Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Автоматизація тестування спеціалізованої комп'ютерної системи на основі вбудованих RFID-датчиків
(тема)

Виконав:
студент _2_ курсу, групи *СКСм-20-1*
Садкова М.В.
(прізвище, ініціали)

Спеціальність
123 Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва освітньої програми)

Керівник Чумаченко С.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри АПОТ _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2021

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Садкова Марія Володимирівна
(прізвище, ім'я, по батькові)

1. Тема роботи
   Автоматизація тестування спеціалізованої комп'ютерної системи на основі вбудованих RFID-датчиків

затверджена наказом університету від «_04_» __11__2021 р. № 1635Ст_

2. Термін подання студентом роботи до екзаменаційної комісії ____ _____ 20__ р.

3. Вихідні дані до роботи : вимоги до комп`ютерної системи; тестовий апаратний додаток;теоретичні відомості

4. Перелік питань, що потрібно опрацювати в роботі
   Аналіз предметної області та постановка задачі.
   Аналіз методів та різновидів тестування. Огляд автоматизації тестування
   Створення тестової документації відносно тестового апаратного додатку.

   Створення програмного додатку додатку автоматизації тестування
   Аналіз отриманих результатів

5. Перелік графічного матеріалу 16 слайдів

6. Дата видачі завдання 13.09.2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи (проекту)	Термін виконання етапів роботи (проекту)	Примітка
1	Отримання завдання		
2	Аналіз літератури по темі мануального тестування		
3	Аналіз літератури по темі автомтатизації тестування		
4	Огляд аппаратного додатку		
5	Написання тестової документації та тестування додатку		
6	Створення програмного додатку для автоматізації тестування		
7	Оформлення пояснювальної записки		

Студент _____
(підпис)

Керівник проекту _____ Чумаченко С.В.
(підпис) (прізвище, ім’я, по батькові, посада, звання)

« » _____ 2021р.

РЕФЕРАТ

Пояснювальна записка містить 57 сторінок, 15 рисунків, 5 таблиці, 3 додатки, 11 джерел за переліком посилань.

МАНУАЛЬНЕ ТЕСТУВАННЯ, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, QML, БЕЗДРОТОВА ПЕРЕДАЧА ДАНИХ, RFID, ВБУДОВАНІ СИСТЕМИ

Задачею кваліфікаційної роботи є вивчення та аналіз методів підвищення якості програмного забезпечення, як мануальним способом так і аналіз області автоматизації тестування, а також створення артефактів на основі тестового апаратного додатку безконтактної передачі даних.

Метою проектування є створення тестової документації та програмного додатку для автоматизації тестування, аналіз методів підвищення якості апаратних та програмних пристроїв.

Результатом кваліфікаційної роботи є розроблена програмна модель для використання під час тестування, автоматизації підвищення якості та ведення тестових артефактів з можливістю подальшого розвитку у рамках додавання користувацьких кастомних артефактів та роботи з сторонніми бібліотеками для автоматизації тестування.

ABSTRACT

The explanatory note contains 83 pages, 15 figures, 5 tables, 3 appendices, 11 sources according to the list of references.

MANUAL TESTING, AUTOMATION TESTING, QML, WIRELESS TRANSFER PROTOCOL, RFID, EMBEDDED SYSTEM

The task of the qualification work is to study and analyze methods of improving the quality of software, both manual and analysis of the field of test automation, as well as the creation of artifacts based on the test hardware application of contactless data transmission.

The purpose of the design is to create test documentation and software applications to automate testing, analysis of methods to improve the quality of hardware and software devices.

The result of the qualification work is a developed software model for use in testing, automation of quality improvement and maintenance of test artifacts with the possibility of further development in the addition of custom custom artifacts and work with third-party libraries to automate testing.

ЗМІСТ

РЕФЕРАТ.....4
ABSTRACT.....5
ЗМІСТ.....6
ВСТУП.....8
1 АНАЛІЗ ПРДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 9
1.1Загальні відомості про забезпечення якості та тестування.....9
1.2 Тестові артефакти.....12
1.3 Класифікація видів та рівнів тестування.....13
1.4 Умови тестування.....17
1.5 Техніки тест дизайну.....18
1.6 Життевий цикл бага.....20
1.7 Мета проектування.....23
2 АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ТА ВИБІР ТЕХНОЛОГІЇ.....24
2.1 Огляд існуючих задач в області автоматизації.....24
2.3 Технологія.QML.....25
2.4 CMake технологія.....27
3 РОЗРОБКА ТЕСТОВОЇ ДОКУМЕНТАЦІЇ 32
4 РОЗРОБКА МОДЕЛІ СИСТЕМИ ВЕДЕННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ 40
5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ.....55
ВИСНОВКИ.....56
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....58
ДОДАТОК А.....59
ДОДАТОК Б.....74
ПРЕЗЕНТАЦІЯ.....74
ДОДАТОК В.....83

В.1 Публікація на тему “ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ ЯЗЫКОВ ОПИСАНИЯ АППАРАТУРЫ VHDL И VERILOG”.....83
В.2 Публікація на тему “МОДЕЛИ ОБЛАЧНЫХ УСЛУГ: IAAS, PASS, SAAS, КАК ОСНОВНЫЕ МОДЕЛИ УСЛУГ ОБЛАЧНЫХ СЕРВИСОВ ”...85

ВСТУП

На сьогоднішній день жоден розроблюваний продукт не може бути створений без тестування. Тестування забезпечує якість, що підвищує попит користувачів та їх довіру до розробників продукту. Користувачі обирають серед безлічі схожих додатків саме той, який не тільки задовольняє бажані потреби, та ще й робить це на відповідному рівні якості. Саме тому тестування починається з організаційних документаційних робіт ще до створення аппаратного чи програмного продукту.

Проте мануальні тестувальники не завжди можуть мають достатні ресурси для самостійної роботи та через це виникло поняття автоматизації та різноманітні системи контролю версій.

Таким чином, зараз все активніше починають набувати популярність різноманітни баг-трекінгові системи та додатки для автоматизації тестування, які базуються на ручній роботі тестувальників та допомогають оптимізувати процес.

Метою кваліфікаційної роботи є створення прототипу системи для автоматизації тестування апаратного додатку з вбудованим RFID датчиком, логування результатів роботи додатку, ведення тестової документації та безпосереднього створення тестових артефактів.

1 АНАЛІЗ ПРДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості про забезпечення якості та тестування

Тестування програмного забезпечення (Software Testing) – це перевірка відповідності між реальною та очікуваною поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином. У більш широкому сенсі, тестування - це одна з технік контролю якості, що включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) і аналізу отриманих результатів (Test Analysis).

Тестування необхідне тому, що всі люди роблять помилки. Деякі із них можуть бути незначними, в той час як інші – мати дуже руйнівні наслідки. Все, що створюється людиною, може містити помилки, які будуть коштувати людського життя. Саме тому будь-який продукт потребує перевірки – тестування, перш, ніж його можна буде ефективно та безпечно використовувати. Те ж саме справедливо і для програмного та аппаратного забезпечення[10].

Комп’ютерні технології все глибше проникають в повсякденне життя. Програмне забезпечення управляє роботою багатьох речей, якими люди користуються кожного дня, від турнікетів у метрополітені до телефонів, складних систем охорони або бухгалтерського обліку . Хоча би один раз кожен зустрічався с багами в програмному або апаратному забезпеченні, наприклад пусте повідомлення з соціальної мережі або невідображені картинки на онлайн-сайті.

Проте, не всі помилки однаково небезпечні – для різних програмних систем рівні ризику можуть відрізнятися.

Ризик (risk) – фактор, який може призвести до негативних наслідків в майбутньому; як правило, виражається через вірогідність виникнення таких наслідків та їх впливу на систему.

Окрім того, рівень ризику буде залежати від вірогідності виникнення негативних наслідків.

Наприклад, одна й та ж незначна помилка, скажімо, опечатка, може мати абсолютно різні рівні ризику для різних програм:

– опечатка в описанні на особистій сторінці в соціальній мережі навряд чи буде мати серйозні наслідки, хіба що викличе посмішку;

– така ж проста опечатка, допущена в описанні діяльності великої компанії, розміщеної на її сайті, вже небезпечна, так як неопосередковано свідчить про непрофесіоналізм її співробітників;

– опечатка в коді програми, для девайсу, який рахує рівень кисню у пацієнта у лікарні, може мати дуже великі та фатальні наслідки.

Наразі існує декілько напрямів заюезпечення якості, такі як Quality Assurance, тестування і Quality Control, які відповідають за різні галузі процессу. Після огляду були віявлені наступні відмінності, які наведені у таблиці 1.1.

Таблиця 1.1 – Різниця між тестуванням,QA та QC

Quality Assurance	Quality Control	Тестування
Комплекс заходів, який охоплює всі технологічні аспекти на всех етапах розробки, запуску та експлуатації систем для забезпечення необхідного рівня якості програмного продукту	Процес контролю відповідності системи, що розробляється, вимогам, що виставлені до неї	Процес, що відповідає безпосередньо застворення та проходження тест-кейсів, знаходження та локалізація дефектів і т.д.
Фокус в більшій мірі на процеси та засоби, ніж на безпосереднє виконання тестування системи	Фокус на виконання тестування шляхом запуску програми з метою визначення дефектів із використанням	Фокус на виконання тестування як такого

	затверджених процесів та засобів	

Розглянемо основні поняття.

Верифікація (Verification) – це процес оцінки системи або її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умов, сформованим на початку цього етапу. Тобто чи виконуються наші цілі, терміни, завдання по розробці проекту, визначені на початку поточної фази.

Валідація (Validation) – це визначення відповідності розробляється ПО очікуванням і потребам користувача, вимогам до системи.

План тестування (Test Plan) – це документ, що описує весь обсяг робіт з тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення .

Тест дизайн (Test Design) – це етап процесу тестування ПО, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування.

Тестовий випадок (Test Case) – це артефакт, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестируемой функції або її частини.

Баг / Дефект Репорт (Bug Report) – це тестовий артефакт, що описує ситуацію або послідовність дій призвела до некоректної роботи об'єкта тестування, із зазначенням причин і очікуваного результату[11].

Тестове Покриття (Test Coverage) – це одна з метрик оцінки якості тестування, що вдає із себе щільність покриття тестами вимог або виконуваного коду.

У програмного та апаратного забезпечення виділяють наступні ключові характеристики якості, на базі яких формуються вимоги:

– функціональність (Functionality) – визначається здатністю ПО вирішувати завдання, які відповідають зафіксованим і очікуваним потребам

користувача, при заданих умовах використання ПЗ. Тобто ця характеристика відповідає за те, що ПО працює справно і точно, функціонально сумісно, відповідає стандартам галузі і захищене від несанкціонованого доступу;

– надійність (Reliability) - здатність ПЗ виконувати необхідні завдання в позначених умовах протягом заданого проміжку часу або вказану кількість операцій. Атрибути даної характеристики - це завершеність і цілісність всієї системи, здатність самостійно і коректно відновлюватися після збоїв в роботі, відмовостійкість;

– зручність використання (Usability) - можливість легкого розуміння, вивчення, використання і привабливості ПО для користувача;

– ефективність (Efficiency) - здатність ПЗ забезпечувати необхідний рівень продуктивності у відповідність з виділеними ресурсами, часом і іншими позначеними умовами;

– зручність супроводу (Maintainability) - легкість, з якою ПО може аналізуватися, тестуватися, змінюватися для виправлення дефектів, для реалізації нових вимог, для полегшення подальшого обслуговування та адаптуватися до наявного оточенню;

– портативність (Portability) - характеризує ПО з точки зору легкості його перенесення з одного оточення (software / hardware) в інше.

1.2 Тестові артефакти

У відповідність з процесами або методологіями розробки ПО, під час проведення тестування створюється і використовується певна кількість тестових артефактів (документи, моделі). Під кожний проект обираються свої основні артефакти, проте були виявлені найбільш поширені:

– план тестування (Test Plan) - це документ описує весь обсяг робіт з тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і

закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення;

– Test Case & Test suite - це послідовність дій, по якій можна перевірити чи відповідає тестована функція встановленим вимогам;

– дефекти / Баг репорт (Bug Reports / Defects) - це документи, що описують ситуацію або послідовність дій призвела до некоректної роботи об'єкта тестування, із зазначенням причин і очікуваного результату. Приклад оформлення бага наведено у рисунку 1.1.

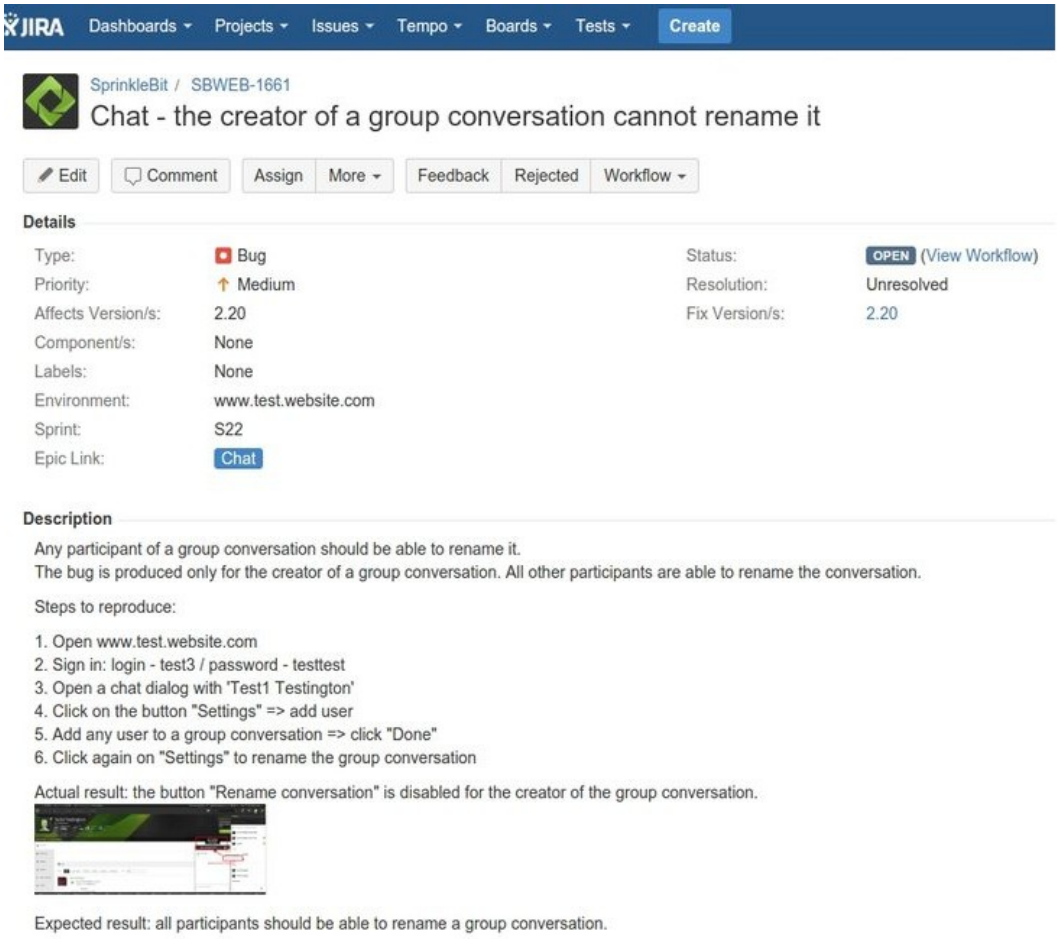

Рисунок 1.1 – Приклад баг-репорту у системі багтрекінгу Jira

1.3 Класифікація видів та рівнів тестування

Програмне та апаратне забезпечення має цілу совокупність вимог та

характеристик, які все поокремо потрібно тестувати, так як не завжди баги образу можно побачити вже в кінцевому продукті, та поділ на умовні группи, як показала практика, сильно скорочує витрати ресурсі, тому було прийнято розділити види на наступні групи:

- функціональні;
- нефункціональні;
- пов'язані зі змінами.

Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component / Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing). Функціональні види тестування розглядають зовнішню поведінку системи[1]. Далі перераховані одні з найпоширеніших видів функціональних тестів:

- Функціональне тестування (Functional testing);
- тестування безпеки (Security and Access Control Testing);
- тестування взаємодії (Interoperability Testing);

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, "Як" система працює. Далі перераховані основні види не функціональних тестів:

Види тестування продуктивності:

- тестування навантаження (Performance and Load Testing);
- стресове тестування (Stress Testing);
- тестування стабільності або надійності (Stability / Reliability Testing);
- об'ємне тестування (Volume Testing);
- тестування установки (Installation testing);
- тестування зручності користування (Usability Testing);
- тестування на відмову і відновлення (Failover and Recovery Testing);

– конфігураційне тестування (Configuration Testing).

Пов'язані зі змінами види тестування. Після проведення необхідних змін, таких як виправлення бага / дефекту, програмне забезпечення повинно бути перетестировано для підтвердження того факту, що проблема була дійсно вирішена. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності додатку або правильності здійсненого виправлення дефекту:

– димове тестування (Smoke Testing);
– регресійне тестування (Regression Testing);
– тестування збірки (Build Verification Test);
– санітарне тестування або перевірка узгодженості (Sanity Testing).

Тестування на різних рівнях проводиться протягом усього життєвого циклу розробки і супроводу програмного забезпечення. Рівень тестування визначає те, над чим виробляються тести: над окремим модулем, групою модулів або системою, в цілому. Проведення тестування на всіх рівнях системи - це запорука успішної реалізації і здачі проекту. Виділяють 4 основних рівня тестування:

Перший: Компонентне (модульне) тестування перевіряє функціональність і шукає дефекти в частинах додатка, які доступні і можуть бути протестовані по-окремо (модулі програм, об'єкти, класи, функції і т.д.). Зазвичай компонентне (модульне) тестування проводиться викликаючи код, який необхідно перевірити і при підтримці середовищ розробки, таких як фреймворки (frameworks - каркаси) для модульного тестування або інструменти для налагодження. Всі знайдені дефекти, як правило виправляються в коді без формального їх опису в системі менеджменту багів (Bug Tracking System).

Один з найбільш ефективних підходів до компонентного (модульного) тестування - це підготовка автоматизованих тестів до початку основного кодування (розробки) програмного забезпечення. Це називається розробка від тестування (test-driven development) або підхід тестування спочатку (test first

approach). При цьому підході створюються і інтегруються невеликі шматки коду, навпроти яких запускаються тести, написані до початку кодування. Розробка ведеться до тих пір, поки всі тести не будуть успішно пройдені.

Другий: Інтеграційне тестування призначене для перевірки зв'язку між компонентами, а також взаємодії з різними частинами системи (операційною системою, обладнанням або зв'язку між різними системами).

Рівні інтеграційного тестування:

1) Компонентний інтеграційний рівень (Component Integration testing) - перевіряється взаємодія між компонентами системи після проведення компонентного тестування;
2) Системний інтеграційний рівень (System Integration Testing) - перевіряється взаємодія між різними системами після проведення системного тестування.

Третій: системне тестування - це перевірка як функціональних, так і не функціональних вимог в системі в цілому. При цьому виявляються дефекти, такі як неправильне використання ресурсів системи, непередбачені комбінації даних користувача рівня, несумісність з оточенням, непередбачені сценарії використання, відсутня або неправильна функціональність, незручність використання і т.д. Для мінімізації ризиків, пов'язаних з особливостями поведінки системи в тому чи іншому середовищі, під час тестування рекомендується використовувати оточення максимально наближене до того, на яке буде встановлено продукт після видачі.

Четвертий: Приймальне тестування - формальний процес тестування, який перевіряє відповідність системи вимогам і проводиться з метою:

- визначення чи задовольняє система приймальним критеріям;
- винесення рішення замовником або іншою уповноваженою особою приймається додаток чи ні.

Приймальне тестування виконується на підставі набору типових тестових випадків і сценаріїв, розроблених на підставі вимог до додатку.

1.4 Умови тестування

Визначення необхідних умов:

- Необхідними умовами істинності твердження А називаються умови, без дотримання яких А не може бути істинним;
- достатніми називаються такі умови, при наявності (виконання, дотриманні) яких твердження А є істинним.

Необхідні умови. Тестування не можливо без об'єкта тестування, звідси перша умова: наявність об'єкта тестування, доступного для проведення випробувань. Далі, щоб тестування все ж відбулося, потрібен виконавець, значить, другим необхідною умовою буде наявність виконавця, причому їм може бути, людина або машина, або комбінація людина + машина.

Визначення достатніх умов. Бо ж не кожна дія, вчинена над програмою, з метою отримання очікуваної поведінки, є тестуванням, постало питання про те, що сама по собі є ціль , а саме: тестування, є одним з достатніх умов. Беручи за основу те, що наявність плану тестування прямо вказує на намір проведення тестування, отримуємо, що тест план є одним з достатніх умов.

При проведенні тестування, людина або машина повинні будуть виконувати якісь дії для перевірки реального і очікуваного поведінки програми. Значить, наявність тест кейсів / тестів також є достатньою умовою.

Для підтвердження, що тестування відбулося, необхідний звіт про результати.

Таким чином маємо наступні необхідні і достатні умови для проведення тестування:

Необхідні умови:

- наявність об'єкта тестування, доступного для проведення випробувань;
- наявність виконавця (ів) (в залежності від виду проведених випробувань їм може бути як людина, так і машина або комбінація людина + машина).

На основі вище перерахованих необхідних умов, достатніми умовами можна виділити:

- наявність об'єкта тестування, доступного для проведення випробувань;
- наявність виконавця(ів) (в залежності від виду діяльності на різних фазах їм може бути як людина, так і машина або комбінація людина + машина);
- наявність плану тестування;
- наявність тест кейсів / тестів;
- наявність звіту, що підтверджує виконання завдань і досягнення цілей, з тестування об'єкта.

1.5 Техніки тест дизайну

Тест-дизайн – це етап процесу тестування ПЗ, на якому проектуються та створюються тестові випадки (тест-кейси), відповідно до визначених раніше критеріїв якості та цілей тестування.

Просто кажучи, завдання тест-аналітиків і дизайнерів зводиться до того, щоб, використовуючи різні стратегії та техніки тест-дизайну, створити набір тестових випадків, що забезпечує оптимальне тестове покриття програми, що тестується. На більшості проектів ці ролі виконує інженер QA[2].

Тестове покриття - це одна з метрик оцінки якості тестування, що представляє собою щільність покриття тестами вимог або виконуваного коду.

Наразі були викремлені наступні основні підходи до оцінки та вимірювання тестового покриття:

- покриття вимог (Requirements Coverage) – оцінка покриття тестами функціональних та нефункціональних вимог до продукту, шляхом побудови матриць трасування (traceability matrix);
- покриття коду (Code Coverage) - оцінка покриття виконуваного коду тестами шляхом відстеження неперевірених у процесі тестування частин

програмного забезпечення;

- тестове покриття на базі аналізу потоку управління - оцінка покриття заснована на визначенні шляхів виконання коду програмного модуля і створення тест кейсів для покриття цих шляхів.

Техніки тест дизайну:

- Еквівалентний Поділ (Equivalence Partitioning – EP). – є діапазон допустимих значень від 1 до 10, тестувальник повинен обрати одне правильне значення всередині інтервалу, скажімо, 5 та одне неправильне значення поза інтервалом – 0, таким чином вважається, що результати дослідженнь із використанням значень в діапазоні будуть мати єквівалетні результати;
- Аналіз Граничних Значень (Boundary Value Analysis – BVA). Якщо взяти приклад вище, як значення для позитивного тестування виберемо мінімальну і максимальну межі (1 і 10) і значення більше і менше меж (0 і 11). Аналіз Граничних значень може бути застосований до полів, записів, файлів, або до будь-яких сутностей, що мають обмеження;
- Причина/Наслідок (Cause/Effect - CE). Це, як правило, введення комбінацій умов (причин) для отримання відповіді від системи (Слідство);
- Передбачення помилки (Error Guessing – EG). Коли тест-аналітик використовує свої знання системи та здатність до інтерпретації специфікації на предмет того, щоб "передбачити", за яких вхідних умов система може видати помилку.;
- Вичерпне тестування (Exhaustive Testing – ET) – це крайній випадок. У межах цієї техніки потрібно перевірити всі можливі комбінації вхідних значень, і в принципі це має знайти всі проблеми. Насправді застосування цього методу неможливо через величезної кількості вхідних значень;
- Парне тестування (Pairwise Testing – PT) – це техніка формування наборів

тестових даних. Сформулювати суть можна, наприклад, таким чином: формування таких наборів даних, в яких кожне тестоване значення кожного з параметрів, що перевіряються хоча б один раз поєднується з кожним тестованим значенням всіх інших параметрів, що перевіряються.

Технік ведення тест дизайну існує безліч, проте найбільш розповсюдженими являються техніки класів еквівалентності та гранічних значень. Таким чином були виделені наступні плюси та мінуси техніки аналізу класів еквівалентності.

До плюсів можна віднести помітне скорочення часу та покращення структурованості тестування, але мінусом є такий фактор як те, що при неправильному використанні техніки ми ризикуємо втратити баги.

Що до техніки аналізу граничних значень були зроблені наступні висновки:

Ця техніка додає до техніки аналізу класів еквівалентності орієнтованість на конкретний тип помилок.

Тобто техніка аналізу класів еквівалентності просто говорити нам про те, що потрібно розбити всі тести на класи та провести тестування всіх класів. А техніка граничних значень спрямована на виявлення конкретної проблеми – виникнення помилок на межах класів еквівалентності.

Але, як і техніки аналізу класів еквівалентності, ефективність техніки аналізу граничних значень залежить від правильності її використання. Ми повинні докласти зусиль, щоб правильно визначити класи еквівалентності та їх межі. Якщо ми поставимося до цього поверхово, то ризикуємо пропустити помилки.

1.6 Життевий цикл бага

Як було зазначено раніше, баг репорт являється одним з найважливіших тестових артефактів. Даний артефакт може аходитися в різних станах так як він

не просто робить помітку про несправність, а супроводжує цілу версію додатку до самого реліза задля запобігання повторенню его виявлення та виявлення йому подібних проблем.

DEFECT LIFE CYCLE або Bug Life Cycle - це особливий набір станів, через які проходить помилка протягом усього свого життя.

Мета життєвого циклу Дефекту полягає в тому, щоб легко координувати зміни стану помилок для різних працівників та систематизувати процес усунення помилок.

Кількість станів, через які проходить дефект, варіюється від проекту до проекту. Були проаналізовані та подібрані основні обов'язкові стани баг-репоту:

- новий: коли новий дефект реєструється та публікується вперше. Йому надано статус Новий;
- призначено: як тільки тестер публікує повідомлення про помилку, провідний тестувальник затверджує помилку і призначає її команді розробників:
- відкрито: розробник починає аналіз і працює над виправленням дефекту;
- Виправлено: Коли розробник вносить необхідні зміни в код і перевіряє цю зміну, він може зробити статус помилки "Виправлено";
- очікування повторного тестування: після усунення дефекту розробник надає конкретний код повторного тестування коду тестувальнику. Оскільки тестування програмного забезпечення залишається в очікуванні з боку тестувальників, надається статус «очікує повторного тестування»;
- повторне тестування: тестер виконує повторне тестування коду на цьому етапі, щоб перевірити, чи виправлений дефект розробником чи ні, та змінює статус на «Повторне тестування»;
- перевірено: Тестер повторно тестує помилку після її виправлення розробником. Якщо в програмному забезпеченні не виявлено помилок, то помилка усунута, і статус «перевірений»;

– повторне відкриття: якщо помилка зберігається навіть після того, як розробник виправив помилку, тестер змінює статус на "відновлено". Ще раз помилка відбувається через життєвий цикл;
– закрито: якщо помилки більше не існує, то тестер надає статус «Закрито»;
– Дублювати: якщо дефект повторюється двічі або дефект відповідає одній і тій же концепції помилки, статус змінюється на дублікат;
– відхилено: якщо розробник вважає, що дефект не є справжнім, він замінює дефект на відхилений;
– відкладено: якщо поточна помилка не має основного пріоритету і очікується, що вона буде виправлена в наступному випуску програми, статус «Відкладено» надається таким помилкам;
– не помилка: якщо це не впливає на функціональність програми, то статус, наданий помилці - "Не помилка".

Нижче розроблена модель життєвого циклу, що охоплює описані стани:

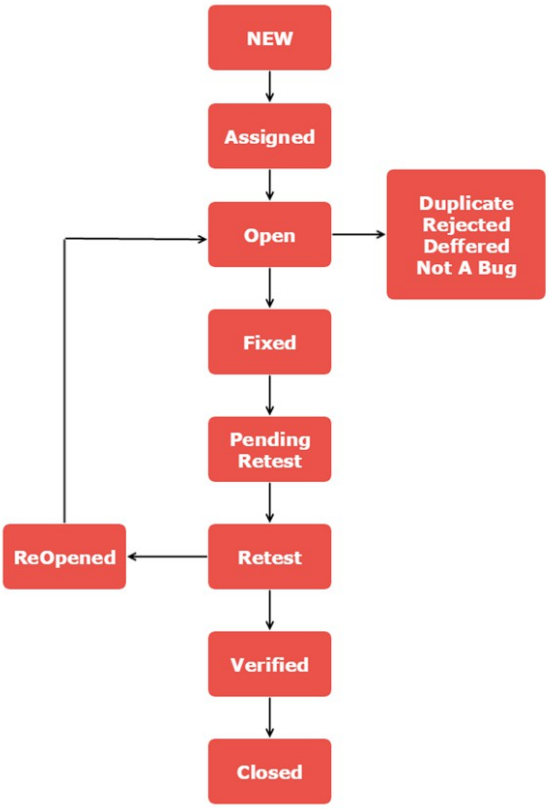

Рисунок 1.2 – Діаграма життевого циклу бага

1.7 Мета проектування

На основі викладеного вище матеріалу можно зробити висновок, що найчастіше розробка починається саме з тестування вхідних даних до проекту та вимог, що є мануальним та ресурсозатратним процесом. Виконується мануальний аналіз проекту та створення оптимальних, по витраченню часу та інших ресурсів, перевірки. Проте цей процесс можно зробити зручнішим шляхом розробки додатку, для автотатичного керування, оновлення, та збереження артефактів та логів: На виході буде отримано: зменшення часу на створення документації проету; прискорення адаптації нових учасників проекту; зменщення затрат на регресійне тестування нової версії; можливість відмовитись від сторонніх платних додатків логування роботи апаратних додатків.

Метою дослідження даної кваліфікаційної роботи є розробка програмної моделі для логування подій на тестовому спеціалізованому пристрої, , який був розроблен у попередній роботі, з вбудованим RFID датчиком та ведення тестової документації на базі технології QML. Дане завдання включає в себе аналіз вимог та написання на їх основі тестових артефактів, а саме баг репортів та чек-листа відповідно до функціоналу тестової моделі безконтактної передачі даних.

Для цього необхідно вирішити такі завдання:

1. провести аналіз функціоналу тестового апаратного додатку;
2. відповідно до теоретичного матеріалу скласти чек-ліст та тест кейси на базі тестового апаратного додатку;
3. розробити систему ведення тестових артефактів та логування системи;
4. виконати тестування додатку на основі створеної документації;
5. провести аналіз отриманих результатів.

2 АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ТА ВИБІР ТЕХНОЛОГІЇ

2.1 Огляд існуючих задач в області автоматизації

Метою будь-якого проекту з тестування є забезпечення якості розроблюваного продукту. Автоматизація підвищує ефективність тестування і, отже, покращує якість.

Як правило, життєвий цикл програмних продуктів досить тривалий, система росте за рахунок нового функціоналу, який необхідно постійно тестувати, але не варто забувати і про старий, раніше протестований функціонал. Після внесення змін до ПЗ старий функціонал може перестати працювати або перестати бути релевантним.

Для того щоб гарантувати якість наявного функціоналу застосовується регресійне тестування, яке часто й стає об'єктом автоматизації.

Автоматизація функціонального тестування може застосовуватися на різних рівнях. На нижчих рівнях (модульному і інтеграційному) автоматизація зводиться до написання unit-тестів, які представляють якусь оболонку, що дозволяє запускати і тестувати окремі процедури[3].

Автоматизація на високому рівні (системному і приймальних) передбачає імітацію роботи реального користувача з використанням інтерфейсу. Виходячи з цих інтерфейсів і вибираються засоби автоматизації, які можуть бути застосовані до:

- Windows - інтерфейсу;
- Web - інтерфейсу;
- Web - сервісів (тут імітується обмін повідомленнями за обраним протоколу).

Метою автоматизації є підвищення ефективності процесу тестування за рахунок вивільнення фахівців зайнятих в регресійному тестуванні. Проведені

автоматичним способом тести вимагають менших людських ресурсів, та , відповідно, коштують дешевше. Крім того, зменшується негативний вплив на якість програмного продукту людського фактора.

Так же існує потреба в автоматизації високрівневих організаціонних процессів. Злагоджена командна робота – швидка командна робота. Вірно налагоджені процесси, грамотний підхід до вибору викристованних технологій та організація коммунікації участників процессу позволяють уникнути місскомунікацій, затримок та зайвої роботи через недостовірню або недостатню інформацію. Тому можно виділити окремий вид автоматизації тестування : розробка моделей програмних та апаратних додатків для автоматизації проведення ручних перевірок, створення та обміном тестовими артефактами з урахуванням специфікацій конкретного проекту та потреб команди.

2.3 Технологія.QML

В якості основної технології для розробки візуального компотента додатку було вирішено обрати QML так як він, на відміну від других подібних технологій, таких як QtWidgets, WinForms та інші, через його швидкість, та більш компатне та читабілні реалізації інтерфейсів та анімаций. Також данна технологія має більш розвинуті графічні бібліотеки, що дозволить підтримати одинаковий дизайн на різних платформах та девайсах, на відміну від вбудованих дефолтних у WinForms та інших технологіях бібліотек.

QML - це декларативна мова, яка дозволяє описувати інтерфейси користувача в термінах їх візуальних компонентів і того, як вони взаємодіють і співвідносяться один з одним. Це добре читається мова, яка була розроблена для забезпечення можливості динамічного з'єднання компонентів, і вона дозволяє легко повторно використовувати і настроювати компоненти в інтерфейсі користувача. Використовуючи модуль QtQuick, дизайнери і розробники можуть легко створювати плавні анімовані інтерфейси користувача

в QML і мати можливість підключати ці інтерфейси до будь-яких внутрішнім бібліотекам C++.

QML – це специфікація користувальницького інтерфейсу і мова програмування. Він дозволяє як розробникам, так і дизайнерам створювати високопродуктивні, плавно анімовані та візуально привабливі додатки. .

Мова QML і інфраструктура двигуна надаються модулем Qt QML. Докладнішу інформацію про мову QML див. у документації модуля Qt QML.

Qt Quick-це стандартна бібліотека типів QML і функціональність для QML. Вона включає візуальні типи,інтерактивні типи,анімації,моделі та уявлення,ефекти частинок і шейдерні ефекти.Розробник QML-програми може отримати доступ до всієї цієї функціональності за допомогою одного оператора імпорту.

Бібліотека QtQuick QML надається модулем Qt Quick. Щоб отримати докладнішу інформацію про різні типи QML та інші функції Qt Quick, зверніться до документації модуля Qt Quick . Qt Quick додає візуальні типи, типи анімації та інші типи QML на додаток до стандартних типів QML Qt QML:

- візуальні типи в QML;
- відповідь на введення користувача в QML;
- фнімації в QML;
- відображення тексту в QML;
- макети в QML;
- підтримка стилю та теми;
- інтеграція JavaScript у QML.

Для набору елементів керування інтерфейсу користувача модуль Qt Quick Controls реалізує кілька елементів керування, таких як кнопки, меню і уявлення. Ці елементи керування мають кілька вбудованих стилів, які можна використовувати, а також підтримують створення стилів користувача.

2.4 CMake технологія

Для реалізації системнї часті розроблюваної моделі було вирішено використовувати технологую CMake. Дана технологія дуже гнучка та позволяє зробити додаток кросплатформеним під Віндоус, МакОс та Лінукса, на відміну від аналогів, MsBuild, Rake та інших, які не мають зручного вбудованого функціоналу для кросплатформеної роботи.

Незалежно від того, чи є розробник досвідченим або тільки починає кар'єру програмного забезпечення, не уникнути процесу ознайомлення з рядом інструментів, щоб перетворити вихідний код проекту в те, що кінцевий користувач може реально використовувати. Компілятори, компонувальники, тестові рамки, системи пакування і більше, все це сприяє складності розгортання високоякісного надійного програмного забезпечення.

Хоча деякі платформи мають домінуюче середовище IDE, яке спрощує деякі аспекти цього (наприклад, Xcode і Visual Studio), проекти, які потребують підтримки кількох платформ, не завжди можуть використовувати їх особливості. Підтримка кількох платформ додає більше ускладнень, які можуть вплинути на все - від набору доступних інструментів до різних доступних можливостей і обмежень. Типовому розробнику можна пробачити за те, що він втратив хоча б частину свого розуму, намагаючись зберегти поверх усієї картини.

На щастя, існують інструменти, які роблять процес приборкання більш керованим. CMake є одним із таких інструментів, або, точніше, CMake — це набір інструментів, який охоплює все, починаючи від налаштування збірки аж до виготовлення пакетів, готових до розповсюдження. Віе не тільки охоплює процес від ід початку до кінця, він також підтримує широкий спектр платформ, інструментів і мов[8].

Простіше кажучи, процес від початку до кінця відповідно до CMake виглядає приблизно так:

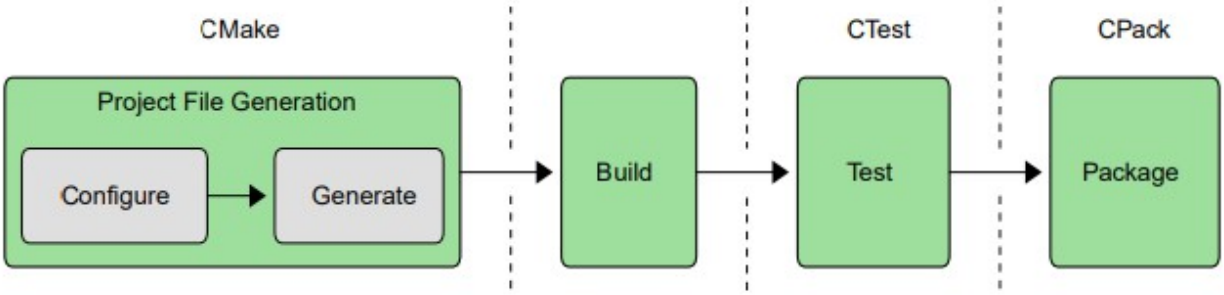

Рисунок 2.1 – Процес роботи CMake

На першому етапі береться загальний опис проекту та генеруються файли проекту для певної платформи, придатні для використання зі звичайним інструментом збірки розробника (наприклад, make, Xcode, Visual Studio, тощо). Хоча цей етап налаштування є тим, чим найбільш відомий CMake, набір інструментів CMake також включає CTest і CPack для керування подальшими етапами тестування та пакування відповідно.

Весь процес від початку до кінця можна керувати з самого CMake, а етапи тестування та пакування доступні просто як додаткові цілі в збірці. CMake може викликати навіть інструмент збірки.

Перш ніж почати працювати з CMake, розробники повинні переконатися, що CMake встановлений у їхній системі. Деякі платформи зазвичай вже мають CMake передвстановлено (наприклад, більшість дистрибутивів Linux мають CMake, доступний через їх менеджер пакетів), але ці версії часто досить старі. Якщо можливо, розробникам рекомендується працювати з останнім випуском CMake. Це особливо актуально при розробці для платформ Apple, де такі інструменти, як Xcode та його пакети SDK, швидко змінюються, а вимоги до магазину додатків змінюються з часом.

Офіційні пакунки CMake можна завантажити та розпакувати до каталогу на машині розробника, не заважаючи встановленню CMake для всієї системи. Розробникам рекомендується скористатися цим і залишатися відносно

близькими до останнього стабільного випуску CMake.

Сьогодні CMake також постачається з досить великою довідковою документацією, яка доступна з офіційного сайту CMake. Цей корисний ресурс дуже корисний для пошуку різних команд, параметрів, ключових слів тощо, і розробники, ймовірно, захочуть додати його в закладки для швидкого використання. Список розсилки користувачів CMake також є чудовим джерелом порад і рекомендованим форумом для запитань, пов’язаних із CMake, якщо документація не містить достатніх вказівок.

Без системи збірки проект — це просто набір файлів. CMake надає певний порядок у цьому, починаючи з файлу, який можна читати людиною під назвою CMakeLists.txt, який визначає, що має бути створено та як, які тести запускати та які пакети(и) створювати. Цей файл є незалежним від платформи описом всього проекту, який CMake потім перетворює у файли проекту інструмента збирання для платформи.

Як випливає з назви, це звичайний текстовий файл, який розробники редагують у своєму улюбленому текстовому редакторі або середовищі розробки. Саме він контролює все, що CMake буде робити під час налаштування та виконання збірки. Фундаментальною частиною CMake є концепція проекту, що має як вихідний каталог, так і бінарний каталог. У вихідному каталозі знаходиться файл CMakeLists.txt, а вихідні файли проекту та всі інші файли, необхідні для збірки, організовані в цьому місці.

Каталог часто перебуває під контролем версій за допомогою такого інструменту, як git, subversion тощо. Бінарний каталог — це місце, де створюється все, що створюється збіркою. Його також часто називають каталог збірки.

CMake зазвичай використовує термін бінарний каталог, але серед розробників термін каталог збірки, як правило, більш поширений.

CMake, вибраний інструмент збірки (наприклад, make, Visual Studio тощо), CTest і CPack створять різні файли в межах збірки каталога і підкаталоги

під ним. Виконувані файли, бібліотеки, тестові результати та пакунки створюються в каталозі збірки. CMake також створює спеціальний файл CMakeCache.txt в каталозі збірки для зберігання різної інформації для повторного використання під час наступних запусків. Зазвичай розробникам не доведеться займатися файлом CMakeCache.txt.

Файли проекту інструмента збирання (наприклад, файли проекту Xcode або Visual Studio, файли Makefiles тощо) також створюються в каталозі збірки і не призначені для контролю версій. Файли CMakeLists.txt є канонічним описом проекту, а згенеровані файли проекту слід вважати частиною результату збірки. Коли розробник починає роботу над проектом, він повинен вирішити, де він хоче, щоб його каталог збірки був по відношенню до його вихідного каталогу. По суті, існує два підходи: джерело і збірки поза джерелом.

Можливо, хоча і не рекомендується, щоб вихідний каталог і каталоги збірки були однаковими. Таке розташування називається збіркою з джерела. Розробники на початку своєї кар’єри часто починають використовувати цей підхід через його простоту.

Основна складність зі збірками з джерелом, однак, полягає в тому, що всі вихідні дані збірки змішуються з вихідними файлами. Ця відсутність поділу призводить до того, що каталоги переповнюються всілякими файлами та підкаталогами, що ускладнює керування джерелами проекту та створює ризик перезапису вихідних файлів збірки. Це також ускладнює роботу з системами контролю версій, оскільки є багато файлів, створених під час збірки, які або інструмент контролю джерел має знати, щоб ігнорувати, або розробник повинен вручну виключити під час комітів.

Ще одним недоліком збірок із вихідним кодом є те, що очистити всі вихідні дані збірки та почати знову з чистого дерева вихідних кодів може бути нетривіально. З цих причин розробникам не рекомендується використовувати збірки з джерелом, де це можливо, навіть для простих проектів.

Краще, щоб вихідний каталог і каталоги збірки відрізнялися, що

називається збіркою поза вихідним кодом. Це повністю відокремлює джерела та вихідні дані збірки один від одного, уникаючи таким чином проблем змішування, які виникають при збірках із вихідним кодом.

Збірки поза вихідним кодом також мають ту перевагу, що розробник може створити кілька каталогів збірок та той самий вихідний каталог, що дозволяє налаштовувати збірки з різними наборами параметрів, наприклад, версії налагодження та випуску тощо. Наприклад:

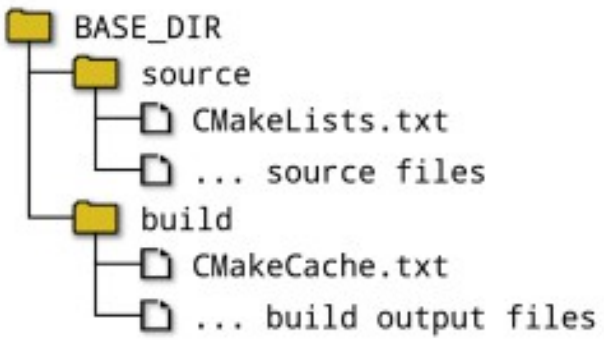

Рисунок 2.2 – Приклад збірки поза вихідним кодом

Варіант цього, який використовують деякі розробники, полягає в тому, щоб зробити каталог збірки підкаталогом вихідного каталогу. Це пропонує більшість переваг збірки поза джерелом, але все одно несе з собою деякі недоліки узгодження з джерелом.

Якщо немає вагомих причин структурувати речі таким чином, замість цього рекомендується зберігати каталог збірки повністю за межами дерева джерел.

3 РОЗРОБКА ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

Першочергово було проведено аналіз існуючего тестового спеціалізованого апаратного додатку з вбудованим RFID-датчиком, розробка якого проводилась протягом 4го курсу. Девайс був презентований під час захисту роботи бакалавра.

В результаті виконання данної роботи було створено макет пристрою-зчитувача RFID міток. Розроблена система відповідає вимогам, що були поставлені на початку курсового проекту:

- наявність компонентів, що можуть бути швидко замінені на аналогічні або швидко демонтовані та використані у іншому пристрої;
- програмна система, що виконує отримання даних методом безконтактної передачі, виконує задані алгоритми обробки та має можливість передати їх у зворотному напрямку;
- є можливість наглядної демонстрації роботи RFID протоколу у зв’язку з платформою Arduino. Фотографію розробленого макету наведено на рисунку 4.1, де 1 – ArduinoUno, 2 – MFRC-522-RFID-KIT, 3 – клавіатура, 4 – дисплей.

Програмна частина була реалізована у середовищі розробки Arduino IDE, з використаним інтерфейсу USART для зв'язку к ПК

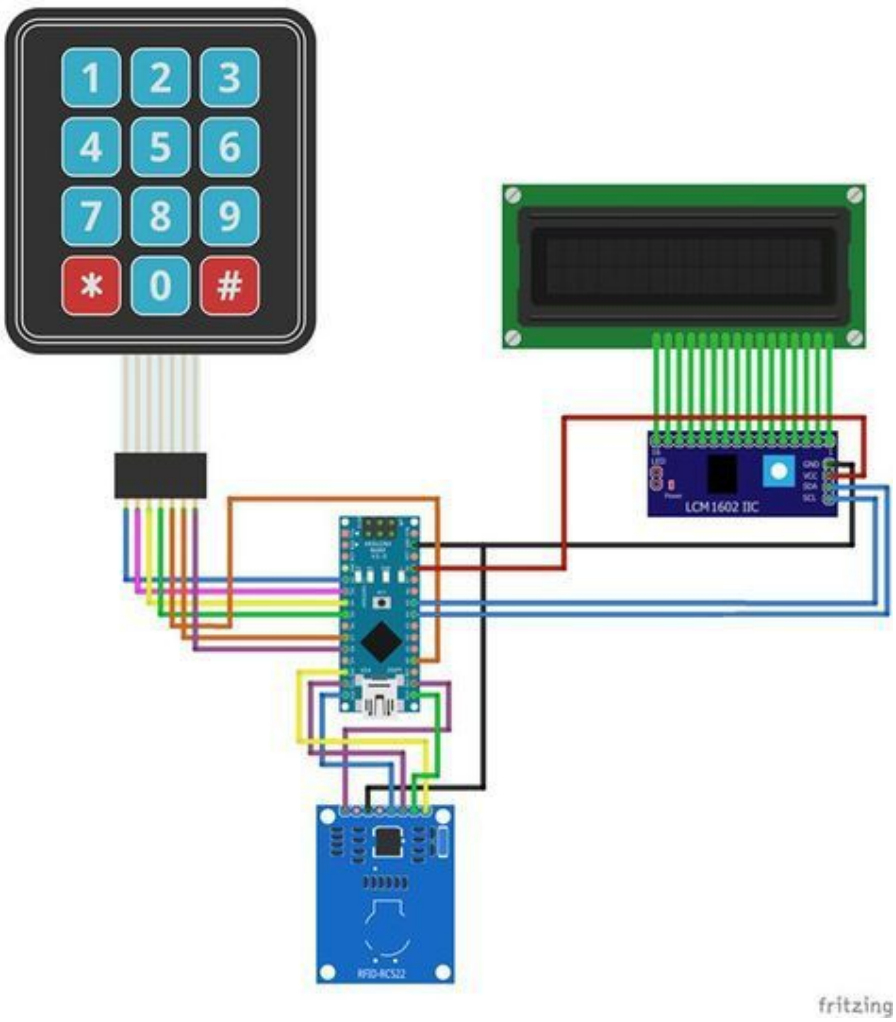

Рисунок 3.1 –Модель прилада-зчитувача

Після повторного аналізу існуючего функціоналу було виявлено наступні вимоги що до фукнціоналу додатку:

- реалізація зчитування поточного балансу карти;
- поповнення картки;
- зменшення балансу картки;
- скидання балансу картки до дефолтних установок;
- відображення балансу та операції на дісплеї терміналу;
- можливість живлення приладу від різних історників.

На основі цих вимог було створено наступний чекліст з тест-кейсами, який покриває функціональні перевірки працездатності та якості розробки апаратного додатку.

Таблиця 3.1 – Чекліст для апаратного додатку з вбудованим RFID

Modul	Action & steps	Expected results	Actual results	Status
Перевірка функціоналу блоку кнопок 1-9 клавіатури	Pre conditions: Для даного блоку тестів необхідно підключити девайс до напруги			
	Кнопка «1»: 1) натиснути на терміналі клавишу 1	На дісплеї відобразилась цифра 1	На дісплеї відобразилась цифра 1	PASS
	Кнопка «2»: 1) натиснути на терміналі клавишу 2	На дісплеї відобразилась цифра 2	На дісплеї відобразилась цифра 2	PASS
	Кнопка «3»: 1) натиснути на терміналі клавишу 3	На дісплеї відобразилась цифра 3	На дісплеї відобразилась цифра 3	PASS
	Кнопка «4»: 1) натиснути на терміналі клавишу 4	На дісплеї відобразилась цифра 4	На дісплеї відобразилась цифра 4	PASS
	Кнопка «5»: 1) натиснути на терміналі клавишу 5	На дісплеї відобразилась цифра 5	На дісплеї відобразилась цифра 5	PASS
	Кнопка «6»: 1) натиснути на терміналі клавишу 6	На дісплеї відобразилась цифра 6	На дісплеї відобразилась цифра 6	PASS
	Кнопка «7»: 1) натиснути на терміналі клавишу 7	На дісплеї відобразилась цифра 7	На дісплеї відобразилась цифра 7	PASS
	Кнопка «8»: 1) натиснути на терміналі клавишу 8	На дісплеї відобразилась цифра 8	На дісплеї відобразилась цифра 8	PASS
	Кнопка «9»: 1) натиснути на терміналі клавишу 9	На дісплеї відобразилась цифра 9	На дісплеї відобразилась цифра 9	PASS
	Кнопка «0»: 1) натиснути на терміналі клавишу 0	На дісплеї відобразилась цифра 0	На дісплеї відобразилась цифра 0	PASS

Перевірка функціоналу блоку кнопок – A,B,C,D, *, # клавіатури	Pre conditions: 1) Для даного блоку тестів необхідно підключити девайс до напруги 2)Перевірити,що для кожного тесту використовуэться нова положня RFID-карта			
	Кнопка «A»: 1)Ввести на клавіатурі число 1 2)Натиснути кнопку у A 3) Прикласти до RFID-датчика RFID-карту 4)Натиснути кнопку C 5) Прикласти RFID-карту з пункту 3	На дісплеї після шагу 3 відобразився надпис «Replenished Ok» Після шагу 5 на дісплеї віобразився надпис «Balance:1»	На дісплеї після шагу 3 відобразився надпис «Replenished Ok» Після шагу 5 на дісплеї віобразився надпис «Balance:1»	PASS
	Кнопка «B» перевірка 1: 1)Ввести на клавіатурі число 100 2)Натиснути кнопку у A 3) Прикласти до RFID-датчика RFID-карту 4)Ввести число 1 на клавіатурі 5)Натиснути кнопку B 6)Прикласти RFID-карту з пункту 3 7)Натиснути кнопку C 8)Прикласти RFID-карту з пункту 3	Після виконання пункту 8 на дісплеї відобразився надпис «Balance 99»	Після виконання пункту 8 на дісплеї відобразився надпис «Balance 99»	PASS
	Кнопка «B» перевірка 2:	Після виконання пункту 8 на	Після виконання пункту 8 на	PASS

	1)Ввести на клавіатурі число 100 2)Натиснути кнопку у A 3) Прикласти до RFID-датчика RFID-карту 4)Ввести число 101 на клавіатурі 5)Натиснути кнопку B 6)Прикласти RFID-карту з пункту 3 7)Натиснути кнопку C 8)Прикласти RFID-карту з пункту 3	дісплеї відобразився надпис «Balance 0»	дісплеї відобразився надпис «Balance 0»	
	Кнопка «C»: 1)Ввести на клавіатурі число 123 2)Натиснути кнопку у A 3) Прикласти до RFID-датчика RFID-карту 4)Натиснути кнопку C 5) Прикласти RFID-карту з пункту 3	Після шагу 5 на дісплеї віобразився надпис «Balance:123»	Після шагу 5 на дісплеї віобразився надпис «Balance:123»	PASS
	Кнопка «D»: 1.1)Натиснути кнопку D 1.2)Прикласти порожню карту 2.1)Натиснути кнопку D 2.2)Прикласти карту з балансом 100 2.3)Перевірити баланс за допомогою кнопки C 3.1)Ввести на клавіатурі число 1	По результату кожної перевірки баланс карт не змінився.	По результату кожної перевірки баланс карт не змінився.	PASS

	3.2)Нажати кнопку D 3.3)Прикласти карту,баланс на котрій становить 3 3.4)Перевірити баланс карти за допомогою кнопки C.			
	Кнопка «*»: 1.1)Натиснути кнопку * 1.2)Прикласти порожню карту 2.1)Натиснути кнопку * 2.2)Прикласти карту з балансом 100 2.3)Перевірити баланс за допомогою кнопки C 3.1)Ввести на клавіатурі число 1 3.2)Нажати кнопку * 3.3)Прикласти карту,баланс на котрій становить 3 3.4)Перевірити баланс карти за допомогою кнопки C	По результату кожної перевірки баланс карт не змінився.	По результату кожної перевірки баланс карт не змінився.	PASS
	Кнопка «#»: 1.1)Натиснути кнопку # 1.2)Прикласти порожню карту 1.3) Перевірити баланс за допомогою кнопки C. 2.1)Натиснути кнопку # 2.2)Прикласти карту з балансом 100	По результату кожної перевірки баланс карти становить 500	По результату кожної перевірки баланс карти становить 500	PASS

	2.3)Перевірити баланс за допомогою кнопки C 3.1)Ввести на клавіатурі число 1 3.2)Нажати кнопку # 3.3)Прикласти карту,баланс на котрій становить 3 3.4)Перевірити баланс карти за допомогою кнопки C			

Таким чином було покрито тестами основний функіонал апаратного додатку. Даний чек-ліст із талиці 3.1 можна використовутати як и для мануального тестування так і як основу для створення автоматичних тестів.

Слід звернути увагу, що кнопки A та B мають у собі арефметичні функції додавання та віднимання, що еквівалентно поповенню та , відповідно, зняття условних кошт з RFID-картки.

Для тестування данного функціоналу слід використати метод гранічних значень. На його основі були сформовані наступни набори данних

Таблиця 3.2 – таблиця гранічних значеннь

Стартовий баланс RFID-картки	Виконувана функція	Гранічне значення перевірки	Очікуваний результат
0	A	1	1
1	A	1000	1001
1	B	1	0
1	B	2	0

Тестових наборів гранічних значень у таблиці 3.2 являється достатнім для логічного та оптимального, відносно часу, затраченого на тестування, покриття додатку зі збереженням якості тестуввання.

Також можна відзначити, що даний набор значень релевантний для

перевірення працездатності дісплею та кнопки C, так як він являється еквівалентним классом по відношенню до перевірки відображення .

Даний девайс має 2 джерела живлення, тому було проведення відповідне тестування.

Таблиця 3.3 – Тестування джерел живлення

Тип живлення	Очікуванний результат	Фактичний результат	Статус перевірки
Зовнішнк від ПК	Подача живлення виконується	Подача живлення виконується	PASS
Від блоку живлення	Подача живлення виконується	Подача живлення виконується	PASS

Під час виконання перевірок був знайден баг мінорного пріорітету. Для документації пробеми був створений баг-репорт, наведений у таблиці 3.4

Таблиця 3.4 – Баг-репорт.

Заголовок	Блимання надписі після використання терміналу
Проект	Апаратний додаток з вбудованим RFID-датчиком
Важливість	Minor
Приорітет	Medium
Статус	Новий
Шаги відтворення	1)Підключити термінал до живлення 2)Ввести на клавіатурі число 1 3)Натиснути кнопку у A 4)Прикласти до RFID-датчика RFID-карту 5)Натиснути кнопку C 6) Прикласти RFID-карту з пункту 4
Очікуваний результат	На дісплеї після шагу 3 відобразився надпис «Replenished Ok» Після шагу 5 на дісплеї віобразився надпис «Balance:1»
Фактичний результат	На дісплеї після шагу 3 відобразився надпис «Replenished Ok» Після шагу 5 на дісплеї віобразився надпис «Balance:1» Після цього відобразивлся надпис «Hello Player!» Актуально для дій A,B,C,#

4 РОЗРОБКА МОДЕЛІ СИСТЕМИ ВЕДЕННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

Наступним кроком була виконана розробка програмного додатку для зручного автоматичного ведення тестових артефактів та логування роботи апаратного додатку.

За основу додатка були обрані такі технологіх, як QML, CMake.

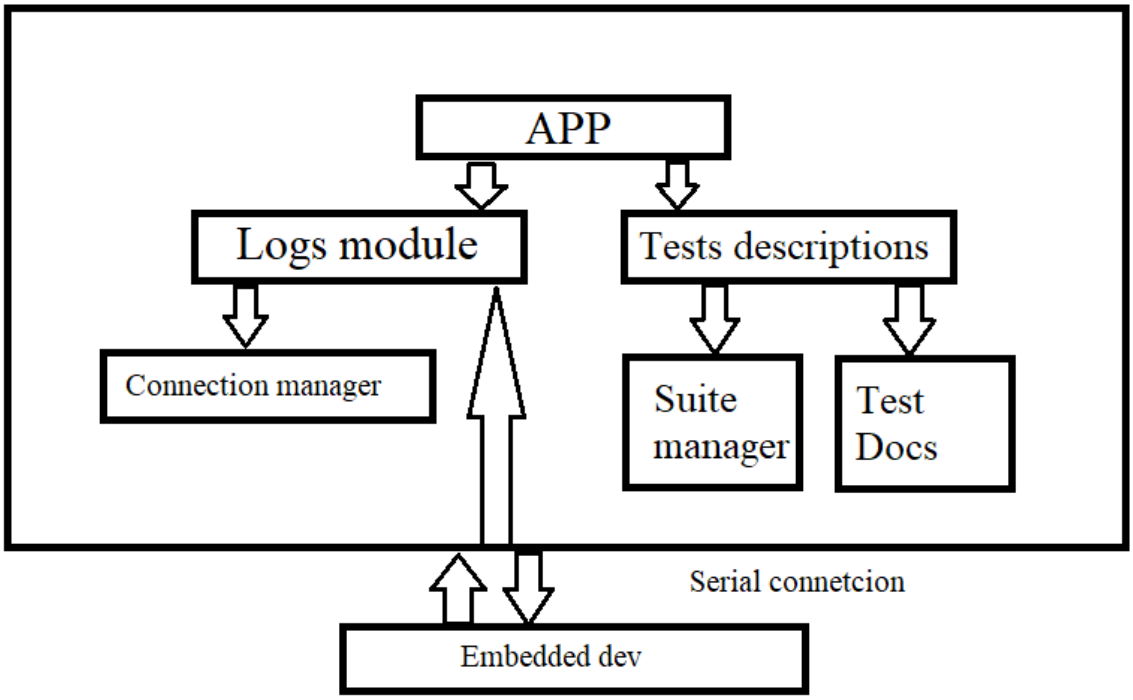

Рисунок 4.1 – Модель прототипу системи

Першим, що користувач бачить після запуску Automation Suite додатку це стартове вікно з опцією вибору бажаної маніпуляції :

- Logger: сторінка логування роботи апаратного девайсу;
- Testing suites:сторінка маніпуляції з тестовими артефактами.

Також перейти до вказаних вище сторінок додатку можно завдяки реалізації бокового свайпу через стрілки на клавіатурі ПК.

Для розробки графічної частини проекту було використану декларативну мову програмування QML з реалізацією бекенду на C++14/17. Реалізацію

основного вікна додатку виконано з використанням компоненту ApplicationWindow з встановленими розмірами вікна та вбудованим SwipeView для переключення екранів додатку. Для відстежування положення активної страниці додатку використовується компонент PageIndicator, де інтекс позначає поточну активну крапку. Фрагмент реалізації головного вікна програми наведено у Лістингу 4.1.

Лістинг 4.1 – Фрагмент реалізації головного вікна програми

```
import QtQuick 2.15
import QtQuick.Window 2.15
import QtQuick.Controls 2.15
import QtQuick.Controls.Material 2.15

ApplicationWindow {
    width: 640
    height: 480
    visible: true
    title: qsTr("Automation Suite")
    id:rootWindow

    SwipeView {
        id: mainView
        currentIndex: 0
        anchors.fill: parent

        MainPageSelector
        {

        }
        LoggerComponentView
        {
            id:loggerView
        }

        PageTestingDocumentsView
        {
            id: testingDocsView
        }

    }
}
```

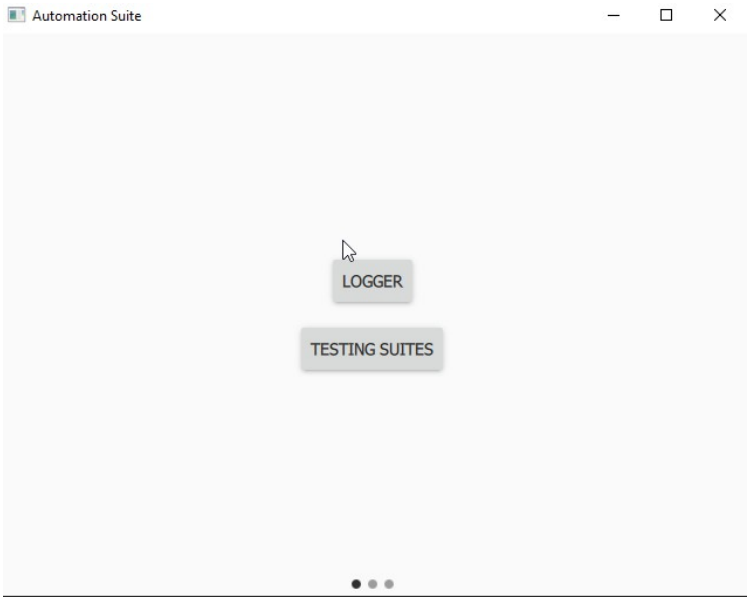

Рисунок 4.2 – Стартове вікно Automation Suite

Якщо підключити до ПК апаратний девайс, на сторіці Logger можно побачити відображення запису логування виконуваних на девайсі функцій у режимі реального часу. Це потрібно для низькорівневого дослідження та аналізу проведених досліджень, а також збору інформації для подальшої автоматизації тестування, а й, відповідно, делегування на це частини мануальних задач по підтримці якості продукту.

Сторінка Testing suites містить у собі богато функціоналу управління тестовими артефактами. Початкова сторінка описується як ListView. У лістингу нижче ми описуємо базові характеристики вікна, його розмір та його модель даних.

Лістинг 4.2 – Фрагмент реалізації списку тестів

```
ListView
{
    Layout.alignment: Qt.AlignTop;
    Layout.fillHeight: true;
    Layout.fillWidth: true;

    model:testingDocumentsModel;
```

```
    delegate:
        RowLayout{
            implicitWidth: parent.width
            implicitHeight: 48;
            Text {
                id: delegateText
                text: `TestSuite: ${documentTitle}`
                font.pointSize: 22;
                elide: Text.ElideRight;

                Layout.fillWidth: true;
            }
            Item
            {
                Layout.fillWidth: true;
                Layout.fillHeight: true;
            }
        }
}
```

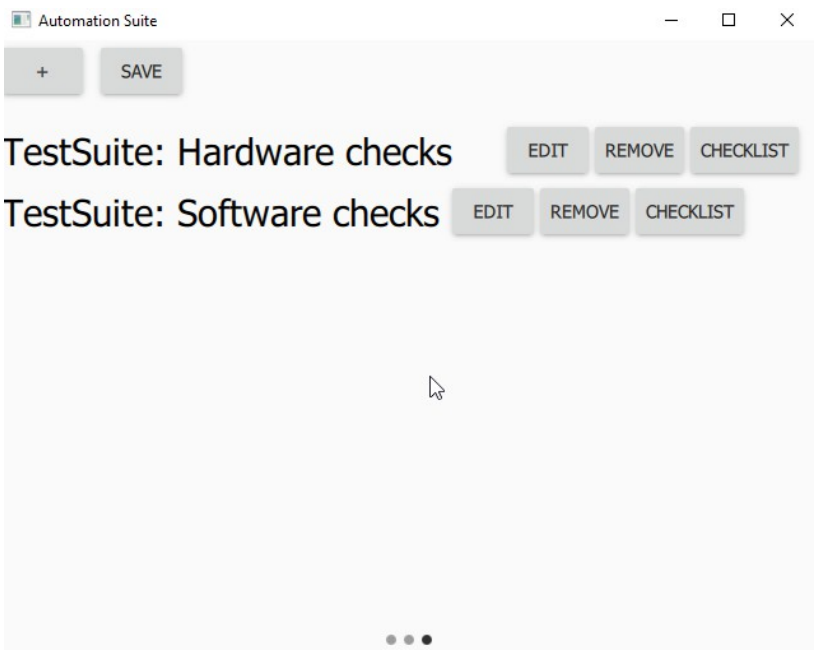

Рисунок 4.3 – Початкове вікно сторінки Testing suites

У верхньому горизонтальному барі находиться дві кнопки швидкого доступу:

- хрестик для додавання нового чеклісту;
- Save для збереження тестових артефактів у форматі json.

По кліку на хрестик підіймається модальне вікно вводу заголовку нового документа, що містить у собі кнопки підтвердження та відміни обраної операції. Базовий текст вікна, відступи та настройки шрифту налаштовуються через властивості макету мовою розмітки QML.

Лістинг 4.3 – Фрагмент реалізації модального вікна

```
ColumnLayout{
    spacing: 10
    anchors.fill: parent;

    TextField
    {
        id: inputText
        text: initialText;
        Layout.fillWidth: true;
        placeholderText: "Testing doc name"
        font.weight: Font.DemiBold
    }


    RowLayout
    {
        spacing:28
        Layout.fillWidth: true;
        Layout.alignment: Qt.AlignHCenter;
        Button
        {
            id: acceptButton;
            text: "OK"
```

```
            onClicked:
            {
                if(inputText.text!=="")
                    addNewItemRequested(inputText.text);
            }
        }
        Button
        {
            id: cancelButton;
            text: "Cancel"
            onClicked:
            {
                addNewDocumentSetPopup.close();
            }
        }
    }
}
```

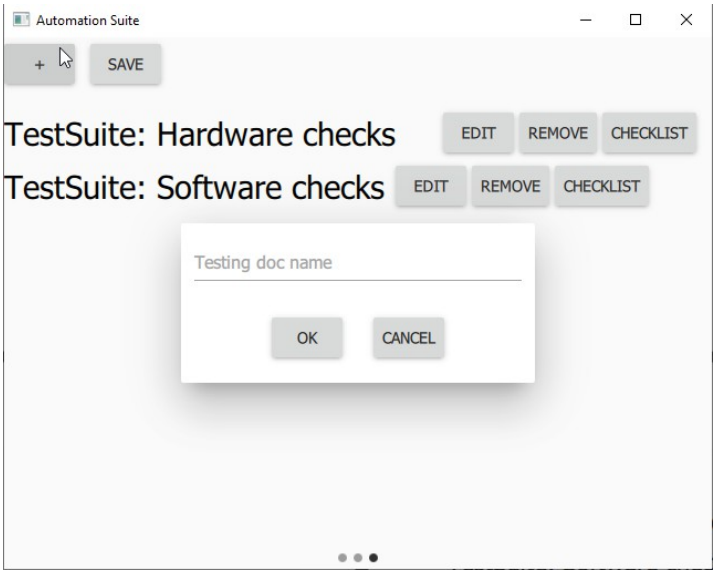

Рисунок 4.4 – Модальне вікно додавання чек-ліста

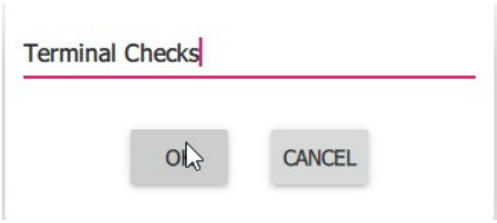

Рисунок 4.5 – Введеня та підтвердження додавання нового чек-ліста

Після кліку на кнопку підтведження, що відображено на рисунку 4.5, виконується успішне додання нові документації у систему

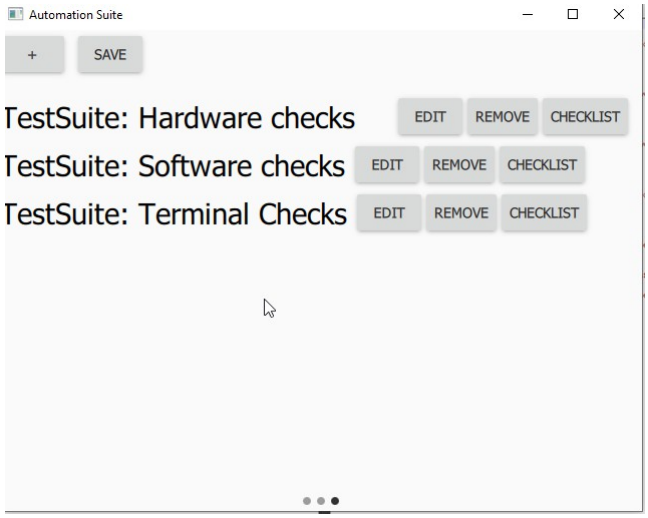

Рисунок 4.6 – Коректне відображення нового чек-ліста

У лістингу нижче описано процес серіалізації C++ моделі у зручний для аналізу та обробки JSON-формат. Використовуючи структуру даних QML моделі ми будуємо дерево у форматі ключ-данні, де ключ.

Лістинг 4.4 – Фрагмент реалізації серіалізації даних

```
    void serializeTo(const QString& outputPath,
gsl::not_null<TestingDocumentsModel*> pModel)
    {
        QJsonArray rootArray;

        pModel->forEachDocument(
            [&rootArray, this](const
```

```
std::shared_ptr<ChecklistModel>& checkListModel) {

rootArray.push_back(SerializeCheckListModel(checkListModel));
            });

        auto correctedPath{QUrl(outputPath).toLocalFile()};
        qDebug() << QString::fromStdString(
            fmt::format("Corrected path is: {}",
correctedPath.toStdString()));
        qDebug() << QString::fromStdString(
            fmt::format("JSON is: ${}",
QJsonDocument(rootArray).toJson(QJsonDocument::Indented)));
        QFile file(correctedPath);
        if (!file.open(QIODevice::ReadWrite | QIODevice::Text))
            return;

        QTextStream out(&file);
        out <<
QJsonDocument(rootArray).toJson(QJsonDocument::Indented);
    }
```

На рисунку 4.7 відображене успушне виконання функції експорту тестової документаці у форматі json. Даний формат був обраний через свою гнучкість відкриття на різних платформах та відсутністю необхідності встановлювати додатковий програмний софт, так як можно використати базовий Блокнок або Notepad++.

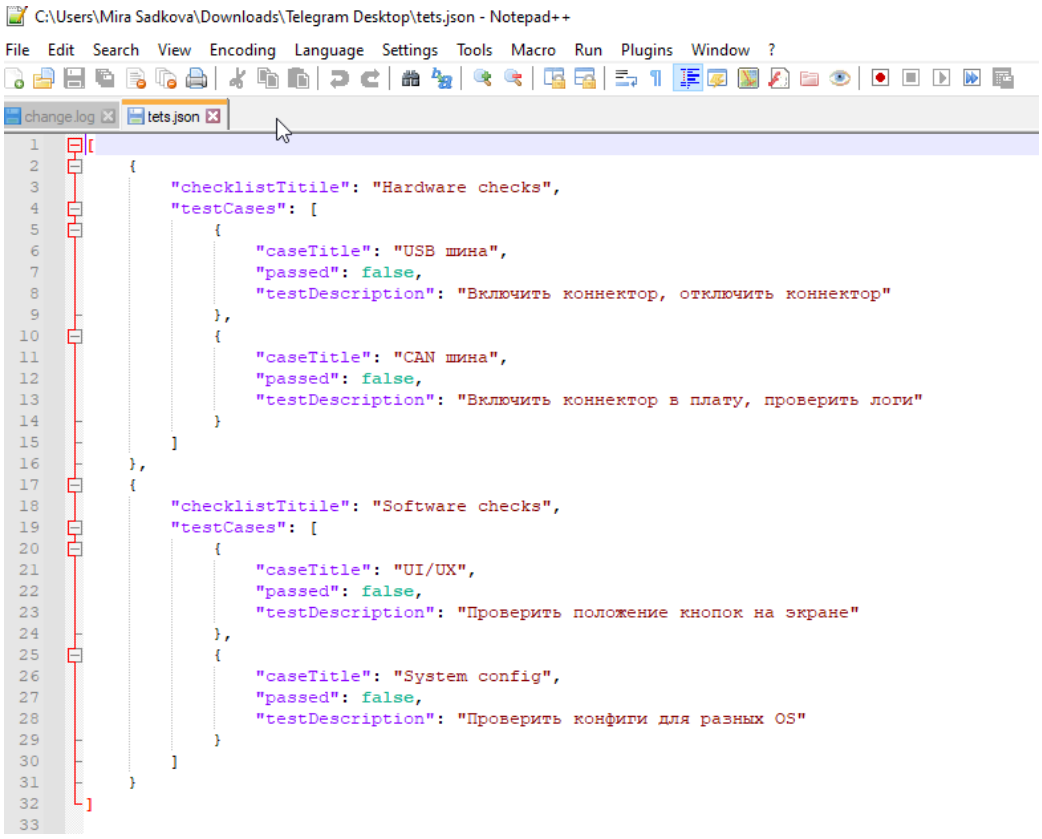

Рисунок 4.7 – Експортований файл створеної документації.

Для обробки даних моделі, інтерпритації типів та подальшої їх серіалізації створюється основний процес програми. Він також виступає обробником всіх подій та відповідає за очищення пам’яті під час завершення роботи.

Лістинг 4.5 – Фрагмент реалізації основного процесу Qt

```
App::App() : m_pAppImpl{std::make_unique<App::AppImpl>()}
{
}

App::~App() = default;

App& App::Instance()
{
    static App appInstance{};
    return appInstance;
}
```

```
void App::registerTypes()
{
    m_pAppImpl->registerTypes();
}

void App::addContextProperties(QQmlApplicationEngine& _qmlEngine)
{
    m_pAppImpl->addContextProperties(_qmlEngine);
}

void App::initAppHandlers()
{
    m_pAppImpl->initAppHandlers();
}

void App::cleanupApplication()
{
    m_pAppImpl->cleanupApplication();
}
```

Сторінка документації логічно розподілина на рядки, у яких можно побачити актуальну назву документа, а також була реалізовани можливості редагувати його заголовок через кнопку Edit, видалити документ через кнопку Delete та переглянути його зміст. Через кнопку Checklist виконується виклик гнучного модального окна із відображенням вмісту документу.

Реалізацію функціоналу наведена на рисунку 4.8.

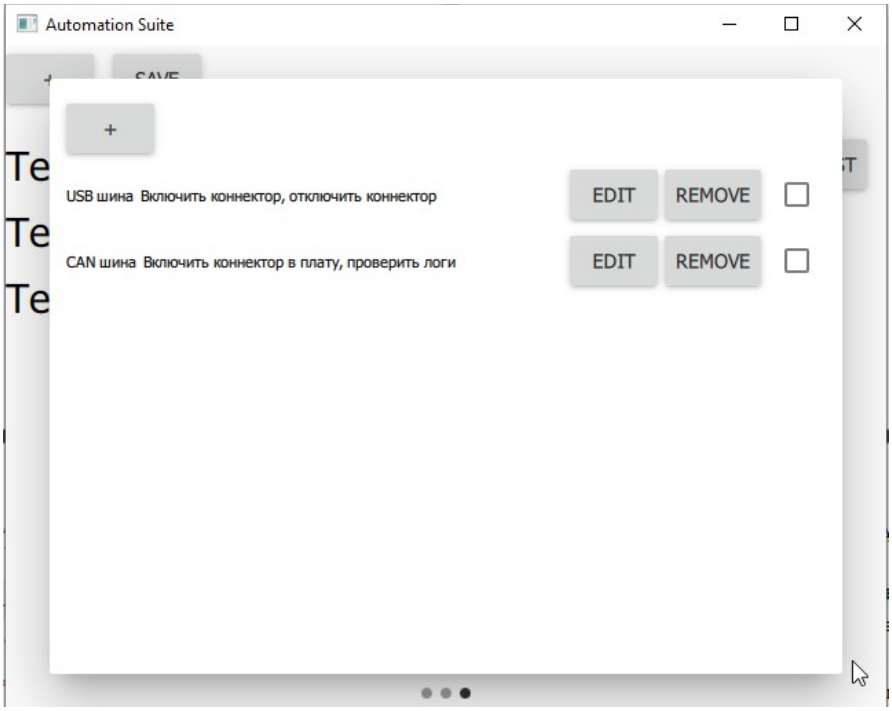

Рисунок 4.8 – Модальне вікно перегляду створених артефактів.

Так як функціонал того чи іного продукту може змінюватись від версії до версії, тестову документацію також потрібно актуалізовувати, для того щоб вона була відповідной поточній версії продукту. Актуальна документація являється еталоном та позволяє швидко вирішити питання відносно того, яка поведінка продукту є істиною, та скорочую час, необхідний на тестування.

Через це був реалізовани функції модифікаці, серед яких:

- Edit – редагування тест-кейсу;
- Remove – видалення засторілої перевірки;
- Плюс – додавання нового тестк-кейсу.

По кліку на іконку додавання підіймається вікно вводу нової перевірки, яке включає поле для ввода заголовку та тіла самої перевірки. Сірим кольором можно побачити підсказки-назви, яки зникають після початку ввода тестку.

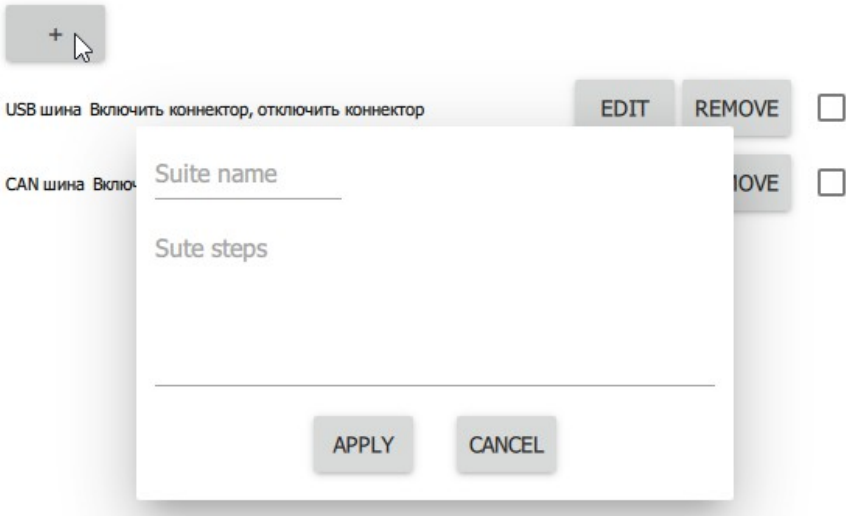

Рисунок 4.9 – Модальне вікно додавання нового тест-кейсу.

Після додавання тесту модель данних має бути оновлена. Для цього ми серіалізуемо дані із нашого вікна та моделі QML у формат QJsonObject для їх подальшої обробки.

Лістинг 4.6 – Фрагмент серіалізації моделі результатів тестів

```
    QJsonObject SerializeCheckListModel(const
std::shared_ptr<ChecklistModel> checkListModel)
    {
        QJsonObject objectModel;
        objectModel["checklistTitile"] = checkListModel-
>getTitle();

        QJsonArray testCases;

        checkListModel->forEachTestCase(
            [&testCases](const std::shared_ptr<ChecklistItem>&
checkListItem) {
                QJsonObject caseItem;
                caseItem["caseTitle"] = checkListItem->getTitle();
```

```
                caseItem["testDescription"] = checkListItem-
>getSteps();
                caseItem["passed"] = checkListItem->isPassed();

                testCases.push_back(caseItem);
            });
        objectModel["testCases"] = testCases;
        return objectModel;
    }
};
```

Успішно доданий тест-кейс можно побачити на рисунку 4.10.

Для зручності відстеження та контролю процесу тестування також був реалізован чекбокс напроти кожного тест-кейсу. Пустий чекбокс відповідає непройденій перевірці, відміченний – перевірці, яка має статус PASS.

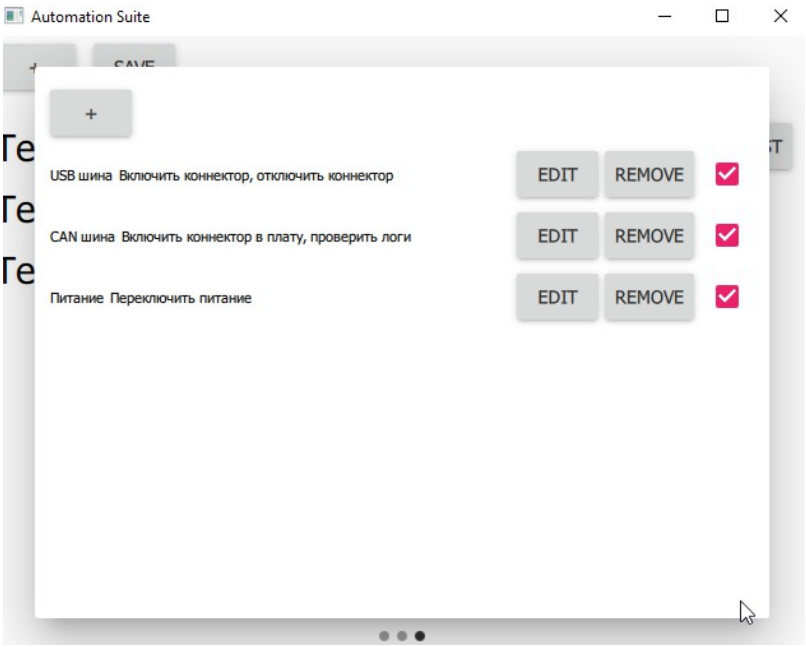

Рисунок 4.10 – Реалізація чекбоксів

Для реалізації функціоналу автоматичного детектування підключеного пристрою з Serial-інтерфейсом було використано WMI-функціонал ОС Windows що дозволяє виконати системний запит на наявність plug and play

пристроїв у системі. Основним компонентом для опрацювання отримування даних з послідовного порту є QSerialPort що дозволяє виконувати асинхронне неблокуюче приймання та передачу даних з використанням моделі сигналів-слотів. Основними параметрами, що потрібно сконфігурувати для налаштування порту є встановлення швидкості обміну, кількості стоп-бітів, підтримки апаратного контролю чіткості. Також необхідно встановити назву порту, що буде використано. Для реалізації періодичного сканування є необхідним використати компонент QTimer що дозволяє у слоті по виконаню сигналу виконати необхідну послідовність дій.

Перевірку наявності порту у системі реалізовано у методі isPortAvailableInSystem який періодично викликається під час обробки подій від таймеру. Реалізацію методу наведено у Лістингу 4.7.

Лістинг 4.7 – Реалізація періодичного опитування послідовних портів у системі

```
bool SerialPortClient::isPortAvailableInSystem(const QString&
connectionPort) const
{

    try
    {
        for (const auto& serialPort :
Wmi::retrieveAllWmi<Win32_PnPEntity>())
        {
            constexpr int kNotFound = -1;
            if
(QString::fromStdString(serialPort.Name).indexOf(connectionPort) !
= kNotFound)
            {
                qCDebug(serialPortsSearch)
                    << "FOUND PORT:" <<
QString::fromStdString(serialPort.Name)
                    << "Matcher:" << connectionPort;
                return true;
            }
        }
        for (const auto& serialPort :
Wmi::retrieveAllWmi<Wmi::Win32_SerialPort>())
        {
```

```
            qCDebug(serialPortsSearch) << "serialPort:" <<
QString::fromStdString(serialPort.Name);
        }
    }
    catch (const Wmi::WmiException& _ex)
    {
    }
    return false;
}
```

Архітектурно для обробки подій від послідовного порту використовується верхній архітектурний шар з MessageHandler що дозволяє виконати передачу даних до QML-графіки без використання надлишкових копіювань, що реалізовані як Copy-On-Write механізм у фреймворку Qt.

Інтеграція з WMI компонентами системи виконана за рахунок використання обгортки, що надає часткові спеціалізації шаблону для типових структур даних, таких як WMI_VIdeoAdapter, WMI_Drive, WMI_MonitorInfo. Для додавання підтримки своєї структури даних необхідно реалізувати клас, що буде мати метод отримання Wmi_ClassName та видавати користувачу ім’я необхідного компоненту. Приклад реалізації такої структури даних для plug-and-play пристрою наведено у Лістингу 4.8.

Лістинг 4.8 – Реалізація підтримки запитів WMI для користувацької структури даних

```
struct Win32_PnPEntity
{
    void setProperties(const Wmi::WmiResult& result, std::size_t
index)
    {
        result.extract(index, "Name", (*this).Name);
    }

    static std::string getWmiClassName()
    {
        return "Win32_PnPEntity";
    }

    std::string Name;
};
```

5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Були проаналізовані існуючі теоретичні відомості, такі як тестові артефакти, типи та види тестування, техніки тест дизайну.

В результаті виконання кваліфікаційного проекту були проаналізовані вимоги та встановлені головні задачі для розробки моделі тестування, а саме чек-лісти та баг репорти із використанням технік тест дизайну, такі як класи еквівалентності та гранічних значень, через те що дані техніки являються найросповсюдженними через оптимальний підхід до роботи з числовими діапазонами шляхом виключення еквивалентних та дуюлюючих перевірок.

Була розроблена модель автоматизації функціонального тестування на базі мови програмування C++ та технологіях QML та Qt. Дана модель являє собою програмний додаток та дозволяє вирішити логування роботи апаратного додатку та проблему ведення тестової документації .

Логування роботи апаратного додатку з вбудованим RFID-датчиком дає можливість точного сберігати логи,які використовуються для подальшого аналізу оптимізації та автоматизації продукту, а також додавати до баг-репортів як низкорівневий доказ наявності несправности, що дозволить розробникам скоротити час на пошут проблеми та й, відповідно, її вирішення.

Також разроблена модель дає можливість вирішити проблему ведення тестової документації для зручного збереження артефактів та обновлення їх до актуальної версії продукту(додавати нове, видаляти старі перевірки, то що). Це дозволяє більш структуровано зберігати інформацію. Це в свою чергу дає можливість проаналізувати проведені тести, віявити недоліки у роботі системи , що тестується, виявити вузькі місця та передбачити критичні помилки.

ВИСНОВКИ

В ході дослідження виконано аналіз теорії тестування та забезпечення якості апаратних та програмних додатків, розглянуті на прикладах тестові артефакти, а саме чек-ліст, тест-кейс та баг-репорт.

Дослідження показало перспективність розвитку автоматизації тестування. Автоматизація слугує для підвищення ефективності процесу тестування за рахунок вивільнення фахівців зайнятих в регресійному тестуванні на різних рівнях розробки та підтримки продукту. Проведені автоматичним способом тести вимагають менших людських ресурсів, відповідно, коштують дешевше. Крім того, скорочується негативний вплив на якість програмного продукту людського фактора та час на проведення тестування.

Також системи контролю версій допомогають структуровона зберігати усю інформацію відносно одного або декількох проєктів, що дозволяє швидко находити та маніполювати інформацією. Також структуровані данні дозволяют знатно скоротити час інтеграції нової люди у проект.

В результаті кваліфікаційної роботи був реалізован прототип нової системи менеджменту контролю якості додатку. До її переваг можно віднести дешевізну у використанні, так як нині всі існуючі аналогічні продукти доступні лише по платній підписці. Даний продукт чудовой підійде фахівцям-початківцям та студентам. Також даний продукт має змогу до комунікації з апаратними додатками, що дозвоялє збегірати все в єдиному місці та не тратити фінаносві та часові ресурси на підтримку та використання декількох утиліт.

Також окремою перевагою можно викремити те, що даний продукт являється кросплатформеною та може повноцінно функціонувати на таких ОС, як Windows та Linux.

У ході роботи були виявлені наступні критерії для розвитку розробленого проекту:

- гдодавання окремого функціоналу баг-трекінгу;
- інтеграція з іншими системами ведення менежменту проектів;
- створення мобільної версії додатку;
- оптимізація існуючої кодової бази;
- додавання різних форматів експорту тестових артефактів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Тестування програмного забезечення [Електронний ресурс] // arobenko. – 2018. – Режим доступу до ресурсу: http://www.protesting.ru/

2. QA, QC і тестування [Електронний ресурс]– Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/qa-qc-i-testuvannya/

3. Автоматизація тестування [Електронний ресурс]– Режим доступу до ресурсу: http://www.nicotech.ru/quality-assurance/test-automation/

4. Автоматическое тестирование ASP.NET [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/147902/

5. Пример создания Unit-теста [Електронний ресурс] - Режим доступу до ресурсу: https://www.bestprog.net/ru/2018/08/27/example-of-creating-a-unit-test-in-ms-visual-studio-2010-c_ru/

6. luncliff. Exploring MSVC Coroutine¶ [Електронний ресурс] / luncliff. – 2019. – Режим доступу до ресурсу: https://luncliff.github.io/coroutine/articles/exploring-msvc-coroutine/.

7. Perfect Software and Other Illusions About Testing / Gerald M. Weinberg, 2010 - 142c.

8. Craig Scott Professional CMake: A Practical Guide. 2018. 429 с.

9. Bug Life Cycle [Електронний ресурс] - Режим доступу до https://coderlessons.com/tutorials/bolshie-dannye-i-analitika/professiia-biznes-analitik/bug-life-cycle-2

10. Чому тестування необхідне? [Електронний ресурс] - Режим доступу до https://qalight.ua/baza-znaniy/chomu-testuvannya-neobhidne/

11. Введення в тестування програмного забезпечення [Електронний ресурс] - Режим доступу до https://qalearning.com.ua/theory/lectures/material/testing-intro/