КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки

на тему:

ПРОГРАМНА СИСТЕМА УПРАВЛІННЯ ЯКІСТЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Виконала студентка 2-го курсу магістратури
Сукач Юлія Вікторівна
_____
(підпис)

Науковий керівник:
доцент
Ткаченко Олексій Миколайович
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«___»_____2021р.,
протокол №
Завідувач кафедри
М.С. Нікітченко _____
(підпис)

КИЇВ - 2021

РЕФЕРАТ

Обсяг роботи 92 сторінки, 67 ілюстрацій, 1 таблиця, 10 джерел посилань.

ДЕФЕКТИ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ, ЖИТТЄВИЙ ЦИКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, КОНТРОЛЬ ЯКОСТІ, ПРОГРАМНА СИСТЕМА, РОЗРОБКА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, , СПОСОБИ ПОКРАЩЕННЯ ЯКОСТІ, СТАНДАРТ ISO 9001, ТЕСТОВІ ВИПАДКИ, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, UML ДІАГРАМИ.

Об’єктом роботи є процес створення тестових випадків, дефектів та відслідковування статистики за допомогою програмної системи управління якістю програмного забезпечення. Предметом роботи є програмний засіб для управління якістю програмного забезпечення.

Метою роботи є створення програмного засобу для управління якістю програмного забезпечення.

Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Sublime Text, мова програмування PHP.

Результати роботи: виконано огляд та аналіз сучасного ринку популярних програмних систем управління якістю програмного забезпечення, проаналізовано переваги та недоліки існуючих систем, створено програмний продукт для управління якістю програмного забезпечення, який дозволяє документувати знайдені дефекти, проектувати тестові випадки та дозволяє наочно демонструвати статистику у розрізі тестових випадків та дефектів.

Програмний продукт для управління якістю програмного забезпечення може використовуватись тестувальниками, менеджерами проектів та розробниками на реальних проектах під час розробки будь-якого програмного забезпечення.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об'єкта розробки. Ідея впровадження систем контролю якості широко розповсюдилась по всьому світу. у статті В.Г.Версана «Криза в стандартизації систем менеджменту. Причини. Шляхи виходу» (2009) за оцінками, зробленим фахівцями ISO / TC 176, зазначалося, що з мільйона підприємств в світі, сертифікат відповідності системи якості стандарту ISO 9001 мають приблизно 50-60 з них не отримали очікуваних результатів з точки зору досягнення цілей в області якості продукції, тобто по суті впровадили систему формально. В результаті протягом восьми років після впровадження стандартів ISO серії 9000: 2000 в світову економіку було вкладено 86,4 млрд доларів.

Розвиток систем якості продовжується і сьогодні у відповідності до нового стандарту ISO 9001. На сьогоднішній день на ринку програмного забезпечення контролю якості представлено досить багато різних програмних засобів: вільних та платних, складних та простих. Кожен з них має певні недоліки, які дають можливість продовжувати створення покращених програмних систем.

Актуальність роботи та підстави для її виконання. Одним з найважливіших чинників зростання ефективності розробки є підвищення якості додатків, що пропонуються користувачам, які безпосередньо визначають її конкурентоспроможність. В даний час тільки конкурентоспроможні проекти можуть успішно подолати виникаючі труднощі і проблеми. Забезпечення якості неможливе без контролю якості.

Контроль якості - це одна з основних функцій в процесі управління якістю. Значення контролю полягає в тому, що він дозволяє вчасно виявити помилки, щоб потім оперативно виправляти їх з мінімальними втратами. Під контролем якості розуміється перевірка відповідності характеристик продукції або процесу встановленим технічним вимогам, зафіксованим в документах, стандартах та інших документах.

Організація контролю якості - це система технічних і адміністративних заходів, спрямованих на забезпечення розробки програмного забезпечення, що відповідає вимогам документації. Інформація, отримана в результаті контролю, повинна служити основою для розробки різних коригувальні та запобіжні дій з метою поліпшення функцій та процесів, а, отже, і підвищення якості програмного забезпечення, що розробляється. [1]

Мета й завдання роботи. Метою кваліфікаційної роботи є створення програмної системи управління якістю програмного забезпечення. Для досягнення цієї мети поставлено такі завдання.

- Дослідити існуючі програмні рішення для управління якістю.
- Визначити недоліки існуючих програм для їх врахування у власній розробці.
- Спроектувати архітектуру розроблюваної системи, розробити UML та DFD діаграми.
- Розробити інтерфейс та дизайн програмного продукту «Система управління якістю програмного забезпечення».

Об’єкт, методи й засоби розроблення. Об’єктом даної кваліфікаційної роботи є процеси управління якістю програмного забезпечення. Предметом дослідження є автоматизовані системи їх підтримки.

Під час розробки даної програмної системи використовувалась еволюційна модель, заснована на таких принципах. Розробляється початкова версія продукту, яка передається кінцевим користувачам для оцінки, після чого продукт доробляється, враховуючи думку замовника. Аналогічно розробляються, передаються й оцінюються проміжні версії програмного продукту, поки не з’явиться повністю готовий продукт, який відповідає всім вимогам замовника.

Під час дослідження використовувались такі методи як системний підхід, що дозволяє розкрити різноманіття проявів досліджуваного об'єкта, визначити місце

предмета дослідження в розроблювальній галузі науки; проектний метод, що визначає цілісність дослідження, стадії і порядок його розроблення; моделювання, як метод дослідження структури, основних властивостей, законів розвитку і взаємодії з навколишнім світом об'єкта моделювання.

В якості інструменту створення програмного засобу було обрано Sublime Text – інтегроване середовище розробки, мовою програмування PHP, яке є вільно поширюваним, з відкритим вихідним кодом.

PHP володіє величезною кількістю бібліотек необхідних для написання веб застосунку. Мова підтримується більшістю хостинг провайдерів, а також є популярною через свою простоту, швидкість, широку функціональність та підтримку багатьма браузерами.

Можливі сфери застосування. Програмний продукт «Система управління якістю програмного забезпечення» може застосовуватись на реальних проектах під час розробки програмного забезпечення.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПРОБЛЕМИ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Особливості програмного забезпечення як продукту: подібності та відмінності від інших типів продуктів

В кінці XX століття сформувалося нове економічне поняття - програмний продукт. Цей продукт є результатом нового виду сучасного промислового виробництва. Програмний продукт включає в себе програму, її текст, представлений на носії і документацію, що її супроводжує. Програмний продукт реєструється в фондах алгоритмів і програм, в функції яких входить розмноження копій програм й документації для користувачів, а також охорона інтелектуальної власності.

Розробка програмного забезпечення для інформаційних систем сьогодні все більше наближається до промислового виробництва своїми технологіями, стандартами, системами управління та іншими ознаками індустріальної галузі. На цьому шляху доводиться долати різноманітні труднощі, властиві як усім галузям, так і специфічні, пов'язані тільки з розробкою програмного забезпечення. Це обумовлено тим, що процес розробки складного програмного забезпечення:

- є відносно новою сферою знань
- носить еволюційний і експериментальний характер
- схильний до впливу кваліфікації та індивідуальних особливостей виконавців
- володіє уявною легкістю внесення змін
- вимагає ґрунтовного вивчення особливостей предметної області і, як правило, результат є унікальним.

Ці властивості є причинами:

- незавершеності багатьох програмних проектів
- значного перевищення бюджету і затримки термінів
- поганої керованості проектів

- відсутності гарантованої якості програмного забезпечення

Світова практика показує, що запорукою успішного подолання цих труднощів є використання світових технологій і стандартів. Адже будь-який стандарт, в першу чергу, це результат узагальнення багаторічного досвіду десятків успішних у своїй області компаній.

Область стандартизації розробки програмного забезпечення постійно розвивається: поряд із стандартами ISO 9000, ISO 12207, з'являються нові, які враховують сучасний досвід.

Однак, не дивлячись на те, що програмне забезпечення це інтелектуальний товар, але він певною мірою відрізняється від інших інтелектуальних продуктів, таких як літературні тексти, музичні записи, відеофільми та інші. Однією з відмінностей програмного забезпечення від інтелектуальних товарів є те, що воно не має споживчої цінності без комплементарного матеріального продукту - апаратного забезпечення.

На ринок програмного забезпечення виходять і стають все більш і більш затребуваними вдосконалені моделі бізнесу, і в кінцевому результаті перед покупцем програмного забезпечення гостро ставатиме питання більш вигідного придбання: покупка ліцензійного комерційного програмного забезпечення (наприклад операційна система Microsoft Windows або офісний пакет Microsoft Office); безкоштовне користування альтернативним некомерційним програмним продуктом, який вільно розповсюджується через мережу інтернет (операційна система Linux, офісний пакет OpenOffice); або використання програмних продуктів як певних послуг (операційна система Ghost, офісний пакет Google Docs); можливість незаконно користуватися піратськими копіями комерційних програмних продуктів. Звичайно ж, і самі виробники програмних забезпечень хочуть знати, яка ж з моделей бізнесу найбільш вигідна в кожному окремому випадку:

- отримання прибутку від поширення ліцензій на програмне забезпечення
- отримання прибутку від продажів підписок на додатки як послуги
- прибуток від візуалізації реклами в програмних продуктах
- безкоштовне розповсюдження програмного забезпечення (можливо, навіть з відкритими початковими кодами)
- прибуток від надання будь-яких інших послуг (встановлення, налаштування, технічна підтримка та інше)

Точної відповіді на поставлені запитання поки що немає. Особливістю взаємодій учасників ринку програмного забезпечення в наш час практично не аналізувалися, і тому проблема конкуренції на ринках програмного забезпечення є актуальною.

На формування і функціонування ринку впливають деякі особливості, котрі відрізняють знання (як інтелектуальний товар) від інших товарів: нематеріальність, бінарність, наявність (або відсутність) інституту захисту авторських прав.

Нематеріальність знань показує їх фізичне відсутність («не існування»), яка призводить до складності в оцінюванні собівартості розробки таких товарів.

Бінарність розуміється в математичному сенсі яка проявляється в тому, що повторне застосування операції до об'єкта дає такий же результат, що і одинарне. Два ідентичні інтелектуальні товари повністю однакові, тобто знання, якщо вже одного разу створене, то воно не втратить своїх властивостей, навіть якщо ним користуватимуться багато разів один або багато споживачів.

Додатковими послугами для товарів на ринку програмного забезпечення є можливі послуги встановлення необмеженого числа копій одного і того ж програмного продукту на різні машини. І вартість копіювання програмного забезпечення мала в порівнянні з ціною на розроблений продукт, це і призводить до збільшення віддачі від масштабності розповсюдження.

Наслідком нематеріальності та бінарності є можливість незаконного копіювання і поширення таких товарів без дозволу їх автора. [1]

Копіювання товару на ринку ПО здійснюється практично без грошових витрат на відміну від розробки нових товарів. Витрати на запис програми на компакт диск мінімальна, а поширення її ж через інтернет ще в два рази знижує витрати. У зв'язку з цим обчислення витрат на ринку програмного забезпечення має власну унікальність, яка полягає в неможливості рознесення витрат за копіями продукції. Постійні витрати, таким чином, на ринку програмного забезпечення наближені до нуля, а собівартість майже однакова з постійними витратами розробки нового товару. Також варто зазначити, що бінарність надалі призводить до відсутності рідкості, тобто знання являють собою загальне благо. А так як у кожного знання є творець, то воно одночасно стає і приватним благом.

У свою чергу, захист авторських прав забезпечує якість, яка полягає в тому, що покупка походить від першоджерела. Цей інститут може існувати на певних ринках знань. Можна пригадати, як ще зовсім недавно багато країн на власні очі підтримували копіювання програм, які були створені в інших країнах. Сьогодні в багатьох країнах світу таке незаконне копіювання і користування програмними продуктами має назву комп'ютерне піратство і відстежується законом. Але все таки, наприклад, в 2009 році 43% всього програмного забезпечення в світі було піратським, в Західній Європі цей показник становив 34%, в США 20 %, в Китаї 79% (за даними компанії IDI).[2] Аналізуючи, можна сказати, що програмне забезпечення, є інтелектуальним продуктом, нематеріальним, бінарним, а його використання напряму залежить від роботи інституту щодо захисту авторського права в кожній країні окремо.

Ринок програмного забезпечення - ринок знань який відрізняється від традиційного ринку, в першу чергу властивостями товарів - нематеріальністю, бінарністю і захистом авторських прав.

Крім того, на відміну від іншого інтелектуального товару, програмне забезпечення може входити до складу інтелектуального капіталу, його неможливо використовувати без свого матеріального продукту, такого як, апаратне забезпечення, хоча цінність має тільки програмне забезпечення, яке представлено в особливій електронній формі, яку визначає апаратне забезпечення, що використовується.

Така особливість, зокрема, лягає в основу невідповідності поведінки виробника традиційного комерційного програмного продукту цілям своєї діяльності: виробник такого продукту змушений викривати розповсюджувача і користувача піратської копії замість займатися тільки створенням. Завдяки якійсь невідповідності можна створити інновацію, і цими інноваціями на ринку програмного забезпечення стають нові моделі бізнесу, які засновані на пропозиціях некомерційних програмних продуктів, програмного забезпечення як послуги. [3][4]

1.1.1 Класифікація програмного забезпечення

Підходи до класифікації програмного забезпечення досить докладно формалізовані в міжнародному стандарті ISO / IEC 12182. Зокрема, перша версія стандарту передбачала 16 критеріїв класифікації програмних засобів:

- за режимом експлуатації;
- за масштабом;
- за стабільністю;
- за функцією;
- за вимогою захисту;
- за вимогою надійності;
- за необхідними принципами роботи даного продукту;
- за вихідною мовою;
- за прикладною областю;
- за обчислювальною системою і середовищем;

- за класом користувача;
- за вимогою до обчислювальних ресурсів;
- за критичністю;
- за готовністю;
- за поданням даних;
- за використанням програмних даних

Прикладами класів функції програмного забезпечення є:

- обробка ділових повідомлень;
- компіляція;
- наукові обчислення;
- обробка текстів;
- медичні системи;
- системи управління.

Прикладами класів прикладної області є:

- наука;
- побутові пристрої;
- обладнання;
- апаратура управління процесом;
- підприємництво;
- система організації мережі.

Прикладами класів масштабу програмного забезпечення є:

- малий;
- середній;
- великий

Прикладами класів критичності є:

- національна безпека. Важливо не допустити витоку в інші країни;
- людське життя. Використання додатку повинно бути безпечним;
- соціальний хаос або паніка. Пріоритетно не дозволити поширення загального панічного стану у населення;
- організаційна безпека. Сторонні особи не мають права перебувати на об'єктах компанії, у них немає доступу до програмного забезпечення;
- приватна власність. Бажання компанії не повинні перекреслювати інтереси окремих громадян;
- секретність. Необхідно забезпечити збереження та цілісність даних.

Прикладами класів користувача є:

- початківець;
- середній;
- фахівець (експерт);
- звичайний;
- випадковий;
- інша система програмного забезпечення;
- технічні засоби.

Прикладами класів стабільності є:

- постійне внесення змін;
- дискретне внесення змін;
- малоймовірне внесення змін

За ступенем переносимості програми ділять на:

- платформо залежні;
- кросплатформені.

Кросплатформеність програмного забезпечення – це можливість виконувати його, без перекомпілювання програми, як на різних апаратних платформах, так і під управлінням різних операційних систем (інакше кажучи, можливість запуску виконуваного файлу на платформах різних операційних систем).

Прикладами програмного забезпечення, що виконується на різних апаратних платформах і під управлінням різних операційних систем, є різноманітні програми, написані на мовах програмування для віртуальних машин, таких як PHP, Perl, Python, Java, і багато інших.

За способом представлення даних програми ділять на:

- індивідуальні - вхід тільки для конкретних користувачів;
- обмежені - допускаються тільки люди з певними правами доступу;
- вільні - дозволено користуватися будь-яким користувачам.

За призначенням програми ділять на:

- системні
- прикладні
- інструментальні

Системне програмне забезпечення – це комплекс програм, який забезпечує управління компонентами комп'ютерної системи, такими як процесор, оперативна пам'ять, пристрої введення-виведення, мережеве обладнання, виступаючи як «міжшаровий інтерфейс», з одного боку якого апаратура, а з іншого - додатки користувача.

На відміну від прикладного програмного забезпечення, системне не вирішує конкретні практичні завдання, а лише забезпечує роботу інших програм, надаючи їм сервісні функції, управляє апаратними ресурсами обчислювальної системи, а деталі апаратної і мікропрограмної реалізації обчислювальної системи абстрагуються. Віднесення того чи іншого програмного забезпечення до

системного є умовним, і залежить від угод, що були використані в конкретному контексті. Як правило, до системного програмного забезпечення відносяться:

- операційні системи
- утиліти
- системи управління базами даних
- широкий клас сполучного програмного забезпечення

Прикладне програмне забезпечення - це програма, призначена для виконання певних завдань користувача і розрахована на безпосередню взаємодію з ним.

Це частина з найбільшим переліком програмного забезпечення. Сюди відносяться графічні та текстові редактори, браузери, бази даних і все, що люди використовують у звичній роботі за комп'ютером. Тут же знаходяться антивірусні пакети, бухгалтерія та різні архіви.

Сенс цього різновиду у виконанні чітко поставленого завдання: малювати, рахувати, відкривати веб-сторінки, набирати текст. Якщо утиліта потрібна для конкретного виконання дії, то вона є прикладним програмним забезпеченням.

Інструментальне програмне забезпечення – це специфічне забезпечення будь-якої комп'ютерної техніки. Його можна було б віднести до прикладного, але через специфіку застосування його виділили в окремий вид. Основна функція - налагодження, налаштування, переписування програмного коду.

Сюди входять компілятори, відладчики, перекладачі високого рівня, редактори, інтерпретатори та інші засоби.

Класифікація програмного забезпечення за сектором індустрії:

Класифікація програмного забезпечення за сектором індустрії включає кілька підходів. В цілому, програмне забезпечення ділять на замовне, тобто створюється для конкретного замовника, і продуктове, тобто створюється для

продажу на ринку. У свою чергу, за типами споживача програмне забезпечення ділять на Business-to-Business (B2B), тобто для підприємств і організацій, Business-to-Government (B2G, зав'язані на державних і муніципальних сервісах) і Business-to-Consumer (B2C), тобто для приватних осіб.

Одним з варіантів класифікації за сектором індустрії є поділ на:

- програмне забезпечення для корпоративного замовника (Enterprise software vendors)
- програмне забезпечення для масового споживача (Mass-market software vendors)
- IT-сервіси

Інший підхід полягає в розподілі індустрії програмного забезпечення на три сектори:

- бізнес-продукти загального призначення (Business Function Software)
- спеціалізовані бізнес-продукти (Industrial Business Software)
- продукти для приватного життя (Consumer Software)

Бізнес-продукти загального призначення створюються для підтримки функціонування підприємств і організацій та включають бухгалтерські системи, фінансові системи, системи кадрового обліку і тому подібне.

Спеціалізовані бізнес-продукти орієнтовані на завдання конкретного типу бізнесу: геоінформаційні системи, медичні системи, логістичні системи і так далі.

Продукти для приватного життя включають антивірусне програмне забезпечення і системи для інформаційної безпеки, різні утиліти, освітнє програмне забезпечення, мультимедійні програми і тому подібне.

За способом використання та поширення програмне забезпечення ділиться на:

- Free

Безкоштовно поширювані програми. Їх дозволяється вільно поширювати, копіювати і використовувати без доплати. При цьому творець компонента може брати оплату за окремі послуги софта - копіювання даних на диск, збільшення обсягу пам'яті і інші.

- Adware

Ще один вид, яким допускається користуватися без внесення грошей. Усередині іноді містяться рекламні ролики або функції, які відкриваються тільки за умови покупки. Ще один варіант - необхідність установки додаткових утиліт для роботи.

- Shareware

Для некомерційного умовно-безкоштовного використання. Тобто один користувач використовує її для особистих потреб. Для регулярного користування компанією будь-якого розміру передбачена оплата або заборона на роботу.

- Trial

Скрипт без внесення фінансових коштів. Обмежено час, який допускає користуватися програмним забезпеченням. Всі функції працюють протягом 10-30 діб або 10-30 запусків. Потім потрібно ввести ключ і оплатити.

- Demo

Програмне забезпечення, яке певний період працює без оплати. В рамках цього часу можна користуватися всім функціоналом або обмеженим набором можливостей ознайомлення. Після закінчення дії пробної версії блокується робота програми, продовжити робочий процес можливо лише після покупки.

- Закрите ПЗ

Це приватна власність авторів. Отримати їх можна тільки за строго зазначеними вимогами власників. Серед таких умов може бути грошова компенсація. Видається з вихідним кодом.

1.1.2 Життєвий цикл програмного забезпечення

Розробка успішного продукту - складний багатоступеневий процес. Створення і розвиток будь-якого продукту відбувається поступово, проходячи ряд обов'язкових етапів, частина з яких може йти паралельно.

Життєвий цикл програмного забезпечення (software development life cycle, SDLC) - умовна схема, що включає фази створення програмного забезпечення. Цикл розробки пропонує шаблон, використання якого полегшує проектування, створення і випуск якісного програмного забезпечення. На виході повинен вийти економічно вигідний продукт, що відповідає вимогам замовника. У структуру циклу розробки входять всі етапи життя програмного забезпечення від його народження у вигляді ідеї до умовної «смерті» (етапи до розробки, під час розробки і після неї).

Життєвий цикл програмного забезпечення складається з наступних етапів: (рис. 1)

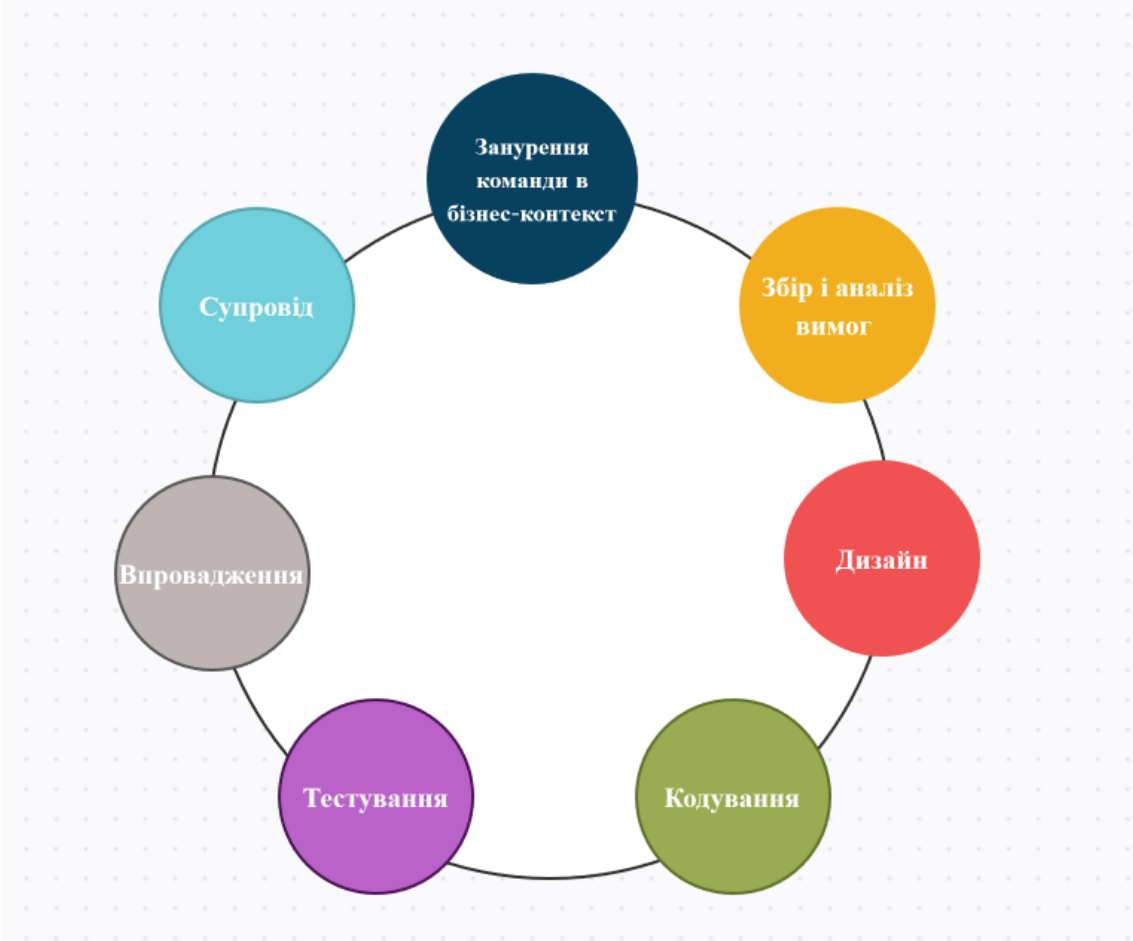


Рисунок 1 - Життєвий цикл програмного забезпечення

Занурення команди в бізнес-контекст - важлива фаза циклу розробки програмного забезпечення, з якої починається робочий процес. Клієнт повинен розуміти, що він витрачає час і ресурси на цьому етапі не просто так. Йому важливо, щоб команда проекту занурилася в деталі бізнесу і ретельніше розібралася, для якого ринку готується продукт, як зробити його по-справжньому корисним. Це актуально для будь-якої галузі.

Збір і аналіз вимог - одна з найбільш важливих фаз життєвого циклу. Вона виконується за участю клієнтів, відділу продажів і відділу аналітики. На цьому етапі важливо максимально точно і однозначно визначити вимоги до проекту і для команди проекту, і для бізнесу. Аналітики допомагають визначити кінцеві цілі і завдання роботи. Також визначаються терміни і вартість розробки програмного

забезпечення, формується і підписується технічне завдання на розробку програмного забезпечення.

Дизайн. До цього етапу переходять, коли є чітке розуміння цілей проекту і досить докладно сформульовані його вимоги. На підставі вимог зазвичай пропонується декілька підходів до проектування архітектури програмної системи, її функцій, зовнішніх умов функціонування, інтерфейсів і розподілу функцій між користувачами і системою, вимоги до програмних і інформаційних компонентів. Проектування системи проводиться на основі результатів формування вимог. А також розробляється та визначається найбільш підходяща система управління базами даних, проектуються структури зберігання даних, обумовлюються вимоги до апаратного забезпечення, визначається набір організаційних заходів, які необхідні для впровадження програмного забезпечення, а також перелік документів що регламентують його використання.

Кодування. Це етап безпосередньої реалізації системи - написання коду на обраній з урахуванням поставлених завдань мові програмування. На даній стадії будуються прототипи як цілої програмної системи, так і її частин, здійснюється фізична реалізація структур даних, розробляються програмні коди, виконується налагоджувальне тестування, створюється технічна документація. В результаті етапу кодування з'являється робоча версія продукту.

Тестування. Цей етап супроводжується перевіркою працездатності системи, виявленням, фіксацією і виправленням багів до тих пір, поки продукт не досягне необхідних стандартів якості. У цей період зазвичай виникає багато не співпадінь, білих плям, дефектів. Їх потрібно оперативно усунути. У систему вбудовуються спеціальні механізми, які дають можливість проводити тестування програмного забезпечення на відповідність вимог до нього, перевірку оформлення і наявність необхідного пакету документації. Результатом тестування є усунення всіх недоліків програмного продукту і висновок про його якість.

Впровадження. Коли продукт протестований і готовий до розгортання, його випускають на ринок. Іноді розгортання продукту відбувається послідовно в рамках бізнес-стратегії. Продукт може бути спочатку випущений в обмеженому сегменті і протестований в реальному бізнес-середовищі, потім, ґрунтуючись на відгуках, буде випущеним таким як є або із поліпшеннями. Введення в експлуатацію програмного забезпечення передбачає встановлення програмної системи, навчання користувачів, документування.

Супровід. На цьому етапі життєвого циклу забезпечується періодична технічна підтримка системи. Це дозволяє переконатися, що система не застаріла і відповідає сучасним стандартам і технологіям. Сюди входить постійна оцінка продуктивності і оновлення компонентів.

Крім зазначених фаз, виділимо і фазу маркетингового просування (рис. 2). Маркетингова діяльність є наскрізною і проводиться протягом усього життєвого циклу проекту. Клієнту важливо створити поінформованість про продукт серед його цільової аудиторії, привернути користувачів, напрацювати клієнтську базу.

Всі процеси розробки типові, але підходи до роботи бувають різні. І в залежності від підходу визначається те, якої довжини будуть етапи, чи будуть вони робитися послідовно або ітеративно, і як їх будуть виконувати.

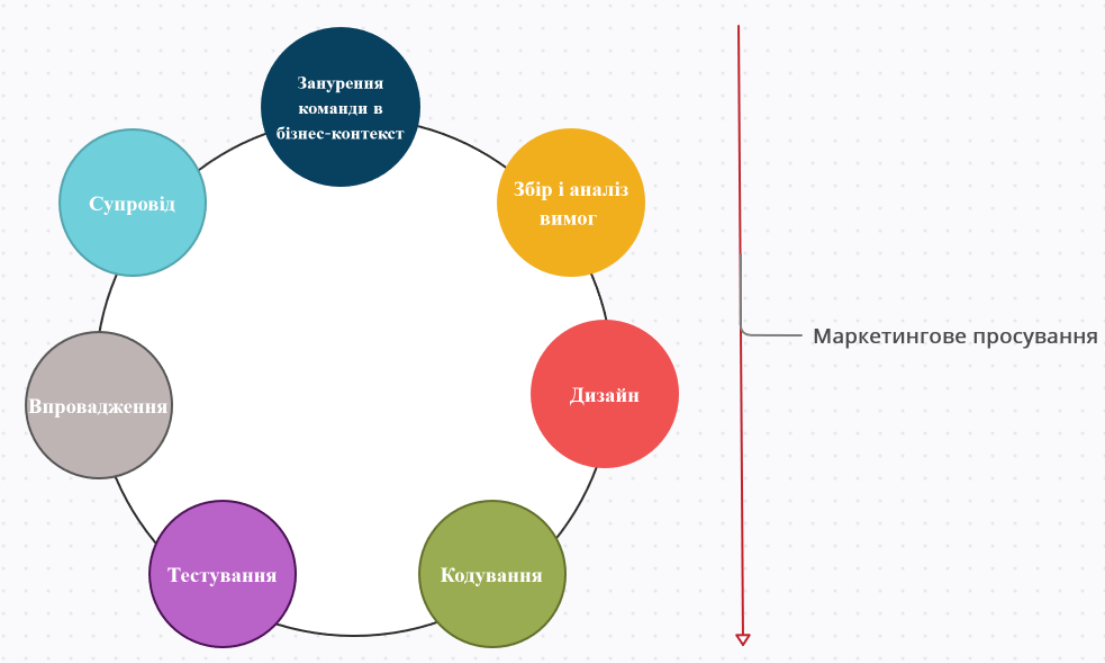


Рисунок 2 - Маркетингове просування в життєвому циклі програмного забезпечення

1.2 Означення якості програмного забезпечення

У контексті програмної інженерії якість програмного забезпечення стосується двох пов'язаних, але різних понять.

Функціональна якість програмного забезпечення відображає, наскільки воно відповідає певній конструкції чи відповідає їй на основі функціональних вимог або специфікацій. Цей атрибут можна також охарактеризувати як придатність певного програмного забезпечення або порівняння його з конкурентами на ринку як продукту, який вартий уваги. Це ступінь виготовлення правильного програмного забезпечення.

Структурна якість програмного забезпечення стосується того, як воно відповідає нефункціональним вимогам, які підтримують виконання функціональних вимог, таких як надійність чи здатність до виправлення.

Багато аспектів структурної якості можна оцінити лише статично шляхом аналізу внутрішньої структури програмного забезпечення, його вихідного коду, на

рівні одиниць, системному рівні (іноді це називають наскрізним тестуванням), що фактично відповідає тому, що її архітектура дотримується обґрунтованих принципів архітектури програмного забезпечення.

Однак деякі структурні якості, такі як зручність використання, можуть бути оцінені лише динамічно (користувачі чи інші, що діють від їх імені, взаємодіють із програмним забезпеченням або, принаймні, деяким прототипом або частковою реалізацією; навіть взаємодія з макетною версією, виготовленою на картоні, представляє динаміку, оскільки така версія може вважатися прототипом). Інші аспекти, такі як надійність, можуть включати не лише програмне забезпечення, а й базове обладнання, тому його можна оцінювати як статично, так і динамічно.

Якість функціональних можливостей зазвичай оцінюється динамічно, але також можна використовувати статичні тести (наприклад, огляди програмного забезпечення).

Історично склалося так, що структура, класифікація та термінологія атрибутів та метрик, застосовних до управління якістю програмного забезпечення, були виведені з ISO 9126 та наступного стандарту ISO / IEC 25000. На основі цих моделей Консорціум з якості програмного забезпечення для IT (CISQ) визначив п’ять основних бажаних структурних характеристик, необхідних для того чи іншого програмного забезпечення для забезпечення ділової цінності:

1) надійність
2) ефективність
3) безпека
4) придатність до виправлення
5) належний розмір

А також додаткові характеристики:

- швидкість доставки
- можливість бути протестованим

- зручність використання

Вимірювання якості програмного забезпечення кількісно визначає якою мірою програмна система оцінює показники по кожному з цих п'яти вимірів. Сукупний показник якості програмного забезпечення може бути обчислений за допомогою якісної або кількісної схеми оцінювання або поєднання обох, а потім системи зважування, що відображає пріоритети.

Цей погляд на позиціонування якості програмного забезпечення на лінійному континуумі доповнюється аналізом «критичних помилок програмування», які за певних обставин можуть призвести до катастрофічних відмов або погіршення продуктивності, які роблять дану систему непридатною для використання незалежно від рейтингу на основі агрегованих вимірювань. Такі помилки програмування, виявлені на системному рівні, становлять до 90 відсотків виробничих проблем, тоді як на рівні одиниць, навіть якщо вони набагато численніші, помилки програмування становлять менше 10 відсотків виробничих проблем. Як наслідок, якість коду без контексту всієї системи має обмежену цінність.

Для перегляду, дослідження, аналізу та передачі вимірювань якості програмного забезпечення концепції та прийоми візуалізації інформації забезпечують корисні візуальні, інтерактивні засоби, зокрема, якщо кілька показників якості програмного забезпечення мають бути пов’язані між собою або компонентами програмного забезпечення чи системи. Наприклад, програмні карти представляють спеціалізований підхід, який може виражати та поєднувати інформацію про розробку програмного забезпечення, якість програмного забезпечення та динаміку системи.

Якість програмного забезпечення також відіграє роль на етапі випуску програмного проекту. Зокрема, якість та налагодження процесів випуску (також процесів виправлення), управління конфігурацією є важливими частинами загального процесу інженерного програмного забезпечення.

Якість програмного забезпечення мотивована принаймні двома основними перспективами:

- Управління ризиками

Помилки програмного забезпечення можуть спричинити смертельні наслідки для людей. Причини можуть бути від погано розроблених користувацьких інтерфейсів до прямих помилок програмування. Наприклад, випадок Boeing 737 або випадки ненавмисного прискорення чи Therac-25. Це призвело до того, що виникли вимоги до розробки деяких типів програмного забезпечення, зокрема історично складених програм, медичних та інших пристроїв, що регулюють критичну інфраструктуру. США, в рамках Федеральної авіаційної адміністрації (FAA) та Служба сертифікації літаків FAA забезпечують програми, політику, керівництво та навчання, зосереджують увагу на програмному забезпеченні та складному електронному обладнанні, що впливає на повітряний продукт. Стандарти сертифікації, такі як DO-178C, ISO 26262, IEC 62304 тощо, надають вказівки.

- Управління витратами

Як і в будь-яких інших галузях техніки, програмний продукт або послуга, що регулюються якістю програмного забезпечення, коштує менше, ніж виправляння помилок, які не були знайдені. Структурна якість основних бізнес-додатків (таких як планування корпоративних ресурсів (ERP), управління взаємовідносинами з клієнтами (CRM) або великі системи обробки транзакцій у фінансових послугах) призводить до додаткових фінансових витрат, часу. Більш того, низька якість програми може спричиняти перебої в роботі через пошкоджені дані, збої в роботі програм, порушення безпеки та проблеми з продуктивністю. CISQ повідомляє про вартість неякісної оцінки впливу:

- 2,08 трильйона доларів у 2020 році
- 2,84 трильйона доларів у 2018 році

1.2.1 Історія розвитку підходів у тестуванні програмного забезпечення

В умовах постійно зростаючої конкуренції одним з найвагоміших аргументів у боротьбі за замовника є висока якість продукції, що виробляється. Для досягнення цієї мети в організації повинна проводитися систематична робота, ця робота повинна бути спрямована на збільшення задоволеності замовника, за допомогою виконання всіх його вимог. Ця робота може бути визначена як система управління якістю.

Перші програмні системи розроблялися в рамках програм наукових досліджень або програм для потреб міністерств оборони. Тестування таких продуктів проводилося строго формалізовано із записом всіх тестових процедур, тестових даних, отриманих результатів. Тестування виділялося в окремий процес, який починався після завершення кодування, але при цьому, як правило, виконувався тим же персоналом.

У 1960-х роках багато уваги приділялося «вичерпному» тестуванню, яке повинно проводитися із використанням всіх шляхів в коді або всіх можливих вхідних даних. Було відзначено, що в цих умовах повне тестування програмного продукту неможливе, тому що, по-перше, кількість можливих вхідних даних дуже велика, по-друге, існує безліч шляхів, по-третє, складно знайти проблеми в архітектурі і специфікаціях. З цих причин «вичерпне» тестування було відхилено і визнано теоретично неможливим.

На початку 1970-х тестування програмного забезпечення позначалося як «процес, спрямований на демонстрацію коректності продукту» або як «діяльність по підтвердженню правильності роботи програмного засобу». В програмній інженерії, що зароджувалася верифікація програмного забезпечення значилася як «доказ правильності». Хоча концепція була теоретично перспективною, на практиці вона вимагала багато часу і була недостатньо всеохоплюючою. Було вирішено, що доказ правильності - неефективний метод тестування програмного

забезпечення. Однак, в деяких випадках демонстрація правильної роботи використовується і в наші дні, наприклад, приймально-здавальні випробування.

У другій половині 1970-х тестування вважалось виконанням програми з наміром знайти помилки, а не довести, що вона працює. Успішний тест - це тест, який виявляє раніше невідомі проблеми. Даний підхід прямо протилежний попередньому. Зазначені два визначення є «парадоксом тестування», в основі якого лежать два протилежних твердження: з одного боку, тестування дозволяє переконатися, що продукт працює добре, а з іншого - виявляє помилки в програмі, показуючи, що продукт не працює. Друга мета тестування є більш продуктивною з точки зору поліпшення якості, так як не дозволяє ігнорувати недоліки.

У 1980-х тестування розширилося таким поняттям як попередження дефектів. Проектування тестів - найбільш ефективний з відомих методів попередження помилок. В цей же час стали висловлюватися думки, що необхідна методологія тестування, зокрема, що тестування має включати перевірки на всьому циклі розробки, і це повинен бути керований процес. В ході тестування треба перевірити не тільки зібрану програму, а й вимоги, код, архітектуру, самі тести. «Традиційне» тестування, яке існувало до початку 1980-х, відносилося тільки до компільованої, готової системи (зараз це зазвичай називається системне тестування), але в подальшому тестувальники стали залучатися в усі аспекти життєвого циклу розробки. Це дозволяло раніше знаходити проблеми у вимогах та архітектурі і тим самим скорочувати терміни і бюджет розробки. В середині 1980-х з'явилися перші інструменти для автоматизованого тестування. Передбачалося, що комп'ютер зможе виконати більше тестів, ніж людина, і зробить це більш надійно. Спочатку ці інструменти були вкрай простими і не мали можливості написання сценаріїв на скриптових мовах.

На початку 1990-х в поняття «тестування» стали включати планування, проектування, створення, підтримку і виконання тестів і тестових оточень, і це означало перехід від тестування до забезпечення якості, що охоплює весь цикл

розробки програмного забезпечення. У цей час починають з'являтися різні програмні інструменти для підтримки процесу тестування: більш просунуті середовища для автоматизації з можливістю створення скриптів і генерації звітів, системи управління тестами, програми для проведення навантажувального тестування. В середині 1990-х з розвитком інтернету і розробкою великої кількості веб-додатків особливу популярність стало отримувати «гнучке тестування» (за аналогією з гнучкими методологіями програмування).

У 2000-х з'явилося ще ширше визначення тестування, коли в нього було додано поняття «оптимізація бізнес-технологій» (business technology optimization, BTO). Воно направляє розвиток інформаційних технологій відповідно до цілей бізнесу. Основний підхід полягає в оцінці і максимізації значущості всіх етапів життєвого циклу розробки програми для досягнення необхідного рівня якості, продуктивності, доступності. [5]

1.2.2 Життєвий цикл тестування програмного забезпечення

Життєвий цикл тестування програмного забезпечення (STLC) - це послідовність конкретних дій, які виконуються в процесі тестування для забезпечення досягнення цілей якості програмного забезпечення (рис. 3). STLC включає в себе як верифікацію, так і валідацію. Всупереч поширеній думці, тестування програмного забезпечення - це не просто окрема діяльність (тестування), а ряд заходів, проведених методологічно, щоб допомогти сертифікувати програмний продукт.

Кожен з цих етапів має певні критерії входу і виходу, пов'язані з ним види діяльності та результати.

Критерії вступу перераховують обов'язкові елементи, які повинні бути виконані до початку тестування. Критерії виходу визначають елементи, які повинні бути виконані до завершення тестування.

Аналіз вимог. На цьому етапі група тестування вивчає вимоги з точки зору тестування. Команда QA для більш детального розуміння може взаємодіяти з різними зацікавленими сторонами (клієнт, бізнес-аналітик, технічні керівники, системні архітектори)

Вимоги можуть бути функціональними (визначення того, що повинно робити програмне забезпечення) або нефункціональними (визначення продуктивності чи безпеки системи)

Можливість автоматизації тестування для даного проекту також виконується на цьому етапі.

Робота, яка відбувається під час цього етапу:

- визначаються типи тестів, які необхідно виконати
- виконується збір детальної інформації про пріоритети тестування
- відбувається підготовка матриці простежування вимог (RTM)
- визначаються подробиці середовища тестування, де передбачається його проведення
- автоматизація техніко-економічного обґрунтування (за необхідності)

Результати цієї фази:

- готова матриця простежування вимог
- автоматизація техніко-економічного обґрунтування

Рисунок 3 - Життєвий цикл тестування програмного забезпечення

Планування. Як правило, на цьому етапі старший менеджер по забезпеченню якості визначає зусилля і кошторис витрат за проектом, а також готує і завершує план тестування. На цьому етапі також визначається стратегія тестування.

Робота, яка відбувається під час цього етапу:

- підготовка плану тестування або стратегічного документу для різних типів тестування
- вибір тестових інструментів
- оцінка зусилля тестування
- планування ресурсів та визначення ролей і обов'язків

Результати цієї фази:

- план тестування або стратегічний документ
- документ про оцінку зусиль

Дизайн. Цей етап включає розробку, перевірку і переробку тестових випадків і тестових сценаріїв. Тестові дані, ідентифіковані, створені і перевірені, а потім перероблені (якщо є така потреба).

Робота, яка відбувається під час цього етапу:

- створення тестових випадків, сценаріїв автоматизації (якщо є така потреба)
- огляд, базові тести і сценарії
- підготовка тестових даних

Результати цієї фази:

- тестові випадки та скрипти
- тестові дані

Налаштування тестового середовища. Середовище тестування визначає умови програмного і апаратного забезпечення, при яких тестується робочий продукт. Налаштування тестового середовища є одним з найважливіших аспектів процесу тестування і може виконуватися паралельно з етапом розробки тестового набору.

Команда тестування може бути не залучена в цю діяльність, якщо клієнт або команда розробників надає середовище тестування, в цьому випадку команда тестування повинна виконати перевірку готовності (димове тестування) даного середовища.

Робота, яка відбувається під час цього етапу:

- ознайомлення з необхідною архітектурою
- підготовка списку вимог до апаратного та програмного забезпечення для середовища тестування

- налаштуванням середовища
- виконання димового тестування на збірці

Результати цієї фази:

- готове середовище з налаштованими тестовими даними
- результати димового тестування

Тестування. На цьому етапі тестувальники проводитимуть тестування на основі планів тестування та підготовлених тестових випадків. При виявленні помилок буде повідомлено команді розробників для виправлення, після чого буде проведено повторне тестування.

Робота, яка відбувається під час цього етапу:

- виконання тестів згідно з планом
- документування результатів випробувань і реєстрація дефектів для невдалих випадків
- карта дефектів для тестових випадків в RTM
- повторне тестування виправленого дефекту
- відстежування дефектів до закриття

Результати цієї фази:

- завершено RTM зі статусом виконання
- оновлені тестові випадки з результатами
- звіти про дефекти

Закриття циклу тестування. Команда тестування зустрінеться, обговорить і проаналізує артефакти тестування, щоб визначити стратегії, які повинні бути реалізовані в майбутньому, використовуючи результати поточного циклу тестування. Ідея полягає в тому, щоб усунути вузькі місця процесу для майбутніх циклів тестування та поділитися передовим досвідом для будь-яких подібних проектів в майбутньому.

Робота, яка відбувається під час цього етапу:

- оцінка критеріїв завершення циклу на основі часу, охоплення тестуванням, вартості, програмного забезпечення, критичних бізнес-цілей, якості
- підготовка тестових показників на основі вищевказаних параметрів
- документування навчання поза проектом
- підготовка звіту про закриття тестування
- якісна і кількісна звітність про якість робочого продукту перед замовником
- аналіз результатів тестування, щоб визначити розподіл дефектів за типом і серйозністю

Результати цієї фази:

- звіт про закриття тестування
- тест метрики

1.3 Місце контролю якості у різних методологіях управління проектами

Методологія управління проектами - це набір керівних принципів і процедур для управління проектом. Методологія, яку вирішено обрати, визначає, як буде організована робота та взаємодія.

В теорії можна використовувати будь-яку методологію незалежно від того, яке програмне забезпечення для управління проектом використовується.

В реальності ж більшість систем управління завданнями і проектами підходять для кількох різних методологій. В цій частині буде розглянуто 9 основних та найбільш відомих методологій управління проектами. Такі як:

1. Waterfall (каскадна модель, «водоспад»)

Методологія Waterfall – найстаріша з усіх. Вперше вона була викладена американським вченим в галузі інформатики Уінстоном Уокером Ройсом в 1970 році у відповідь на потребу управління, що все більш ускладнювалась процесом

розробки програмного забезпечення. З тих пір вона набула широкого поширення, особливо в сфері програмного забезпечення.

Каскадна модель характеризується послідовністю. Крім цього, вона в значній мірі орієнтована на вимоги. Необхідно мати абсолютно чітке уявлення про те, що потрібно проекту, щоб продовжити роботу з каскадною методологією.

Методологія Waterfall ділиться на три окремих етапи. Спочатку необхідно зібрати і проаналізувати вимоги, потім розробити рішення і підхід, впровадити рішення і виправити проблеми, якщо вони з'явилися.

Кожен етап цього процесу автономний. Щоб перейти до наступного, необхідно завершити попередній етап (рис. 4).

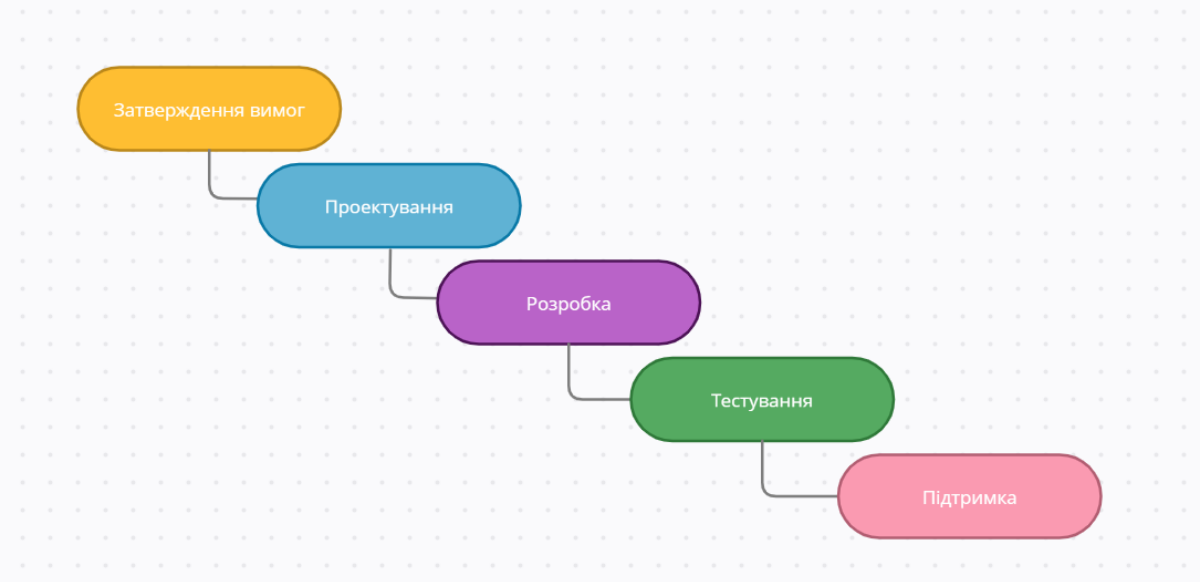


Рисунок 4 - Каскадна модель

Переваги каскадної моделі:

Витрачаючи час на ранніх стадіях розвитку проекту, менеджери створюють умови для своєчасного виконання вимог. Це дозволяє заощадити час і сили на виправлення недоліків і вирішення проблем в подальшому.

- Простота використання

Цю модель просто зрозуміти і використовувати. Розподіл на етапи досить інтуїтивний, його просто освоїти незалежно від досвіду.

- Структура

Жорсткість методології Waterfall одночасно і недолік, і явна перевага. Чіткий поділ на етапи дозволяє організувати і розподілити роботу. Оскільки назад повернутися не можна, необхідно ідеально справлятися з виконанням кожного етапу, що часто дозволяє домогтися кращих результатів.

- Документація

Оскільки багато уваги приділяється збору і розумінню вимог, модель Waterfall в значній мірі спирається на документацію. Завдяки цьому новим ресурсам простіше влитися в проект і почати роботу над ним.

Недоліки каскадної моделі:

- Підвищений ризик

Жорсткість методології означає, що, якщо виявлено помилку або знадобиться додати зміни, доведеться починати проект спочатку. А це означає, що є вірогідність зовсім не завершити проект вчасно.

- Складність першого етапу

Весь підхід Waterfall залежить від того, наскільки правильно всі зрозуміють і проаналізують вимоги. Якщо не вдасться зробити це або якщо вимоги зміняться, доведеться починати спочатку. Тому ця методологія управління не підходить складним довгостроковим проектам.

Проекти, для яких найкраще підійде каскадна модель:

- короткі нескладні проекти
- проекти з чітко встановленими вимогами
- проекти, в яких змінюються ресурси, залежні від детальної документації.

2. Agile (гнучка методологія)

Agile - це ще одна методологія управління з акцентом на розробці програмного забезпечення. З'явилася вона як результат незастосовності методології Waterfall в рамках складних проектів.

Хоча ідеї, властиві Agile, вже давно використовуються в сфері розробки програмного забезпечення, формально методологія з'явилася лише в 2001 році, коли кілька представників з інформаційних технологій випустили Agile-маніфест.

Agile повністю протилежна методології Waterfall за підходом і ідеологією. Сама назва з англійської мови перекладається як «гнучкий», а це значить, що в управлінні використовується швидкий і гнучкий підхід.

У Agile проектах не потрібен ретельний збір вимог.

Методологія характеризується невеликими циклічними змінами, які впроваджують у відповідь на зміну вимог (рис. 5).

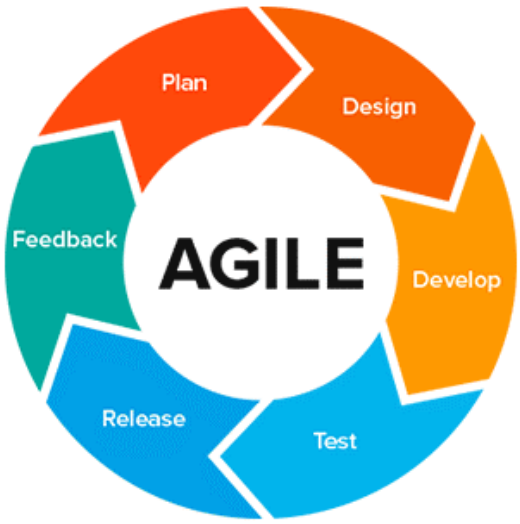


Рисунок 5 - Agile методологія

Переваги гнучкої методології Agile:

- гнучкість і свобода

Оскільки тут не потрібно чітко позначати етапи і робити акцент на вимогах, у виконавців проекту з'являється можливість експериментувати і вносити зміни поступово. Саме тому Agile відмінно підходить творчим проектам.

- Знижений ризик

Методологія Agile передбачає регулярне отримання зворотного зв'язку від зацікавлених учасників і подальше внесення змін. Це значно скорочує ризик провалу проекту, так як потрібні ресурси залучені в процес.

Недоліки Agile:

- відсутність чіткого плану

У Agile підході реагування на зміни відбувається тоді, коли вони виникають. Відсутність чіткого плану ускладнює управління ресурсами і планування. Доведеться постійно балансувати і у випадковому порядку переводити ресурси з одного завдання на інше.

- Складність взаємодії

Відсутність чіткого плану означає, що всім зацікавленим сторонам, включаючи замовників і спонсорів, доведеться працювати в набагато більш тісній співпраці, щоб кожен учасник проекту знав про всі зміни, завдання і їх актуальність.

Гнучкість підходу Agile дозволяє адаптувати його до проектів різного типу. Методологія найкраще працює в випадках:

- коли немає впевненості, яким повинен бути кінцевий результат, але є загальне уявлення про продукт
- коли проект потрібно швидко підлаштовуватися під зміни
- якщо взаємодія і комунікація – це сильні сторони, а планування відсутнє

3. Гібридна модель

Гібридний підхід - це поєднання методологій Waterfall і Agile. Йому притаманне все найкраще, що є в цих методиках. Це гнучкий і при цьому добре структурований метод, який можна використовувати для різних проектів.

Гібридна методологія приділяє особливу увагу первинного збору та аналізу вимог, у чому вона і схожа на Waterfall. На наступному етапі вона характеризується гнучкістю, притаманною підходу Agile, і швидким внесенням змін.

Поєднуючи властивості Waterfall і Agile, гібридна методологія, яку іноді називають «структурованим Agile», дозволяє скористатися перевагами обох складових (рис. 6).

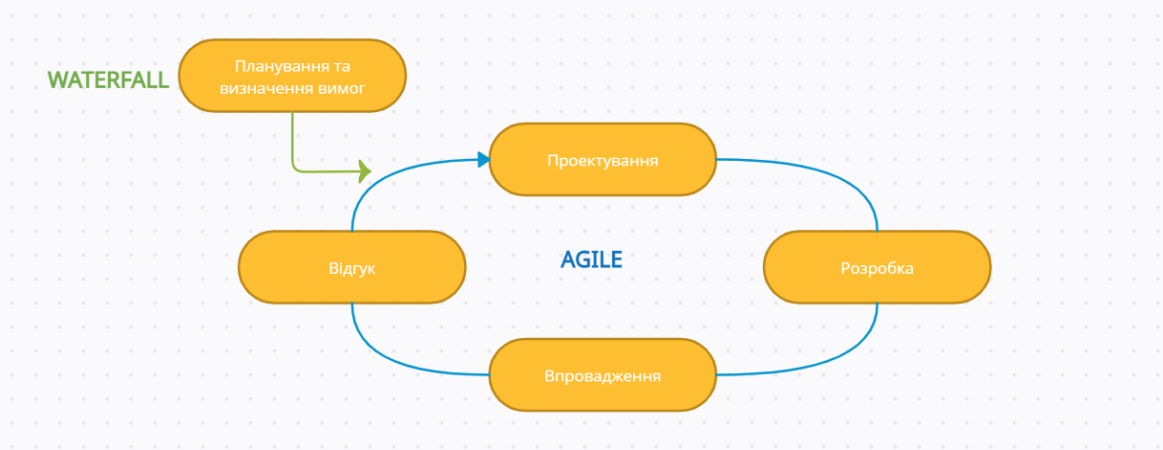


Рисунок 6 - Гібридна методологія

Переваги гібридної моделі:

- більша гнучкість

Якщо не брати до уваги етап планування, гібридній методології властива значно більша гнучкість, ніж методу Waterfall. Якщо вимоги не будуть часто змінюватися, в проект можна буде вносити зміни в міру необхідності.

- Більша структурованість

Запозичивши етап первісного планування з Waterfall, гібридна методологія вирішує одну з основних проблем підходу Agile - недостатню організованість і

відсутність плану. Таким чином, ця методологія поєднує в собі найкраще від цих двох підходів.

Недоліки гібридної моделі:

- необхідність компромісів

Оскільки доведеться підтримувати баланс між двома абсолютно протилежними підходами, потрібно буде шукати компроміси в області вимог і гнучкості.

- Поєднання найкращого від обох підходів

Методологія, що поєднує в собі все найкраще від двох підходів, позбавляє гнучкості Agile і стабільності Waterfall. Будь-які зміни, які будуть внеситись, повинні будуть відповідати бюджету та плану, позначених заздалегідь.

Типи проектів яким підійде гібридний підхід:

- гібридна методологія найбільше підійде для проектів з розмитими вимогами, в яких важливі і планування, і гнучкість.
- проекти середнього обсягу з високою складністю і фіксованим бюджетом. Швидше за все, існує певне уявлення про кінцеву мету, але при цьому можливі експерименти. З зацікавленими сторонами знадобиться тісний контакт, особливо після етапу планування.

4. Scrum

Scrum - це не повнофункціональна методологія управління проектами. Це скоріше підхід до методології Agile з акцентом на командах проекту, спринтах і щоденних зборах.

Незважаючи на те що Scrum запозичує принципи і процеси з Agile, до цього підходу властиві свої методи і тактики управління проектами.

В рамках підходу Scrum в центрі проекту - команда. Найчастіше менеджера проекту немає. Тому передбачається, що команда характеризується самоорганізацією і самоврядуванням. Саме тому такий підхід ідеально підійде для досвідчених мотивованих команд, але навряд чи підійде всім іншим (рис. 7).

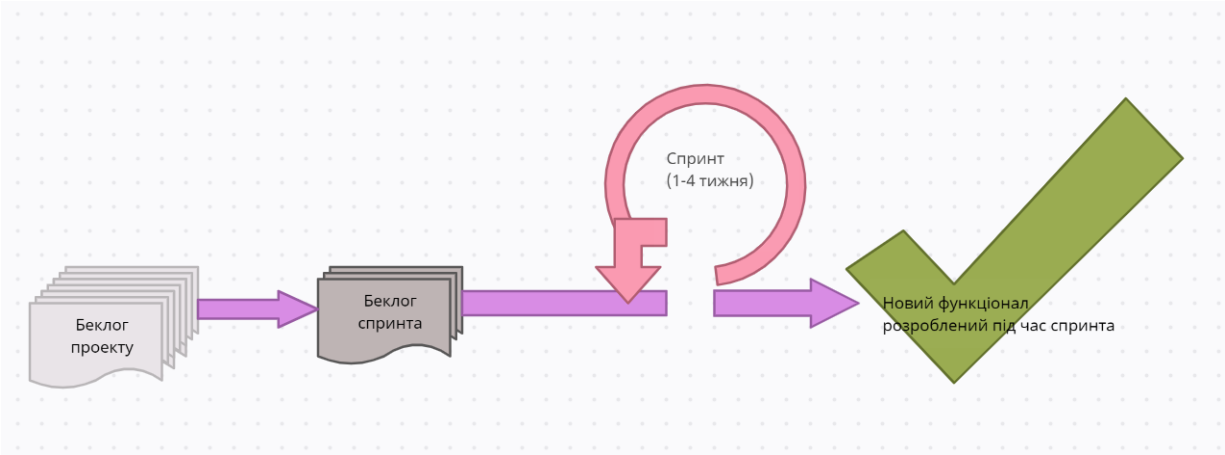


Рисунок 7 - Методологія Scrum

Переваги Scrum:

- спринти

У підході Scrum акцент робиться на тижневі спринти, або відрізки часу. Так, команда проекту ділить список кінцевих цілей на невеликі завдання, а потім працює над ними протягом одного-чотирьох періодів з щоденними зборами. Завдяки такому підходу простіше справлятися з великими складними проектами.

- Динамічність

Завдяки розбивці роботи на тижневі періоди з щоденними зборами, розробка і внесення змін відбуваються досить динамічно.

- Командна робота

Оскільки мається на увазі самоорганізація команди проекту, учасники чіткіше розуміють і знають проект. А ще лідери проекту можуть самостійно розставляти пріоритети відповідно до своїх знань і можливостей.

- Крім перерахованих, цій методології властиві всі переваги Agile: швидке внесення змін і регулярний зворотний зв'язок із зацікавленими сторонами.

Недоліки Scrum:

- неконтрольоване розширення масштабів

Оскільки дата завершення проекту не встановлена і відсутній менеджер проекту, який займався б плануванням і бюджетом, Scrum може стати причиною неконтрольованого розширення масштабів проекту.

- Підвищений ризик

Оскільки команда проекту займається самоорганізацією, збільшується ризик провалу, якщо команда недисциплінована і немотивована. Якщо у команди недостатньо досвіду, робота в рамках Scrum з великою ймовірністю закінчиться невдачею.

- Недостатня гнучкість

Акцент на команді проекту означає, що звільнення будь-якого ресурсу матиме значний вплив на результат. Також цей підхід недостатньо гнучкий для великих команд.

Проекти яким існує сенс застосовувати Scrum:

- методологія Scrum найкраще підходить досвідченим, дисциплінованим і мотивованим командам, які вміють розставляти свої пріоритети і мають чітке уявлення про вимоги проекту
- використовуйте на проекті Scrum, якщо розробляєте складне програмне забезпечення із досвідченою командою

5. Метод критичного шляху (Critical Path Method, CPM)

Метод критичного шляху - це покрокова система для управління проектами та планування процесів в них. Він допомагає визначати критичні і некритичні завдання і попереджає проблеми з термінами.

Критичний шлях в галузі управління проектами - це ряд завдань, які необхідно виконати в чіткому порядку і за необхідну кількість часу.

Тимчасової резерв - це відрізок часу, на який планова активність може бути затримана або продовжена з її ранньої дати до дати закінчення.

Графічна схема проекту (діаграма) - це графічний опис логічних відносин завдань проекту.

В рамках методу критичного шляху ви класифікуєте всі дії, які необхідно виконати, щоб досягти мети проекту в рамках ієрархічної структури робіт (Work breakdown structure). Після цього ви визначаєте тривалість всіх завдань і залежності між ними. Таким чином, ви зрозумієте, які завдання можна виконувати одночасно, а які - до того, як почнуться інші завдання (рис. 8).

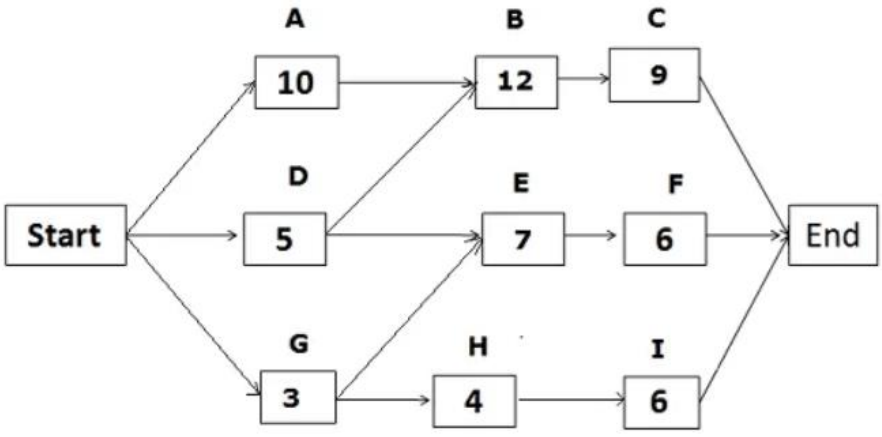


Рисунок 8 - Метод критичного шляху

Переваги методу критичного шляху:

- детальне планування

Завдяки акценту на тривалості діяльності і взаємозв'язках між ними є можливість краще спланувати завдання. Якщо для виконання завдання X спочатку потрібно завершити завдання Y, CPM допоможе визначити це заздалегідь і розпланувати все належним чином.

- Розстановка пріоритетів

Успіх методу критичного шляху залежить від визначення і планування критично важливих завдань і завдань другорядного значення. Визначивши завдання, ви зможете оптимально розподілити ресурси.

Недоліки методу критичного шляху:

- для планування необхідний досвід

Будь-який досвідчений менеджер з управління проектами скаже, що на все завжди йде більше часу, ніж планувалося. Якщо у вас немає реального досвіду планування, ви напевно неправильно розрахуєте час на виконання завдань.

- Відсутність гнучкості

Як і в Waterfall, перший етап роботи тут громіздкий. Необхідно все розпланувати з самого початку. Якщо з'являться зміни, весь план буде вже не важливий. Саме тому ця методологія не підходить для проектів з мінливими вимогами.

Для яких проектів найкраще підійде метод критичного шляху:

- метод критичного шляху найкраще підходить проектам, в яких є взаємозалежні частини. Якщо необхідно виконати завдання одночасно або необхідно завершити одну задачу перед тим, як перейти до іншої, ця методологія управління проектами підходить.
- CPM добре підходить для складних проектів, в рамках яких часто повторюються завдання і дії, наприклад, для промислових проектів. Для динамічних проектів, наприклад, творчих, вона підходить в меншій мірі.

4. Kanban

Kanban - це модель, яка допомагає візуалізувати і контролювати роботу. Її мета - наочно відслідковувати роботу за допомогою дошки та карток із завданнями. Дошка зазвичай ділиться на три основні колонки:

- to do («зробити»)
- in progress («в роботі»)
- done («готово»)

Але може бути застосовано і більше колонок залежно від потреб конкретної команди (наприклад, тестування, перевірка коду і так далі). Картки зазвичай переміщаються у відповідну секцію в залежності від прогресу (рис. 9).

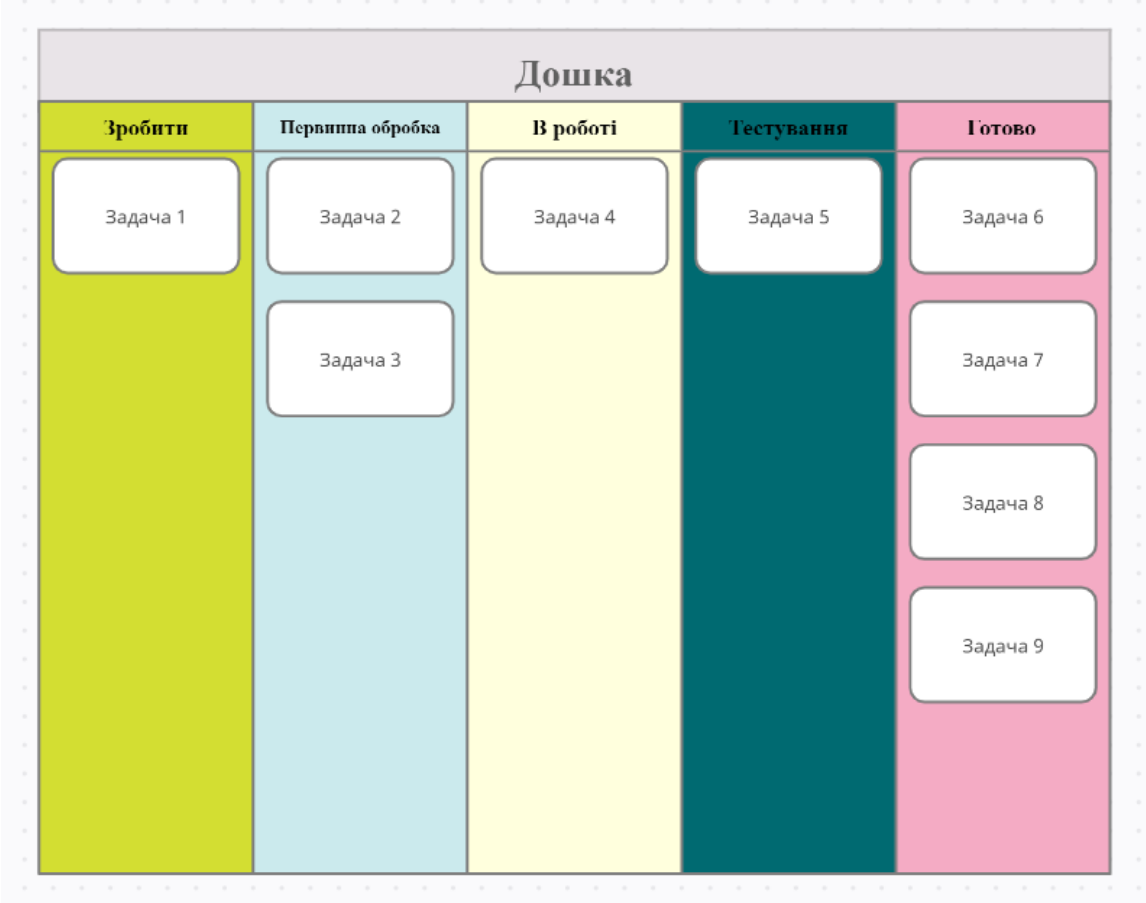


Рисунок 9 - Методологія Kanban

Переваги Kanban:

- як і Scrum добре підходить для згуртованих команд з хорошою комунікацією. Але на відміну від Scrum, в Kanban немає встановлених чітких дедлайнів, що добре підходить для мотивованих і досвідчених команд.
- точний розрахунок навантаження на команду
- правильна розстановка обмежень
- концентрація на постійне поліпшення
- все вище перераховане в поєднанні із гнучкістю

Недоліки Kanban:

- дана методологія погано працює з великими командами (більше 5 осіб)
- Kanban погано працює з крос-функціональними командами. Важко поєднати тестування і розробку в одній команді. Більш вдалою думкою є розбити процес на «станцію» розробки і «станцію» тестування з окремими керівниками і списками задач
- Kanban не призначений для довгострокового планування

РОЗДІЛ 2 ОСОБЛИВОСТІ СИСТЕМ УПРАВЛІННЯ ЯКІСТЮ

2.1 Особливості систем управління якістю

Система управління якістю визначається як формалізована система, яка документує процеси, процедури та відповідальність за досягнення політики та цілей у галузі якості. Вона допомагає координувати та керувати діяльність організації з метою задоволення вимог споживачів і регуляторних вимог та постійно підвищувати її результативність та ефективність.

ISO 9001: 2015, міжнародний стандарт, що визначає вимоги до систем управління якістю, вважається найвидатнішим підходом для систем управління якістю. Є найбільш визнаним та впровадженим стандартом системи управління якістю у світі. ISO 9001: 2015 визначає вимоги, які організації можуть використовувати для розробки власних програм.

Інші стандарти, що стосуються систем управління якістю, включають решту серій ISO 9000 (включаючи ISO 9000 та ISO 9004), серії ISO 14000 (системи екологічного менеджменту), ISO 13485 (системи управління якістю для медичних виробів), ISO 19011 (управління аудитом системи) та ISO / TS 16949 (системи управління якістю для автомобілів).

Впровадження системи управління якістю впливає на всі аспекти діяльності організації. Переваги задокументованої системи управління якістю включають:

- виконання вимог замовника, що допомагає вселити довіру до організації, що, в свою чергу, призводить до збільшення кількості клієнтів, збільшення продажів
- виконання вимог організації, що забезпечує дотримання нормативних актів та надання продуктів та послуг більш економно та зі збереженням ресурсів, створюючи простір для розширення, зростання та прибутку
- визначення, вдосконалення та контроль процесів
- запобігання помилок

- зниження витрат
- збільшення продуктивності праці
- покращення внутрішнього зв'язку між командами [8]

Перш ніж створювати систему управління якістю, проект повинен визначити список багатофункціональних процесів. На дизайн системи впливають цілі та потреби проекту. Основними етапами впровадження системи управління якістю є такі:

- проектування
- побудова системи
- контроль
- аналіз результатів
- внесення покращень

Проектування та побудова

На цих етапах розробляється структура системи контролю якості, її процеси та функції. Потрібно впевнитись, що система включає в себе всі потрібні функції для вирішення конкретних задач проекту.

Контроль та аналіз результатів

На даному етапі перевіряється коректність роботи створеної системи управління якістю та аналіз результатів виконання.

Внесення покращень

Після попереднього етапу визначено основні недоліки та дефекти програми. На етапі покращення є можливість виправити їх та додати новий функціонал для більшої зручності використання. [6][7]

2.2 Огляд відомих програмних систем управління якістю та порівняння їх функціональності із власноруч розробленою системою

На сьогоднішній день сучасний ринок програмних систем управління якістю досить різноманітний. В рамках цієї кваліфікаційної роботи було розглянуто 3 найбільш популярні системи та проаналізовано їх особливості та недоліки.

1. Testrail

Це програмне забезпечення зручне як для команд QA, так і для розробки. План тестування можна вибудувати як за сценарієм гнучкою методології, так і для більш традиційного підходу. Інструмент дозволяє отримати уявлення про хід тестування в реальному часі. Можливо будувати звіти по необхідних метриках. Можливо налаштувати інтеграцію з JIRA, Jenkins.

Мінуси Testrail:

1. Не безкоштовно (рис. 10). Для інтеграції з JIRA потрібно додатково платити за її використання, яка в свою чергу має досить високу вартість (рис. 11).

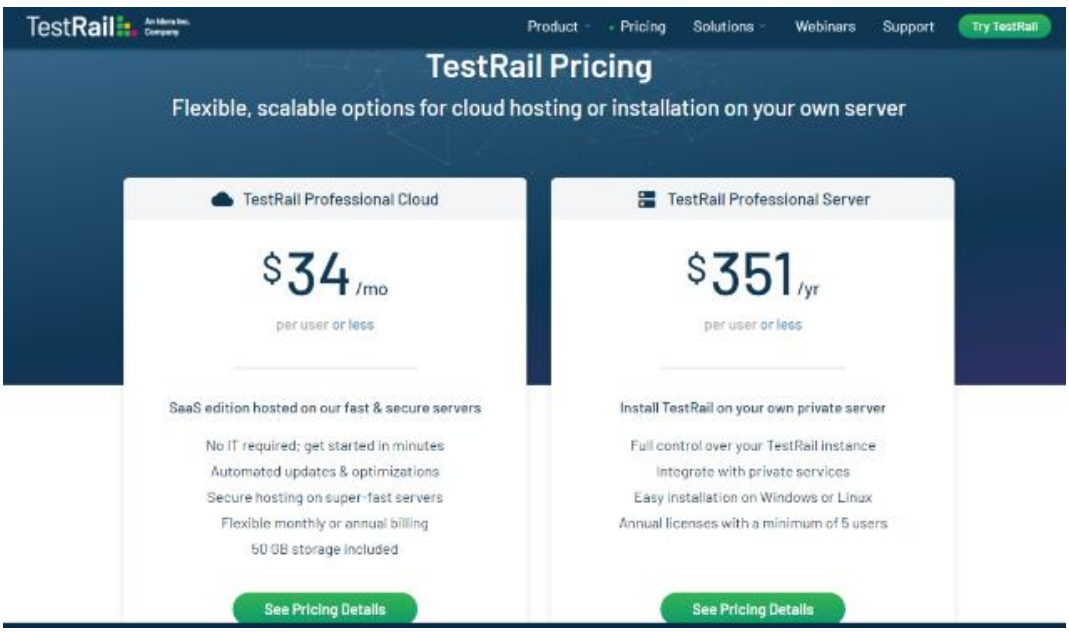


Рисунок 10 - Вартість послуг Testrail

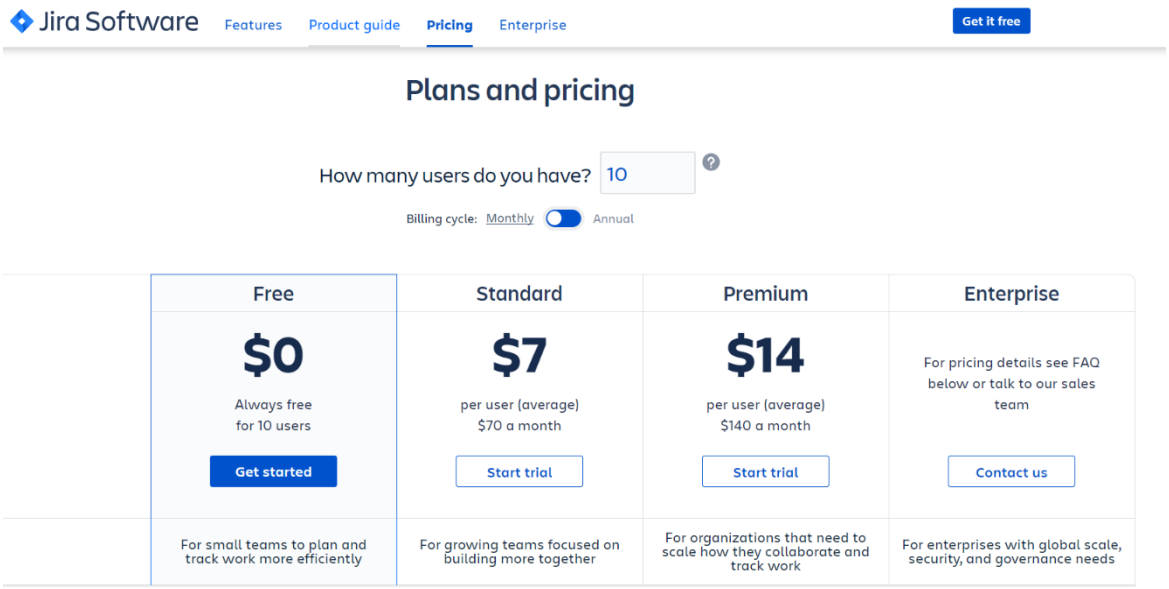


Рисунок 11 - Вартість послуг JIRA

2. Велика кількість вбудованих функцій, які зазвичай не використовуються
3. Суворе обмеження на кількість символів для введення заголовку тест кейсу (200 символів)
4. При великій кількості даних часто можна зіштовхнутись з проблемами продуктивності

Zephyr

Zephyr - це плагін для JIRA, який інтегрує тестування в проектний цикл, дозволяючи відстежувати якість програмного забезпечення та приймати рішення в стилі go\no-go. У нових версіях була підтримана робота з автоматизованими тестами.

Мінуси Zephyr

1. Не безкоштовно (рис. 12). Якщо інтегрувати з JIRA, то як і для TestRail потрібен додатковий бюджет на користування послугами JIRA.

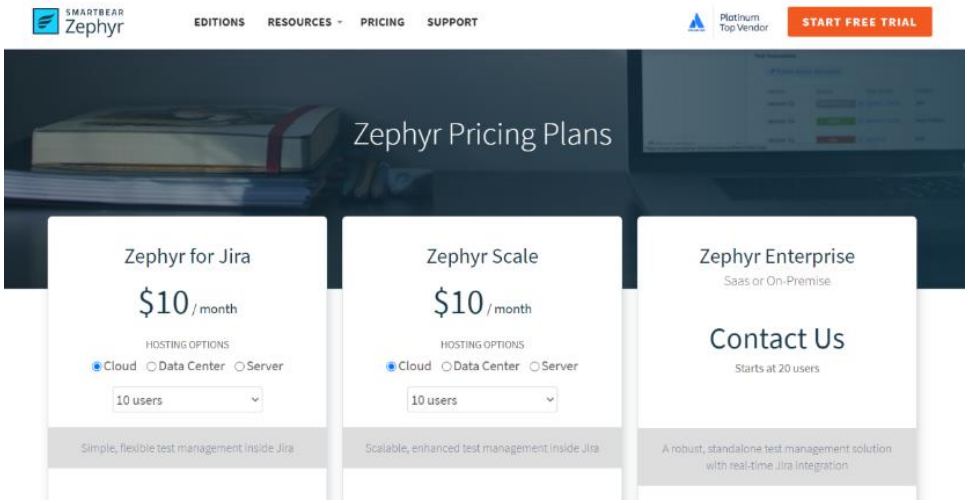


Рисунок 12 - Ціна послуг Zephyr

2. Згідно відгуків користувачів - не зручний інтерфейс, потрібен час для звикання
3. Інтеграція тільки з JIRA

TestLink

TestLink - це інструмент управління тестами, який використовується для управління проектами, відстеження помилок і управління тестами. Він синхронізує специфікацію вимог і тестову специфікацію.

Мінуси TestLink

1. Застарілий інтерфейс. Для зручного налаштування потрібно витратити багато часу.
2. Так як використовується підхід написання в текстових HTML вікнах, не працює стандартна орфографія браузера
3. Для створення баг репортів потрібна інтеграція з системою відслідковування дефектів, що може бути платною.

Далі наведено порівняльну таблицю Testrail, Zephyr, TestLink та власноруч розробленої системи (рис. 13).

Рисунок 13 - Порівняльна таблиця відомих програмних систем із власною

РОЗДІЛ 3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ УПРАВЛІННЯ ЯКІСТЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Етапи створення додатку

3.1.1 Визначення цілей та задач проекту

Першою з’явилася ідея. Так як на сьогоднішній день є дуже актуальним програмне забезпечення для розробки якісних програмних продуктів, а також через те, що я цікавлюсь тематикою контролю якості, ідея виникла досить швидко. Після цього був розпочатий перший етап розробки, а саме визначення мети та задачі проекту.

Мета роботи: розробка програмної системи обліку, контролю та управління якістю програмного забезпечення.

Задачами проекту були:

- аналіз існуючих програмних рішень та виявлення їх недоліків
- проектування власної програмної системи
- розробка додатку з урахуванням виявлених недоліків
- додавання унікальної функції створення дефекту під час проходження тест кейсів

3.1.2 Проектування основних функцій

На стадії проектування програмної системи було розроблено 11 UML діаграм та 2 DFD. Серед яких:

- діаграма варіантів використання (Use case)
- діаграма послідовності (Sequence)
- діаграма потоків даних (DFD)
- діаграма діяльності (Activity)
- діаграма класів (Class)

Метою створення цих діаграм була наочна демонстрація можливостей програмної системи, її розгляд з різних ракурсів та швидке ознайомлення з функціями.

В усіх створених діаграмах під поняттям User мається на увазі тестувальник, також можливі й інші додаткові ролі користувачів (розробник, менеджер проекту), котрі можуть бути додані у подальших версіях додатку.

Програмна система представляє собою веб сторінку із трьома вкладками: Test Cases, Bugs, Dashboard, що дозволяє працювати з тест кейсами, дефектами та статистикою відповідно. Для створення діаграм використовувались наступні веб ресурси:

- creately
- online.visual-paradigm

Діаграма варіантів використання

Діаграма варіантів використання в UML - діаграма, що відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні.

Основне призначення діаграми - опис функціональності і поведінки, що дозволяє замовнику, кінцевому користувачеві і розробнику спільно обговорювати проектовану систему.

Діаграми варіантів використання, що була створена для програмної системи контролю якості описує можливі варіанти використання функціоналу з вкладки Test cases та частково з вкладки Dashboard. Складається з актора, що має назву користувач (User) та наступних прецедентів як: аутентифікація користувача, перегляд списку тест кейсів, зміна їх статусу та можливість видалення, а також додавання нового тест кейсу та перегляд статистики (рис. 14).

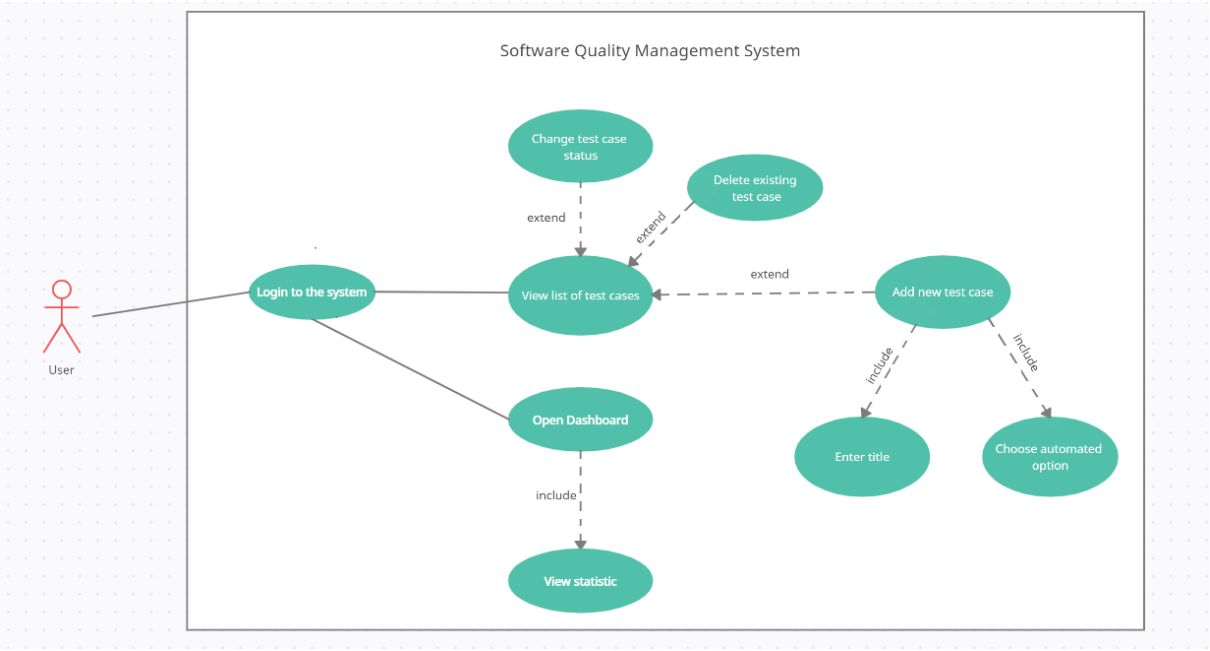


Рисунок 14 - Діаграма варіантів використання для вкладки Test Cases

Діаграма варіантів використання для вкладки Bugs складається з актора User та наступних прецедентів: аутентифікація користувача, перегляд списку дефектів, зміна їх статусу та можливість додати новий дефект, а також перегляд статистики (рис. 15).

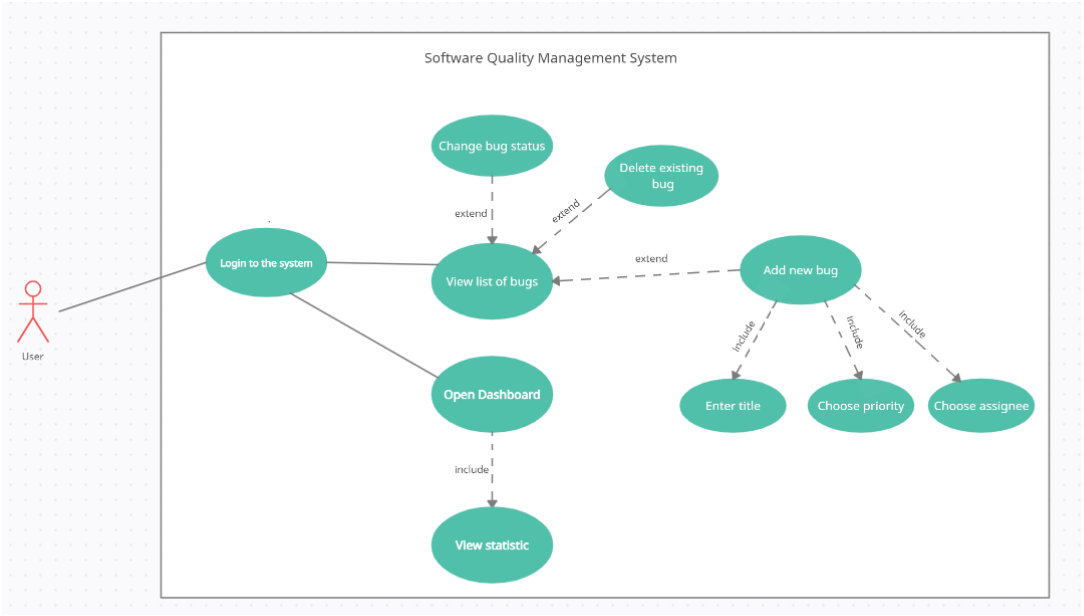


Рисунок 15 - Діаграма варіантів використання для вкладки Bugs

Для розуміння загальної картини варіантів використання було створено загальну діаграму (рис. 16).

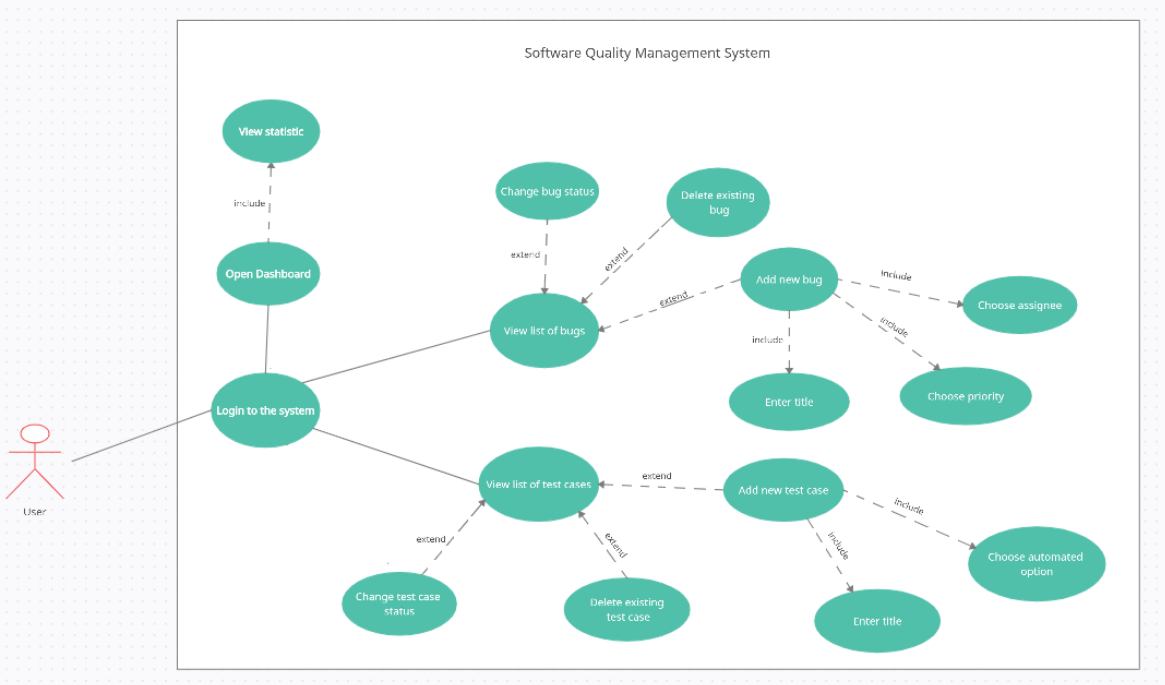


Рисунок 16 - Загальна діаграма варіантів використання для програмної системи, що розробляється

Діаграма послідовності

Діаграма послідовності - UML-діаграма, на якій для деякого набору об'єктів на єдиній тимчасовій осі зображений життєвий цикл об'єкта (створення-діяльність-знищення якоїсь сутності), а також взаємодія акторів (дійових осіб) інформаційної системи в рамках прецеденту.

Діаграми послідовностей використовується для уточнення діаграми варіантів використання, більш детального опису логіки сценаріїв використання.

Створено окрему діаграму послідовності для аутентифікації користувача, що складається з учасника User, класу розмежовувача Login form, класу контролеру Access manager та класу розмежовувача Homepage (рис. 17).

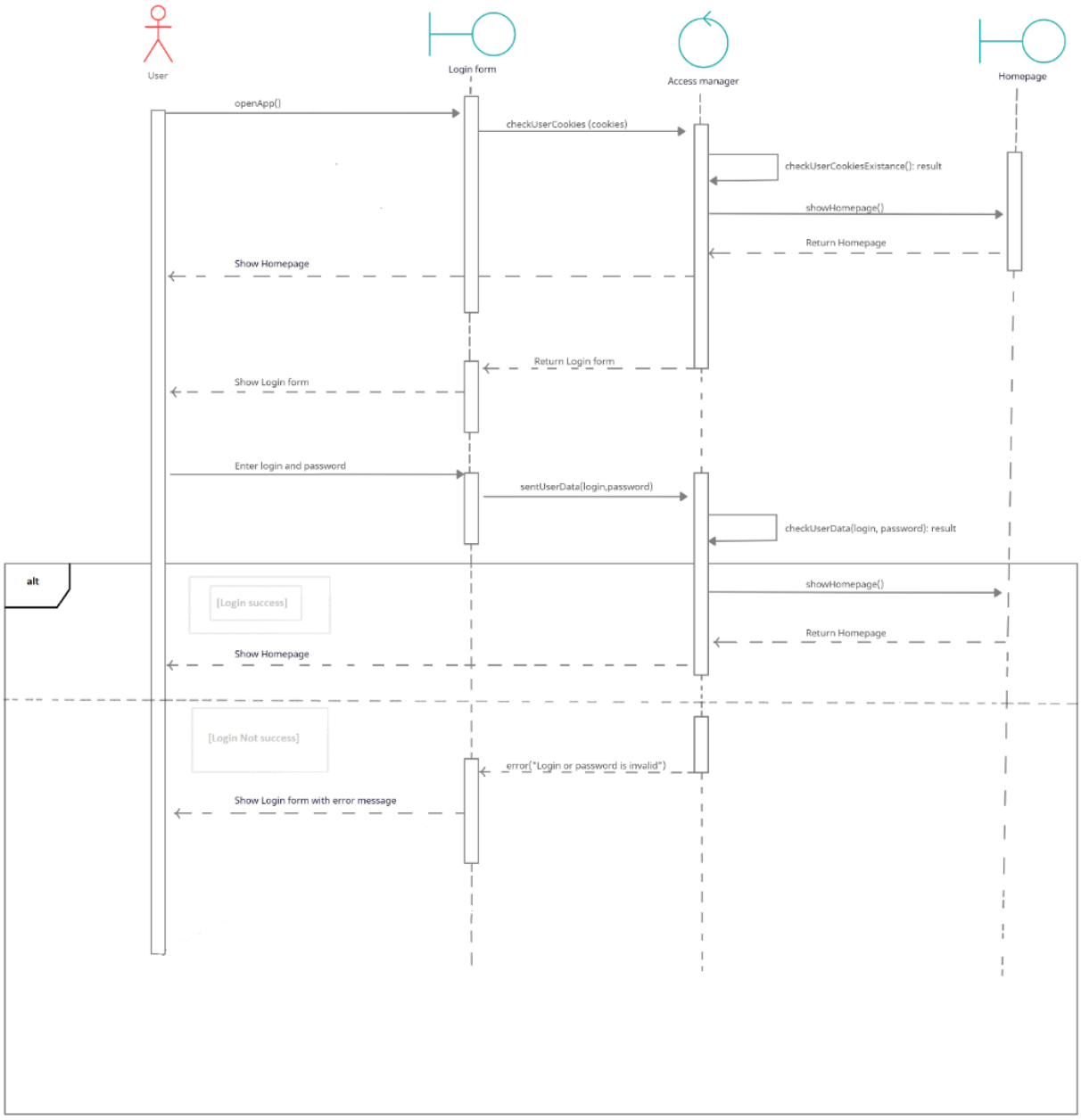


Рисунок 17 - Діаграма послідовності використання для аутентифікації користувача

Додатково створено окрему діаграму послідовності використання для сторінки функцій створення та видалення тест кейсів (рис. 18). В цій діаграмі використовуються інші позначення об'єктів для демонстрації різних варіантів синтаксису UML.

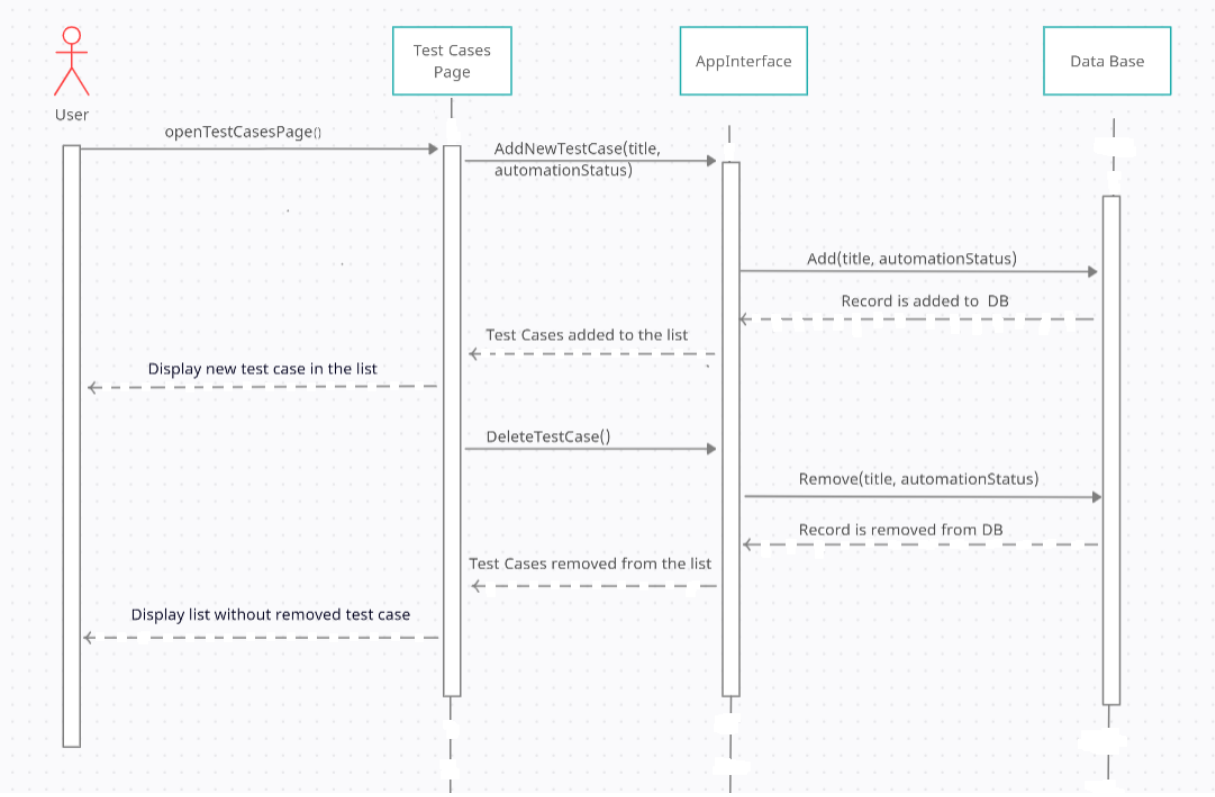


Рисунок 18 - Діаграма послідовності для функцій створення та видалення тест кейсів

Додатковою можливістю роботи з тест кейсами є функція зміни їх статусу на пройдено чи не пройдено (рис. 19).

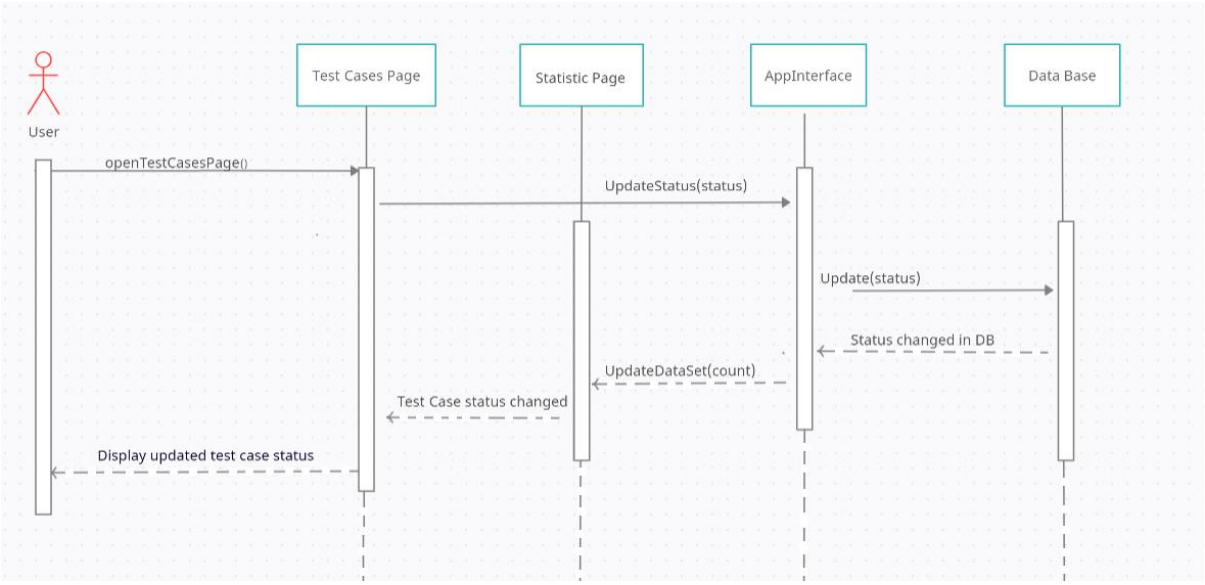


Рисунок 19 - Діаграма послідовності використання для функції зміни статусу тест кейсу

Аналогічно було створено діаграму послідовностей для додавання та видалення дефектів (рис. 20).

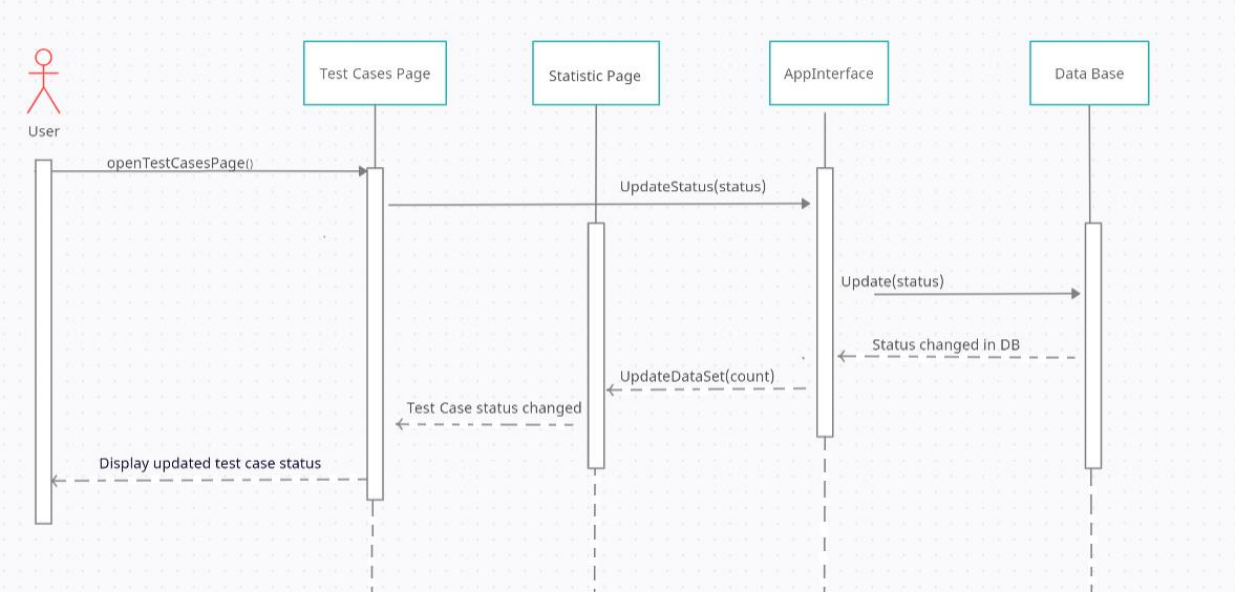


Рисунок 20 - Діаграма послідовностей для додавання та видалення дефектів

Також для демонстрації функції оновлення статусу дефекту створено діаграму послідовностей (рис. 21).

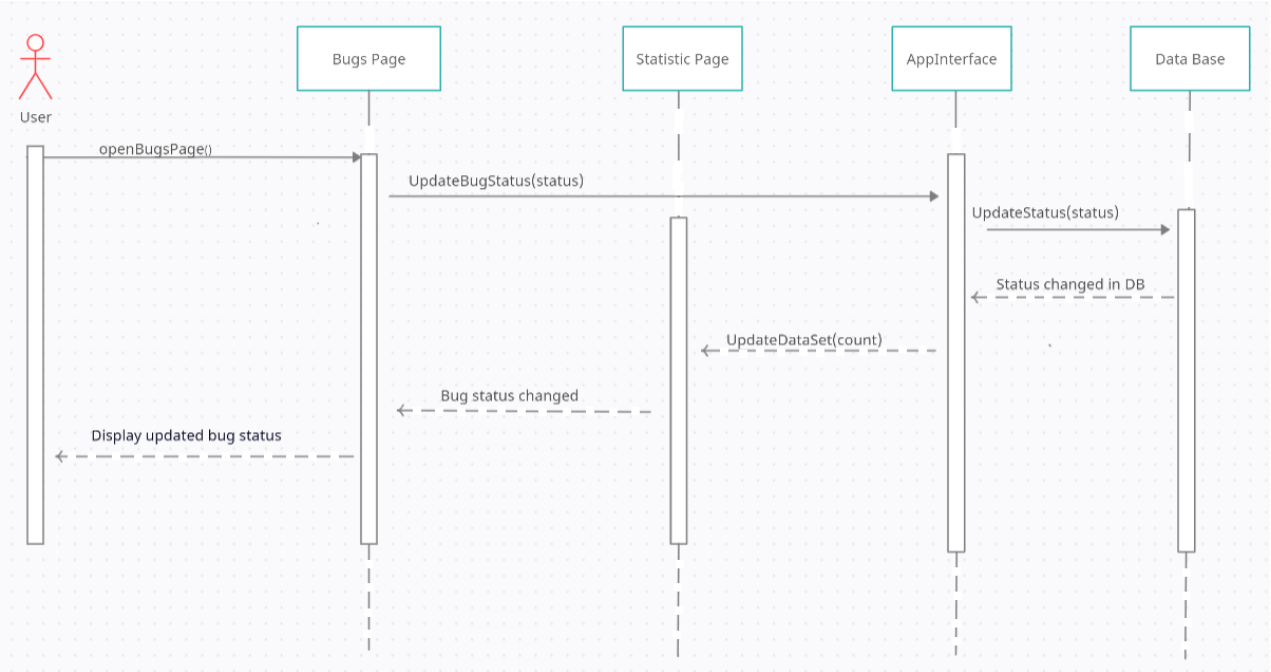


Рисунок 21 - Діаграма послідовностей для функції оновлення статусу дефекту

Діаграма потоків даних

DFD - загальноприйняте скорочення від англійської data flow diagrams - діаграми потоків даних. Так називається методологія графічного структурного аналізу, що описує зовнішні по відношенню до системи джерела і адресати даних, логічні функції, потоки даних і сховища даних, до яких здійснюється доступ. Діаграма потоків даних один з основних інструментів структурного аналізу і проектування інформаційних систем, що існували до широкого поширення UML.

Діаграма створена для наочно зображення перебігу інформації в межах процесів створення та видалення тест кейсів та дефектів.

Існує декілька варіантів систем нотацій DFD схем, що названі в честь їх засновників:

- Йордон Коуд
- Йордон і Де Марко
- Гейн і Сарсон

Основна різниця цих систем у тому, що методи Йордона-Коуді і Йордона-Де Марко для позначення процесів застосовують кола, а метод Гейне-Сарсона - прямокутні блоки з округленими кутами (які іноді називають «цукерками»).

У випадку розробки власної системи було використано варіант Гейне-Сарсона для побудови діаграм потоків даних як для тест кейсів, так і для дефектів (рис. 22) (рис. 23).

Відмінність DFD та UML

Діаграми DFD наочно показують переміщення даних по системі, тоді як UML - це мова моделювання, що застосовується в об'єктно-орієнтованій розробці програм з метою показати проект в деталях. Безперечно, діаграма DFD може стати чудовою відправною точкою для роботи, однак коли справа дійде саме до

створення системи, розробникам зручніше звернутися до діаграм UML і вже з їх допомогою добитися бажаного рівня деталізації.

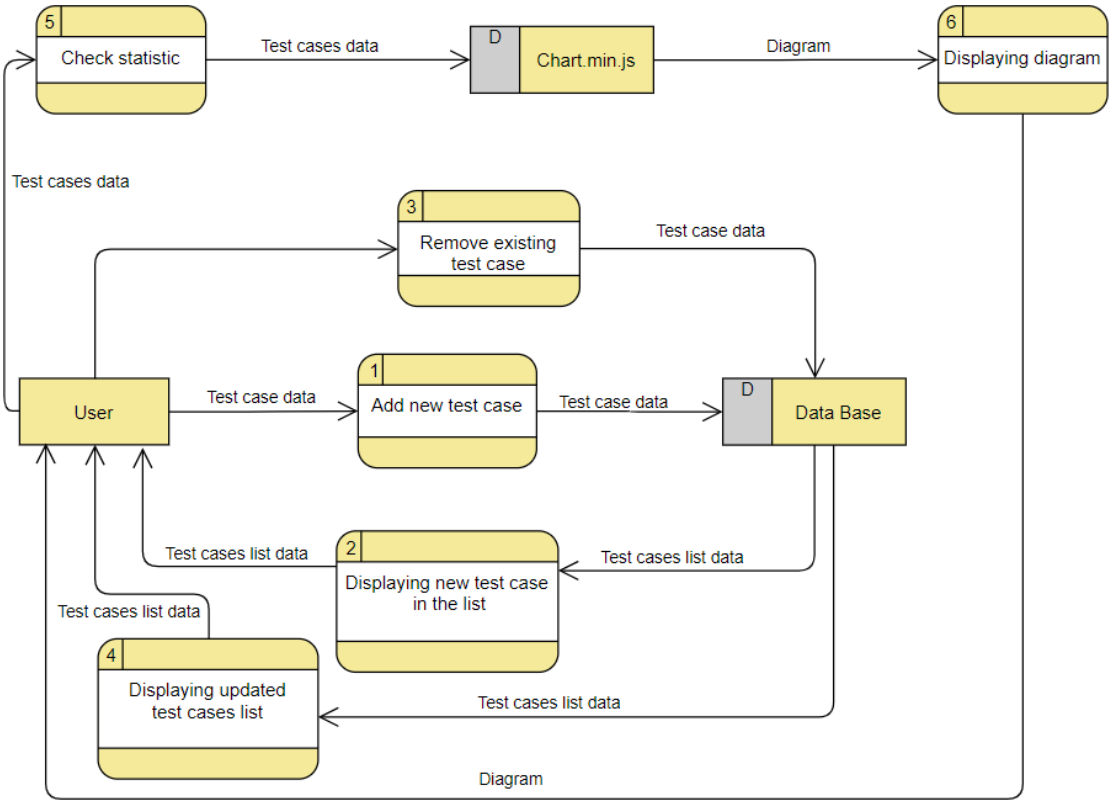


Рисунок 22 - Діаграма потоків даних для тест кейсів

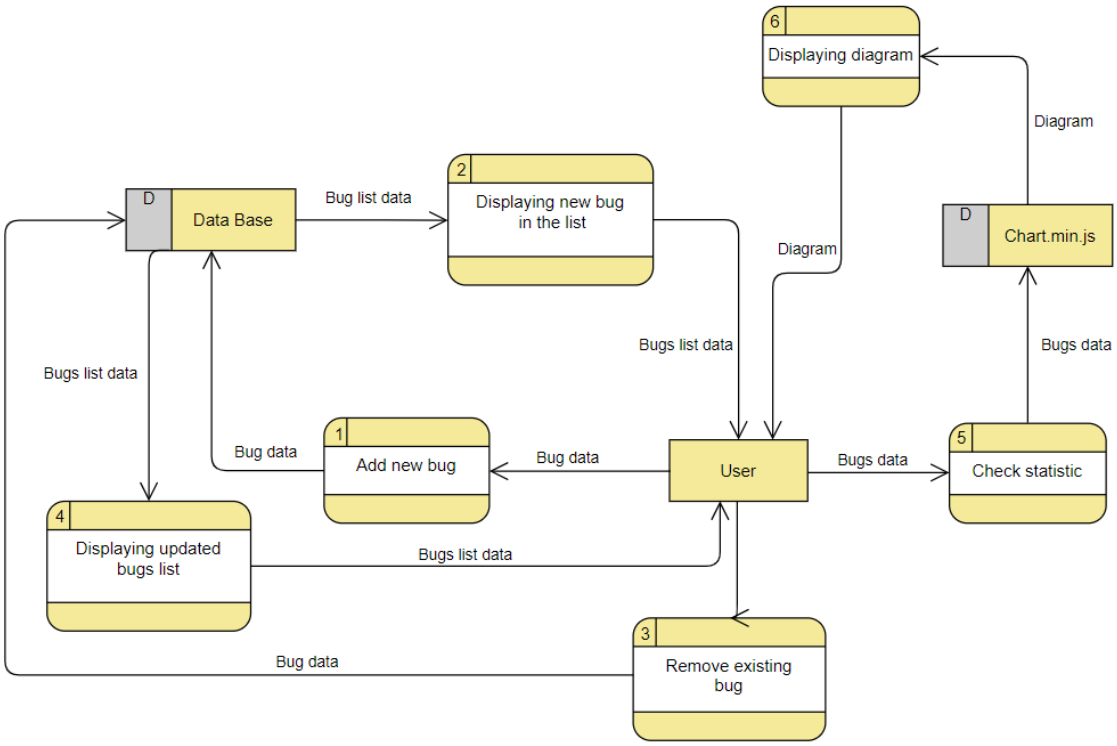


Рисунок 23 - Діаграма потоків даних для дефектів

Діаграма діяльності

Діаграма діяльності - це технологія, що дозволяє описувати логіку процедур, бізнес-процеси і потоки робіт. У багатьох випадках вони нагадують блок-схеми, але принципова різниця між діаграмами діяльності і нотацією блок-схем полягає в тому, що перші підтримують паралельні процеси.

Діаграма активностей дозволяє будь-кому, хто виконує цей процес, вибирати порядок дій. Іншими словами, діаграма тільки встановлює правила обов'язкової послідовності дій, яким користувач повинен слідувати. Це важливо для моделювання бізнес-процесів, оскільки ці процеси часто виконуються паралельно. Така діаграма також корисна при розробці паралельних алгоритмів, в яких незалежні потоки можуть виконувати роботу паралельно.

Для більшої наочності діаграму діяльності було розділено на дві: для тест кейсів і для дефектів, як і для попередніх видів діаграм. Діаграма діяльності для роботи з тест кейсами починається із початкового стану і переходить в об’єкт в

стані перегляду списку тест кейсів. Також використовуються такі елементи як розгалужувач, з'єднувач, логічні розгалуження та з'єднання (прийняття рішень), кінцевий стан (рис. 24).

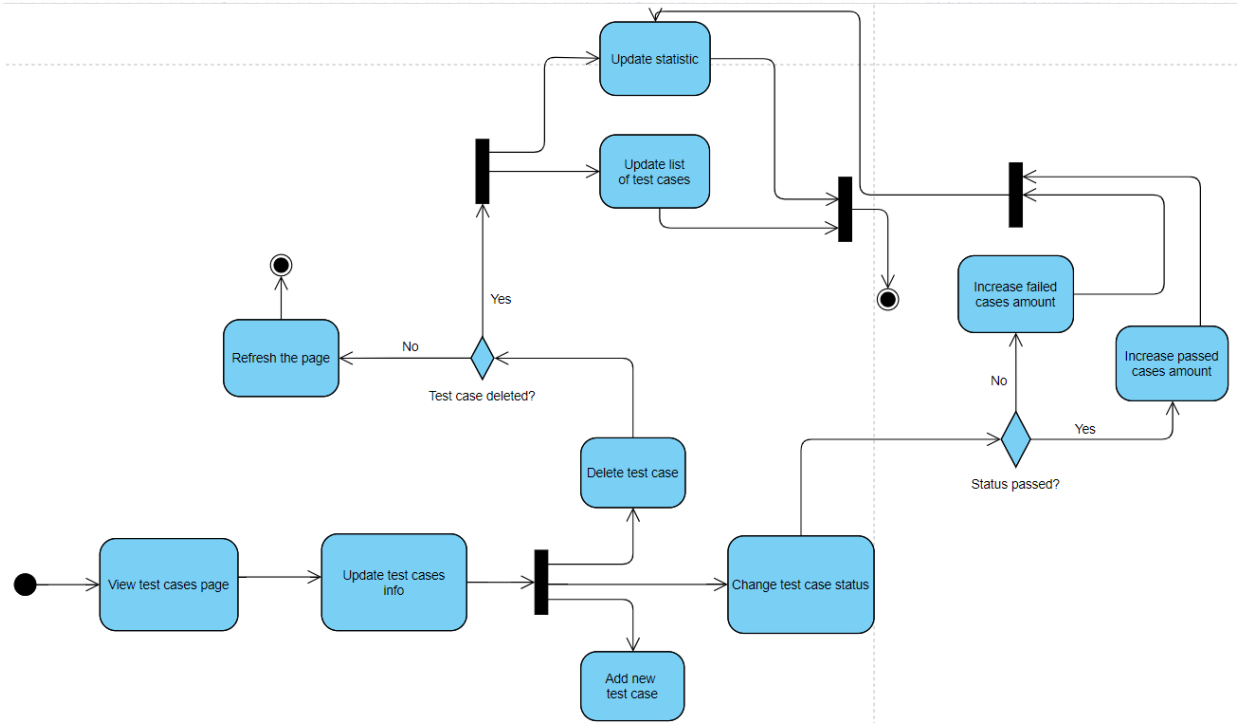


Рисунок 24 - Діаграма діяльності для роботи з тест кейсами

Діаграма діяльності для роботи з дефектами містить аналогічні елементи та має таку ж саму структуру (рис. 25).

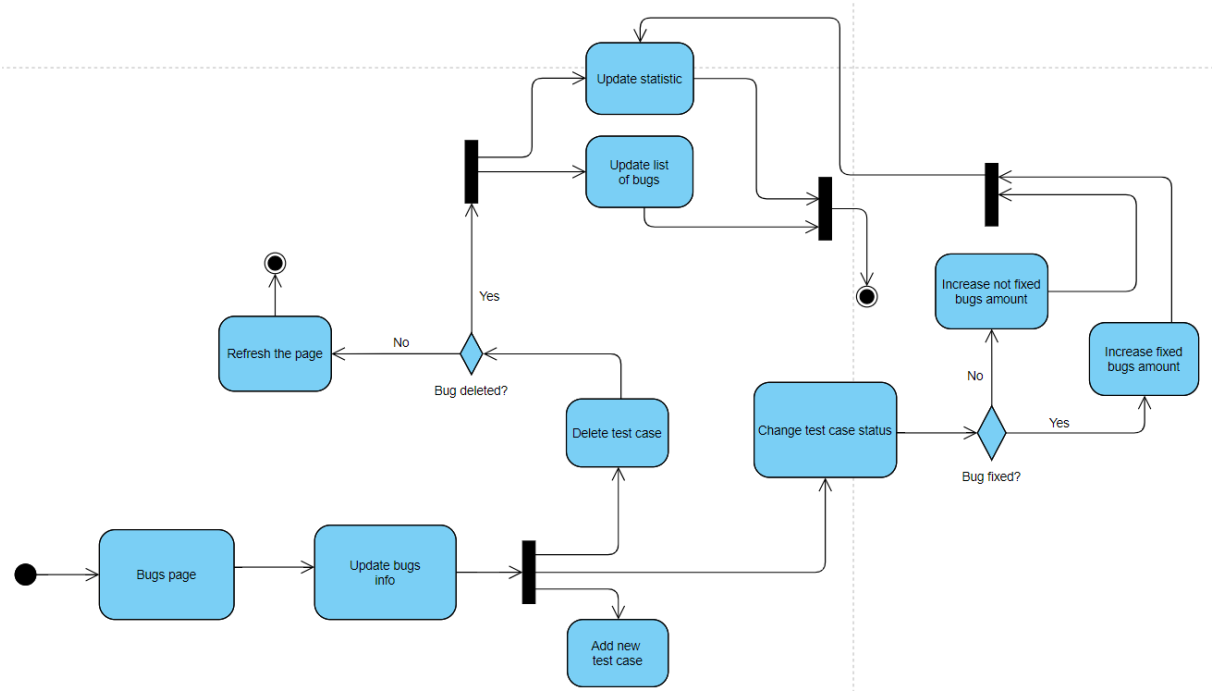


Рисунок 25 - Діаграма діяльності для роботи з дефектами

Діаграма класів

Діаграма класів визначає типи класів системи і різного роду статичні зв'язки, які існують між ними. На діаграмах класів зображуються також атрибути класів, операції класів та обмеження, які накладаються на зв'язки між класами. Вид і інтерпретація діаграми класів істотно залежить від точки зору (рівня абстракції): класи можуть представляти сутності предметної області (в процесі аналізу) або елементи програмної системи (в процесах проектування і реалізації).

В даній діаграмі використовується відношення композиції (1 до багатьох) (рис. 26).

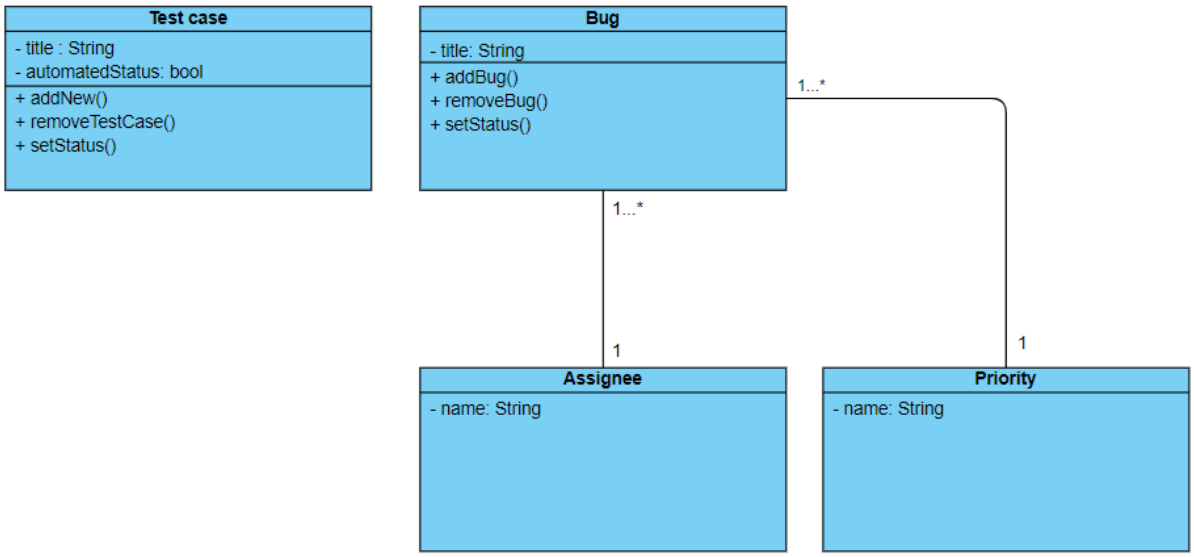


Рисунок 26 - Діаграма класів для програмної системи, що розроблюється

3.1.3 Вибір інструментів

Для створення додатку були використані наступні технології.

- Мова програмування: PHP
- IDE: Sublime Text
- Система управління базою даних: MySQL
- Для розгортки на веб-сервері пакет програм: XAMPP
- Веб-сервер: Apache
- Фреймворк для розробки інтерфейсу: Bootstrap

Програма написана мовою PHP так як ця мова чудово підходить для розробки веб додатків. Однією з причиною обрання саме цієї мови програмування була її універсальність. Так як додаток – це мінімально життєздатний продукт, що володіє мінімальними, але достатніми для перших потреб користувачів функціями (Minimal viable product, MVP), важливо було подбати про його перспективність у майбутньому. У цьому на допомогу якраз і приходить PHP, як мова, котра чудово поєднується з іншими мовами програмування (є можливість написати необхідні

доповнення мовою C), також PHP володіє достатньо великою кількістю доступних бібліотек та фреймворків, які надають широкі можливості у розробці.

Також до причин вибору PHP можна додати хорошу швидкість завантаження веб сторінки, що є важливим в перспективі для цього проекту, так як на сайт будуть завантажуватись великі обсяги даних (тисячі тест кейсів, сотні багів), а також широкі можливості для підключення баз даних.

СУБД було обрано MySQL. Перше за все, за простоту встановлення, налаштування і в цілому роботу з нею. Також було враховано те, що MySQL працює досить швидко завдяки внутрішньому кешуванню та її легкої переносимість.

Для написання користувацького інтерфейсу використано фреймворк Bootstrap, так як він безкоштовний, швидкий, а також містить велику кількість готових добре продуманих компонентів.

3.1.4 Розробка структури сайту

На цьому етапі було визначено, яка саме інформація буде міститись у додатку, створено наочну карту сайту, яка демонструє взаємозв'язки сторінок і їхні значущі функції (рис. 27).

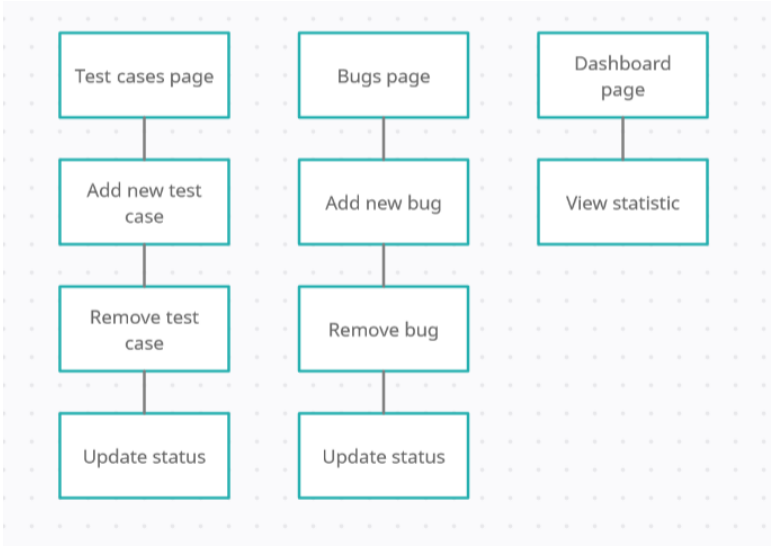


Рисунок 27 - Наочна карта сайту

А також XML карту сайту, котра буде потрібна для систем пошуку задля просування додатку. Так як додаток був створений на локальному веб-сервері, потрібна була допомога сторонніх додатків.

Було використано сучасну утиліту ngrok за допомогою котрої створено віддалений доступ до власного веб-ресурс, що був запущений на комп'ютері. Доступ організовується через створений завдяки ngrok безпечний тунель. Для роботи з ngrok було:

1. Встановлено утиліту на комп'ютер
2. Створено власний акаунт на сайті ngrok.com
3. Збережено свій унікальний токен за допомогою команди: ngrok.exe authtoken <my_token>
4. Створено віддалений доступ до власного додатку використавши команду ngrok. exe http 80, де 80 це номер порту локального веб-сервера (рис. 28).

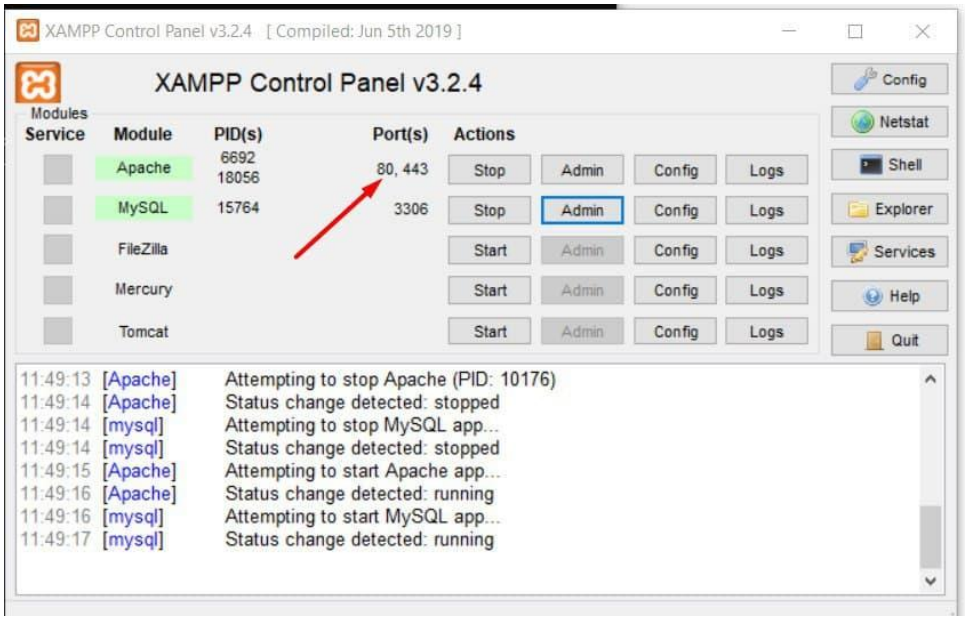


Рисунок 28 - Номер порту локального сервера

5. Таким чином доступ до веб-ресурсу був створений (рис. 29).

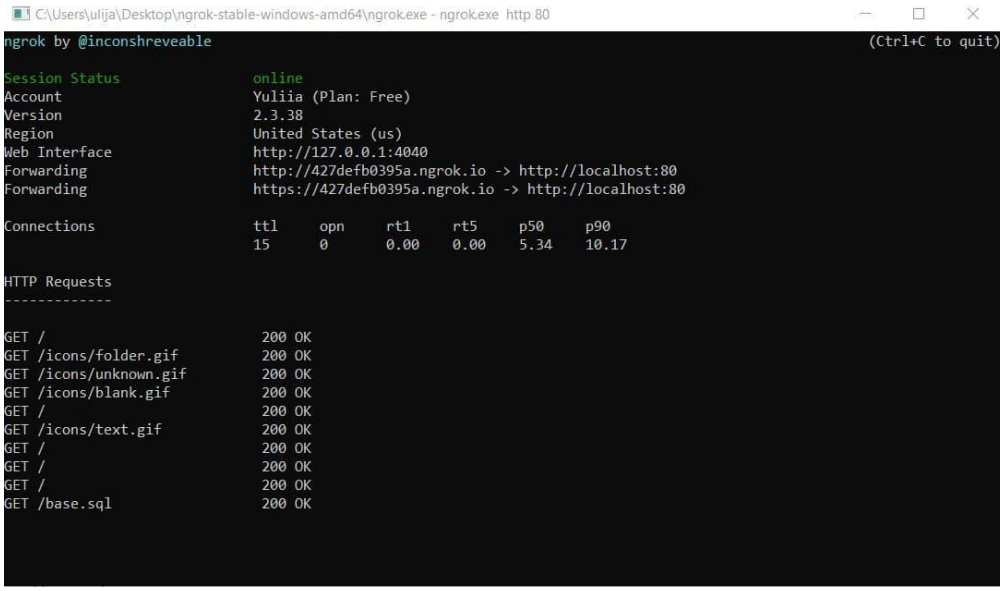


Рисунок 29 - Демонстрація створеного віддаленого доступу до локального додатку

Перейшовши по адресі http://127.0.0.1:4040 можна побачити сервісну сторінку із статистикою та логуванням запитів (рис. 30).

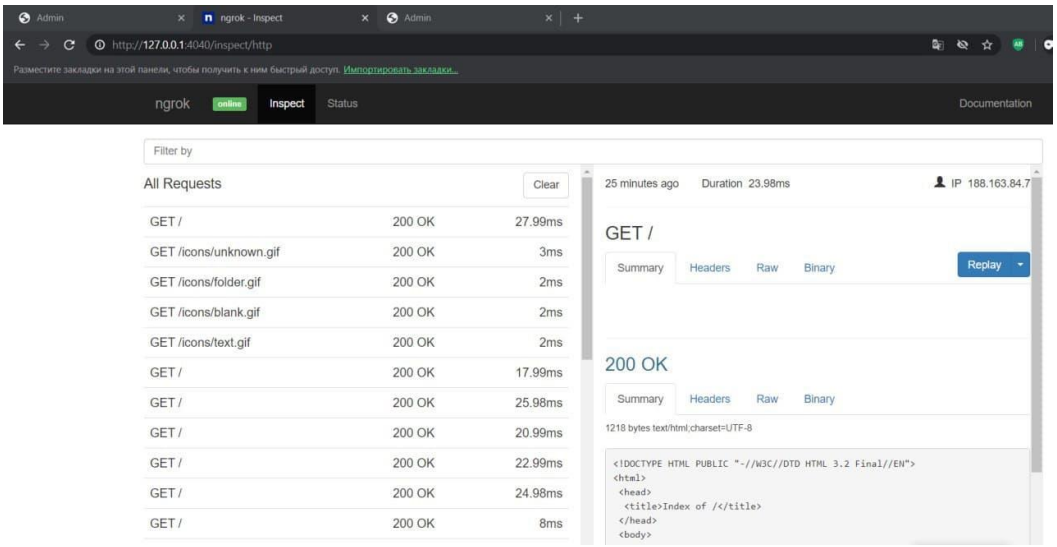


Рисунок 30 - Сервісна сторінка із статистикою та логом запитів

Якщо перейти по створеній адресі http://427defb0395a.ngrok.io , то можна побачити, що створена система тепер доступна за цією адресою (рис. 31).

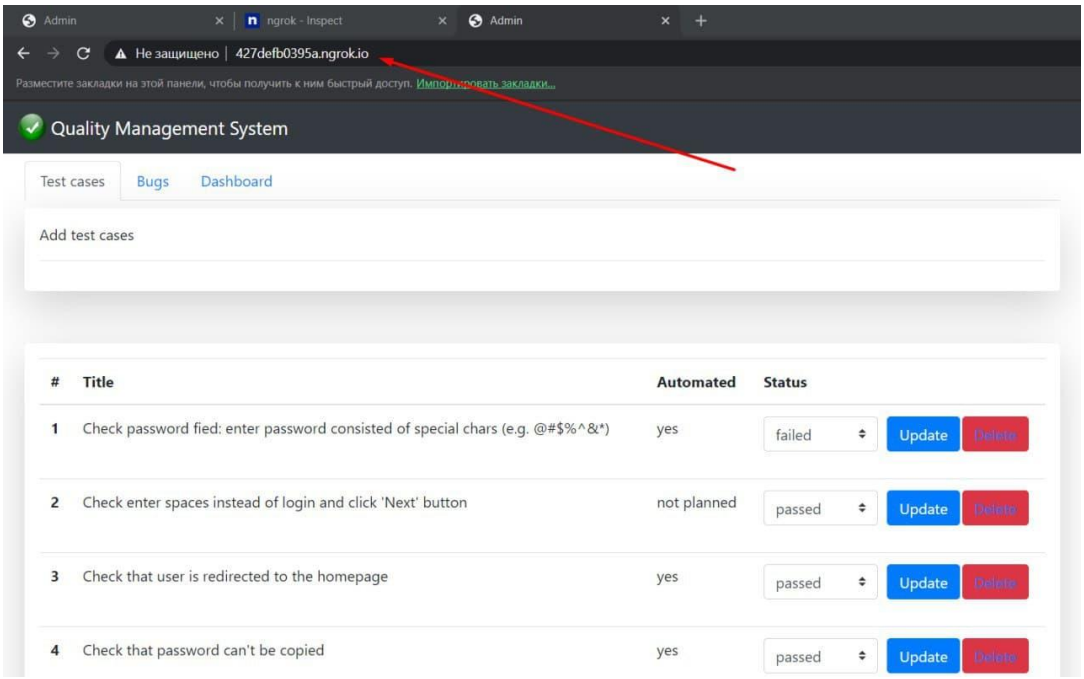


Рисунок 31 - Розроблювана система доступна за глобальною адресою

Таким чином тепер є можливість створити XML карту сайту використовуючи mysitemapgenerator (https://www.mysitemapgenerator.com/ru/). Після введення адреси програмної системи, було обрано XML Sitemap значення та натиснено кнопку «Перейти до створення» (рис. 32).

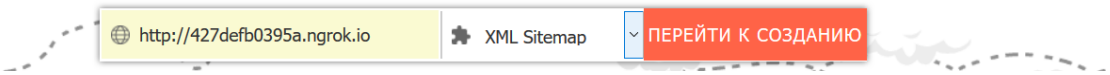


Рисунок 32 - Створення XML карти сайту

Отримано XML карту сайту системи контролю якості програмного забезпечення (рис. 33).

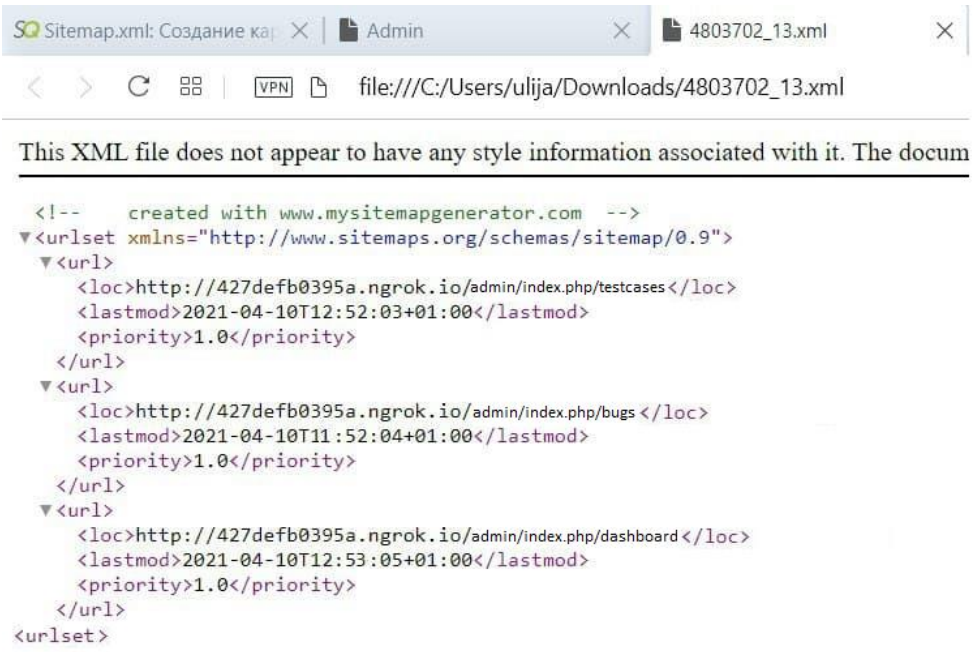


Рисунок 33 - XML карта сайту системи контролю якості програмного забезпечення

3.1.5 Проектування структури бази даних

База даних складається з двох таблиць: test_cases та bugs. Таблиця test_cases, у свою чергу, містить наступні стовпці (рис. 34):

- id
- title
- automated
- status

id – це унікальний ідентифікатор тест кейсу, первинний ключ типу integer, який автоматично збільшується з кожним наступним записом на одиницю.

title – це назва тест кейсу із типом varchar.

automated – це поле для відміток чи є конкретний тест кейс автоматизовано чи ні. Поле має тип integer. Його можливі значення: 0 – не автоматизовано, 1 – автоматизовано, 2 – не планується бути автоматизованим.

status – це можливий статус тест кейсу з типом int. Можливі значення: passed та failed. Я вирішила обрати саме цей тип, а не boolean, так як зазвичай статусів набагато більше і, а passed та failed – це достатній мінімум. При подальшому розвитку даної системи буде зручніше додавати нові статуси (наприклад, blocked, N/A і так далі) не роблячи змін в структурі бази даних.

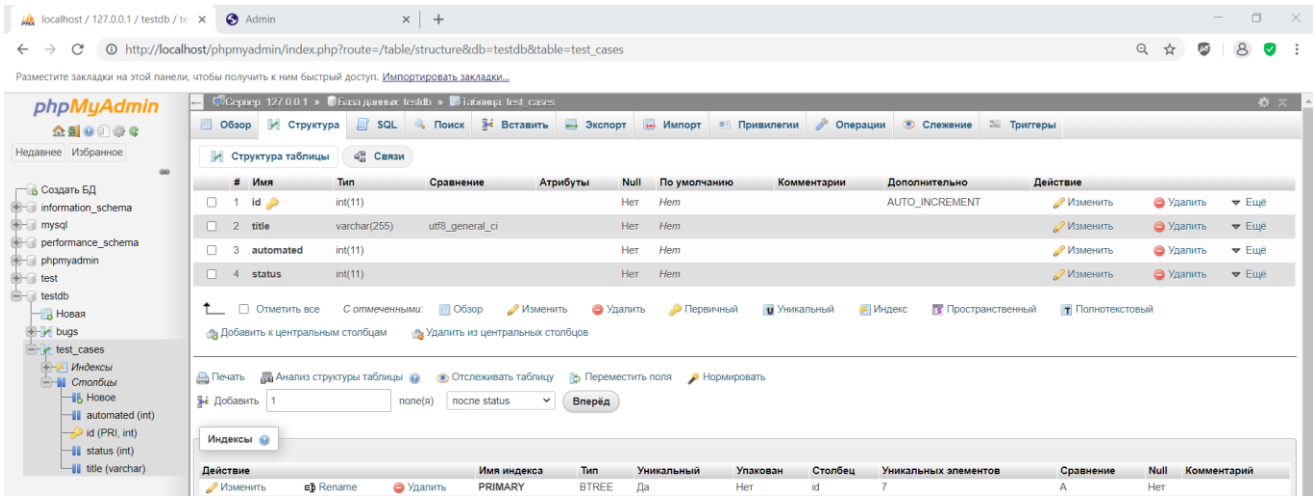


Рисунок 34 - Структура таблиці test_cases

Таблиця bugs містить наступні поля (рис. 35):

- id
- title
- priority
- assignee
- status

id – це унікальний ідентифікатор дефекту, перший ключ типу integer, який так само як і аналогічний ідентифікатор в таблиці test_cases, автоматично збільшується з кожним наступним записом на одиницю.

title – це назва дефекту із типом varchar

priority – це пріоритет конкретного дефекту із типом enum, що може приймати значення із списку допустимих значень, що були явно перераховані під час створення таблиці. В даному випадку – «Low», «Medium», «High».

assignee – це ім’я відповідального за виправлення дефекту з типом integer.

status – це статус виправлення конкретного дефекту з типом integer. Можливі значення: 0 – not fixed, 1 – fixed.

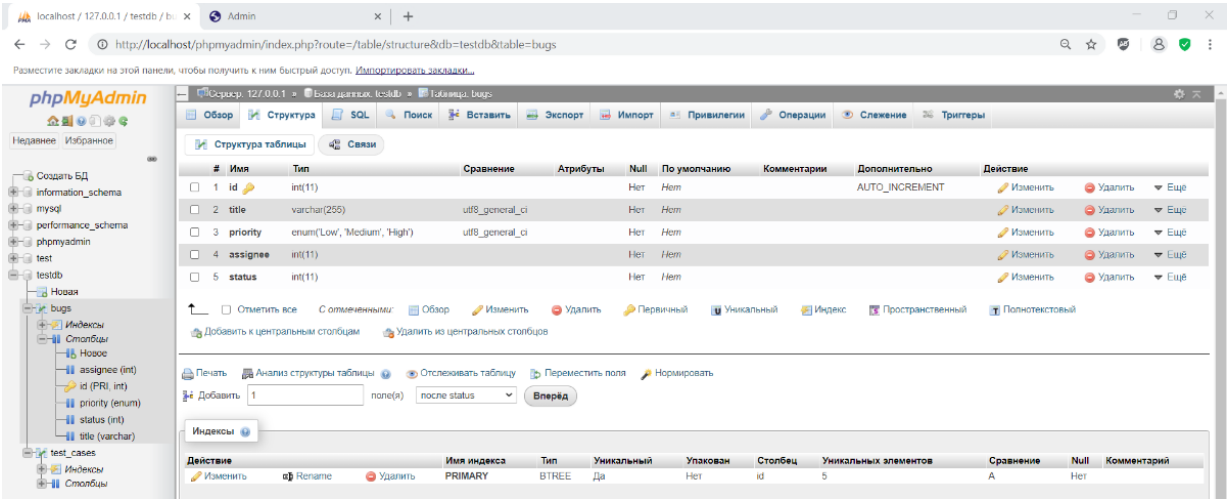


Рисунок 35 - Структура таблиці bugs

3.1.6 Написання backend частини

На цьому етапі велась розробка всіх спроектованих на попередніх кроках функцій. Спочатку я з'єднала створену базу даних mySQL та власний програмний додаток наступним фрагментом коду:

```
public static function connect ($type = 'mysql', $param = []) {

    {  $dsn = 'mysql:host='.(! empty($param['host']) ? $param['host'] :
'localhost').';'.(! empty($param['port']) ? 'port='.$param['port'].';' :
null).'dbname='.$param['base'].';charset='.(! empty($param['charset']) ?
$param['charset'] : 'utf8');  }

    if (isset($dsn)){

        try {

            self::$pdo = new \PDO($dsn, (! empty($param['user']) ? $param['user'] :
'root'), (! empty($param['pass']) ? $param['pass'] : '')
```

```
self::info('Successfully connected to the '.$type.' database', 2);
return self::$pdo;  } }  }
```

Далі для розмежування ролей було створено аутентифікацію користувача. Перевіряємо наявність cookies, якщо вони збережені, то користувач одразу потрапляє на сторінку Test Cases. Якщо cookies відсутні, користувач опиняється на сторінці авторизації:

```
if ($_COOKIE['auth']): // checking cooking existence
    header('Location: index.php');
    exit();~
endif;
$login = 'Admin';
$password = '123456';
if ($_POST['login'] == $login && $_POST['password'] == $password):
    setcookie('auth', 'ok', time() + 86400 * 7); // saving cookies for 7 days
    header('Location: index.php');
    exit();
endif;
```

Після цього реалізовувались безпосередні функції додатку. Далі наведено приклад однієї з функцій, а саме додавання нового тест кейсу:

```
include '../Base.php';
if (isset($_POST['title']) && isset($_POST['priority']) &&
isset($_POST['assignee'])) {  'title' => $_POST['title'],
    'priority' => $_POST['priority'],
```

```
        'assignee' => $_POST['assignee'] ]); }
    header('location: index.php');
    exit();
```

3.1.7 Написання frontend частини

Після завершення попереднього етапу, я розпочала роботу над frontend частиною. Далі буде наведено фрагмент коду, який демонструє частину функціоналу відображення статистики:

```
<div class="tab-pane fade" id="nav-dashboard" role="tabpanel" aria-labelledby="nav-dashboard-tab">
            <div class="row">
              <div class="col">
                <div class="col-12 shadow-lg p-3 mb-5 bg-white rounded mt-2">
                  <div class="mb-2">Statistic</div>
                  <hr>
                  <div class="row">
                    <div class="col">
                      Test Cases
                      <div id="canvas-holder" style="width:80%">
                        <canvas id="chart-area"></canvas>
                      </div>
                      <script>
                        var config = {
```

```
type: 'doughnut',
data: { datasets: [{  data: [
<?= Base::query('select count(id) from test_cases where status = 1') ?>,
<?= Base::query('select count(id) from test_cases where status = 0') ?>,],
backgroundColor: [
'rgb(255, 99, 132)',
'rgb(255, 159, 64)', ],
label: 'Dataset 1' }],
labels: [ 'Passed',
'Failed', ]},
options: {  responsive: true,
plugins: { legend: {
position: 'top',},
title: { display: true,
text: 'Chart.js Doughnut Chart'  },},
animation: { animateScale: true,
animateRotate: true}}};
var ctx = document.getElementById('chart-area').getContext('2d');
window.myDoughnut = new Chart(ctx, config);
</script>
</div>
```

3.2 Робота з програмою

Система контролю якості програмного забезпечення створена для вирішення такої задачі як відслідковування якості певного додатку, можливість оцінити загальний рівень якості завдяки діаграмам, а також завдяки можливості написання тестових випадків запобігти появі дефектів в програмі.

Перший крок для роботи з програмою це запуск сервера та бази даних в додатку XAMPP (рис. 36).

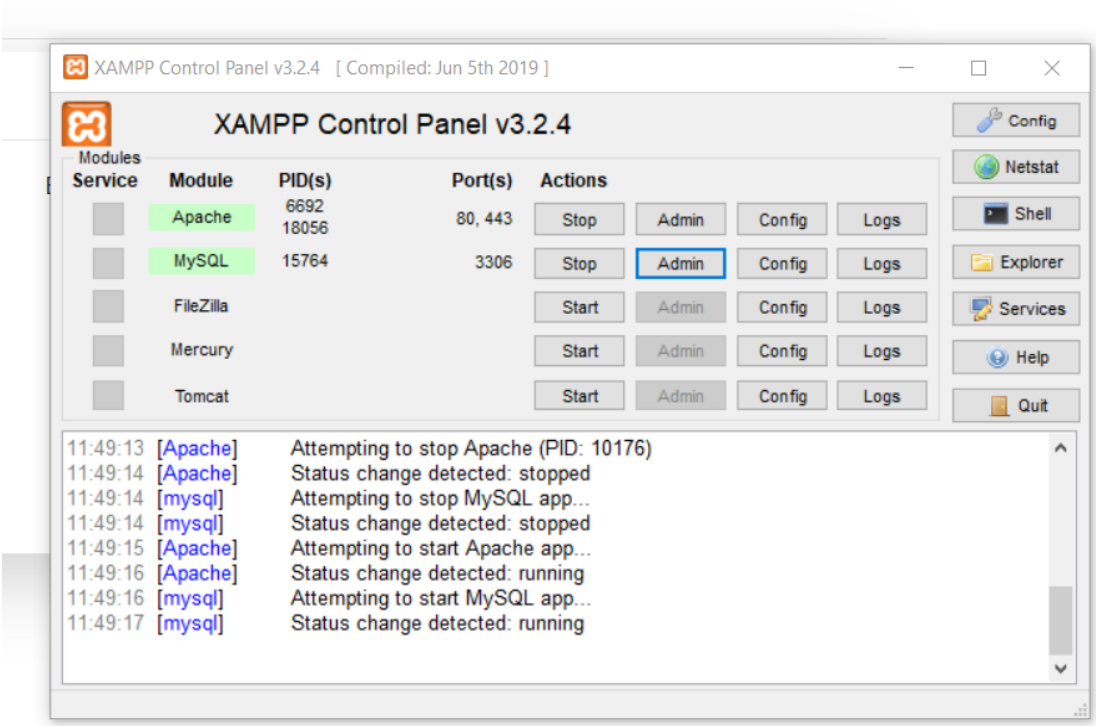


Рисунок 36 - Запуск Apache та MySQL в XAMPP

Після цього є два сценарії роботи програми:

1. Якщо користувач був авторизованим, тоді йому одразу відкривається сторінка Test Cases за замовчуванням
2. Якщо користувач не був авторизованим, тоді він перенаправляється на сторінку авторизації, де має ввести логін та пароль (рис. 37). Після успішного введення відкриється сторінка Test Cases за замовчуванням (рис. 38).

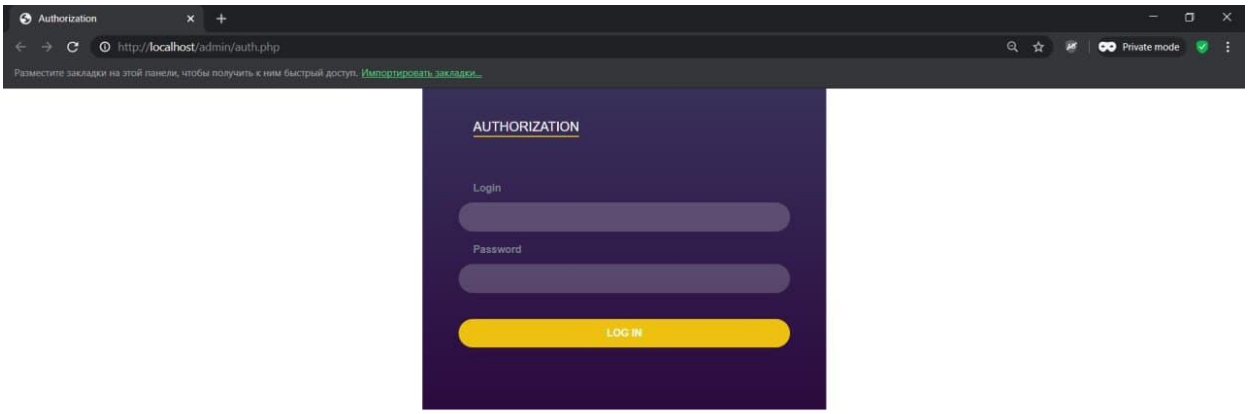


Рисунок 37 - Сторінка авторизації

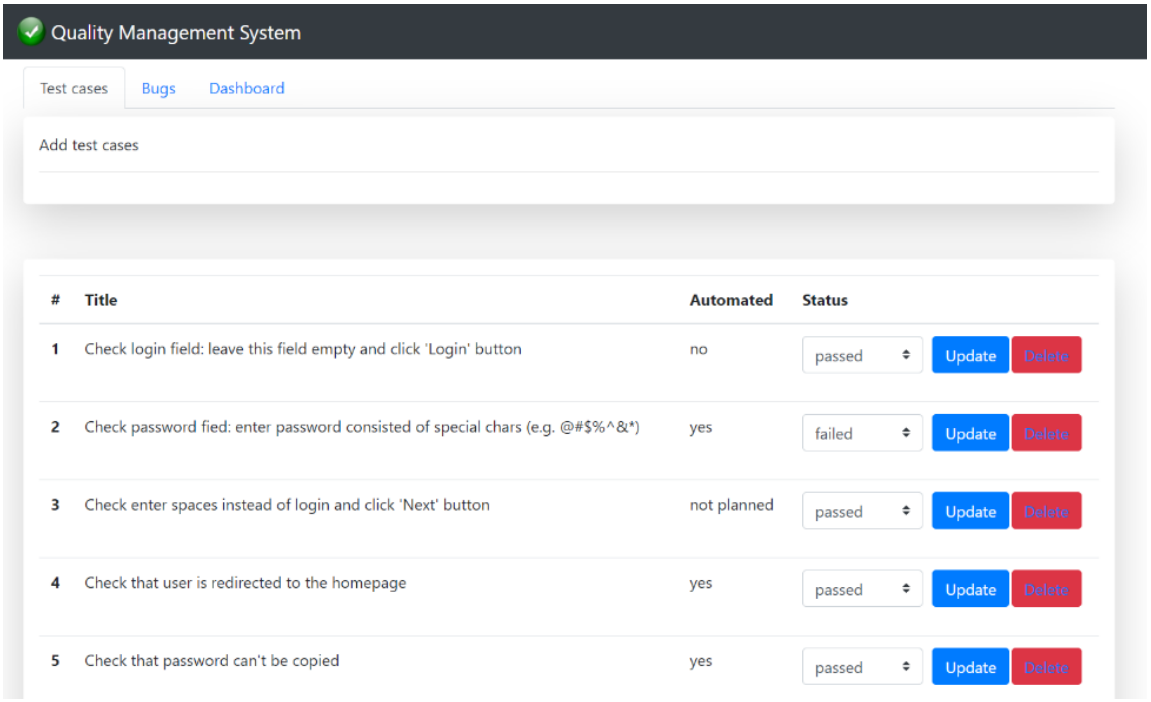


Рисунок 38 - Сторінка Test Cases

3.2.1 Робота з тест кейсами

Користувач має можливість додати новий тест кейс. Для цього йому потрібно натиснути кнопку «Add test cases». Після натиснення кнопки розгорнеться область введення даних: назва та вибір статусу автоматизації (рис. 39).

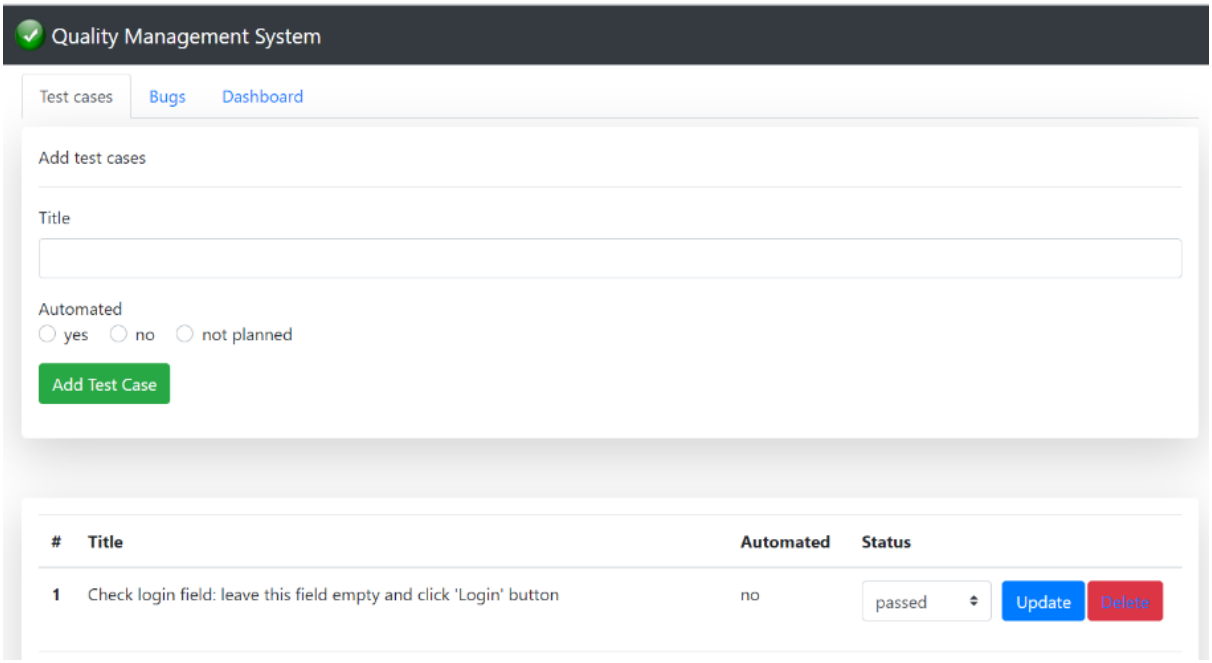


Рисунок 39 - Створення нового тест кейсу

Після того як всі поля заповненні користувач натискає кнопку «Add Test Case».

Варто виконати перевірку коректного додавання запису до бази даних. Відкриємо базу даних за адресою: localhost/phpMyAdmin. Відкриємо таблицю test_cases та знайдемо новостворений тест кейс (рис. 40).

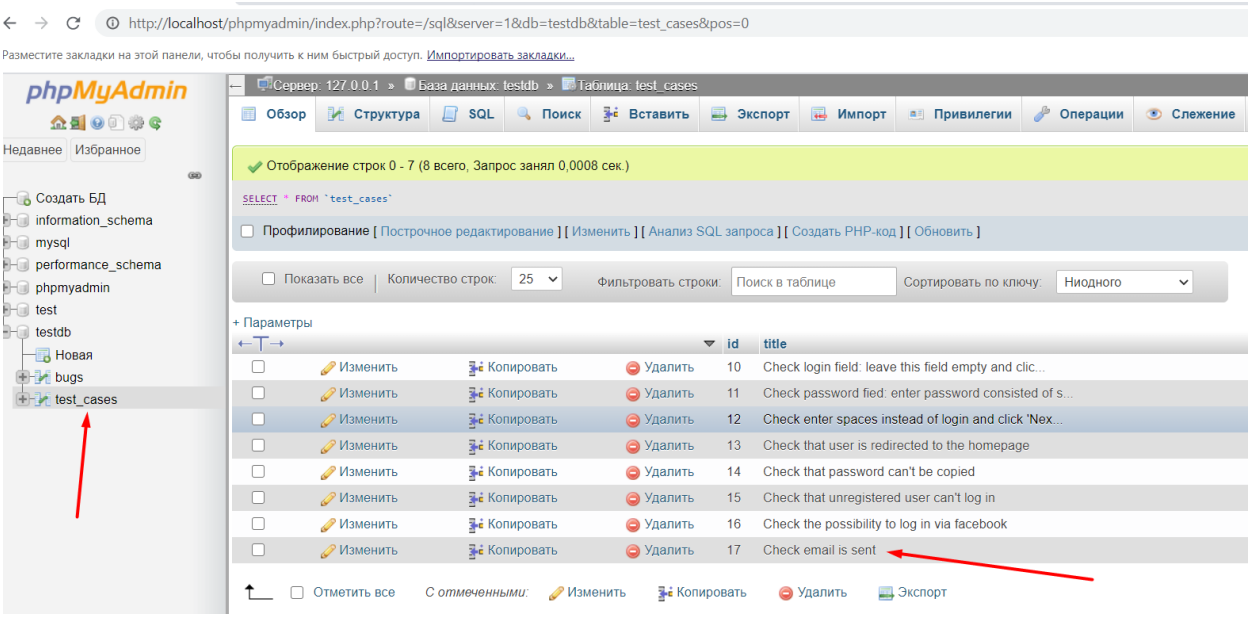


Рисунок 40 - Новий запис в базі даних в таблиці test_cases

Повернувшись на сторінку Test Cases в додатку можна побачити, що новий тест кейс з'явився в кінці списку (рис. 41).

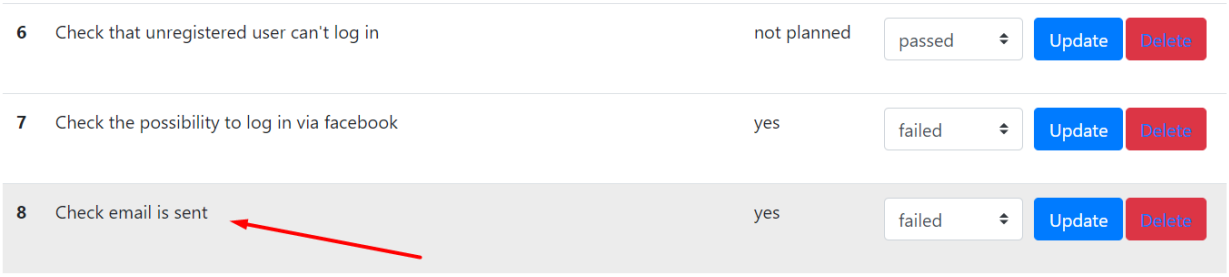


Рисунок 41 - Зображення нового тест кейсу в списку всіх тест кейсів

Наступною можливістю є зміна статуса тест кейса на пройдено чи не пройдено. Змінимо створеному на попередньому кроці тест кейсу статус на passed (рис. 42).

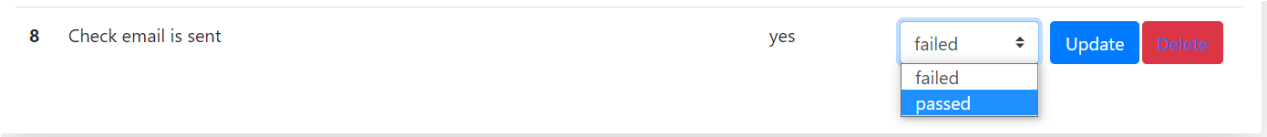


Рисунок 42 - Зміна статуса тест кейса на passed

Перевіримо зміни в базі даних. Як бачимо з рис. 43 статус тест кейса змінився і став «1», що означає passed.

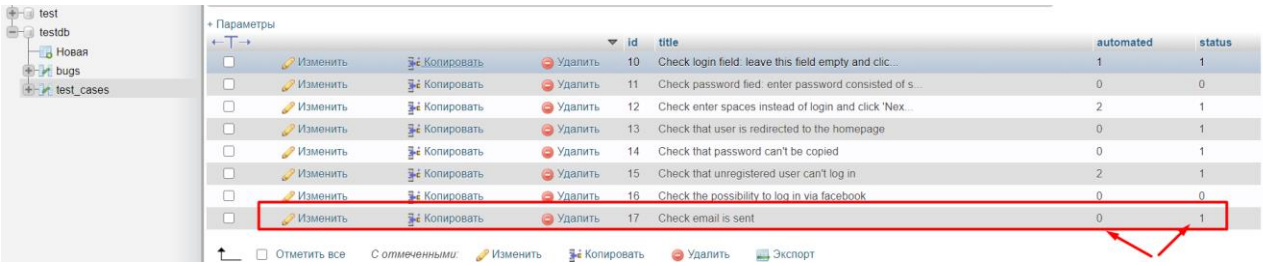


Рисунок 43 - Демонстрація зміненого статусу тест кейса

Перейдемо на сторінку Dashboard, котра демонструє статистику тест кейсів у розрізі кількості пройдених (passed) по відношенню до кількості не пройдених (failed) (рис. 44).

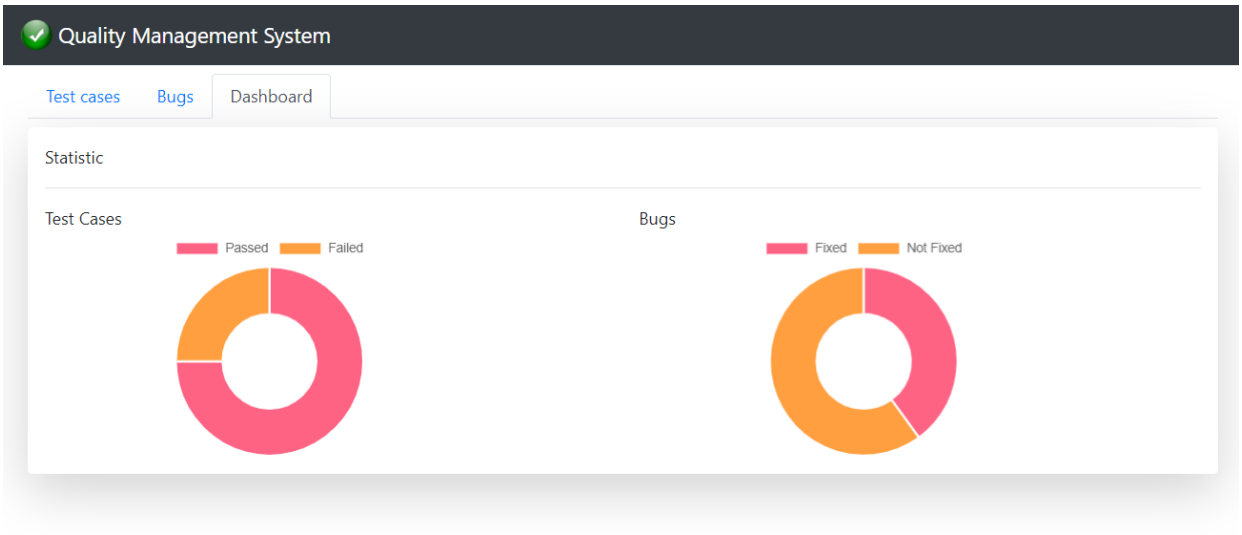


Рисунок 44 - Сторінка Dashboard

Загальна кількість тест кейсів дорівнює вісім, серед них 6 пройдених та 2 не пройдених (рис. 45).

#	Title	Automated	Status		
1	Check login field: leave this field empty and click 'Login' button	no	passed	Update	Delete
2	Check password fied: enter password consisted of special chars (e.g. @#$%^&*)	yes	failed	Update	Delete
3	Check enter spaces instead of login and click 'Next' button	not planned	passed	Update	Delete
4	Check that user is redirected to the homepage	yes	passed	Update	Delete
5	Check that password can't be copied	yes	passed	Update	Delete
6	Check that unregistered user can't log in	not planned	passed	Update	Delete
7	Check the possibility to log in via facebook	yes	failed	Update	Delete
8	Check email is sent	yes	passed	Update	Delete

Рисунок 45 - Загальна кількість тест кейсів з відповідними статусами

Розглянемо зображення цих даних на круговій діаграмі і впевнимося в правильності розрахунків (рис. 46).

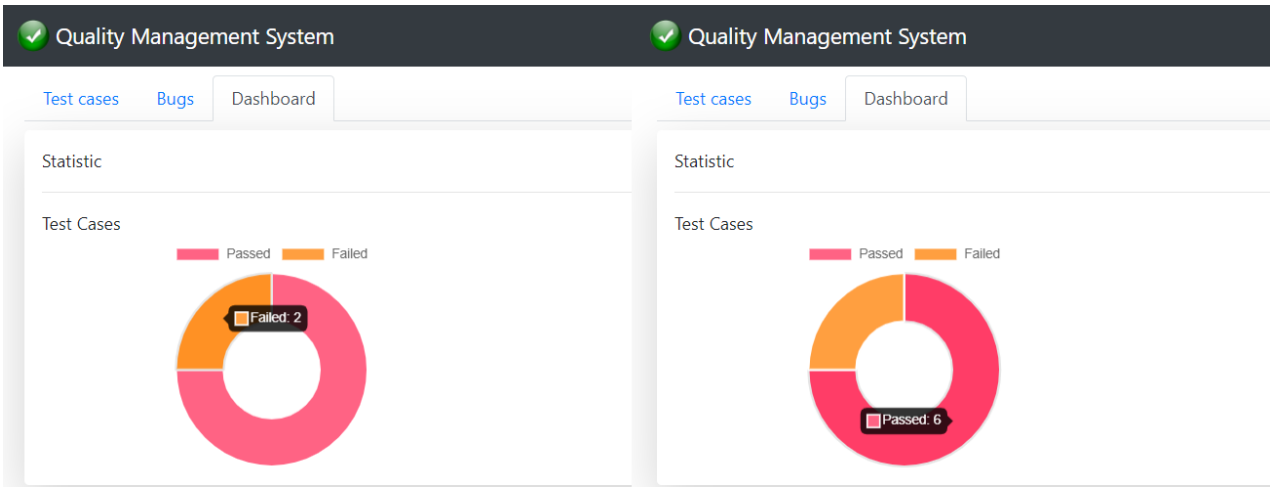


Рисунок 46 - Співвідношення пройдених та не пройдених тест кейсів на круговій діаграмі

Видалення тест кейсів

Для того аби видалити певний тест кейс потрібно відкрити вкладку Test Cases та натиснути кнопку «Delete» навпроти обраного тест кейсу. Перед цим я продемонструю, що даний тест кейс наявний в базі даних (рис. 47).

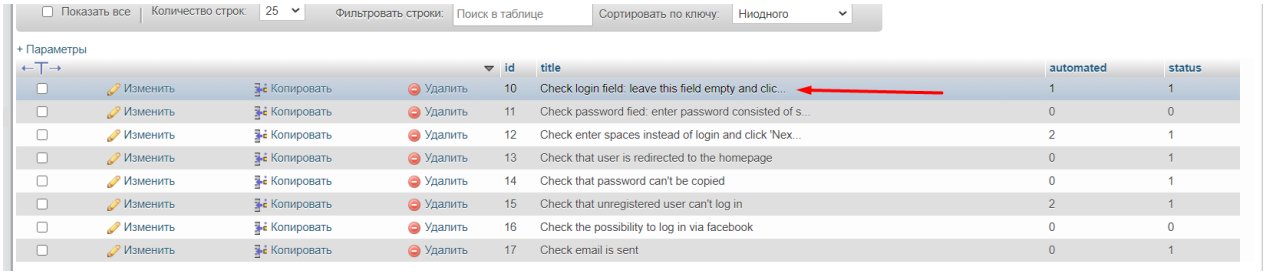


Рисунок 47 - Запис обраного тест кейсу наявний в базі даних

Далі видаляємо цей тест кейс (рис. 48).

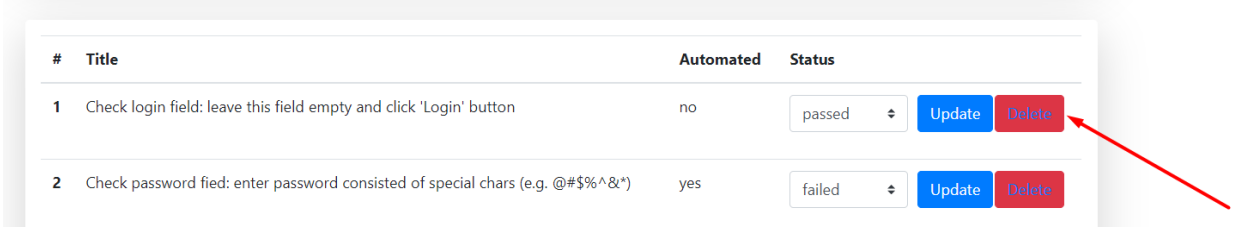
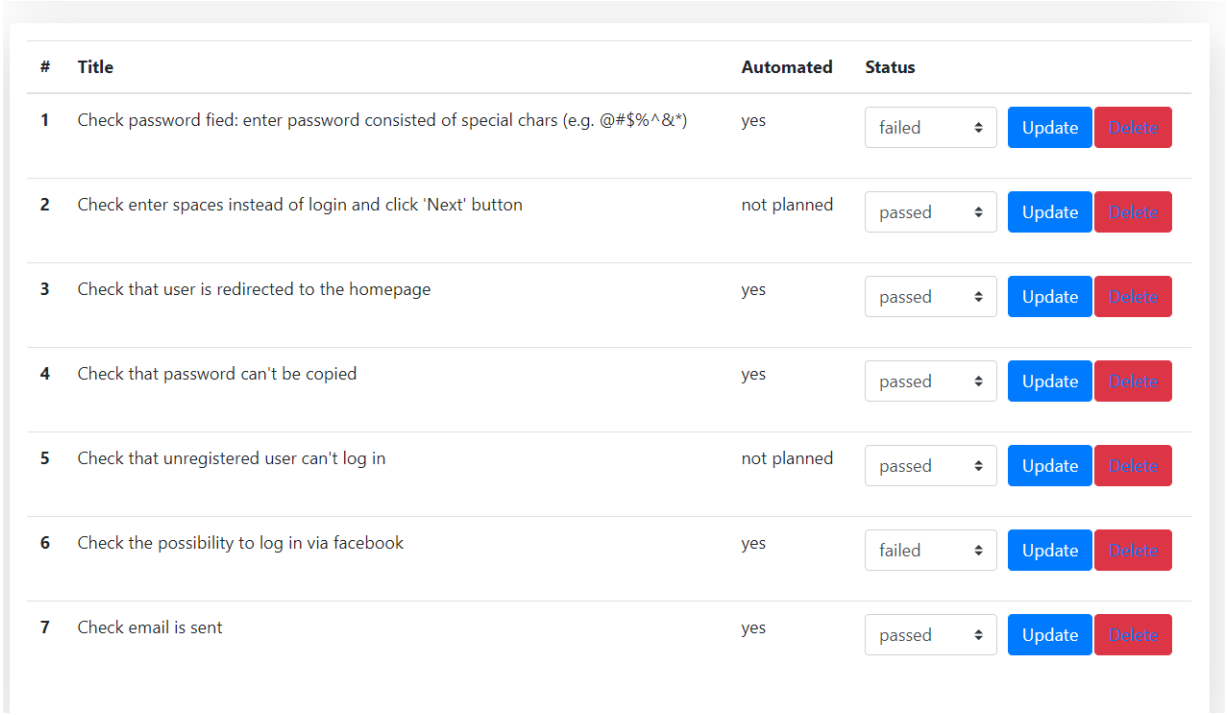


Рисунок 48 - Видалення обраного тест кейсу

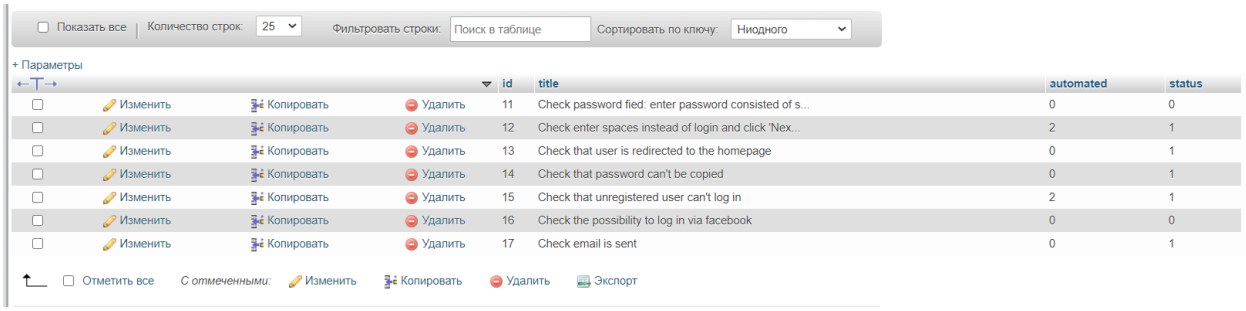
Тест кейс видалено із загального списку, кейс, що був під номером два тепер відображається першим (рис. 49).



#	Title	Automated	Status		
1	Check password fied: enter password consisted of special chars (e.g. @#$%^&*)	yes	failed	Update	Delete
2	Check enter spaces instead of login and click 'Next' button	not planned	passed	Update	Delete
3	Check that user is redirected to the homepage	yes	passed	Update	Delete
4	Check that password can't be copied	yes	passed	Update	Delete
5	Check that unregistered user can't log in	not planned	passed	Update	Delete
6	Check the possibility to log in via facebook	yes	failed	Update	Delete
7	Check email is sent	yes	passed	Update	Delete

Рисунок 49 - Оновлений список тест кейсів

В базі даних цього запису більше немає (рис. 50).



Показать все | Количество строк: 25 | Фильтровать строки: Поиск в таблице | Сортировать по ключу: Ниодного

+ Параметры

				id	title	automated	status
☐	Изменить	Копировать	Удалить	11	Check password fied: enter password consisted of s...	0	0
☐	Изменить	Копировать	Удалить	12	Check enter spaces instead of login and click 'Nex...	2	1
☐	Изменить	Копировать	Удалить	13	Check that user is redirected to the homepage	0	1
☐	Изменить	Копировать	Удалить	14	Check that password can't be copied	0	1
☐	Изменить	Копировать	Удалить	15	Check that unregistered user can't log in	2	1
☐	Изменить	Копировать	Удалить	16	Check the possibility to log in via facebook	0	0
☐	Изменить	Копировать	Удалить	17	Check email is sent	0	1

Отметить все С отмеченными: Изменить Копировать Удалить Экспорт

Рисунок 50 - Оновлений вигляд таблиці test_cases після видалення тест кейсу

Зміни були застосовані і до кругової діаграми (рис. 51).

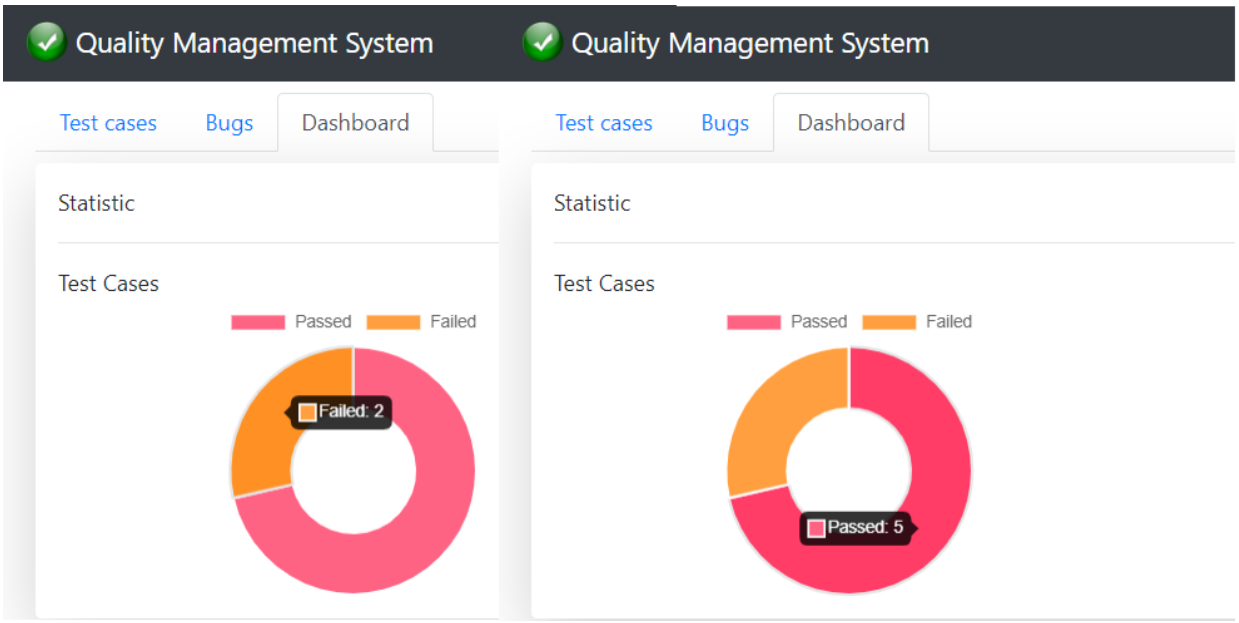


Рисунок 51 - Вигляд оновлених діаграм після видалення тест кейсу

Додатково було реалізовано модуль прийняття рішень, який базується на двох власних метриках оцінки ефективності тест кейсів, а також покриття автоматизацією.

Перша метрика «Test cases efficiency» відображає розподіл між кількістю написаних тест кейсів та кількістю дефектів, які були знайдені завдяки ним (рис. 52). Також можна побачити еталонне значення, а саме: один тест кейс знаходить мінімум один дефект. Завдяки цій метриці можна оцінити ефективність створених тест кейсів, а саме чи знаходять вони дефекти та наскільки багато, можна визначити проблемну область додатку.

В результаті цього аналізу метрики можна прийняти управлінське рішення, наприклад: необхідно збільшити кількість тест кейсів або підняти рівень їх деталізації.

Друга метрика рахує відсоток автоматизованих тест кейсів по відношенню до загальної кількості (рис. 53). Еталоне значення для проекту прямує до 100%. Таким чином користуючись результатами цих підрахунків можна робити певні припущення та висновки. Наприклад, що половина написаних тест кейсів

автоматизовано, отже заощадили час на їх проходження. На основі даної метрики можна приймати управлінські рішення, наприклад необхідні додаткові ресурси для покриття автоматизованими тестами.

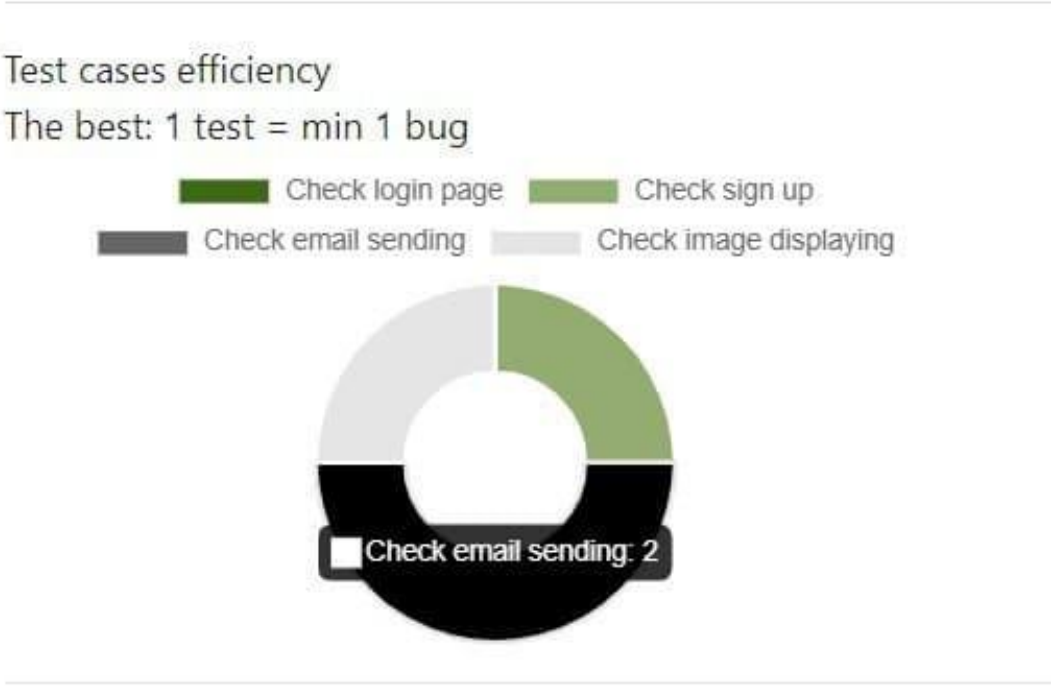


Рисунок 52 - Метрика ефективності тест кейсів

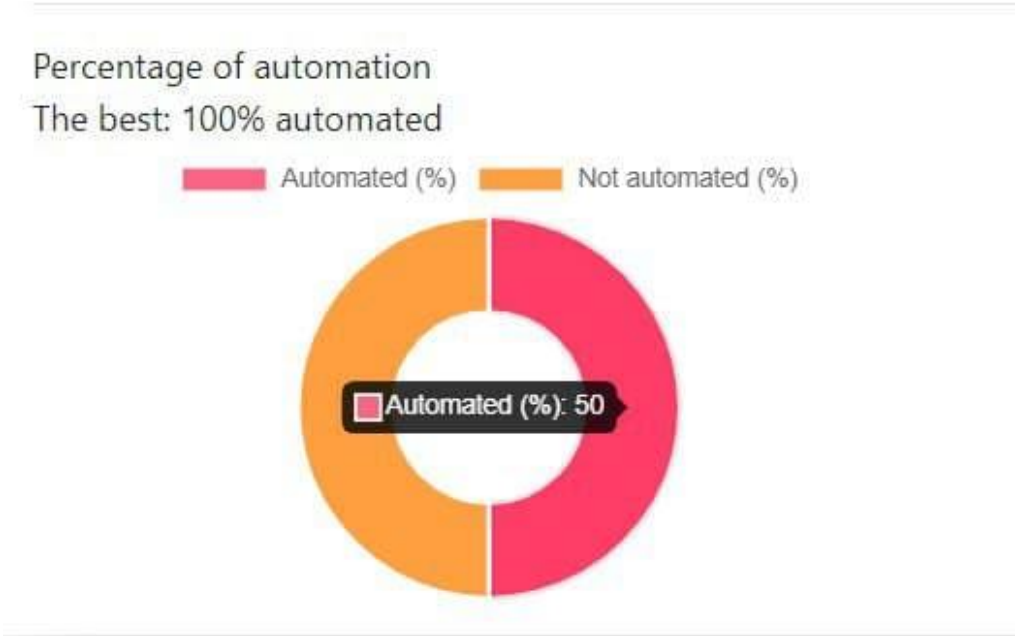


Рисунок 53 - Метрика покриття тест кейсів автоматизованими тестами

3.2.2 Робота з дефектами

Перейдемо на сторінку Bugs (рис. 54).

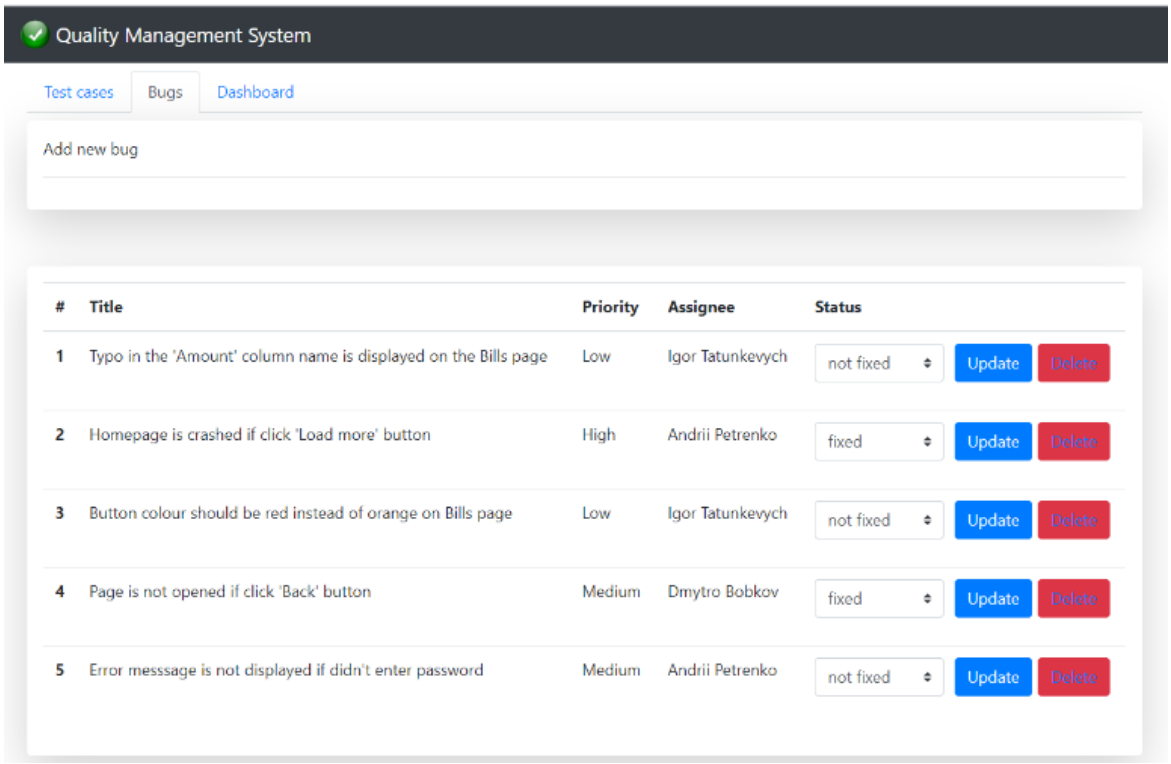


Рисунок 54 - Сторінка Bugs

Натиснувши кнопку «Add new bug» користувачеві розгорнеться область для вводу вхідних даних, так як назва, пріоритет та виконавець (рис. 55).

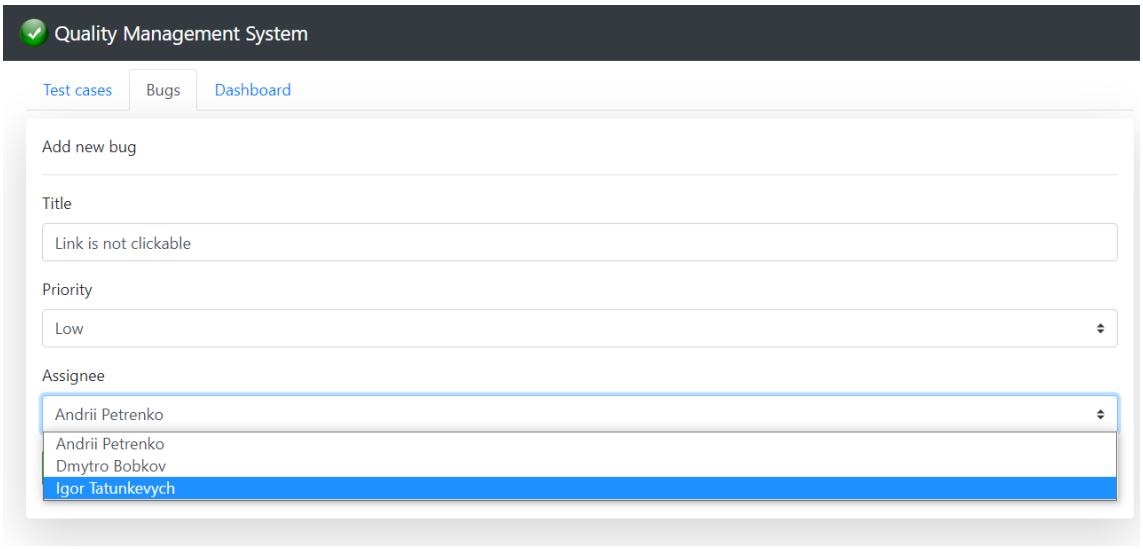


Рисунок 55 - Демонстрація процесу створення нового дефекту

Після натискання кнопки «Add new bug» новий дефект з’явився в кінці списку дефектів (рис. 56).

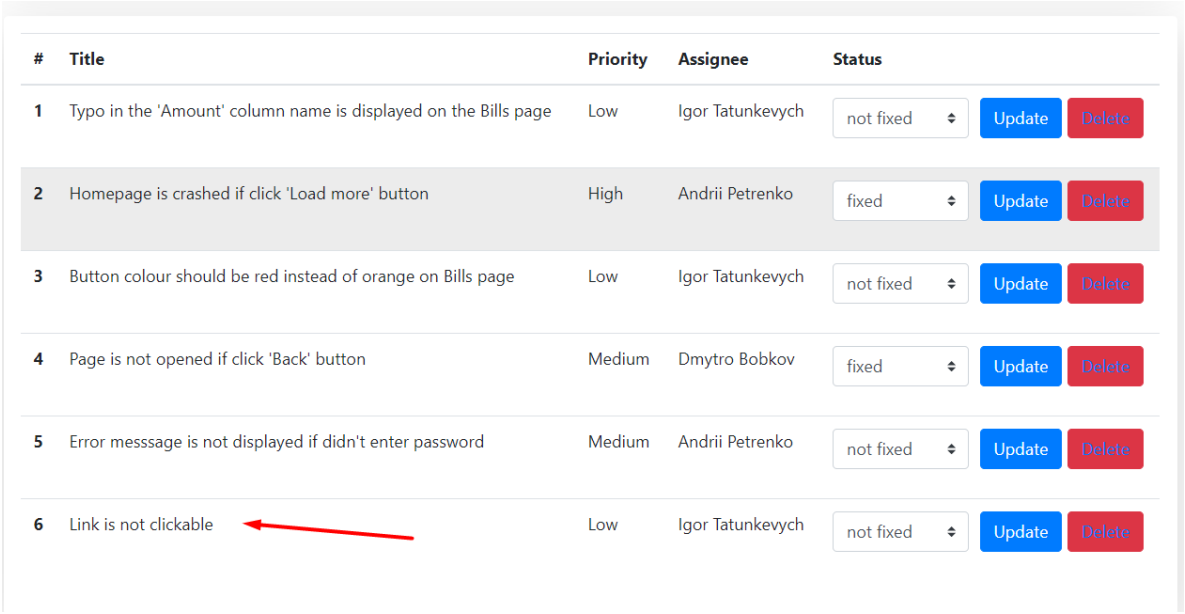


Рисунок 56 - Зображення нового дефекту в списку існуючих

Новий запис з’явився і в таблиці bugs в базі даних із обраним пріоритетом та виконавцем (рис. 57).

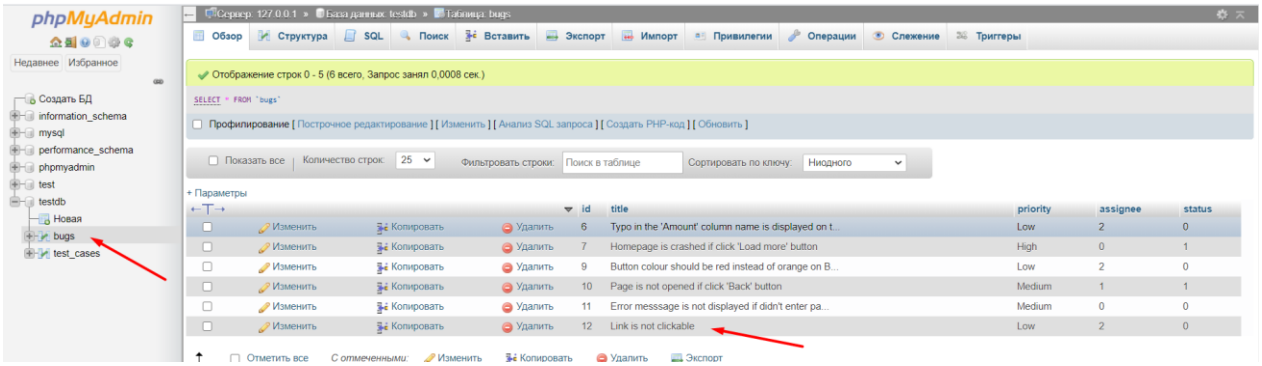


Рисунок 57 - Новий запис в таблиці bugs

Користувач має можливість змінити статус дефекту на виправлено або не виправлено (fixed, not fixed). Змінимо статус створеного на попередньому кроці дефекту на виправлено (рис. 58).

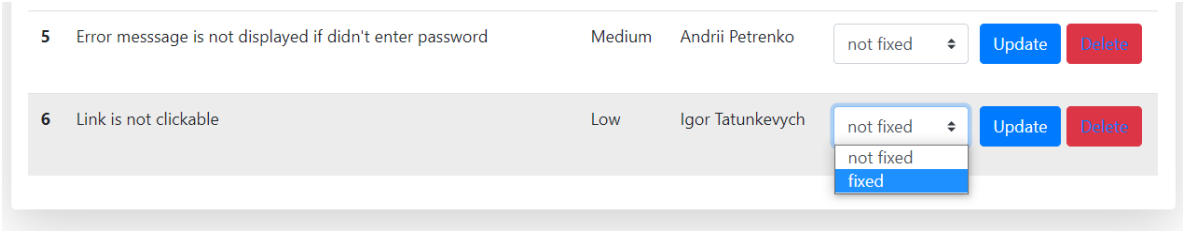


Рисунок 58 - Зміна статусу дефекту на «виправлено»

Наступне зображення демонструє, що статус було успішно змінено і в базі даних (рис. 59).

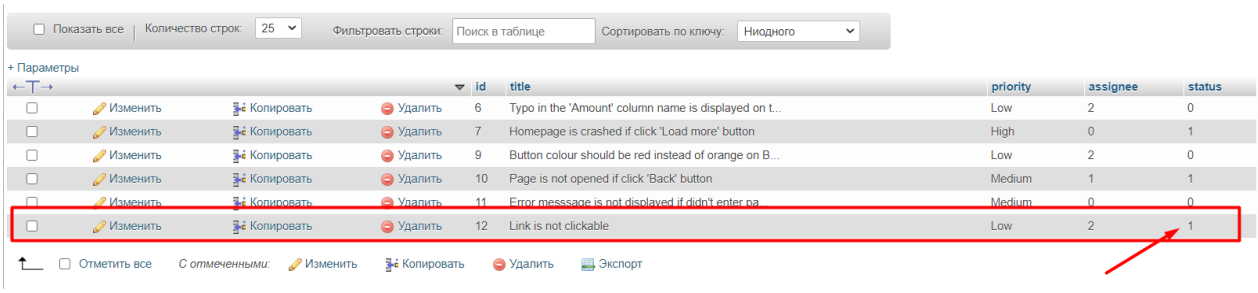


Рисунок 59 - Змінений статус дефекту в базі даних.

Для перевірки правильності роботи кругової діаграми для дефектів, порахуємо кількість виправлених та не виправлених дефектів. Всього 6 дефектів: 3 виправлених та 3 не виправлених (рис. 60).

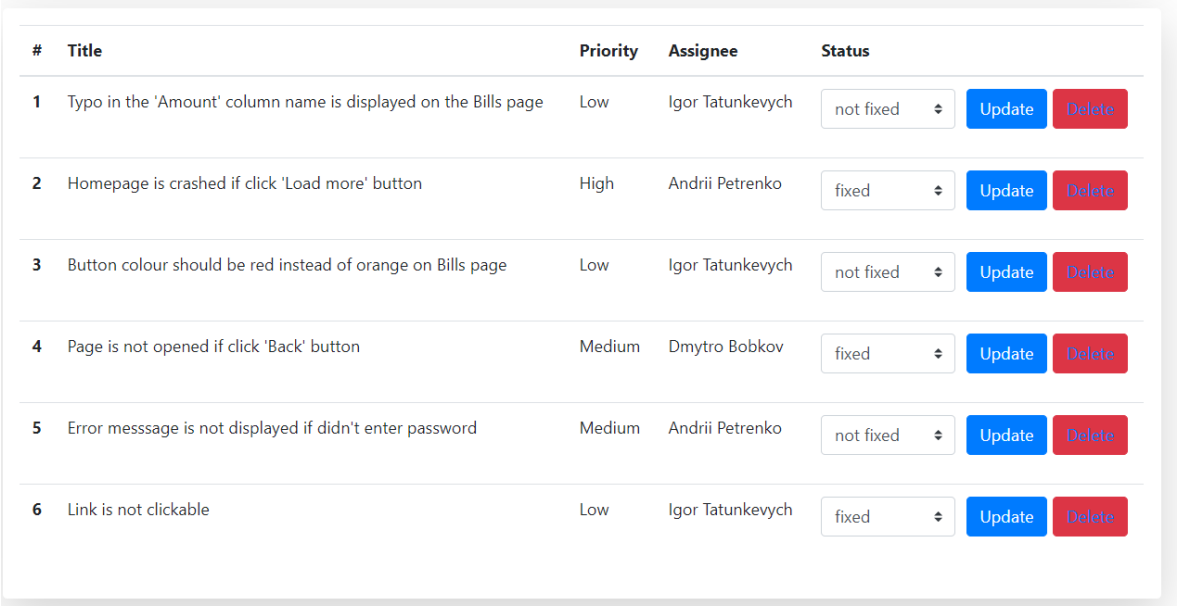


Рисунок 60 - Загальний вигляд списку дефектів

Кругова діаграма демонструє ті ж самі дані (рис. 61).

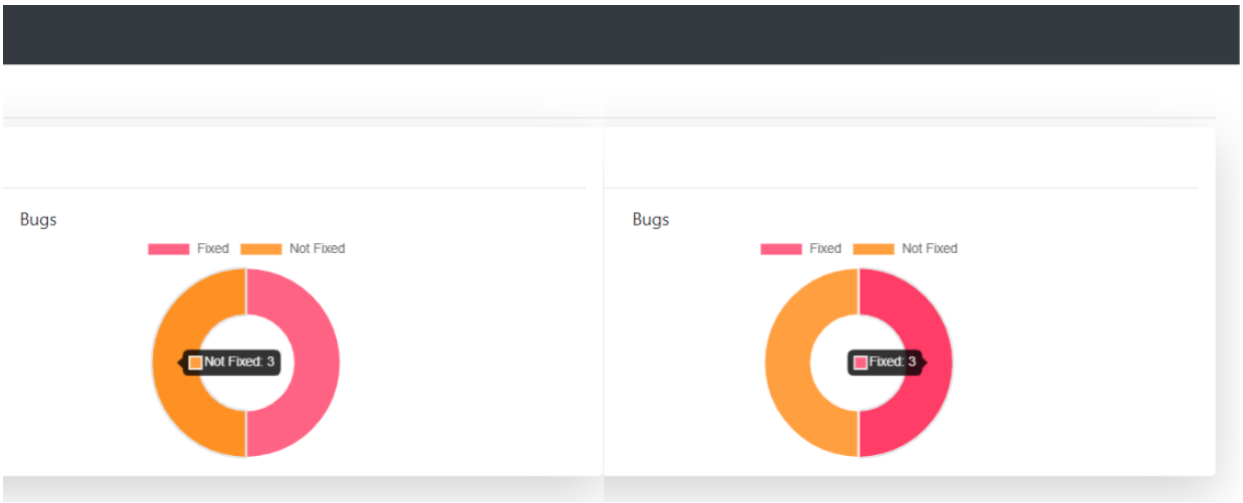


Рисунок 61 - Співвідношення виправлених та не виправлених дефектів на круговій діаграмі

Видалення дефектів

Для прикладу буде видалено дефект №4 «Page is not opened if click ‘Back’ button». Заздалегідь переконаємось, що такий запис є в таблиці bugs (рис. 62).

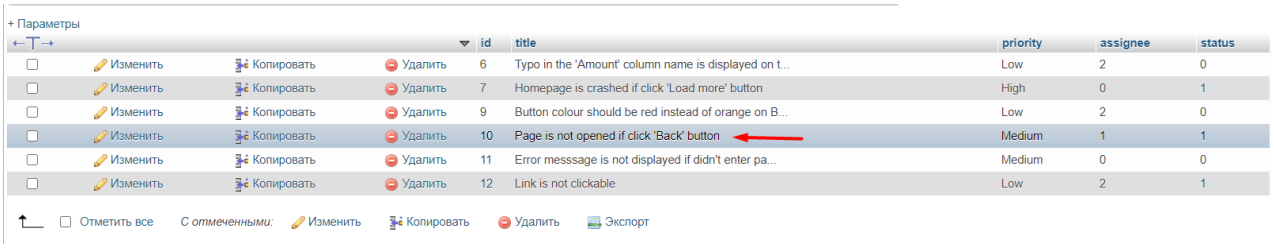


Рисунок 62 - Запис дефекту в таблиці bugs

Натискаємо на кнопку «Delete» (рис. 63).

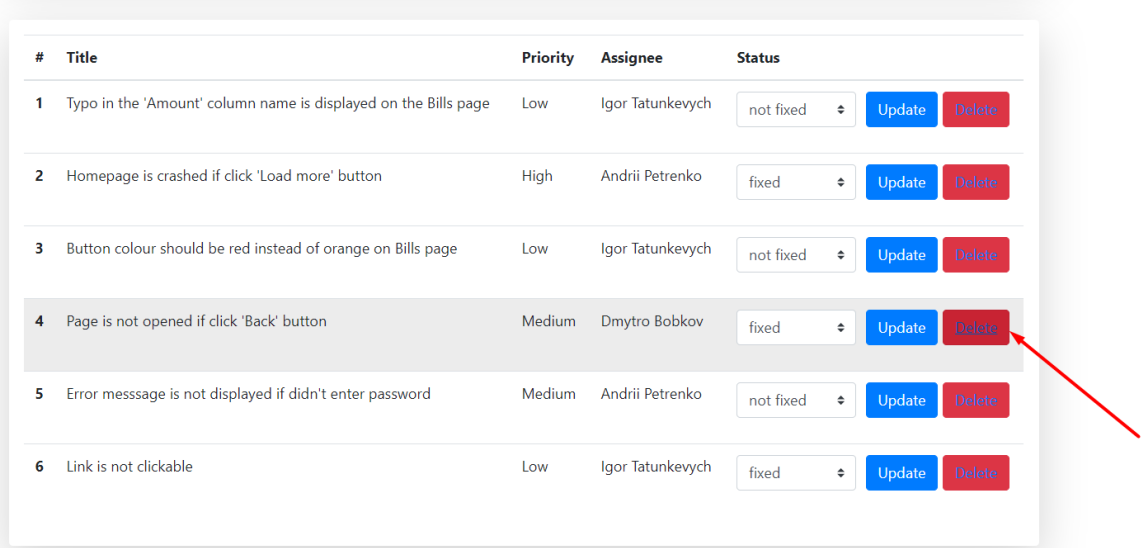


Рисунок 63 - видалення дефекту

Оновлений список дефектів має наступний вигляд (рис. 64).

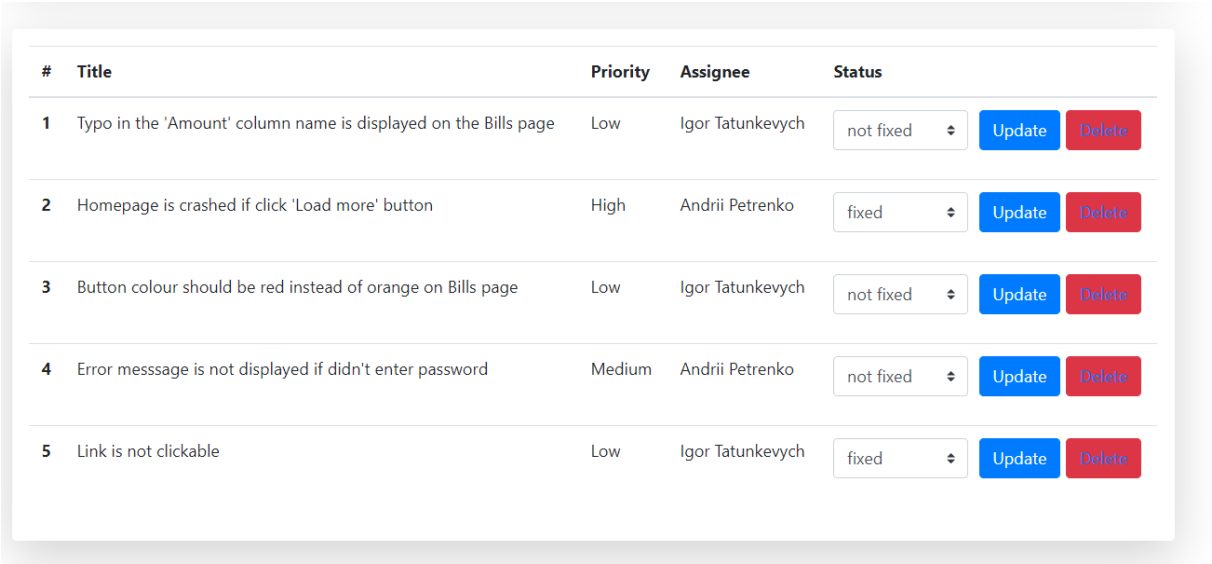


Рисунок 64 - Зображення оновленого списку дефектів після видалення одного з них

В таблиці цей запис також видалено (рис. 65).

+ Параметры

				id	title	priority	assignee	status
☐	Изменить	Копировать	Удалить	6	Typo in the 'Amount' column name is displayed on t...	Low	2	0
☐	Изменить	Копировать	Удалить	7	Homepage is crashed if click 'Load more' button	High	0	1
☐	Изменить	Копировать	Удалить	9	Button colour should be red instead of orange on B...	Low	2	0
☐	Изменить	Копировать	Удалить	11	Error messsage is not displayed if didn't enter pa...	Medium	0	0
☐	Изменить	Копировать	Удалить	12	Link is not clickable	Low	2	1

Отметить все *С отмеченными:* Изменить Копировать Удалить Экспорт

Рисунок 65 - Оновлений вигляд таблиці bugs після видалення дефекту

Зміни також були застосовані до кругової діаграми (рис. 66).

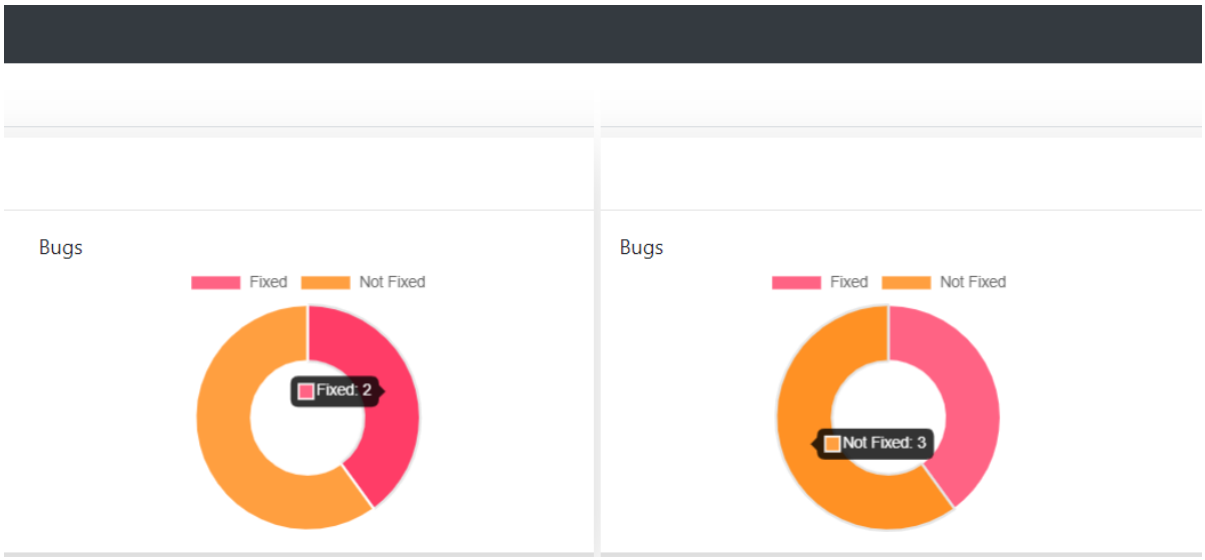


Рисунок 66 - Оновлений вигляд кругової діаграми після видалення дефекту

В доповнення до цього було створена метрики оцінки кількісті дефектів, що було «породжено» певним розробником (рис. 67). Еталоне значення прямує до 0 дефектів.

Це значення сигналізує про особливо складний для розробки та підтримки модуль, якщо ним займається конкретний розробник. Результати не вказують на найслабшого розробника, що допустив велику кількість дефектів, але допомагає звернути на це увагу та розібратись у причинах.

Також результати допомагають прийняти управлінське рішення, таке як зміна вектору роботи команди, можливо є необхідність підключити інших розробників для допомоги конкретному; впровадження огляду коду (code review) і тому подібне.

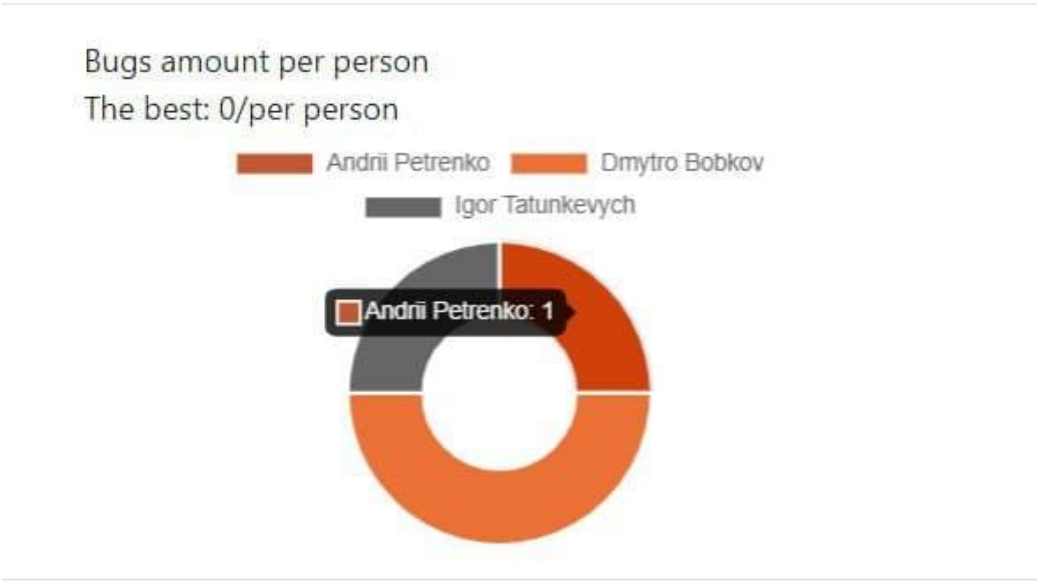


Рисунок 67 - Метрика оцінки кількості дефектів в коді конкретного розробника

ВИСНОВКИ

В кваліфікаційній роботі було виконано наступні поставлені задачі:

- проаналізовано існуючі програмні рішення для розв'язання задачі управління якістю програмного забезпечення
- проведено весь цикл робіт по розробці, починаючи від проектування закінчуючи тестуванням та внесенням покращень
- створено власну програмну систему, яка враховує недоліки існуючих продуктів, а саме:

  - являється безкоштовною
  - має вбудований функціонал для документування дефектів
  - має збільшений допустимий діапазон кількості символів для введення назви тест кейсу\дефекту
  - містить унікальну функцію створення дефекту під час зміни статусу тест кейса на «failed»
  - має сучасний інтерфейс, який не захаращений непотрібними функціями

Розроблена програмна система може бути використана на реальних проектах під час розробки програмного забезпечення, а також може бути доповнена додатковими функціями в майбутніх реалізаціях, такими як нові ролі користувачів (менеджер, розробник), фільтрація та сортування по тест кейсам та дефектам, формування звітів і так далі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. American Society for Quality. Електронний ресурс: https://asq.org/quality-resources/quality-management-system
2. Machlup F. The Branches of Learning. — Princeton, USA: Princeton University Press, 1982. -776с.
3. Machlup F. The Economics of Information and Human Capital. — Princeton, USA: Princeton University Press, 1984. -664с.
4. Machlup F. The Production and Distribution of Knowledge in the United States. — Princeton, USA: Princeton University Press, 1962. (Махлуп Ф. Виготовлення і розповсюдження знань США. ФМахлуп переклад з англ. — М.: Прогресс, 1966).-332с.
5. Тампе, Л. Введення в тестування програмного забезпечення, 2003, -368 с.
6. Machlup F. Knowledge and Knowledge Production.Ф Махлуп — Princeton, USA: Princeton University Press, 1980. -304с.
7. Alka Jarvis, Luis Morales, Ulka Ranadive. Achieving Customer Experience Excellence Through A Quality Management System, 2016, -256с.
8. What is the Quality Management System? Електронний ресурс: http://www.prismvs.com/what-is-the-quality-management-system.html
9. B. Purushothama. Effective Implementation of Quality Management Systems, 2010, -158с.
10. Сорокин М.А. Управление качеством продукции массового производства на основе оптимизации процессов контроля и испытания, 2011, -135с.