сценарії, які автоматичне тестування може пропустити.

Враховуючи існування таких складних сценаріїв, тестування програмного забезпечення постійно розвивається з появою нових інструментів, технологій і методологій. QA-фахівці повинні бути в курсі тенденцій галузі, вивчати нові методи тестування та вдосконалювати свої навички.

2.2 Постановка задачі автоматизованої генерації тест-плану для тестування web-додатку за допомогою інструментів штучного інтелекту

Використання інструментів штучного інтелекту в процесі написання тест-плану для тестування web-додатку значно зменшить робоче навантаження QA-тестувальників.

Проте, на сьогоднішній день немає спеціалізованих інструментів штучного інтелекту для автоматизованої генерації тест-плану. Для роботи буде використано інструменти “загального” користування. Початковим етапом буде провести аналіз існуючих інструментів та обрати найкращий.

Важливим критерієм оцінювання буде технологія NLP, адже для автоматизованої генерації тест-плану, штучний інтелект повинен проаналізувати технічне завдання. Також, важливу роль грає кількість текстів, на яких інструмент штучного інтелекту навчався, адже потенційний замовник може надати технічне завдання для проекту пов’язаним з абсолютно будь-якою сферою життя і інструмент штучного інтелекту повинен розуміти контекст області застосування. В обраному інструменті має бути можливість завантаження документа для подальшого аналізу та опрацювання.

Після вибору інструменту, потрібно визначитись із програмними технологіями, за допомогою яких буде реалізовано систему. Також, потрібно визначитись із параметрами, які користувач буде вносити для подальшої генерації тест-плану, адже саме чіткість і коректність запиту до інструменту штучного інтелекту гарантує задовільний результат. Проте, найважливішу роль у якості отриманого результату відіграє правильно обраний інструмент.

2.3 Огляд доступних інструментів штучного інтелекту

2.3.1 Chatsonic

Writesonic - це стартап AI, який отримав підтримку інвесторів і став популярним в останні роки. Серед різних продуктів виділяється один: Chatsonic AI. Chatsonic AI дуже схожий на ChatGPT. Це чат-бот на основі штучного інтелекту, призначений для взаємодії з користувачами та надання їм допомоги у виконанні таких завдань, як створення тексту та зображень за допомогою штучного інтелекту.

Chatsonic є додатковим додатком до Writesonic, який використовує генеративну технологію штучного інтелекту для спрощення написання завдань для письменників усіх смуг. Хоча Writesonic в першу чергу зосереджується на впровадженні генеративного ШІ в редактор документів, щоб легко інтегрувати різні функції написання ШІ в цілісний, уніфікований контент, Chatsonic використовує інший підхід.

Chatsonic, який використовує ту ж технологію штучного інтелекту, що і Writesonic, служить інтерактивним чат-ботом. Цей інноваційний чат-бот дозволяє користувачам більш ефективно взаємодіяти з системою штучного інтелекту, оптимізуючи продуктивність завдань і підвищуючи зручність користувача. Ці взаємодії з Chatsonic характеризуються повторюваним і залежним від контексту характером, який відображає нюанси людських розмов.

Розширені чат-боти, такі як Chatsonic, дозволяють їм спілкуватися природно, демонструючи розумний рівень критичного мислення. Ця функція робить Chatsonic надзвичайно цінним для досліджень та письмових цілей, пропонуючи допомогу, подібну до розмови зі співробітником.

Одна з найважливіших речей, щоб зрозуміти про Chatsonic є його здатність розширити контекст розмов і отримати доступ до останньої інформації. Можливо, ви вже знаєте, що моделі OpenAI GPT навчали використовувати дані лише до середини 2021 року. Це також стосується базових моделей, які використовує Chatsonic, оскільки вони також засновані на моделях OpenAI. Однак Chatsonic відрізняє його складна система, яка може автоматично використовувати дані результатів пошуку Google, коли ви берете участь у розмові.

Для вас це означає, що коли ви починаєте спілкуватися з Chatsonic, він може надати вам оновлення глобальних подій у реальному часі та поточну інформацію. Chatsonic повністю здатний обробляти останні події, такі як визначення останнього переможця Суперкубка або надання детальної інформації про останній випуск iPhone. Ця функція дозволяє бути в курсі останніх світових подій, роблячи Chatsonic цінним інструментом для отримання інформації.

Крім використання даних Google, Chatsonic має можливість запам'ятовувати кожне повідомлення в розмові, що знижує ймовірність його повторення або втрати корисності. Якщо увімкнено, ця функція дозволяє більш ефективно навчатися на основі поточної розмови. Однак важливо відзначити, що ця покращена здатність до навчання іноді може сповільнювати час відгуку. Таким чином, користувачі можуть час від часу вимикати цю функцію для більш плавної роботи.

2.3.2 Microsoft Bing Chat

Microsoft заявила, що її Bing AI - це «новий день у пошуку», і якщо ви думаєте про це, це правильно. Пошук в Інтернеті залишається майже незмінним вже більше двох десятиліть. За цей час кількість сайтів збільшилася в тисячу разів. З такою кількістю інформації навіть пошукові системи насилу справляються. На сайтах ставлять фільтри, і все ж це не допомагає - кількість інформації зростає, як дріжджі.

Але проблема не тільки в кількості даних, але і в відверто поганій якості видачі. За даними Microsoft, близько 4 з 10 користувачів натискають на пошукові посилання і відразу повертаються в пошукову систему знову. Причина банальна - система не дає людям того, що вони шукають. Отже, користувачам потрібна допомога в обробці інформації, доступної в Інтернеті.

Bing прагне задовольнити цю потребу, надаючи можливість пошуку на основі свого чат-бота.

Chat Bing AI - це нова функція пошукової системи Bing, яка дозволяє спілкуватися зі штучним інтелектом на різні теми. Чат-бот Bing AI використовує технологію ChatGPT, засновану на потужному мовному моделюванні від OpenAI.

Ви можете обговорити різні теми з чат-ботом, такі як новини, погода, спорт, мистецтво, наука тощо. Чат-бот також може писати код на різних мовах програмування, розповідати анекдоти і вірші або грати з вами в ігри. Ви також можете встановити голосові запити чат-бота в мобільній версії Edge.

Замість того, щоб пропонувати користувачам посилання в результатах пошуку, чат-бот надає результати безпосередньо, тим самим заощаджуючи час користувачів і усуваючи необхідність відсіювати неактуальну інформацію. Microsoft заявила, що Bing набагато потужніший, ніж інші чат-боти, включаючи ChatGPT, завдяки використанню мовних моделей наступного покоління.

Чат-бот Bing здатний розуміти запити до 1000 символів більш ніж на 100 мовах. Bing AI працює, написавши статтю або керівництво по темі, що цікавить користувача.

Bing Chat Bot дивиться на інформацію, доступну в Інтернеті, і відображає ту, яка представляє інтерес для користувача. На виході у нас є персоналізований матеріал, в якому зібрані все найважливіше і корисне.

Чат-бот Bing поважає авторське право - вказує джерело кожного тексту або факту.

2.3.3 Google Bard

Google Bard AI - універсальний чат-бот, який демонструє свої навички поета, математика та співрозмовника. Чат-бот призначений для надання свіжих та якісних відповідей шляхом підключення до Інтернету.

Під час свого першого випуску, Bard був представлений в легшій версії LaMDA. Таке рішення було прийнято для забезпечення масштабованості, що дозволило чат-боту обслуговувати велику базу користувачів за рахунок значного скорочення необхідних обчислювальних ресурсів.

Після етапу тестування Bard тепер доступний користувачам по всьому світу в більш ніж 180 країнах. Щоб отримати доступ до Bard, користувачі можуть

підключитися до нього за допомогою облікового запису Workspace, який є альтернативою особистим обліковим записам Google.

- Він може відповісти на детальні питання, підлаштовуючись під співрозмовника.
- Бард використовує PaLM 2, мовну модель з розширеними можливостями міркування, мови та кодування.
- Якщо цей бот надасть вам інформацію від когось (блог, відео, стаття), він підкреслить її і вкаже джерело, звідки вона прийшла.
- Bard дозволяє користувачам експортувати код Python через Replit.
- Підтримка до 40 мов.
- Bard може генерувати генеративні зображення з тексту завдяки співпраці з Adobe та Ініціативою автентичності вмісту.
- Інтеграція з Adobe Firefly дозволяє користувачам додатково редагувати згенеровані зображення AI в Adobe Express.
- Можливості AI Bard виходять за рамки можливостей ChatGPT (враховуючи безкоштовну версію без плагінів). Він покращив міркування, кодування та спеціалізовані інструменти, такі як Med-PaLM 2 та Sec-PaLM. Він також включає в себе такі функції, як поліпшення цитування, можливість експорту коду Python і створення генеративних зображень у співпраці з Adobe.

2.3.4 ChatGPT

GPT-3 - це нейромережева модель машинного навчання, випущена в травні 2020 року. Він здатний використовувати дані з Інтернету для створення текстів будь-якого типу. GPT-3 може відповідати на питання, генерувати пости для соціальних мереж на різні теми, створювати статті для блогу, писати контент-план, робити переклади, редагувати тексти, писати код і знаходити в ньому жучки, переказувати фільми, складати вірші та інше. Поява цієї моделі породило справжній фурор, адже взаємодія з нею максимально наближена до діалогу з живою людиною.

ChatGPT (Chat Generative Pre-trained Transformer) здатний відповідати на питання, визнавати помилки, оскаржувати неправильні припущення, генерувати тексти, відхиляти невідповідні запити, шукати ключові слова для заданих тем, знаходити помилки в коді та виконувати інші завдання.

ChatGPT має чотири мовні моделі, які можна підключити через OpenAI API. Нижче розглянемо кожну з них. Почнемо з найшвидшого і найдешевшого, а закінчимо з найпотужнішого.

- Ada. Ця мовна модель має найнижчу вартість і здатна виконувати прості завдання, такі як парсинг і парсинг тексту, фіксація адреси та ключових слів. Продуктивність можна покращити, надаючи більше контексту. Ціна становить $0,0004 за 1000 токенів, де 1000 токенів означає приблизно 750 слів.
- Babbage. Він також виконує прості завдання, але краще розуміє шаблони в тексті, який пізніше використовується як посилання для генерації іншого тексту. Беббідж здатний ранжувати дані і присвоювати категорії. Ціна становить $0,0005 за 1000 жетонів.
- Curie. Ця модель більш потужна, ніж дві попередні. Він здатний аналізувати текст, визначати його тон, відповідати на питання. Кюрі найчастіше використовується в чат-ботах. Ціна використання цієї мовної моделі становить $0,0020 за 1000 токенів.
- Davinci Найбільш дієздатна модель мови в сім'ї ChatGPT, але найповільніша. Він розуміє зміст тексту, виявляє причинно-наслідкові зв'язки, узагальнює і обробляє великі обсяги даних. Давінчі вимагає найменше інструкцій для запуску. Ця мовна модель коштує $0,0200 за 1000 токенів.

2.4 Розробка блок-схеми функціонування додатку для автоматизованої генерації тест-плану для тестування web-додатку

Для подальшого розуміння бажаного результату було створено схему функціонування додатку для автоматизованої генерації тест-плану для тестування web-додатку яку можна побачити на схемі 2. Цю схему було використано також при розробці програмного забезпечення.

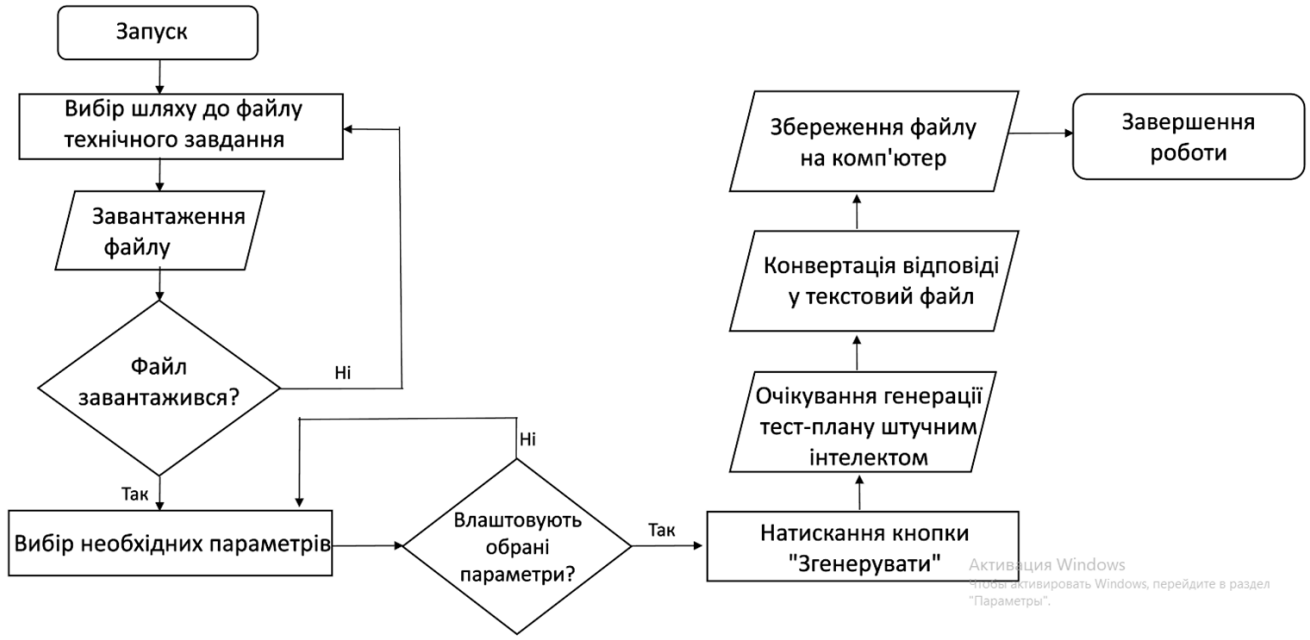


Схема 2: Блок-схема роботи додатку

Після запуску додатку, потрібно буде обрати глях до технічного завдання. Технічне завдання автоматизовано буде проаналізовано інструментами штучного інтелекту за допомогою NLP.

Обробка природної мови (NLP) - це технологія машинного навчання, яка дозволяє комп'ютерам інтерпретувати, маніпулювати та розуміти людську мову. Сьогодні організації мають велику кількість голосових та текстових даних з різних каналів зв'язку, таких як електронна пошта, текстові повідомлення, стрічки новин соціальних мереж, відео, аудіо тощо. Вони використовують програмне забезпечення NLP для автоматичної обробки цих даних, аналізу намірів або настрою повідомлення та реагування на людське спілкування в режимі реального часу.

Обробка природної мови (NLP) має вирішальне значення для повного та ефективного аналізу тексту та мови. Таким чином можна подолати відмінності в діалектах, сленгу і граматичних розладах, характерних для повсякденних розмов.

Компанії використовують цей метод для декількох автоматизованих завдань, таких як:

- Обробка , аналіз та архівація великих документів
- Аналіз відгуків клієнтів або записів колл-центру
- Запуск чат-ботів для автоматизованого обслуговування клієнтів
- Відповіді на питання «хто, що, коли і де»
- Класифікація та вилучення тексту

Ви також можете інтегрувати NLP в клієнтоорієнтовані програми для більш ефективного спілкування з клієнтами. Наприклад, чат-бот аналізує і сортує запити клієнтів, автоматично відповідає на поширені питання і перенаправляє складні запити на підтримку. Ця автоматизація допомагає скоротити витрати, позбавити агентів від необхідності витрачати час на надлишкові запити і підвищує задоволеність клієнтів.

Після завантаження файлу з технічним завданням, потрібно буде обрати необхідні параметри для подальшого формування тест-плану. Серед яких:

- Кількість людей у команді.
- Тип тестування.
- Дату початку тестування.
- Дату кінця тестування.
- Інструменті тестування.
- Кількість сторінок тест-плану.
- Критерії завершення тестування.

Визначення дати початку тестування є одним з ключових моментів планування тестування. Ця дата вказує початок тестового циклу і визначає очікувані терміни завершення.

Як правило, дата початку тестування встановлюється після завершення всіх етапів розробки продукту, включаючи розробку компонентів, інтеграцію та тестування. Таким чином, попереднє планування процедури тестування є важливим кроком.

На ранніх етапах розробки продукту, коли його функціональність тільки знаходиться в стадії розробки, може бути важко встановити точну дату початку тестування. При цьому планування може базуватися на приблизних датах і очікуваних термінах завершення етапів розробки.

Коли дата початку тесту встановлена, ви можете почати складати тестовий план, визначати критерії завершення тесту та вибирати сценарії тестування.

- Важливі моменти в установці дати початку тесту:
- Завершення продукту
- Очікувані терміни завершення етапів розробки
- Можливість зміни продукту
- Наявність ресурсів для тестування
- Загальний графік проекту

Використовуючи цю інформацію, ви можете визначити дату початку тестування, яка відповідатиме вимогам проекту та досягне необхідної якості розроблюваного продукту.

Встановлення дати початку тестування є важливим кроком у процесі розробки програмного забезпечення. Початок тестування може залежати від різних факторів, таких як:

- Готовність продукту. Тестування зазвичай починається після завершення розробки продукту і створення оригінальної версії.
- Завершення попередніх етапів розробки. У деяких випадках тестування може початися тільки після успішного завершення попередніх етапів розробки, таких як проектування та впровадження.
- Наявність тестових інструментів. Тестування може початися тільки після створення та налаштування відповідного тестового середовища, включаючи необхідні інструменти та тестові дані.

Встановлення дати початку тестування може бути процесом, який вимагає узгодження між розробниками, тестувальниками та іншими зацікавленими сторонами. Зазвичай це робиться в рамках планування розвитку та враховує терміни, ресурси та пріоритети проекту.

Крім того, важливо розглянути можливість тестування паралельно з розробкою, щоб прискорити процес і виявити проблеми на ранній стадії розвитку.

В ідеалі, дата початку тестування повинна бути визначена, щоб забезпечити достатній час для підготовки тестових сценаріїв, планування та налаштування тестового середовища. Це також дозволяє змінити дату початку тестування, якщо відбуваються затримки розробки або зміни в планах.

Після вибору всіх необхідних параметрів, потрібно натиснути кнопку “Згенерувати”, після чого буде відправлений запит до інструментарію штучного інтелекту. Після отримання даних, вони автоматично конвертуються у текстовий файл та зберігаються на комп’ютер.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ТЕСТ-ПЛАНУ ДЛЯ ТЕСТУВАННЯ WEB-ДОДАТКУ

Продукт було розроблено за допомогою операційної системи Windows 10, та може бути використаним на комп’ютерах з операційною системою Windows.

3.1 Опис використаних програмних засобів

3.1.1 Visual Studio

Visual Studio - це потужний інструмент розробника, який може бути використаний для запуску всього циклу розробки в одному місці. Це всеосяжне інтегроване середовище розробки (IDE), яке можна використовувати для запису, редагування, налагодження та збирання коду, а потім розгорнути програму. На додаток до редагування та налагодження коду, Visual Studio включає в себе компілятори, інструменти завершення коду, контроль версій, розширення та багато іншого для поліпшення кожного етапу процесу розробки програмного забезпечення.

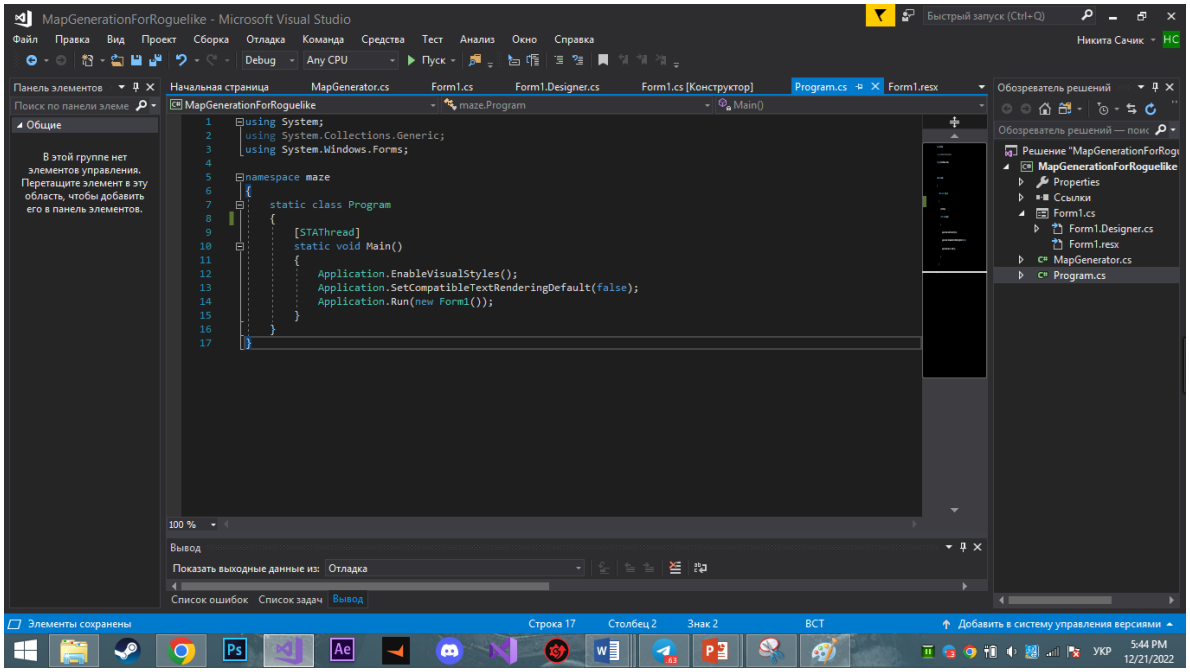


Рисунок 3.1 – Інтерфейс Visual Studio

Лінійка продуктів Microsoft включає програмне забезпечення інтегрованого середовища розробки (IDE) та ряд інших інструментів. Ці продукти дозволяють розробляти як консольні програми, так і ігри та графічні додатки, включаючи підтримку технології Windows Forms, UWP, а також веб-сайти, веб-додатки, веб-сервіси як у рідному, так і керованому коді для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE,. NET Framework,. NET Ядро,. NET, MAUI, Xbox, Windows Phone. NET Compact Framework і Silverlight. Після покупки Xamarin Company компанією Microsoft з'явилася можливість розробки програм для iOS і Android.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю простого рефакторингу коду. Вбудований зневаджувач може працювати як зневаджувач рівня джерела, так і зневадник рівня машини. Інші вбудовані інструменти включають редактор форм для спрощення створення графічного інтерфейсу програми, веб-редактор, дизайнер класів та конструктор схем баз даних. Visual Studio дозволяє створювати і підключати сторонні доповнення (плагіни) для розширення функціональності майже на кожному рівні, включаючи додавання підтримки систем контролю версій джерела (наприклад, Subversion і Visual SourceSafe) додавання нових наборів інструментів (наприклад, для редагування та візуалізації коду в предметно-орієнтованих мовах програмування) або інструменти для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server).

3.1.2 Мова програмування C#

C# (вимовляється як «C Sharp») - сучасна об'єктно-орієнтована і безпечна для типів мова програмування. C# дозволяє розробникам створювати різні типи безпечних і надійних додатків, що працюють в .NET. C# належить до відомої сім'ї мов C, і буде здаватися знайомим кожному, хто працював з C, C++, Java або JavaScript. Ось огляд основних компонентів C# 8 і більш ранніх версій.

C# - це об'єктно-орієнтована, компонентно-орієнтована мова програмування. C# надає мовні конструкції для безпосередньої підтримки цієї концепції роботи. Це робить C# придатним для створення та застосування програмних компонентів. З моменту свого створення C# був збагачений функціями для підтримки нових робочих навантажень і сучасними рекомендаціями з розробки програмного забезпечення. В основному, C# є об'єктно-орієнтованою мовою. Ви визначаєте типи та їх поведінку.

Ось лише кілька функцій C#, які дозволяють створювати надійні та стійкі програми. Збір сміття автоматично звільняє пам'ять, зайняту невикористаними об'єктами. Типи, які приймають null, забезпечують захист від змінних, які не посилаються на вибрані об'єкти. Обробка винятків забезпечує структурований та масштабований підхід до виявлення та відновлення помилок. Лямбда-вирази підтримують функціональні методи програмування. Синтаксис LINQ створює загальний шаблон для роботи з даними з будь-якого джерела. Підтримка мови для асинхронних операцій забезпечує синтаксис для створення розподілених систем. У C # існує єдина система типів. Всі типи C #, включаючи примітиви, такі як int і double, успадковуються від одного типу кореневих об'єктів. Всі типи використовують загальний набір операцій, і значення будь-якого типу можуть зберігатися, передаватися і оброблятися аналогічним чином. Крім того, C# підтримує як визначені користувачем типи посилань, так і типи значень. C# дозволяє динамічно вибирати об'єкти і зберігати спрощені структури в стеку. C# підтримує універсальні методи та типи, які забезпечують підвищену безпеку та продуктивність. C# надає ітератори, які дозволяють розробникам класів збірки визначати власні параметри поведінки клієнтського коду.

C# підкреслює контроль версій для забезпечення сумісності програм і бібліотек з плином часу. Проблеми керування версіями значно вплинули на аспекти розробки C#, такі як окремі віртуальні та перевизначення модифікаторів, правила перевантаження та підтримка явного оголошення членів інтерфейсу.

3.1.3 Windows Forms

Windows Forms - платформа інтерфейсу користувача для створення класичних додатків Windows. Він надає один з найефективніших способів створення класичних додатків за допомогою візуального дизайнера в Visual Studio. Такі функції, як перетягування розміщення візуальних елементів управління спрощують створення класичних додатків.

У Windows Forms ви можете розробляти графічно складні програми, які легко розгортати, оновлювати та працювати як в автономному режимі, так і в Інтернеті. Програми Windows Forms можуть отримати доступ до локального обладнання та файлової системи комп'ютера, на якому працює програма.

Windows Forms - це технологія інтерфейсу користувача. NET - це набір керованих бібліотек, які спрощують виконання стандартних завдань, таких як читання та запис у файлову систему. За допомогою середовища розробки, такого як Visual Studio, ви можете створювати інтелектуальні клієнтські програми Windows Forms, які відображають інформацію, запитують введення користувача та взаємодіють з віддаленими комп'ютерами через мережу.

У Windows Forms форма - це візуальна поверхня, на якій відображається інформація для користувача. Як правило, програма Windows Forms створюється шляхом додавання елементів керування до форм і створення коду для відповіді на дії користувача, такі як натискання клавіш або натискання клавіш. Управління - це окремий елемент інтерфейсу користувача, призначений для відображення або введення даних.

Коли користувач виконує дію за допомогою форми або одного з елементів керування, створюється подія. Додаток реагує на ці події з кодом і обробляє події, коли вони відбуваються.

Windows Forms надає багато елементів керування, які можна додати до форм. Наприклад, елементи керування можуть відображати текстові поля, кнопки, випадаючі списки, перемикачі та навіть веб-сторінки. Якщо надані елементи керування не підходять для ваших цілей, Windows Forms може створювати власні елементи керування за допомогою класу UserControl.

Windows Forms має багатофункціональні елементи керування інтерфейсом користувача, які дозволяють емулювати функції складних програм, таких як Microsoft Office. За допомогою елементів керування ToolStrip та MenuStrip можна створювати панелі інструментів та меню, які містять текст та зображення, відображати підменю та розміщувати інші елементи керування, такі як текстові та спискові поля.

Використовуючи функцію перетягування конструктора Windows Forms у Visual Studio, ви можете легко створювати програми Windows Forms. Просто виберіть управління з курсором і помістіть його в потрібне місце в формі. Щоб подолати труднощі керування вирівнюванням, дизайнер надає такі інструменти, як лінії сітки та опорні лінії. За допомогою FlowLayoutPanel, TableLayoutPanel і SplitContainer можна створювати складні макети форм набагато швидше.

Нарешті, якщо ви хочете створити власні елементи інтерфейсу користувача, простір імен System.Drawing містить широкий набір класів, необхідних для малювання ліній, кіл та інших фігур безпосередньо на формі.

3.2 Опис програмної реалізації

Для початку створюємо проект додатку C# з користувацьким інтерфейсом на основі Windows:

- Відкриваємо Visual Studio.
- У вікні запуску обираємо “Створення нового проекту”.
- У вікні “Створення проекту” обираємо шаблон “Додаток Windows Form (.NET Framework) для C#”.
- У полі “Ім’я проекту” вікна “Налаштувати новий проект” прописуємо назву нашого проекту “Test Plan Generator” і натискаємо “Створити”.

Коли ми обрали шаблон проекту C# і задали ім’я файлу, Visual Studio відкриває форму. Форма є користувацьким інтерфейсом Windows. Ми створюємо додаток, додаючи елементи управління на форму.

Створюємо файл з основним кодом та прописуємо початкові елементи інтерфейсу.

За ініціалізацією усіх елементів керування відповідає метод InitializeComponent. Він генерується автоматично при створенні форми.

Створюємо елементи, які будуть відповідати за:

- Кількість людей у команді.
- Дату початку тестування.
- Дату закінчення тестування.
- Умови завершення тестування.
- Елемент додавання файлу.
- Тип тестування.
- Кнопка “Згенерувати”.

За показник кількості людей у команді буде відповідати елемент TextBox. Для його створення у Microsoft Visual Studio відкриваємо “Панель елементів”, відкриваємо загальні елементи управління та обираємо TextBox. Перетаскуємо елемент на нашу форму.

Знаходимо у властивостях System.Windows,Forms.TextBox. Далі поле (Name) і змінюємо значення на “Кількість людей в команді”.

Додаємо дату початку тестування та дату закінчення тестування. За ці елементи буде відповідати DateTimePicker. За аналогією з минулим елементов, змінюємо назву на необхідну.

За умови завершення тестування буде відповідати елемент TextBox.

За додавання файлу з технічним завданням буде відповідальний клас OpenFileDialog за допомогою TextBox.

У блоці Window задаємо клас x:Class=”WpfTutorialSamples.Dialogs.OpenFileDialogSample”. Далі задаємо параметри xmlns=”http://schemas.microsoft.com/winfx/2006/xaml/presentation ” та xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" .

Прописуємо WrapPanel HorizontalAlignment="Center" DockPanel.Dock="Top" Margin="0,0,0,10" .

Задаємо параметри кнопки: Button Name="btnOpenFile" Click="btnOpenFile_Click".

При натисканні на кнопку “Open” відобразиться вікно OpenFileDialoge в залежності від нашої версії Windows.

Метод ShowDialog () повертає логічне нульове значення (може бути true, false або null). Якщо користувач вибере файл і натисне кнопку «Відкрити», результат буде True, в цьому випадку ми постараємося завантажити файл в елемент TextBox. Ми отримали повний шлях до вибраного файлу за допомогою властивості FileName у OpenFileDialog.

Зазвичай при виборі файлу користувач хоче обмежити виділення одним або декількома типами файлів. Наприклад, Word в основному відкриває файли Word (з розширенням .doc або .docx), а Notepad - текстові файли (.txt).

Ми можемо визначити фільтр для діалогового вікна OpenFileDialog з можливістю вибору та використання типів файлів та обмежень файлу для більшої чіткості. Це робиться за допомогою властивості Filter, яку ми додамо до попереднього прикладу (після ініціалізації вікна):

openFile.Dialoge.Filter = “Text files (*.txt)|*.txt|All files(*.*)|*.*”

На перший погляд, формат визначення фільтра може здатися трохи дивним, але він представляє зручну для користувача версію, а інший, для комп'ютера, який зможе «читати» і розділити їх символом роздільника «|». Якщо ви хочете визначити більше одного типу (як у прикладі), просто додайте необхідне для вас, використовуючи роздільник, введений вище.

Підсумовуючи, наступний код означає, що ми даємо типу ім'я «Текстові файли (*.txt)» (розширення в дужках є знаком уваги користувача, він буде знати, які типи включені в «Текстові файли»), а друга частина визначає, що файли з розширенням будуть показані. Txt:

Text files (*.txt)|*.txt

Звичайно, кожен створений тип може мати багато розширень. Наприклад, зображення можуть бути визначені для файлів JPEG і PNG, так:

openFileDialog.Filter = "Image files (*.png;*.jpeg)|*.png;*.jpeg|All files (*.*)|*.*";

Початковий каталог, який використовується класом OpenFileDialog, вже визначено у Windows, але ви можете перевизначити його за допомогою властивості InitialDirectory. Як правило, ви визначите це значення спеціально для користувача, всієї програми або, можливо, як каталог, який використовувався минулого разу. Ви можете визначити його як шлях у форматі рядка, наприклад:

openFileDialog.InitialDirectory = @"c:\temp\";

Можливо, ви помітили, що використання OpenFileDialog у WPF дуже просте, до того ж цей механізм робить більшу частину роботи для вас. Але зверніть увагу, що в цьому прикладі, щоб заощадити місце в коді, я не присвятив жодного рядка обробці винятків. При роботі з файлами і взагалі при роботі з вводом-виводом завжди слід побоюватися винятків, так як вони легко можуть з'явитися при роботі з заблокованим файлом, неіснуючим способом або іншим.

Далі, нам потрібен маркер для відправки запитів. Його можна створити в особистому кабінеті на сайті openai.

Нові користувачі отримують безкоштовну пробну версію з певною кількістю токенів, виділених на їх обліковий запис. Оскільки вони закінчуються або використовуються, вам знадобиться кредитна картка для продовження служби. OpenAPI має повне керівництво по ціні.

Після натискання кнопки «Створити новий секретний ключ» ви побачите вікно зі згенерованим маркером, який потрібно скопіювати і зберегти. Цей маркер з'явиться у списку ключів.

Далі потрібно визначити ряд класів для відправки і отримання json повідомлень.

Для відповіді визначені наступні класи: Message, ResponseData, Select and Use.

Для порівняння з даними, отриманими під час десеріалізації відповіді, ми встановлюємо відповідний атрибут JsonPropertyName для кожної властивості класу.

Формат запиту, що надсилається, буде мати вигляд:

“model”: “gpt-3.5-turbo”,

“messages”: [{“role”:”user”, “content”:TestPlan]}.

Для представлення цих даних визначено тип Request, що має наступний вигляд:

[JsonPropertyName(“model”)],

public string ModelId{get; set} = “”;

[JsonPropertyName(“messages”)],

public List<Message> Message{get; set} = new();

У програмі встановлюємо токен, який буде використовуватися для надсилання запитів до ChatGPT, і адресу запиту:

string apiKey у значення якого прописуємо наш згенерований токен. А у string endpoint адресу інструменту штучного інтелекту.

Далі ми визначаємо список повідомлень (порожніх за замовчуванням) і об'єкт HttpClient для відправки повідомлень:

httpClient.DefaultRequestHeaders.Add(“Authorization”, $”Bearer {apiKey}).

Для виконання запиту створюємо об'єкт Request, властивості якого передаються за назвою моделі та списком повідомлень. Після цього від змінної response ми можемо отримати відповідь чат-бота.

Після отримання відповіді від чат-боту, нам потрібно зберегти текстовий файл на комп’ютер.

Для цього будемо використовувати клас StreamWriter для відкриття, запису та закриття текстового файлу. Цей код передає конструктору два додаткових параметри. Першим параметром є шлях до файлу та ім'я файлу. Другий параметр, true, вказує, що файл відкривається в режимі додавання. Якщо ви встановите

другий параметр false, вміст файлу буде перезаписано кожного разу при запуску коду. Третій параметр визначає Unicode для використання Unicode для кодування файлу в StreamWriter.

На малюнку нижче можна побачити результат роботи:

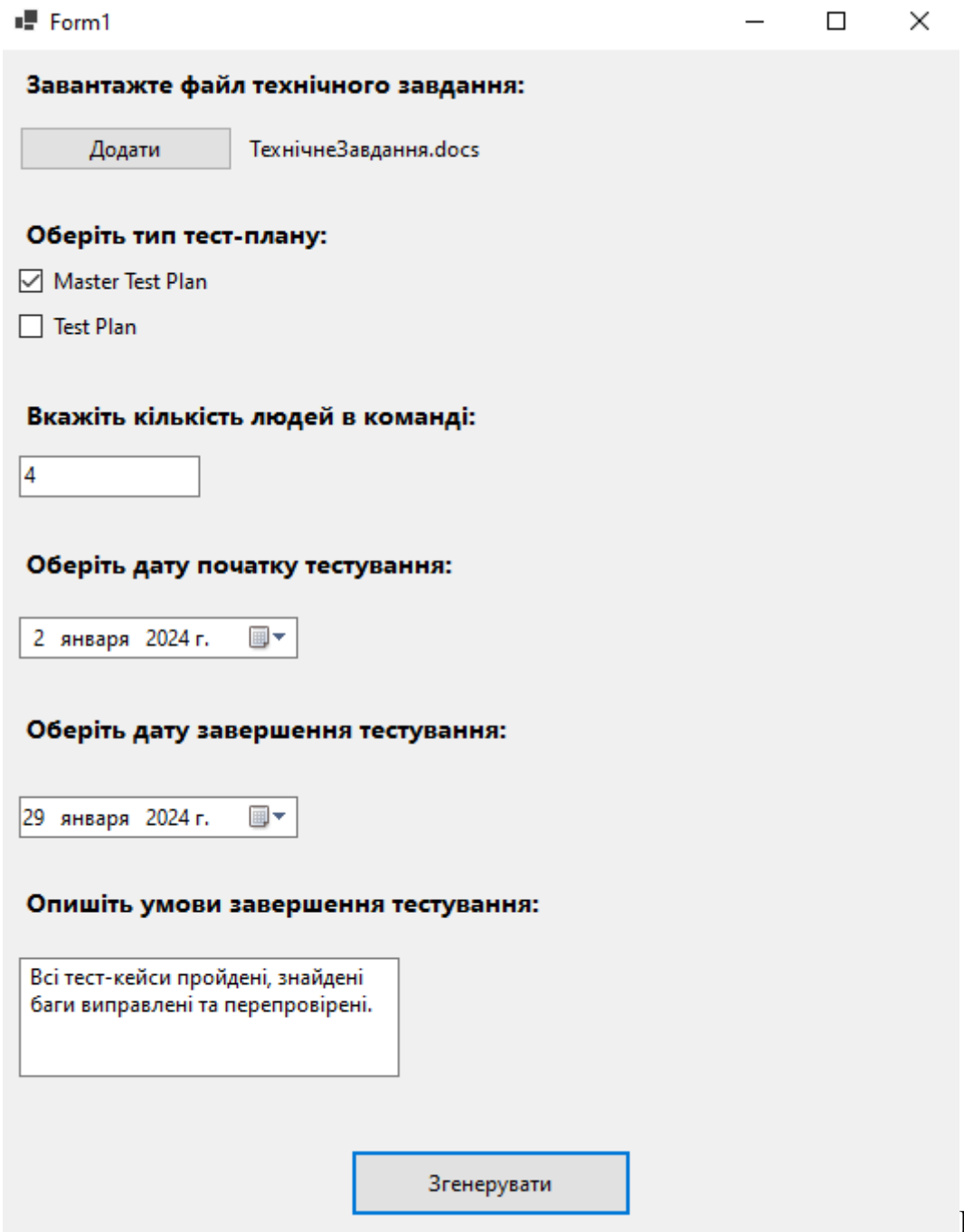
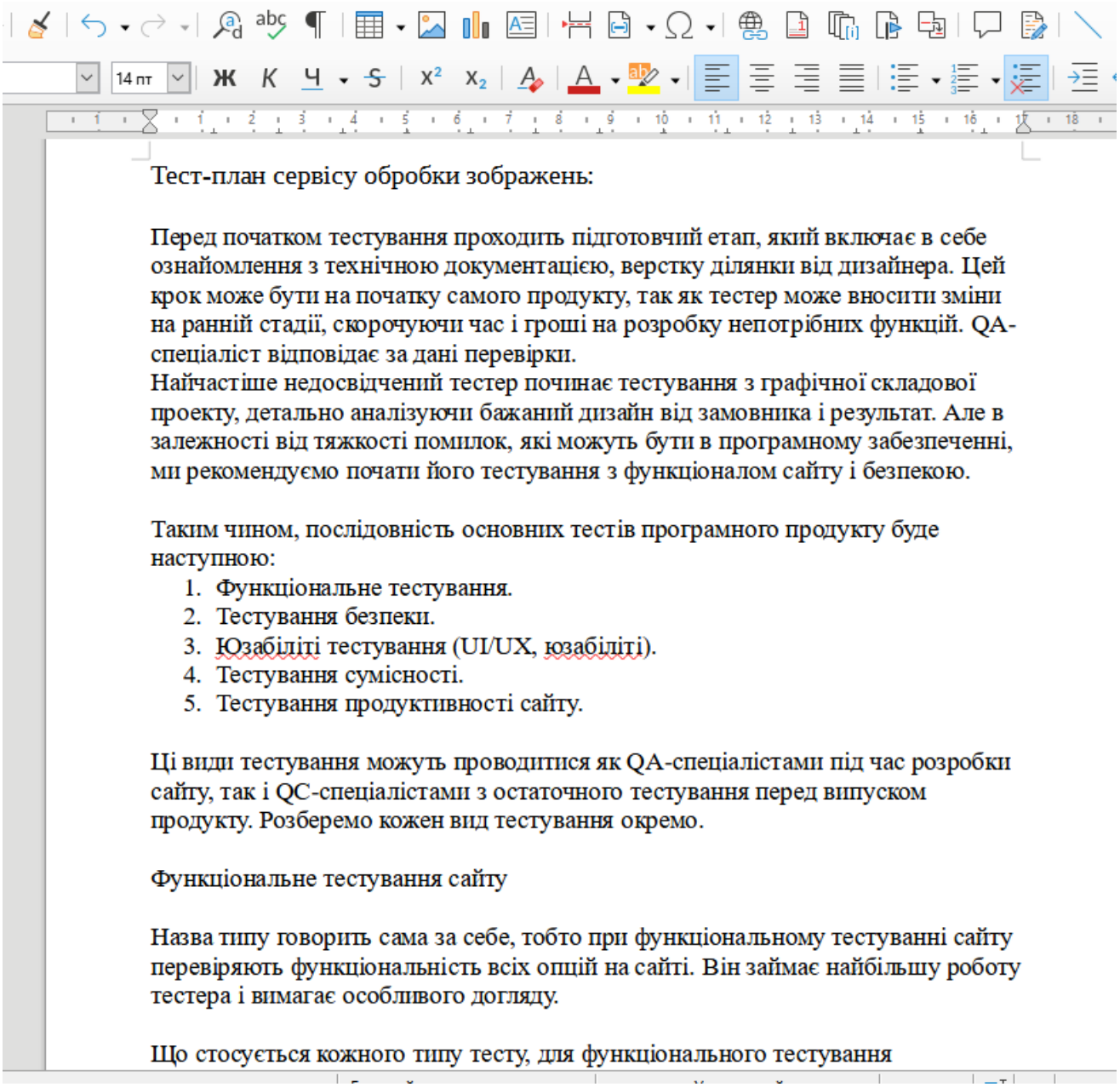


Рисунок 3.2: Інтерфейс додатку



Тест-план сервісу обробки зображень:

Перед початком тестування проходить підготовчий етап, який включає в себе ознайомлення з технічною документацією, верстку ділянки від дизайнера. Цей крок може бути на початку самого продукту, так як тестер може вносити зміни на ранній стадії, скорочуючи час і гроші на розробку непотрібних функцій. QA-спеціаліст відповідає за дані перевірки.
Найчастіше недосвідчений тестер починає тестування з графічної складової проекту, детально аналізуючи бажаний дизайн від замовника і результат. Але в залежності від тяжкості помилок, які можуть бути в програмному забезпеченні, ми рекомендуємо почати його тестування з функціоналом сайту і безпекою.

Таким чином, послідовність основних тестів програмного продукту буде наступною:

1. Функціональне тестування.
2. Тестування безпеки.
3. Юзабіліті тестування (UI/UX, юзабіліті).
4. Тестування сумісності.
5. Тестування продуктивності сайту.

Ці види тестування можуть проводитися як QA-спеціалістами під час розробки сайту, так і QC-спеціалістами з остаточного тестування перед випуском продукту. Розберемо кожен вид тестування окремо.

Функціональне тестування сайту

Назва типу говорить сама за себе, тобто при функціональному тестуванні сайту перевіряють функціональність всіх опцій на сайті. Він займає найбільшу роботу тестера і вимагає особливого догляду.

Що стосується кожного типу тесту, для функціонального тестування

Рисунок 3.3: Тест-план згенерований штучним інтелектом

На малюнку 3.2 можна побачити критерії, які ми обираємо для подальшої генерації тест плану за допомогою інструментів штучного інтелекту. Можна обрати тип тест-плану, кількість людей у команді, дату початку та завершення тестування та критерії завершення тестування. Також, є можливість додати файл з технічним завданням.

На малюнку 3.3 видно результат генерації тест-плану для тестування web-додатку. А саме, тестування сервісу обробки зображень.

ВИСНОВКИ

У результаті виконання магістерської роботи було розглянуто головні підходи для реалізації автоматизованої генерації тест-плану для тестування web-додатку.

Проведено аналіз процесу написання тест-плану і розроблено схему написання тест-плану за міжнародним стандартом IEEE 829. В якому вказано, що тест-план повинен включати в собі такі пункти як: оцінка часу та зусиль, тестові результати у вигляді тест кейсів, тест-скриптів, списку дефектів та тест-репортів; критерії виходу з тестування, підхід до тестування, а саме тип тестування та методі тестування; оцінка ризиків, управління дефектами, ідентифікатор тест-плану, область тестування, а саме тестові кейси, об’єкти тестування та функції для тестування; підготовка тестових сценаріїв, тестова документація, етапи тестування, ролі та відповідальності, та інструменти для тестування.

Проведено аналіз існуючих інструментів штучного інтелекту. Хоч на сьогоднішній день не існує спеціалізованих інструментів штучного інтелекту для автоматизованої генерації тест-плану, за результатами роботи можна зробити висновки, що можна використовувати інструменти штучного інтелекту “загального користування”, головне правильно задати параметри бажаного результату.

Розроблено систему для автоматизованої генерації тест-плану для тестування web-додатку за допомогою таких інструментів як C#, Visual Studio та Windows Form. Також розроблено блок-схему функціонування додатку. Дана система дозволяє автоматично генерувати тест-план відповідно до міжнародного стандарту IEEE 829. Використання системи значно полегшує роботу QA-мастера, адже написання тест-плану, в середньому, займає 2,5 тижні. Використовуючи дану систему, ми отримуємо результат за ~10 хвилин. Отриманий тест план потребує редагування відповідно до деталей проекту, проте редагування займає в середньому 1 тиждень в залежності від масштабу проекту. В результаті маємо скорочення часу при написанні тест-плану у 1,5 тижні.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кріспін Л. (2011). Гнучке тестування: практичне керівництво для тестувальників ПО та гнучких команд.
2. Бейзер Б. (2004). Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем
3. Дудка В.О. (2021) Система автоматичного тестування веб-додатків.
4. Бондаренко О.В. (2020) Дослідження інструментів тестування веб-додатків.
5. Коцюбинський В.Ю. (2019) Аналіз ефективних сучасних інструментів для тестування web-додатків.
6. Регресійне тестування. URL: https://qalearning.com.ua/theory/lectures/ material/regression-testing-crossbrowser/ (дата звернення 02.10.2022).
7. Джек Фолк, Сем Канер, Енг Кек Нгуєн (2001). Тестування програмного забезпечення.
8. Sankar R. Burpsuite – A Beginner’s Guide For Web Application Security or Penetration Testing. URL: https://kalilinuxtutorials.com/ burpsuite/ (дата звернення 03.10.2022).
9. Слободян Р. (2020) Автоматизоване тестування веб-додатків шляхомвзаємодії з користувацьким інтерфейсом.

ДОДАТОК

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Вдосконалення процесу створення тест-плану для тестування Web-додатку з використанням штучного інтелекту»

Виконав: студент групи ПДМ-61 Лозінський Станіслав Вадимович

Керівник: к.пед.н., доц., зав. кафедри ІТ Корецька В.О.

Київ - 2024

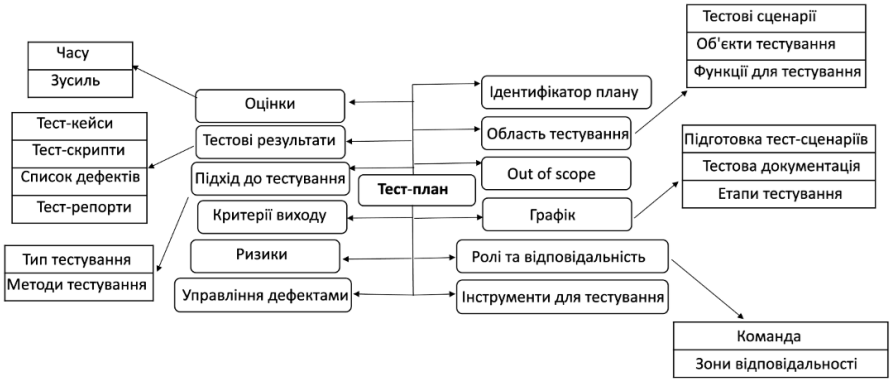
МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: розробка методу автоматизованої генерації тест-плану для тестування Web-додатку з використанням штучного інтелекту.

Об'єкт дослідження: автоматизована генерація тест-плану для тестування Web-додатку.

Предмет дослідження: методи автоматизованої генерації тест-плану для тестування Web-додатку з використанням інструментів штучного інтелекту.

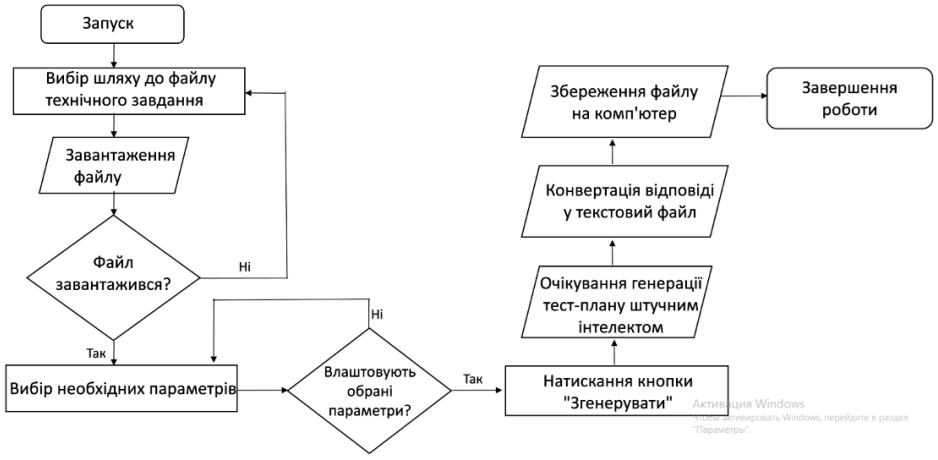
СХЕМА ТЕСТ-ПЛАНУ ЗА МІЖНАРОДНИМ СТАНДАРТОМ IEEE 829



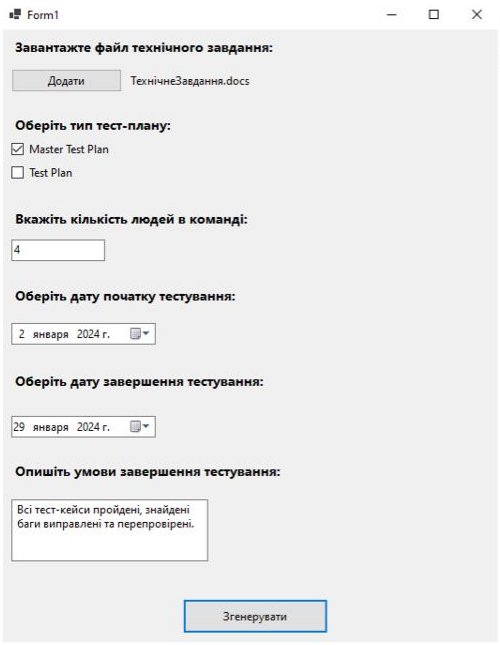
ХАРАКТЕРИСТИКИ ПОПУЛЯРНИХ ІНСТРУМЕНТІВ ШТУЧНОГО ІНТЕЛЕКТУ

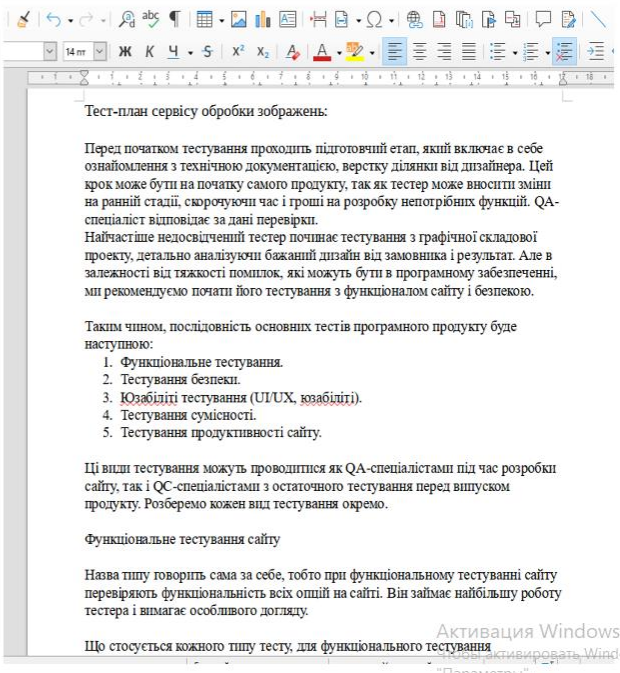
Назва інструменту	Основні характеристики	Недоліки
Chatsonic	Здатність отримувати найновішу інформацію через зв'язок інструменту із Google. Здатність генерувати зображення через підключення нейромереж до Stable Diffusion та DALL-E	Місяць безкоштовного користування. Надалі лише платна основа
Microsoft Bing Chat	Найближчий за можливостями аналог відомого OpenAI, працює на основі моделі GPT-4. Прив'язан до актуальних мережевих даних і може отримувати інформацію з інтернету у режимі реального часу. Повністю безкоштовний	Працює лише у браузері Edge. Розробники попереджають про можливість викривлення даних, не дивлячись на те, що відповіді виглядають переконливо
ChatGPT	Найпопулярніший безкоштовний інструмент. Універсальність обумовлена тренуванням на величезній кількості текстів, завдяки чому практично не має обмежень за тематиками	Використання шаблонів та кліше. Відповідь може бути створена англійською з подальшим перекладом на українську. Нездатність надавати актуальну інформацію

БЛОК-СХЕМА РОБОТИ ДОДАТКУ ДЛЯ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ТЕСТ-ПЛАНУ



ПРАКТИЧНИЙ РЕЗУЛЬТАТ РОБОТИ





Тест-план сервісу обробки зображень:

Перед початком тестування проходить підготовчий етап, який включає в себе ознайомлення з технічною документацією, верстку ділянки від дизайнера. Цей крок може бути на початку самого продукту, так як тестер може вносити зміни на ранній стадії, скорочуючи час і гроші на розробку непотрібних функцій. QA-спеціаліст відповідає за дані перевірки.
Найчастіше недосвідчений тестер починає тестування з графічної складової проекту, детально аналізуючи бажаний дизайн від замовника і результат. Але в залежності від тяжкості помилок, які можуть бути в програмному забезпеченні, ми рекомендуємо почати його тестування з функціоналом сайту і безпекою.

Таким чином, послідовність основних тестів програмного продукту буде наступною:

1. Функціональне тестування.
2. Тестування безпеки.
3. Юзабіліті тестування (UI/UX, юзабіліті).
4. Тестування сумісності.
5. Тестування продуктивності сайту.

Ці види тестування можуть проводитися як QA-спеціалістами під час розробки сайту, так і QC-спеціалістами з остаточного тестування перед випуском продукту. Розберемо кожен вид тестування окремо.

Функціональне тестування сайту

Назва типу говорить сама за себе, тобто при функціональному тестуванні сайту перевіряють функціональність всіх опцій на сайті. Він займає найбільшу роботу тестера і вимагає особливого догляду.

Що стосується кожного типу тесту, для функціонального тестування

ТАБЛИЦЯ ПОРІВНЯННЯ ВИТРАЧЕНОГО ЧАСУ ДЛЯ НАПИСАННЯ ТЕСТ-ПЛАНУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ І БЕЗ

	Середній показник часу для написання тест-плану
З використанням штучного інтелекту	7 днів
Без використання штучного інтелекту	17 днів

ВИСНОВКИ

1. Створено схему компонентів тест-плану згідно з міжнародним стандартом IEEE 829.
2. Проведено аналіз та порівняння існуючих інструментів штучного інтелекту.
3. Розроблено схему та додаток для автоматизованої генерації тест-плану.
4. Проведено практичне застосування штучного інтелекту в процесі написання тест-плану для тестування Web-додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кріспін Л. (2011). Гнучке тестування: практичне керівництво для тестувальників ПО та гнучких команд.
2. Бейзер Б. (2004). Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем
3. Дудка В.О. (2021) Система автоматичного тестування веб-додатків.
4. Бондаренко О.В. (2020) Дослідження інструментів тестування веб-додатків.
5. Коцюбинський В.Ю. (2019) Аналіз ефективних сучасних інструментів для тестування web-додатків. Регресійне тестування. URL: https://qalearning.com.ua/theory/lectures/ material/regression-testing-crossbrowser/ (дата звернення 02.10.2022).
6. Джек Фолк, Сем Канер, Енг Кек Нгуєн (2001). Тестування програмного забезпечення.
7. Sankar R. Burpsuite – A Beginner's Guide For Web Application Security or Penetration Testing. URL: https://kalilinuxtutorials.com/ burpsuite/ (дата звернення 03.10.2022).
8. Слободян Р. (2020) Автоматизоване тестування веб-додатків шляхомвзаємодії з користувацьким інтерфейсом.

АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Корецька В.О., Лозінський С.В. Використання інструментів штучного інтелекту для тестування програмного забезпечення. // Розвиток науки та техніки: виклики 2024 року — Чернівці, 2024. (Подано до публікації)