МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп’ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Бодров Олександр Миколайович

Програмний модуль для тестування та верифікації веб-орієнтованих додатків на основі технології ASP.NET / The software module for web-based applications testing and verifying based on ASP.NET technology

напрям підготовки: 123 Комп’ютерна інженерія
фахове спрямування - Комп’ютерна інженерія
Бакалаврська робота

Виконав студент групи КСМ-43/2
Бодров Олександр Миколайович

Науковий керівник:
І.В. Гураль

Тернопіль - 2018

РЕЗЮМЕ

Дипломний проект містить 80 сторінок пояснюючої записки, 32 рисунок, 12 таблиць, 2 додатки та 2 аркуші формату А3.

Для реалізації системи тестування проведено аналіз універсального та спеціалізованого апаратного та програмного забезпечення комп'ютерних систем і мереж, яке використовується організацією. Сформовано рекомендації, щодо покращення існуючих в організації технологічних процесів створення та використання комп'ютерних систем і мереж, їх програмного забезпечення, а саме:

– рекомендовано розробити тести, які б покривали абсолютно усю систему;

– рекомендовано оновити використовувані організацією системи, а саме SQL Server, Visual Studio та TeamCity;

– рекомендовано провести глобальний рефакторинг існуючого коду та архітектури проектних рішень, для більш ефективного використання простору та ресурсів системи.

Розроблено модульні тести, які проводять порівняння результатів та станів окремих модулів системи. Всі розроблені тести використовують фреймворк для розробки юніт-тестів NUnit. Для запуску тестів було використано менеджер Nunit Runner.

Результатом виконання завдання дипломної роботи стали автоматизовані модульні тести, розроблені засобами фреймворку NUnit. Тести розроблено для веб-додатку на основі технології ASP.NET.

Ключові слова: АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ASP.NET.

RESUME

The diploma project contains 80 pages of explanatory note, 32 figures, 12 tables, 2 appendices and 2 A3 sheets.

To implement the testing system, an analysis of the universal and specialized hardware and software of computer systems and networks used by the organization. Recommendations for improving the existing technological processes in the organization of the creation and use of computer systems and networks, their software, namely:

– it is recommended to develop tests that would cover absolutely the entire system;

– it is recommended to update the systems used by the organization, namely SQL Server, Visual Studio and TeamCity;

– it is recommended to conduct a global refactoring of the existing code and architecture of design solutions, for more efficient use of space and system resources.

Modular tests have been developed that compare the results and states of individual modules of the system. All developed tests use the framework to develop NUnit unit tests. The Nunit Runner manager was used to run the tests.

The result of the thesis was automated modular tests developed by the NUnit framework. The tests are designed for a web application based on ASP.NET technology.

Keywords: AUTOMATED TESTING, ASP.NET.

ЗМІСТ

ВСТУП

Для розробки та реалізації тестової системи було проведено підготовку, яка включала в себе підготовку особистих систем, дослідження реалізованої організацією системи, яка підлягає тестування та складено план розробки.

Розробка систем тестування програмного забезпечення, зокрема веб-додатків на сьогоднішній день є досить актуальною темою. Правильно спроектовані та реалізовані системи автоматичного тестування можуть гарантувати високий рівень стійкості програмного забезпечення до помилок. Також, тести, розроблені в процесі розробки програмного забезпечення можуть деяким чином впливати на архітектуру самого програмного засобу. Наприклад, досить поширена практика розроблення такої архітектури додатків, яка б легко піддавалась тестуванню, тобто архітектури з низьким рівнем залежності між елементами системи та дотриманням принципів SOLID.

Основною ідеєю автоматизованого тестування являється, звичайно, автоматизація процесу тестування. Тобто, головна мета таких тестів це автоматизація рутинних процесів та економія часу та людських ресурсів на ручне тестування. Проте, за останні кілька років автоматизовані тести стали дечим більшим, ніж просто заглушки для економії часу, а надійним засобом для гарантії робото-здатності додатку. Доказом цього є досить висока популярність методу розробки через тестування – Test Driven Development (TDD). При такому методі спочатку як правило розробляють базову архітектуру, або каркас програмного забезпечення, далі розробляють автоматизовані тести, які перевіряють роботу окремих модулів системи і тоді розробляють безпосередньо логіку, тобто сам продукт. При такому підході вся бізнес-логіка (і не тільки) розробляється таким чином, щоб розроблені модулі проходили розроблені тести.

Метою даної дипломної роботи є розробка деякої кількості автономного тестування окремих модулів веб-додатку на основі технології ASP.NET. Дані тести будуть мати вигляд окремого програмного модуля, інтегрованого в проект.

1 АНАЛІЗ ТА ОБГРУНТУВАННЯ ВИБОРУ ВЕБ-ОРІЄНТОВАНИХ ДОДАТКІВ

1.1 Класифікація засобів розробки веб-додатків

Зважаючи на особливості та розвиток сучасних Інтернет-технологій, засоби розробки веб-додатків можна класифікувати наступним чином:

– за типом мережевої архітектури (клієнт-сервер, локальні системи, P2P (Peep-To-Peer)-системи, тощо);

– за технологіями (MVC (Model View Controller), WebForms, тощо);

– доступністю (коммерційні, SAAS (Software As A Service) засоби, бешкоштовні).

На сьогоднішный день в багатьох системах, які використовують бази даних, методи клієнтського інтерфейсу (API – Application Programming Interface) викликаються безпосередньо з коду додатку, написаного на мові програмування високого рівня, як правило на C або C++. Більшість таких модулів API клієнтської частини будь-якої серверної системи управління баз даних (СУБД) можна знайти чимало екземплярів найбільш використовуваних модулів, наприклад, для авторизації клієнтів, виконання запитів до бази даних і т.д. Проте витрати часу та трудових ресурсів на написання подібних систем, можна значно скоротити, якщо створити бібліотеку з найбільш використовуваними модулями додатків і популярними елементами, призначеними для користувацького інтерфейсу. І якщо оформити дані модулі в окремій бібліотеки або навіть в фреймворки можна та приєднати їх до процесу розробки та використання систем можна було отримати зручний функціональний інструмент. Саме так і з'явилися перші засоби розробки, орієнтовані на конкретні СУБД, такі як Oracle, SQL Server, MySQL, PostgreSQL, Microsoft Access, тощо.

Продукти даного типу дуже популярні серед розробників і не тільки до сьогодні. Більшість розробників СУБД розробляють також і засоби розробки

додатків. Як результат, розробникам доступний широкий спектр засобів розробки, які в свою чергу заточені під зручний доступ до СУБД відповідної компанії. Зазвичай, розроблені фреймворки, бібліотеки та модулі підтримують взаємодію з СУБД інших компаній, як правило, за допомогою універсальних механізмів доступу до даних (ODBC, OLE DB, BDE). Однак доступ до СУБД, розробленої однією компанію, що і клієнтські API, здійснюється максимально ефективним способом по зрозумілим причинам. Результатом таких стандартів стали клієнтські API, модулі та методи, що знаходяться в бібліотеках та фреймворках клієнтської частини додатків, а також спеціальні класи для організації доступу до даних конкретної (фірмової) СУБД або за допомогою реалізації провайдерів для універсальних механізмів доступу до даних, здатних враховувати специфічні особливості конкретної СУБД.

Окремо можна відзначити СУБД, які пропонують певний функціональний набір інструментів для розробників веб-додатків. На даний момент найпопулярнішими СУБД є Microsoft Visual FoxPro, Microsoft Access, Corel Paradox, Visual dBase, SQLServer, Oracle, MySQL, PostgreSQL. Більшість з перерахованих СУБД володіють методами для забезпечення взаємодії з спеціально розробленими системами – бібліотеками та фреймворками для реалізації веб-додатків. Також, дані СУБД підтримують багато універсальних методів взаємодії з додатком. На даний момент, створення веб-додатків архітектури "клієнт-сервер" з допомогою даних СУБД є розповсюдженим явищем. Окремо можна відзначити засоби розробки веб-додатків ASP.NET та Microsoft SQL Server. Популярність даної пари є результатом грамотної політики Microsoft, яка прагне до максимальної сумісності своїх продуктів і забезпечує найбільш комфортну для користувачів заміну настільних СУБД власними ж серверами баз даних.

Інструменти розробки, які позиціонуються як універсальні по відношенню до СУБД представляють собою послідовниками звичайних інструментів розробки додатків, які не мають прямого відношення до баз

даних. Приклади подібних засобів розробки - Borland Pascal, Borland C++, Microsoft QuickC, здатні використовувати бібліотеки сторонніх розробників. Дані інструменти дозволяють звертатися до методів та класів клієнтських API, а з розвитком універсальних механізмів доступу до даних, наприклад, ODBC - і до функцій API бібліотек, що реалізують такі механізми. Досить часто, за допомогою даних засобів створювалися середовища настільних СУБД (dBase, FoxBase) або псевдо-компілятори для мов сімейства xBase (наприклад, Clipper).

Пізніші версії описаних засобів розробки вже включали безліч сторонніх бібліотек, функцій та класів, призначених для доступу до даних за допомогою універсальних механізмів. Подальший розвиток засобів розробки призвів до появи двох категорій продуктів подібного призначення.

Веб-додаток - клієнт-серверний додаток, в якому клієнт взаємодіє з сервером за допомогою браузеру, а за сервер відповідає - веб-сервер. Бізнес логіка веб-додатку розділена між сервером і клієнтом, збереження даних здійснюється, переважно, на сервері, обмін інформацією проводиться за допомогою мережі. Однією з переваг даного підходу є незалежність клієнтів від конкретної операційної системи користувача, тому веб-додатки являють собою крос-платформенні системи.

Веб-додатки почали широко використовуватися в кінці 1990-х – початку 2000-х [1]. Важлива перевага побудови веб-додатків для підтримки класичних функцій браузера полягає в тому, що функції повинні виконуватися незалежно від операційної системи клієнта. Замість того, щоб розробляти різні версії для Microsoft Windows, Mac OS X, GNU/Linux та інших операційних систем, додаток створюється для довільно обраної платформи і на ній розгортається. Проте, різна реалізація HTML, CSS, DOM і інших специфікацій в браузерах може стати причиною проблем при розробці веб-додатків і подальшої їх підтримки. Крім того, можливість користувача змінювати багато параметрів браузеру (наприклад, розмір шрифту, кольору,

відключення підтримки сценаріїв, відображення Flash) може завадити коректній роботи програми.

Інший підхід полягає у використанні Adobe Flash, Silverlight або Java-аплетів для повної або часткової реалізації призначеного для користувача графічного інтерфейсу. Так, як більшість браузерів підтримує дані технології (як правило, за допомогою плагінів), Flash- або Java-додатки можуть виконуватися без проблем, спричинених особливостями браузерів. Так, як вони надають програмісту більший контроль над інтерфейсом, вони здатні обходити безліч несумісностей в конфігураціях браузерів, хоча несумісність між Java- або Flash-реалізаціями на стороні клієнта може призводити до різних ускладнень.

Від 2015 року технологію Adobe Flash не підтримують Chrome, Safari, і інші популярні браузери [2].

У зв'язку з архітектурною схожістю з звичайними клієнт-серверними додатками, існують суперечки, щодо коректності віднесення подібних систем до веб-додатків.

Веб-додаток складається з клієнтської і серверної частин, тим самим реалізуючи технологію «клієнт-сервер». Клієнтська частина реалізує користувацький графічний інтерфейс, формує запити до серверу і обробляє відповіді. Серверна частина отримує запит від клієнта, виконує обчислення, після цього формує веб-сторінку і відправляє її клієнту засобами мережі з використанням протоколу HTTP (Hyper Text Transfer Protocol).

Саме веб-додаток може виступати в якості клієнта інших служб, наприклад, бази даних або іншого веб-додатку, розташованого на іншому сервері. Яскравим прикладом веб-додатку є система управління вмістом статей Вікіпедії: безліч її учасників можуть брати участь у створенні мережевої енциклопедії, використовуючи для цього браузери своїх операційних систем (Microsoft Windows, GNU/Linux або будь-яка інша операційна система) і не завантажуючи додаткових виконуваних модулів для роботи з базою даних статей.

На даний момент набирає популярність новий підхід до розробки веб-додатків, так званий AJAX (Asynchronous Javascript and XML (eXtensible Markup Language)). При використанні AJAX сторінки веб-додатки не перезавантажуються цілком, а лише довантажують необхідні дані з сервера, що робить їх більш інтерактивними і продуктивними.

Також, останнім часом набирає велику популярність технологія WebSocket, яка не вимагає постійних запитів від клієнта до сервера, а створює дво-направлене з'єднання, при якому сервер може відправляти дані клієнта без запиту від останнього. Таким чином з'являється можливість динамічно керувати контентом в режимі реального часу.

Для створення веб-додатків на стороні сервера використовуються різноманітні технології та будь-які мови програмування, здатні здійснювати вивід в стандартну консоль.

Будь-який веб-додаток являє собою набір статичних і динамічних веб-сторінок. Статична веб-сторінка - це сторінка, яка завжди відображається перед користувачем в незмінному вигляді. Веб-сервер відправляє сторінку за запитом веб-браузера без будь-яких змін. На противагу цьому, сервер вносить зміни в динамічну веб-сторінку перед відправкою її браузеру. У зв'язку з тим що сторінка змінюється, вона називається динамічною.

Наприклад, можна створити сторінку, на якій будуть відображені результати програми оздоровлення. При цьому деяка інформація (наприклад, ім'я співробітника і його результати) буде визначатися в момент запиту сторінки співробітником.

Ізоморфізм - можливість виконання однієї і тієї ж бази коду на сервері і клієнті. Своєю появою він зобов'язаний можливості виконувати серверний JS за допомогою Node.js, а по справжньому популярним став завдяки поширенню React. На даний момент ізоморфізм - один з найгарячіших трендів веб розробки і це найкращий час що б оцінити наслідки його появи, розглянути ізоморфізм як щабель до абсолютно новим архітектурам і рішенням.

Завдяки поточної затребуваності ізоморфізм обріс інструментами. Найпопулярнішими з них є веб сервер Express, який використовується для доставки додатків клієнту і UI бібліотека React, яка дозволяє рендерити компоненти як в DOM в браузері, так і в строковий HTML на сервері. І якщо використання Express є практично безальтернативним, то клієнтських бібліотек, які підтримують ізоморфізм існує досить велика кількість, проте незалежно від того яку з них було обрано, основна концепція ізоморфного коду не змінюється. Головним досягненням ізомофрного JavaScript стала уніфікація коду в браузері і на сервері, яка практично розмила грань між ними. Тепер головною відмінністю клієнта і сервера стає не специфічність коду, а можливість сервера роздавати контент. Але що якщо піти далі ізоморфізму і припустити, що можлива передача додатків з клієнта на клієнт? Що, якщо можливо повністю знищити різницю в функціональності серверного та клієнтського коду? Таким чином кожен клієнт, який отримав код додатка ставав би його розповсюджувачем або носієм, якщо провести аналогію з поширенням мікроорганізмів в природі, а саму техніка отримала назву віральний JavaScript (Viral Java Script).

P2P поширення контенту дозволяє зменшити навантаження на сервер, що буде цікаво громадським організаціям і некомерційним проектам, а також само зменшити затримки при доставці контенту, адже пірінгова мережа може бути налаштована таким чином, щоб контент поставлявся з найближчого доступного вузла. Наприклад, одного разу проникнувши в корпоративну мережу підприємства додаток буде поширюватися всередині по корпоративним, високошвидкісним каналам зв'язку, не навантажуючи інтернет-канал підприємства.

1.2 Аналіз застосування технології ASP.NET для створення веб-додатків

ASP.NET (Active Server Pages для .NET) - платформа розробки веб-додатків, до складу якої входять: веб-сервіси, програмна інфраструктура, модель програмування, від компанії Майкрософт. ASP.NET входить до складу платформи .NET Framework і являє собою нову версію технології Microsoft ASP.

ASP.NET зовні багато в чому зберігає схожість із старішою технологією ASP, що дозволяє розробникам відносно легко перейти на ASP.NET. У той же час внутрішній устрій ASP.NET істотно відрізняється від ASP, оскільки вона заснована на платформі .NET і, отже, використовує всі нові можливості, що надаються цією платформою.

Оскільки ASP.NET ґрунтується на Common Language Runtime (CLR), яка є основою всіх додатків Microsoft .NET, розробники можуть писати код для ASP.NET, використовуючи мови програмування, що входять в комплект .NET Framework (C #, Visual Basic.NET, J # і JScript .NET). ASP.NET має перевагу в швидкості в порівнянні з скриптовими технологіями, так як при першому зверненні код компілюється і поміщається в спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на обробку, оптимізацію і так далі.

Програмна модель ASP.NET ґрунтується над протоколом HTTP і використовує його правила взаємодії між сервером і браузером. При формуванні сторінки закладена абстрактна програмної модель Web Forms і на ній заснована основна частина реалізації програмного коду.

Переваги технології ASP.NET перед ASP:

– компільований код виконується швидше, більшість помилок відловлюється ще на стадії розробки;

– значно поліпшена обробка помилок під час виконання запущеної готової програми, з використанням блоків try..catch;

– призначені для користувача елементи управління (controls) дозволяють виділяти часто використовувані шаблони, такі як меню сайту;

– використання метафор, вже застосовуються в Windows-додатках, наприклад, таких як елементи управління і події;

– розширюваний набір елементів управління і бібліотек класів дозволяє швидше розробляти додатки;

– ASP.NET спирається на багатомовні можливості .NET, що дозволяє писати код сторінок на VB.NET, Delphi.NET, Visual C#, J# і т.д;

– можливість кешування всієї сторінки або її частини для збільшення продуктивності;

– можливість кешування даних, що використовуються на сторінці;

– можливість поділу візуальної частини та бізнес-логіки по різних файлах («code behind»);

– розширювана модель обробки запитів;

– розширена подієва модель;

– розширювана модель серверних елементів управління;

– наявність master-сторінок для завдання шаблонів оформлення сторінок;

– підтримка CRUD-операцій (Create Read Update Delete) при роботі з таблицями через GridView;

– вбудована підтримка AJAX;

– ASP.NET має перевагу в швидкості в порівнянні з іншими технологіями, заснованими на скриптах.

Платформа ASP.NET MVC являє собою фреймворк для створення сайтів і веб-додатків за допомогою реалізації паттерна MVC. Концепція паттерна (шаблону) MVC передбачає поділ додатка на три компоненти.

Контролер (controller) представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує ведені

користувачем дані та обробляє їх. В залежності від результатів обробки відправляє користувачу певний висновок, наприклад, у вигляді подання.

Подання (view) - це візуальна частина або призначений для користувача інтерфейс програми. Як правило, html-сторінка, яку користувач бачить, зайшовши на сайт.

Модель (model) представляє клас, що описує логіку використовуваних даних.

Загальну схему взаємодії цих компонентів можна представити таким чином:

у даній схемі модель є незалежним компонентом - будь-які зміни контролера або подання не зачіпають модель. Контролер і уявлення є відносно незалежними компонентами, і нерідко їх можна змінювати незалежно один від одного.

Завдяки цьому реалізується концепція поділу відповідальності, в зв'язку з чим легше побудувати роботу над окремими компонентами. Крім того, внаслідок цього додаток краще піддається тестуванню. І якщо, припустимо, важлива візуальна частина або фронтенд, то можна тестувати уявлення незалежно від контролера. Або можна зосередитися на бекенді і тестувати контролер.

Конкретні реалізації та визначення даного патерну можуть відрізнятися, але в силу своєї гнучкості і простоти він став дуже популярни, особливо в сфері веб-розробки.

Свою реалізацію паттерна представляє платформа ASP.NET MVC. 2013 рік ознаменувався виходом нової версії ASP.NET MVC - MVC 5, а також релізом Visual Studio 2013, яка надає інструментарій для роботи з MVC5.

Хоча в багатьох аспектах MVC 5 не дуже сильно відрізнятиметься від MVC 4, багато принципів з однієї версії цілком можна застосувати до іншої, але в той же час є і суттєві відмінності:

У MVC 5 змінилася концепція аутентифікації і авторизації. Замість SimpleMembershipProvider була впроваджена система ASP.NET Identity, яка використовує компоненти OWIN і Katana.

Для створення адаптивного і розширюваного інтерфейсу в MVC 5 використовується css-фреймворк Bootstrap.

Додані фільтри аутентифікації, а також з'явилася функціональність перевизначення фільтрів. У MVC 5 також додані атрибути маршрутизації.

Це найбільш важливі нововведення в MVC 5. Крім того, є ще ряд менш значимих, наприклад, використання за замовчуванням Entity Framework 6, деякі зміни при створенні проекту (концепція One ASP.NET), додаткові компоненти і т.д.

У будь-якому випадку всі отримані при роботі з MVC 4 навички можна успішно застосовувати при використанні MVC 5, враховуючи, звичайно, нововведення.

1.3 Вибір програмного засобу для тестування

Для розробки системи автоматизованого тестування обрано незалежне середовище розробки Visual Studio NUnit.

NUnit - відкрите середовище юніт-тестування додатків для .NET. Вона була перенесена з мови Java (бібліотека JUnit). Перші версії NUnit були написані на J#, але потім весь код був переписаний на C# з використанням таких нововведень .NET, як атрибути.

Існують також відомі розширення оригінального пакету NUnit, велика частина з них також з відкритим вихідним кодом. NUnit.Forms доповнює NUnit засобами тестування елементів призначеного для користувача інтерфейсу Windows Forms. NUnit.ASP виконує ту ж задачу для елементів

інтерфейсу в ASP.NET. Всі фреймворки з сімейства NUnit мають такі базові компоненти архітектури, які в різних реалізаціях можуть злегка варіюватися.

Модуль представляє собою виконувану програму, яка виконує тести, реалізовані за допомогою фреймворка, і відображає інформацію про хід їх виконання.

Варіанти тестування (тестові сценарії/випадки) є базовими елементами модульних тестів.

Конфігурація тестування (також звана контекстом) - це набір попередньо заданих умов або станів об'єктів, необхідний для запуску тесту. Розробник повинен задати свідомо коректну конфігурацію перед виконанням кожного тесту, а потім повернути оригінальну конфігурацію після завершення тесту.

Тестовий набір - це кілька тестів, що мають загальну конфігурацію. Черговість виконання тестів не повинна мати значення.

Виконання кожного тесту відбувається за такою схемою:

```
setup (); / * Спочатку готується 'контекст' тестування * /
...
/ * Тіло тесту - тут вказується тестовий сценарій * /
...
teardown (); / * Після проходження тесту (незалежно від його результату) контекст тестування "очищається" * /
```

Модуль, який виконує тестування, повинен вивести результати в одному або декількох заданих форматах. На додаток до звичайного тексту, сприймається людиною, часто результати виводяться у форматі XML.

Затвердження в тесті - це функція або макрос, яка перевіряє поводження або стан модуля, що тестується. Часто твердженням є перевірка рівності або нерівності деякого параметра модуля очікуваного результату. Невдале проходження перевірки призводить до провалу всього тестового

сценарію і (якщо необхідно) до виключення, яке зупиняє сценарій без переходу до наступного твердження.

Автоматизовані тести мають низку переваг:

- висока якість програми;
- зниження вартості;
- безпека регресії мережі.

Чим вище якість програми, тим менше коштів витрачається на усунення недоліків проекту. Тобто, якщо знайти недоліки в проекті на ранньому етапі, вирішити їх буде дешевше.

Автоматизоване тестування програмного забезпечення - частина процесу тестування на етапі контролю якості в процесі розробки програмного забезпечення. Воно використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити його процес.

Існує два основні підходи до автоматизації тестування: тестування на рівні коду і тестування користувацького інтерфейсу (зокрема, GUI-тестування). До першого типу відносяться, зокрема, модульне тестування. До другого - імітація дій користувача - функціональне тестування (за допомогою спеціальних тестових фреймворків).

Найбільш поширеною формою автоматизації є тестування додатків через графічний користувацький інтерфейс (англ. GUI). Популярність такого виду тестування пояснюється двома факторами: по-перше, додаток тестується тим же способом, яким його буде використовувати людина, по-друге, можна тестувати додаток, не маючи при цьому доступу до вихідного коду.

GUI-автоматизація розвивалася протягом 4 поколінь інструментів і технік:

Утиліти запису і відтворення (англ. Capture / playback tools) записують дії тестувальника під час ручного тестування. Вони дозволяють виконувати тести без прямої участі людини протягом тривалого часу, значно збільшуючи

продуктивність і усуваючи повторення одноманітних дій під час ручного тестування. У той же час, будь-яке мале зміна тестованого ПЗ (Програмного Забезпечення) вимагає перезапису ручних тестів. Тому це перше покоління інструментів не ефективно і не масштабоване.

Написання сценарію (англ. Scripting) - форма програмування на мовах, спеціально розроблених для автоматизації тестування ПЗ - пом'якшує багато проблем інструментів запису і відтворення. Але розробкою займаються програмісти високого рівня, які працюють окремо від тестувальників, безпосередньо запускають тести. До того ж скрипти найбільше підходять для тестування GUI і не можуть бути впровадженими, пакетними або взагалі будь-яким чином об'єднані в систему. Нарешті, зміни в тестованому ПЗ вимагають складних змін у відповідних скриптах, і підтримка все зростаючої бібліотеки тестують скриптів стає врешті-решт непереборної завданням.

Кероване даними тестування (англ. Data-driven testing) - методологія, яка використовується в автоматизації тестування. Особливістю є те, що тестові скрипти виконуються і верифицируются на основі даних, які зберігаються в центральному сховищі даних або базі даних. Роль бази даних можуть виконувати ODBC-ресурси, csv або xls файли і т. Д. Керований даними тестування - це об'єднання декількох взаємодіючих тестових скриптів і їх джерел даних у фреймворк, який використовується в методології. У цьому фреймворку змінні використовуються як для вхідних значень, так і для вихідних перевірочних значень: в тестовому скрипті зазвичай закодовані навігація по додатком, читання джерел даних, ведення логів тестування. Таким чином, логіка, яка буде виконана в скрипті, також залежить від даних.

Тестування за ключовими словами (англ. Keyword-based) автоматизація передбачає поділ процесу створення кейсів на 2 етапи: етап планування і етап реалізації. У цьому випадку кінцевий тест являє собою не програмний код, а опис послідовності дій з їх параметрами (наприклад, «завести в базі даних користувача з логіном XXX і паролем YYY»). При цьому фреймворк відповідає за безпосередню реалізацію ключових слів (дій), а дизайнерові

тестів досить мати уявлення про всьому наборі дій, реалізованих у фреймворку. Це дає можливість створювати тести людям, які не мають навичок програмування.

Приклад тесту NUnit:

```
using NUnit.Framework;

[TestFixture]
public class ExampleTestOfNUnit
{
   [Test]
   public void TestMultiplication()
   {
      Assert.AreEqual(4, 2 * 2, "Множення");
   }
}
```

Приклад написано на мові C#, з використанням функцій .NET. Атрибут класу ExampleTestOfUnit [TestFixture] позначає для NUnit даний клас, як тестовий, тобто клас, який містить в собі тестові методи або інші дані, необхідні для тестів. Атрибутом [Test] помічено метод, який являє собою тестовий метод і даний метод буде відображено в менедженрі тестів NUniet Runner.

1.4 Постановка задачі

Основна задача розробки модульного тестування – створення максимально самостійних тестових модулів, які підлаштовуються під зміни у системі, розробка тестів, які виконуються за мінімальний проміжок часу та не навантажують систему, написання модулів, які перевірятимуть усі аспекти та можливі сценарії роботи з системою.

Необхідність у розробці модульних тестів продиктована наступними вимогами до системи.

Автоматичні тести дають впевненість, що ваша програма працює як задумано. Такі тести можна запускати багато разів. Успішне виконання тестів покаже розробнику, що його зміни не зламали нічого, що ламати не планувалося.

Провалений тест дозволить виявити, що в коді зроблені зміни, які змінюють або ламають його поведінку. Дослідження помилки, яку видає провалений тест, і порівняння очікуваного результату з отриманим дає можливість зрозуміти, де виникла помилка, будь вона в коді або у вимогах.

Юніт-тести можуть служити в якості документації до коду. Грамотний набір тестів, який покриває можливі способи використання, обмеження та потенційні помилки, нітрохи не гірше спеціально написаних прикладів, і, крім того, його можна скомпілювати і переконатися в коректності реалізації.

Якщо тести легко використовувати (а їх повинно бути легко використовувати), то іншої документації (наприклад, коментарів doxygen) не потрібно.

Для досягнення поставленої мети роботи, потрібно вирішити ряд таких завдань :

– провести аналіз існуючих систем тестування та веб-додатку вцілому;

– розробити та описати методи та алгоритми, які будуть використані для реалізації тестового модулю;

– впровадити та реалізувати розроблений модуль;

– описати та обрахувати економічної складової проекту, витрат та кошторисів.

2 МЕТОДИ ТА АЛГОРИТМИ ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЇ ВЕБ-ДОДАТКУ

2.1 Обґрунтування вибору методу

Нижче коротко описані загально-прийняті методи тестування веб-додатків.

Функціональне тестування - це тестування програмного забезпечення (ПЗ) з метою перевірки можливості бути реалізованим функціональних вимог, тобто здатності ПЗ в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить ПЗ, які завдання воно вирішує.

Функціональні вимоги включають в себе:

- функціональна придатність (англ. Suitability);
- точність (англ. Accuracy);
- здатність до взаємодії (англ. Interoperability);
- відповідність стандартам і правилам (англ. Compliance);
- захищеність (англ. Security).

Перевірка ергономічності (юзабіліті-тестування, англ. Usability testing) - дослідження, яке виконується з метою визначення, чи зручний деякий штучний об'єкт (такий як веб-сторінка, призначений для користувача інтерфейс або пристрій) для його передбачуваного застосування. Таким чином, перевірка ергономічності вимірює ергономічність об'єкта або системи. Перевірка ергономічності зосереджена на певному об'єкті або невеликому наборі об'єктів, в той час як дослідження взаємодії людина-комп'ютер в цілому - формулюють універсальні принципи.

Перевірка ергономічності - метод оцінки зручності продукту у використанні, заснований на залученні користувачів в якості тестувальників, випробувачів і підсумовуванні отриманих від них висновків.

При випробуванні багатьох продуктів користувачеві пропонують в «лабораторних» умовах вирішити основні завдання, для виконання яких цей продукт розроблявся, і просять висловлювати під час виконання цих тестів свої зауваження.

Процес тестування фіксується в протоколі (балці) і / або на аудіо-та відеотехніка - з метою подальшого більш детального аналізу.

Якщо перевірка ергономічності виявляє будь-які труднощі (наприклад, складності в розумінні інструкцій, виконання дій або інтерпретації відповідей системи), то розробники повинні доопрацювати продукт і повторити тестування.

Спостереження за тим, як люди взаємодіють з продуктом, нерідко дозволяє знайти для нього більш оптимальні рішення. Якщо при тестуванні використовується модератор, то його завдання - тримати респондента сфокусованим на завданнях (але при цьому не «допомагати» йому вирішувати ці завдання).

Тестування крос-браузерності - вид тестування, спрямований на підтримку і правильне повне відображення програмного продукту в різних браузерах, мобільних пристроях, планшетах, екранах різного розміру.

Крос-браузерне тестування (cross-browser testing) - важливий етап при розробці будь-якої програми. Адже зовнішній вигляд сайту і його коректне відображення на будь-якому сучасному пристрої грає визначальну роль для замовника.

Cross-browser testing сайту починається з вибору браузерів. Замовник сам визначає, з якими саме веб-оглядачами буде працювати його додаток. Але завдання розробника і тестувальника - підказати клієнтові, який браузер буде головним, слід вивчити статистику заходів подібних додатків, визначити якими браузерами користується така аудиторія.

Як правило, розглядають найпопулярніші браузери: Google Chrome, Mozilla Firefox, Internet Explorer, Opera.

Тестування навантаження (англ. Load testing) - підвид тестування продуктивності, збір показників і визначення продуктивності і часу відгуку програмно-технічної системи або пристрої у відповідь на зовнішній запит з метою встановлення відповідності вимогам, що пред'являються до даної системи (пристрою).

Для дослідження часу відгуку системи на високих або пікових навантаженнях проводиться стрес-тестування, при якому створюється на систему навантаження перевищує нормальні сценарії її використання. Не існує чіткої межі між навантажувальним і стрес-тестуванням, однак ці поняття не варто змішувати, так як ці види тестування відповідають на різні бізнес-питання і використовують різну методологію.

Термін тестування навантаження може бути використаний в різних значеннях в професійному середовищі тестування ПЗ. У загальному випадку він означає практику моделювання очікуваного використання додатка за допомогою емуляції роботи декількох користувачів одночасно. Таким чином, подібне тестування найбільше підходить для багатокористувацьких систем, частіше - використовують клієнт-серверну архітектуру (наприклад, веб-серверів). Однак і інші типи систем ПЗ можуть бути протестовані подібним способом. Наприклад, текстовий або графічний редактор можна змусити прочитати дуже великий документ; а фінансовий пакет - згенерувати звіт на основі даних за кілька років. Найбільш адекватно спроектований навантажувальний тест дає більш точні результати.

Основна мета навантажувального тестування полягає в тому, щоб, створивши певну очікувану в системі навантаження (наприклад, за допомогою віртуальних користувачів) і, звичайно, використавши ідентичне програмне і апаратне забезпечення, спостерігати за показниками продуктивності системи.

Для розробки модульних тестів для організаційної системи було обрано функціональний та навантажувальний методи тестування.

2.2 Алгоритми тестування на основі ASP.NET

До запуску програми в виробництво, коли воно стане доступним користувачам, важливо переконатися, що дане додаток функціонує, як і повинно, що в ньому немає помилок. Для перевірки додатку ми можемо використовувати різні схеми і механізми тестування. Одним з таких механізмів є юніт-тести.

Юніт-тести дозволяють швидко і автоматично протестувати окремі компоненти програми незалежно від іншої його частини. Не завжди юніт-тести можуть покрити весь код програми, але тим не менше вони дозволяють істотно зменшити кількість помилок вже на етапі розробки.

Ми не повинні тестувати код використовуваного фреймворка або використовуваних залежностей. Тестувати треба тільки той код, який написали ми самі.

Треба відзначити, що в цілому концепція юніт-тестів не є непорушною вимогою до веб-розробці, та й взагалі до розробки. Хтось вважає, що юніт-тести обов'язково повинні покривати весь код проекту, хтось вважає, що юніт-тести можна використовувати переважно для особливо складних моментів у коді програми, якоїсь складної логіки. Деякі не використовують юніт-тести.

Але тим не менше юніт-тести несуть потенційні переваги при розробці, до яких слід віднести не тільки власне перевірку результату і тестування коду, але і інші, як наприклад, написання слабо-зв’язанних компонентів відповідно до принципів SOLID. Адже щоб тестувати компоненти програми незалежно один від одного, нам треба, щоб вони були слабо зв'язаної. А подібне побудова додатки в подальшому може позитивно позначитися на його подальшій модифікації і підтримки.

Для створення юніт-тестів вибираються невеликі ділянки коду, які треба протестувати. Тестований ділянку, як правило, повинен бути менше

класу. У більшості випадків тестується окремий метод класу або навіть частина функціоналу методу. Упор на невеликі ділянки дозволяє досить швидко писати простенькі тести.

Одного разу написаний код нерідко читають багато разів, тому важливо писати зрозумілий код. Особливо це важливо в юніт-тестах, де в разі невдачі при тестуванні розробник повинен швидко прочитати вихідний код і зрозуміти в чому проблема і як її виправити. А використання невеликих ділянок коду значно спрощує подібну роботу.

При тестуванні важливо ізолювати тестований код від решти програми, з якою він взаємодіє, щоб потім чітко визначити можливість помилок саме в цьому ізольованому коді. Що спрощує і підвищує контроль над окремими компонентами програми.

Створення юніт-тестів для невеликих ділянок коду веде до того, що кількість цих юніт-тестів стає дуже велике. Якщо процес отримання результатів і проведення тестів не автоматизовано, то це може привести до непродуктивної витраті робочого часу і зниження продуктивності. Тому важливо, щоб результати юніт-тестів представляли собою просте рішення, яке означає, пройдений тест чи ні. Для автоматизації процесу розробники зазвичай звертаються до фреймворками юніт-тестування

Навіть невеликі зміни в класі можуть привести до невдачі багатьох юніт-тестів, оскільки реалізація використовуваного класу змінилася. Тому при написанні юніт-тестів обмежуються тільки загальнодоступними кінцевими точками, що дозволяє ізолювати юніт-тести від багатьох деталей внутрішньої реалізації компонента. В результаті зменшується ймовірність, що зміни в класах можуть привести до провалу юніт-тестів.

Інфраструктура ASP.NET MVC Framework спроектована для максимального полегшення підготовки автоматизованих тестів і використання таких методологій, як розробка через тестування (Test-Driven Development - TDD), яка буде описана далі. ASP.NET MVC надає ідеальну платформу для автоматизованого тестування, а в Visual Studio є ряд зручних

засобів тестування. Всі разом це перетворює розробку і прогін тестів в просту і легко вирішуване завдання.

У широкому сенсі сучасні розробники веб-додатків зосереджують свою увагу на двох видах автоматизованого тестування. Перший вид - це модульне тестування, що є способом вказівки і перевірки поведінки окремих класів (або інших невеликих блоків коду) в ізоляції від решти частини програми. Другий вид - інтеграційне тестування, яке представляє собою спосіб вказівки і перевірки поведінки безлічі компонентів, які працюють спільно, аж до цілого веб-додатки.

У веб-додатках можуть бути корисні обидва види тестування. Модульні тести, які легко створювати і запускати, блискуче точні при роботі з алгоритмами, бізнес-логікою та іншими частинами внутрішньої інфраструктури. Цінність інтеграційного тестування полягає в тому, що воно може моделювати взаємодію користувача з призначеним для користувача інтерфейсом і покривати весь стек технологій, застосовуваних в додатку, включаючи веб-сервер і базу даних.

Як правило, інтеграційне тестування краще підходить для виявлення нових програмних помилок, що виникають в уже готових функціональних засобах; таке тестування називають регресійний тестуванням.

У світі .NET для зберігання тестових оснасток створюється окремий тестовий проект в рішенні Visual Studio. Цей проект буде створюватися при першому додаванні модульного тесту або ж встановлюватися автоматично в разі використання шаблону проекту Model-View-Controller (MVC). Тестова оснащення - це клас C#, який визначає набір тестових методів, по одному для кожного поведінки, що потребує перевірки. Тестовий проект може містити безліч класів тестових оснасток.

Далі буде описано приклад просто тесту. Для початку створимо клас з уявного додатку, як показано в прикладі нижче. Цей клас має ім'я AdminController і визначає метод ChangeLoginName, який дозволяє уявним

користувачам змінювати свої паролі. Для цієї демонстрації класи створені в новому проекті Visual Studio під назвою TestingDemo (додаток А).

Контролер спирається на ряд класів моделі і інтерфейс, код яких наведено в прикладі нижче. Ці класи не відносяться до якогось реального проекту; вони навмисно спрощені для полегшення демонстрації тестування. Наприклад, тут не передбачається, що ви будете створювати класи, що представляють користувачів, які мають всього лише одне строкове властивість (додаток Б).

Клас User представляє користувача всередині програми. Користувачі створюються, керуються і зберігаються в сховищі, функціональність якого визначена інтерфейсом IUserRepository, і неповна реалізація цього інтерфейсу виконана в класі DefaultUserRepository.

Мета, яка переслідується в цьому розділі, полягає в написанні модульного тесту для функціональності, що надається методом ChangeLoginName() класу AdminController (додаток В).

Тестової оснащенням є метод CanChangeLoginName(). Зверніть увагу, що цей метод декорований атрибутом Test, а клас, якому він належить - AdminControllerTests - атрибутом TestFixture. Саме так NUnit Runner знаходить тестову оснащення.

Структура методу CanChangeLoginName() слідує шаблоном, який відомий під назвою організація/дію/твердження (Arrange/Act/Assert - AAA). Організація відноситься до налаштування умов тесту, дія - до виконання тесту, а твердження - до перевірки того, що результат виявився тим, який був потрібний. Дотримання узгодженості структури методів, що реалізують модульні тести, сприяє їх кращої читабельності - характеристика, яку ви оціните по достоїнству, якщо проект буде містити багато сотень модульних тестів.

Тестова оснащення з прикладу використовує специфічну для тесту фіктивну реалізацію інтерфейсу IUserRepository, щоб емулювати конкретна умова - в даному випадку ситуацію, коли єдиний об'єкт учасника аукціону,

Bob, присутній в сховище. Створення фіктивного сховища і об'єкта User здійснюється в розділі організації тесту.

Потім викликається тестований метод - AdminController.ChangeLoginName. Це розділ дії тесту. І, нарешті, результат перевіряється за допомогою пари викликів методів класу Assert (це розділ затвердження тесту). Клас Assert надається тестовим набором Visual Studio (простір імен NUnit.Frameworki дозволяє перевіряти специфічні результати.

Запуск тесту проводиться за допомогою менеджеру тестів NUnit Runner. У міру виконання тестів ми отримуємо візуальний відгук, як показано на рисунку 2.1.

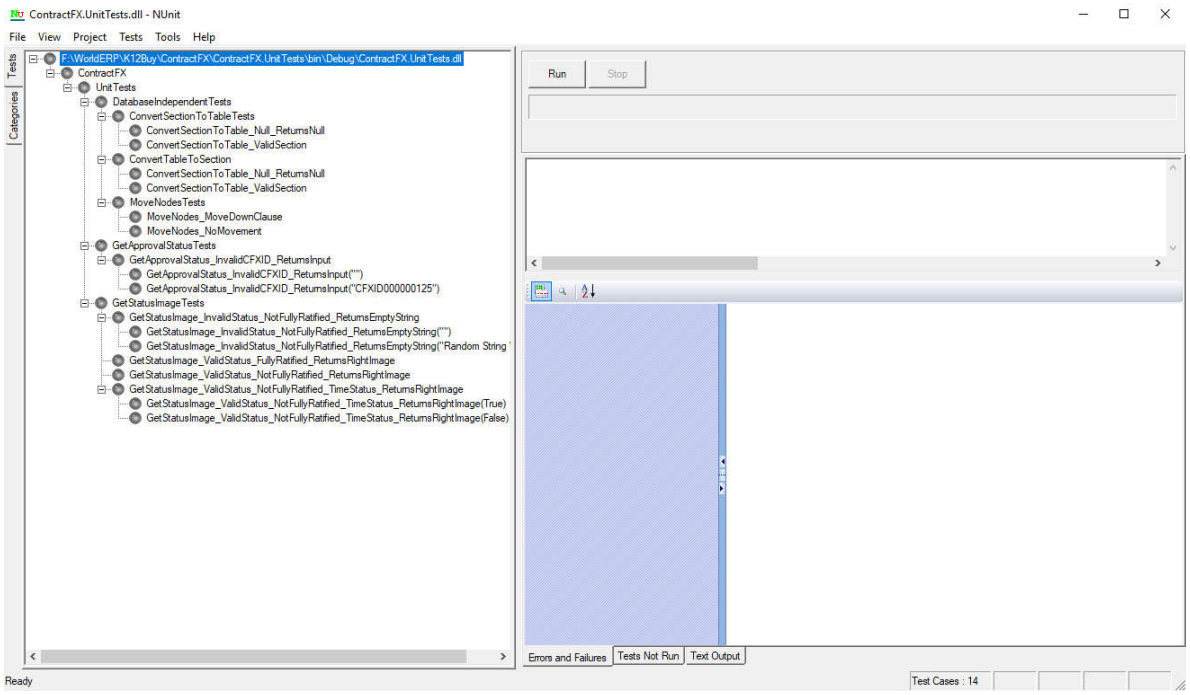

Рисунок 2.1 – Меню запуску тесту

Якщо тестова оснащення запускається без генерації будь-яких необроблених винятків і всі оператори виклику методів класу Assert виконуються без проблем, вікно Test Explorer (Провідник тестів) відображає зелений маркер. В іншому випадку ми побачимо червоний маркер і подробиці про проблему.

Як бачите, застосування DI (Dependency Injection – Впровадження Залежностей) полегшило модульне тестування. Ми змогли створити фіктивну реалізацію сховища і впровадити її в контролер для створення специфічного сценарію. Це одна з причин популярності шаблону проектування DI.

Може здатися, що тестування простого методу вимагало непропорційно великих зусиль, однак обсяг коду для тестування чогось значно більш складного виявився б набагато більшими. Якщо виникає бажання відмовитися від дрібних тестів, аналогічних описаним вище, то врахуйте, що такі тестові оснащення допомагають виявити помилки, які іноді залишаються непоміченими при виконанні більш складних тестів. Одне з можливих удосконалень, яке можна було б зробити - виключити специфічні для тесту фіктивні класи, подібні FakeRepository за рахунок застосування інструменту імітації.

2.3 Математична модель та алгоритм для верифікації програмної системи

Нехай *f(T)* – тестовий метод, тоді:

$$f(T) = F(f(D)/f(A)),$$

де *F* – методи фреймворку,

f(D) – функція ініціалізації даних,

f(A) – функція порівняння

Так, як автоматизовані тести не представляють математичної складності, для доповнення математичної моделі далі приведено декілька схем (додаток Д).

Схема алгоритму тестування являє собою досить просту схему, яку зображено на рисунку 2.2.

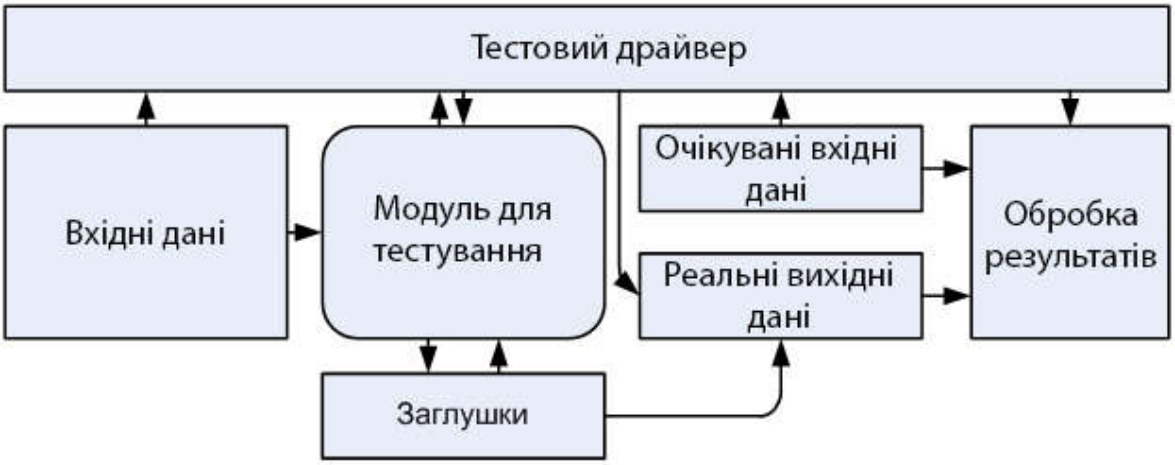

Рисунок 2.2 – Схема алгоритму тестування

Тестовим драйвером в даній схемі виступає фреймворк NUnit. На схемі відображено взаємодію між абстрактними елементами тестового модулю. Майже всі обрахування проводяться функціями фреймворку. Вхідні дані – це набір функцій та методів, необхідних для ініціалізації даних, які далі будуть використовуватись тестовим модулем. Модуль для тестування – безпосередньо модулі веб-додатку, які піддаються тестування. Заглушки – це методи та дані, необхідні для ізоляції модуль тестування. Очікувані та реальні вхідні дані - результат гіпотетичний результат роботи модулю. Обробка результатів – засоби порівняння очікуваного результату роботи та рельного.

3 ЗАСТОСУВАННЯ РОЗРОБЛЕНИХ ТЕСТІВ ТА ПРОГРАМНИХ МОДУЛІВ ДЛЯ ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЇ ДОДАТКІВ

3.1 Структура програмного модулю

Структура програмного модулю складається з наступних елементів:

- розділ визначення змінних;
- розділ процедур і функцій;
- розділ основної програми.

Розділ визначення змінних представляє собою набір методів та полів, а також «магічних чисел», необхідних для виконання тестів. Наприклад, для тестування методу Add() в програмі-калькуляторі, необхідно передати методи будь-які (які підходять по типу та стандарту) дані, а результат перевірити.

Розділ процедур і функцій включає в себе безпосередньо самі тестові методи та функції, які застосовується в процесі роботи тестового модулю. В даних методах виконуються виклики, обробки, порівняння, ініціалізація «фейків», тощо.

Розділ основної програми включає в себе заглушки та «стаби». Тобто, даний розділ представляє собою набір класів та методів для «підробки» функціональних одиниць, які використовуються тестованим модулем. Також, даний розділ включає в себе набір полів, необхідних для роботи модулю.

Для обробки даних було створено структура даних. Вона вміщує в собі поля, необхідні для проведення тесту на створення контракту в системі ContractFX. Перед виконанням тестів на створення контракту викликається клас Data.cs, який ініціалізує поля структури.

Для генерації відповідних екземплярів класів, необхідних для тестування окремих функцій було створено додаткові методи, які викликаються за необхідності.

Структура модулю ділиться на незалежні від бази даних та залежні тести, як зображено на кресленні ДП.КСМ.07238/16.00.00.000 С1.

Код веб-додатку – це модуль або абстрактний процес, який безпосередньо викликається з тестового модуля для перевірки правильності роботи модулю та/або перевірки на наявність помилок.

Фреймворк NUnit – функції та інструменти фреймворку NUnit. NUnit Framework вміщує в собі безліч інструментів для розробки та роботи модульних тестів. Деякі з даних інструментів використовуються в даному модульному тесті.

Проект модульного тестування включає в себе тестові методи та класи – засоби, використовувані для тестування.

В структурі описано п’ять класів та п’ять тестових методів. Методи класів ConvertSectionToTableTests, ConvertTableToSectionTests та MoveNodesTests являють собою незалежні тести від бази даних, тобто для тестування окремих модулів веб-додатку немає потреби в базі даних. Всі інші класи ж навпаки, залежні від бази даних.

Метод Setup() в дані схемі представляє собою ініціалізатор, який готує дані, необхідні для тесту. Метод Setup() є абстрактним, тому кожен клас, якому для роботи необхідно оголосити деякі дані наслідує інтерфейс ITestBase і реалізовує метод.

3.2 Опис алгоритмічної реалізації модульних тестів для ContractFX

Реалізація алгоритму для розробленого модуля тестування зображено на кресленні ДП.КСМ.07238/16.00.00.000 А2.

Після запуску обраного тесту модуль спробує завантажити фреймворк для використання додаткових функцій. Якщо завантаження пройшло успішно, тест продовжить своє виконання. Якщо спроба завантажити

фреймворк провалилась, тест буде завершено з повідомленням та описом відповідної помилки.

Далі, буде викликано метод Setup(), яки проведе ініціалізацію даних, необхідних для виконання тесту. Лише після успішного виконання попередніх функцій буде викликано тестовий метод, який, безпосередньо, розпочне тестове створення контракту.

Якщо під час порівняння результатів не виникло помилок та всі порівняння виявились вдалими, тест вважається успішним і тестовий модуль завершує роботу з повідомленням про успішне виконання. Якщо ж тест провалився, модуль завершить свою роботу та виведе повідомлення з описанням помилки.

В даній алгоритмічній схемі етап звернення до класу Data.cs представляє собою стандартну для всіх кдасів ініціалізацію певного набору даних, необхідних для успішної роботи окремих одиниць тестового модулю вцілому.

Підготовка даних включає в себе процеси обробки даних, а також, що більш важливо – налаштування станів використовуваних класів веб-додатку, тобто встановлення полів, перевизначення віртуальних методів тощо.

Також, підготовка даних включає безпосередньо виклики тестованих методів, властивостей та конструкторів для подальшої перевірки правильності їх функціонування через порівняння результатів. Якщо ж функціональна одиниця не повертає ніяких результатів, але змінює стан класів, правильність роботи даних методів перевіряється через порівняння значень окремих полів класів.

Етап виконання перевірок являє собою набір інструкцій, які виконують порівняння результатів, які повертає тестований метод і визначених наперед результатів.

Для методів, які повертають значення, алгоритм представлено на рисунку 3.3.

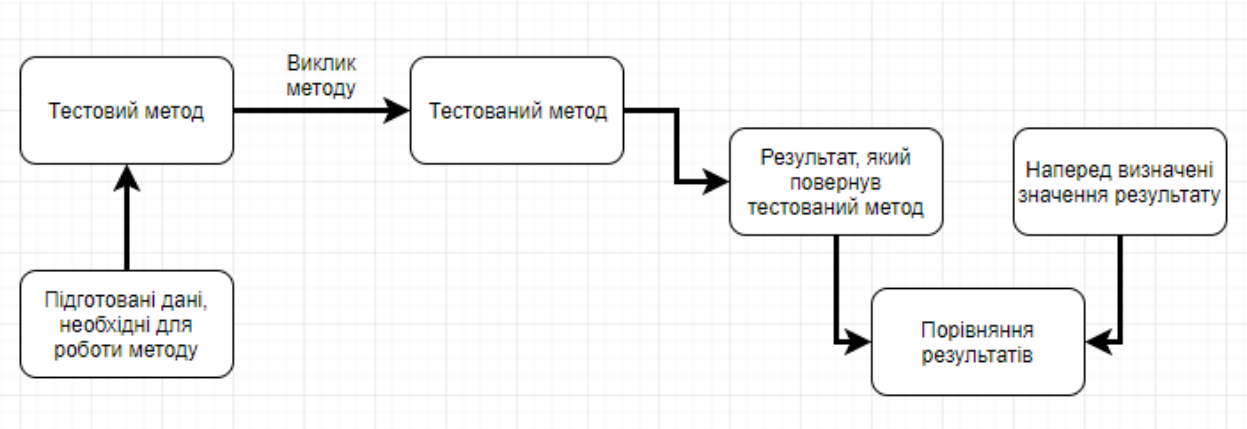

Рисунок 3.3 – Алгоритм тестування методу, який повертає значення

А на рисунку 3.4 зображено алгоритм для методу, який не повертає ніяких значень.

Рисунок 3.4 – Алгоритм тестування методу, який змінює стан об’єкту

Етап виведення повідомлення про помилку виникає лише тоді, коли тест провалився, тобто тестований метод працює не так, як очікувалось або тест написаний невірно. На даному етапу фреймворк NUnit виводить повідомлення про помилку. Як правило (залежно від інтерфейсу) повідомлення включає назву помилки в .NET, рядок тестового методу, на якому сталася помилка та опис причини помилки, наприклад якщо очікувалось число 20, а метод повернув 21, опис причини помилки виглядатиме так:

Помилка в рядку 515 файлу Tests.cs:

Метод AreEqual класу Assert повернув виключення:

Очікувано: 20

Але було: 21

На даному етапі виконання тестового методу завершується. Для окремих методів алгоритм виконання може дещо відрізнятись, але ці різниця достатньо не суттєва, для включення її в схему.

3.3 Порівняльний аналіз ефективності створеного програмного модулю для тестування та верифікації веб-додатку

Так, як розроблювана системи є нововведенням, елементів для проведення порівняльного аналізу недостатньо, тому далі просто перераховано переваги та недоліки даної системи.

Як і все в нашому недосконалому світі, модульне тестування має не тільки переваги, а й недоліки.

По-перше, модульне тестування дозволяє бути впевненим, що після виправлення однієї помилки у вас не з'явиться інша. Оскільки тести працюють в автоматичному режимі, ви завжди можете відстежити описану вище ситуацію (провівши, тим самим, регресійні тестування). Ще один плюс, який, за великим рахунком, групується з першим, - unit-тести дозволяють безболісно проводити рефакторинг коду (тобто зміна самого коду без якого б то не було зміни його функціональності). Багато хто навіть стверджують, що модульні тести заохочують рефакторинг коду розробниками.

Друга перевага полягає в тому, що unit-тести служать деяким чином документацією до коду і путівником по ньому. І справді, тести можуть розповісти, як та чи інша функції повинні реагувати на різні вхідні параметри

- опис цієї поведінки часто по не цілком зрозумілих причин (всі розуміють, що основна причина - лінь і робота в режимі "це потрібно було реалізувати ще вчора") відсутня в коментарях, що залишаються програмістами.

Третя перевага полягає в тому, що застосування unit-тестів дає більш якісне відділення інтерфейсу від реалізації. Оскільки в рамках одного тесту перевіряється один клас, то все "порочні" зв'язку з іншими класами, не передбачені архітектурою додатки, спливають на поверхню і безжальним чином розриваються.

Перший недолік - цілком логічне продовження останнього плюса. Якщо ви тестуєте кожен елемент програмного коду окремо від інших, ніхто не зможе сказати напевно, що станеться, коли всі елементи коду зібрати в єдине ціле. Тому не обійтися без додаткових інтеграційних тестів.

Наступний недолік полягає в тому, що при недостатньо продуманій архітектурі проекту тести можуть заважати зміни коду, а не сприяти йому. Ситуація не така вже й рідкісна, а тому і далеко не остання в списку неприємностей.

Але найгірше те, що навіть тести, успішно виконані і перевиконані після внесення усіляких змін, не дають ніякої гарантії, що в коді немає помилок. І справа тут навіть не в підготовці тестів під код, а в людському факторі. Іноді дещо змінивши значення вхідних параметрів в рамках тестів програміст може, сам того не бажаючи, пропустити помилку в коді, який тестується або, гірше того, внести її туди. Але, на жаль, від помилок, пов'язаних з людським фактором, не застрахований ніхто і ніколи.

3.4 Демонстрація роботи тестового модулю

Для демонстрації роботи тестового модуля, тобто для запуску тестів було використано менеджер тестів NUnit Runner, вікно якого показано на рисунку 3.5.

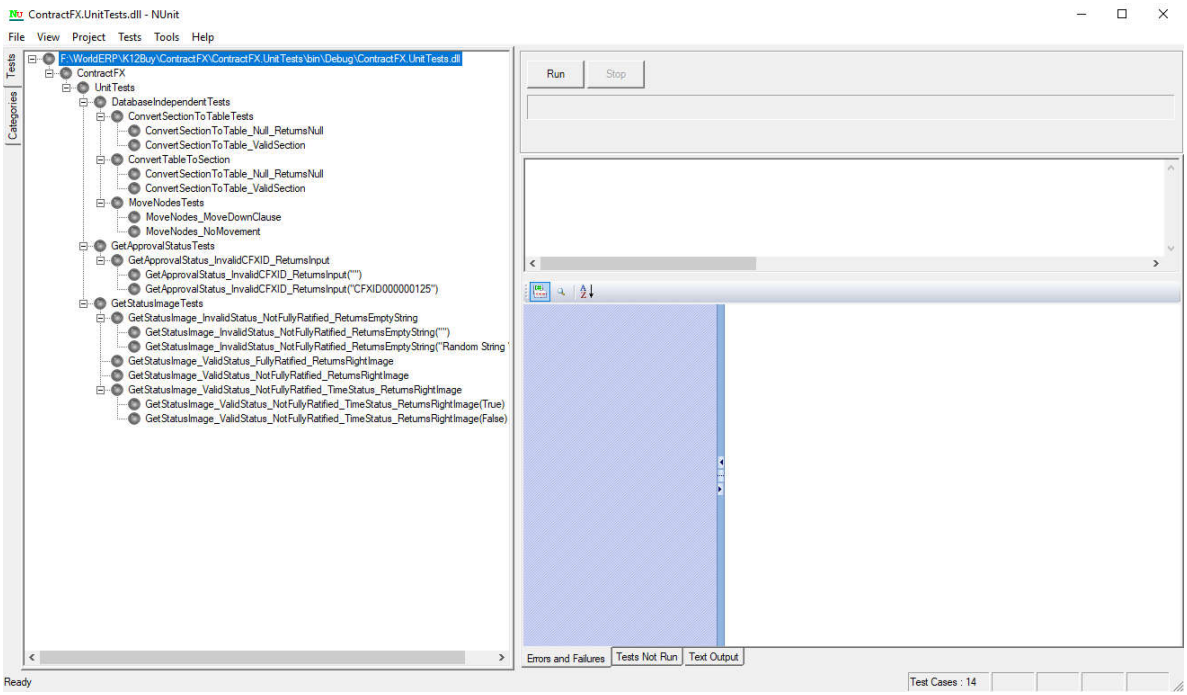

Рисунок 3.5 – Вікно NUnit Runner

У розділі справа менеджер відображає усі наявні тести, завантажені через динамічну бібліотеку (dll) або за допомогою проектного рішення NUnit зображено на рисунку 3.6.

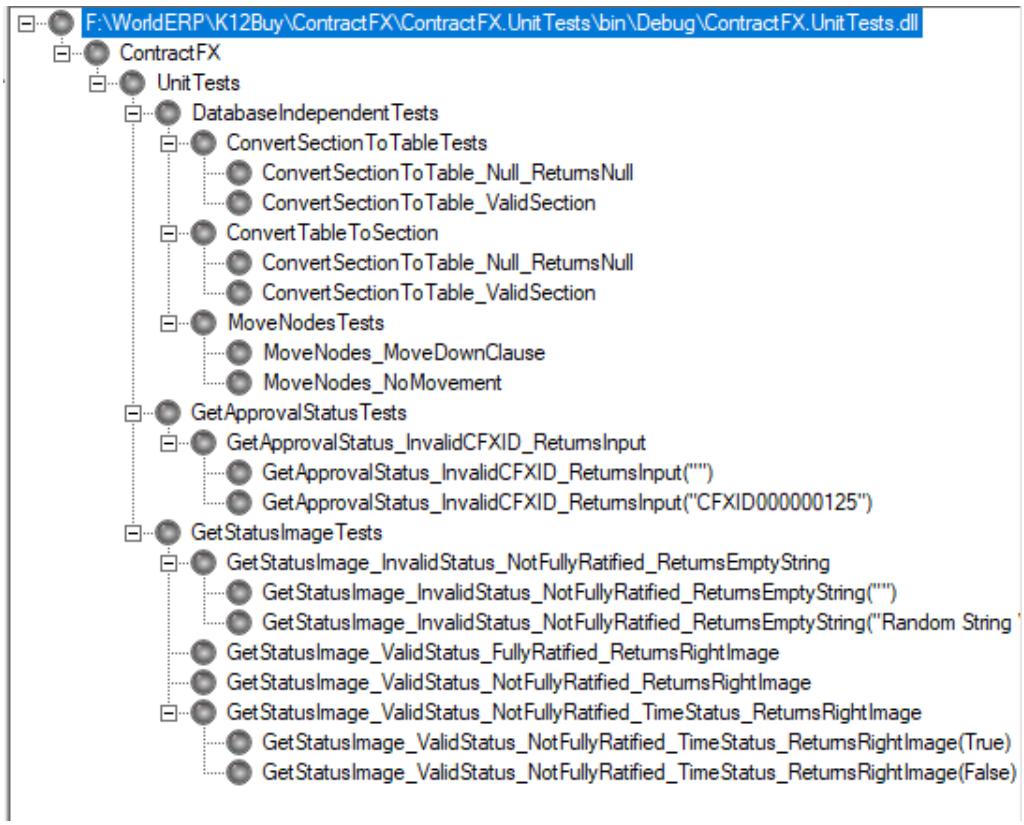

Рисунок 3.6 – Перелік тестів, отриманих з бібліотеки

Для роботи з NUnit було створено окремий проект у проектному рішенні – веб-додатку (рисунок 3.1) та скомпільовано його в динамічну бібліотеку.

Для запуску окремого тесту необхідно вибрати один тест та натиснути Run. NUnit виконая тестовий метод. Результат виконання зображено на рисунку 3.7.

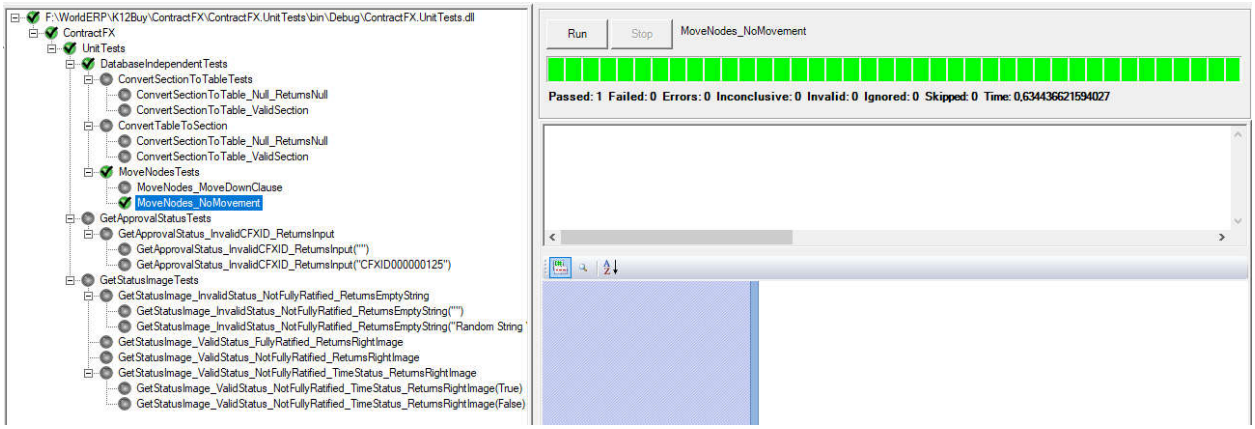

Рисунок 3.7 – Виконання тесту

Зелений індикатор свідчить про успішне виконання тестового методу. Також можна виконати усі виявлені тести в бібліотеці. Для цього досить вибрати саму бібліотеку та натиснути Run.

В фреймворку NUnit для .NET за допомогою атрибутів можна помітити тести, які необхідно проігнорувати. Тест, помічений атрибутом [Ignore] буде ігноруватись менеджером, як зображено на рисунку 3.8.

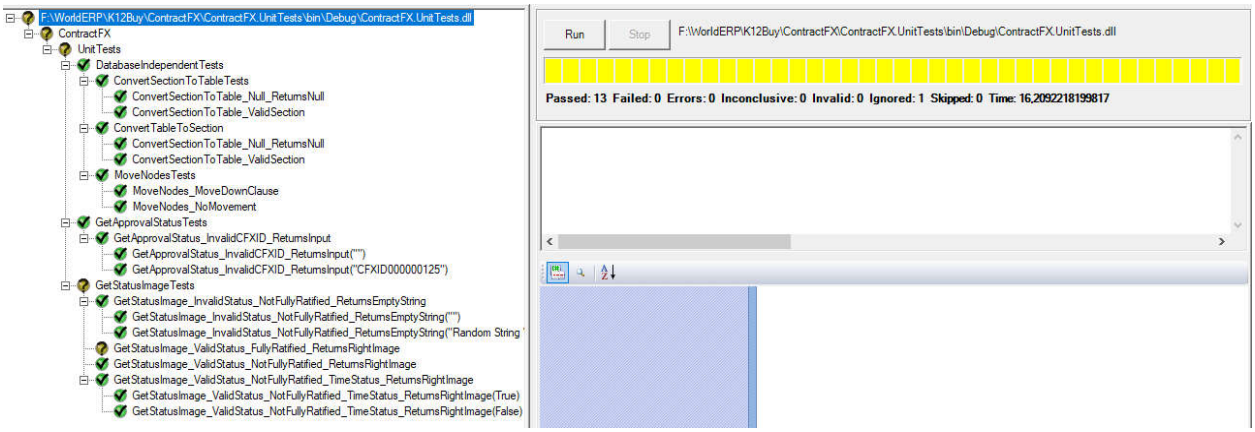

Рисунок 3.8 – Виконання всіх тестів з наявністю ігнорованих тестів

Тестові методи, помічені жовтим маркером, а не зеленим як усі звичайно, ігноруються менеджером. Жовтий колір індикатора означає, що один або більше тестів з виконуваних був проігнорований.

Однією з головних особливостей менеджеру NUnit є виведення результату тестів, які було провалено. За допомогою окремих команд при описані тестів можна визначити повідомлення, яке буде виведено в разі провалу виконання тесту. Також, менеджер відображає посилання на рядок, на якому «впав» тест та дозволяє дослідити ієрархію викликів (callstack), як зображено на рисунку 3.9.

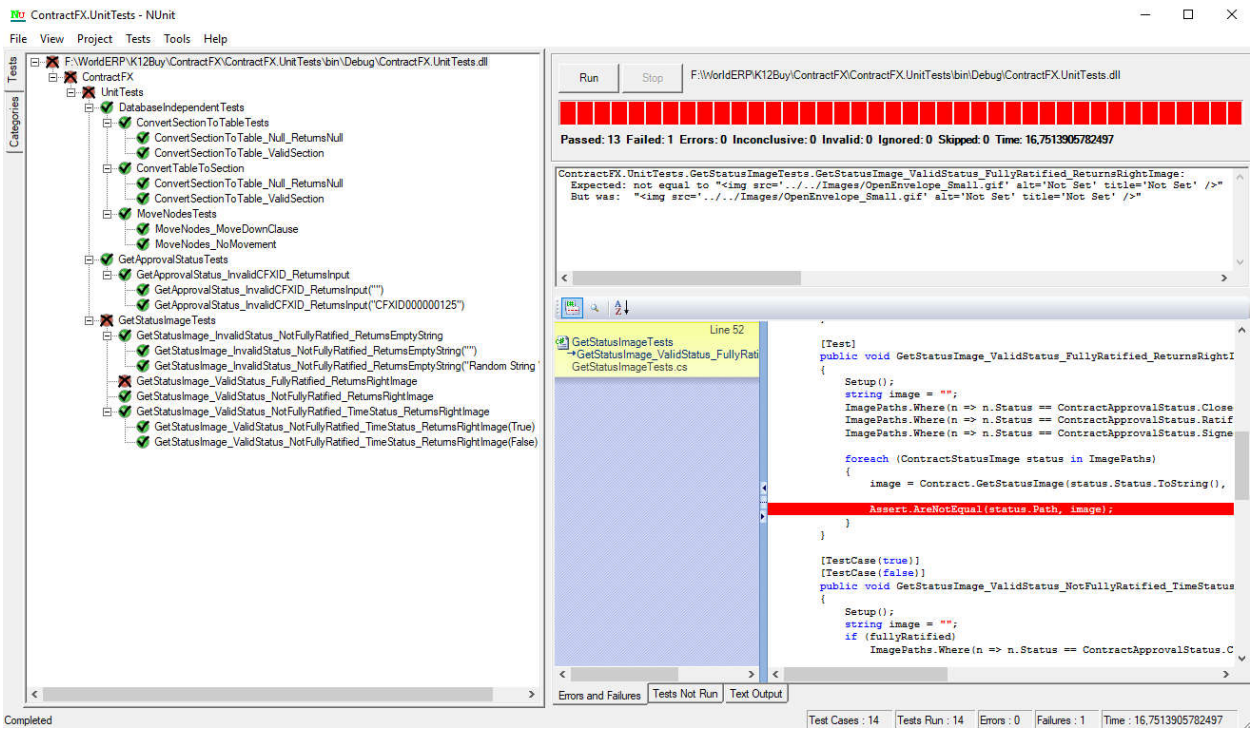

Рисунок 3.9 – Результат виконання тестів

Провалені тести помічені червоним маркером в переліку тестів. Червоний індикатор означає, що один або більше тестів «впали». Не зважаючи на те, що один із тестів провалився, NUnit не припиняє роботу та продовжує виконувати всі інші тести. У вікні справа можна побачити callstack. Також, червоною лінією помічений рядок, в якому тест «впав».

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

4.1 Розрахунок витрат на розробку програмного забезпечення

В першу чергу розробка подібної системи значно покращить точність знаходження помилок та збоїв в системі. Також розроблена система тестування веб - додатку призначена для пришвидшення тестування, його автоматизації та автоматизації створення журналу для запису результатів і характеризується підвищеною ефективністю виконання алгоритму, що призводить до зменшення часу тестування в ручну об’єкту дослідження.

Витрати на розробку і впровадження програмних засобів ($К$) включають [30]:

$$К = К_1, + К_2$$

де $К_1$ - витрати на розробку системи, грн.

$К_2$ - витрати на налаштування і тестову експлуатацію системи рішення задачі на комп’ютері, грн.

Витрати на розробку системи включають:

- витрати на придбання спец - обладнання для проведення експериментальних робіт ($Об$);
- витрати на оплату праці розробників ($Р_{ОП}$);
- витрати на покупні вироби ($Пв$);
- накладні витрати ($Н$);
- витрати на відрахування у спеціальні державні фонди ($Вф$);
- інші витрати ($Ів$).

4.1.1 Розрахунок витрат на оплату праці

Витрати на оплату праці включають заробітну плату (ЗП) всіх категорій працівників, безпосередньо зайнятих на всіх етапах проектування. Розмір ЗП обчислюється на основі трудомісткості відповідних робіт у людино-днях та середньої ЗП відповідних категорій працівників.

У розробці проектного рішення задіяні наступні спеціалісти - розробники, а саме: керівник проекту; студент-дипломник; консультант техніко-економічного розділу.

Таблиця 4.1 - Вихідні дані для розрахунку витрат на оплату праці

№ п/п	Посада виконавців	Місячний оклад, грн.
1	Керівник проекту	4916
2	Консультант техніко-економічного розділу, доцент	6026
3	Студент	1400

Витрати на оплату праці розробників проекту визначаються за формулою:

$$B_{ОП} = \sum_{i=1}^{N} \sum_{j=1}^{M} n_{ij} \cdot t_{ij} \cdot C_{ij} \,, \qquad (4.1)$$

де n_{ij} – чисельність розробників i-ої спеціальності j-го тарифного розряду, осіб;

t_{ij} – затрачений час на розробку проекту співробітником i-ої спеціальності j-го тарифного розряду, год;

C_{ij} – годинна ставка працівника i-ої спеціальності j-го тарифного розряду, грн.

Середньо годинна ставка працівника може бути розрахована за формулою:

$$C_{ij} = \frac{C_{ij}^{0}(1+h)}{РЧ_{i}}, \qquad (4.2)$$

де C_{ij} – основна місячна заробітна плата розробника *i*-ої спеціальності *j*-го тарифного розряду, грн.;

h – коефіцієнт, що визначає розмір додаткової заробітної плати (при умові наявності доплат);

$РЧ_i$ - місячний фонд робочого часу працівника *i*-ої спеціальності *j*-го тарифного розряду, год. (приймаємо 168 год.).

В даній таблиці коефіцієнт додаткової заробітної плати (*h*) для керівника проекту становить 0. Для консультанта техніко-економічного відділу даний коефіцієнт становить 1,47.

Таблиця 4.2 - Розрахунок витрат на оплату праці

№ п/п	Посада виконавців	Час розробки, год	Погодинна заробітна плата, грн/год.	Витрати на розробку, грн
1	Керівник проекту	16	29	580
2	Консультант техніко-економічного розділу, доцент	2	30,9	61,8
3	Студент	72	8,3	597,6
Разом				1837

4.1.2 Відрахування на соціальні заходи

Величну відрахувань у спеціальні державні фонди визначають у відсотковому співвідношенні від суми основної та додаткової заробітних плат. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 20,5 % від суми заробітної плати:

$$B_{\Phi} = \frac{20,5}{100} \cdot 1837 = 376,585\,\text{грн.} \qquad (4.3)$$

4.1.3 Розрахунок витрат на матеріали та комплектуючі

У таблиці 4.3 наведений перелік купованих виробів і розраховані витрати на них.

Таблиця 4.3- Розрахунок витрат на матеріали та комплектуючі

№ п/п	Найменування купованих виробів	Одиниця виміру	Ціна, грн	Кількість купованих виробів	Сума, грн	Транспортні витрати (10% від суми)	Загальна сума, грн
1	Папір (формат А4)	уп	90,0	2	180,00	10,0	190,0
2	Диски CD-R	шт	5,0	1	5,00	5,0	10,0
3	Тонер для принтера	уп	95	1	95	5,0	100,0
Разом							300,00

4.1.4 Витрати на використання комп’ютерної техніки

Витрати на використання комп’ютерної техніки включають витрати на амортизацію комп’ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп’ютером. Вартість години роботи ноутбука Lenovo T-530 становить 3 грн. Середній щоденний час роботи на ноутбуці – 6 годин. Розрахунок витрат на використання комп’ютерної техніки приведений в таблиці 4.4.

Таблиця 4.4- Розрахунок витрат на використання комп'ютерної техніки

№ п/п	Назва етапів робіт, при виконанні яких використовується комп'ютер	Час використання комп'ютера, год.	Витрати на використання комп'ютера грн.
1	Проведення досліджень та оформлення їх результатів	120	360
2	Оформлення техніко-економічного розділу	8	24
3	Оформлення ДП	12	36
Разом		140	420

4.1.6 Накладні витрати

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими відсотками до витрат на оплату праці. Середньостатистичний відсоток накладних витрат приймемо 150% від заробітної плати:

$$H = 1{,}5 \cdot 1837 = 2755{,}5 \text{ (грн.)} \quad (4.4)$$

4.1.6 Інші витрати

Інші витрати є витратами, які не враховані в попередніх статтях. Вони становлять 10% від заробітної плати:

$$I = 1837 \cdot 0{,}1 = 183{,}7 \text{ (грн.)} \quad (4.5)$$

Витрати на розробку програмного забезпечення складають:

$$K_1 = B_{ОП} + B_Ф + B_{ПВ} + H + I \quad (4.6)$$

$$K_1 = 1837 + 376{,}585 + 221 + 2755{,}5 + 183{,}7 = 5373{,}785 (\text{грн.}) \quad (4.7)$$

Витрати на налаштування і тестову експлуатацію програмного продукту визначаємо за формулою:

$$K_2 = S_{м.г.} \cdot t_{від} \quad (4.8)$$

де $S_{м.г.}$ - вартість однієї машино-години роботи ПК, грн./год.

$t_{від}$ - комп'ютерний час, витрачений на налаштування і тестову експлуатацію створеного програмного продукту, год.

Загальна кількість днів роботи на комп'ютері дорівнює 20 днів. Середній щоденний час роботи на комп'ютері – 6 години. Вартість години роботи комп'ютера дорівнює 3 грн. Тому:

$$K_2 = 3 \cdot 120 = 360 \text{ грн.} \quad (4.9)$$

На основі отриманих даних складаємо кошторис витрат на розробку програмного забезпечення.

Таблиця 4.6 - Кошторис витрат на розробку системи тестування

№ п/п	Найменування витрат	Сума витрат, грн.
1	Витрати на оплату праці	1837
2	Відрахування у спеціальні державні фонди	376,585
3	Витрати на куповані вироби	221
4	Накладні витрати	2755,5
5	Інші витрати	183,7
6	Витрати на відлагодження і дослідну експлуатацію програмного продукту	360,0
Разом		5733,785

4.2 Визначення експлуатаційних витрат

Для оцінки економічної ефективності розроблюваного програмного продукту слід порівняти його з аналогом, тобто існуючим програмним забезпеченням ідентичного функціонального призначення.

Експлуатаційні одноразові витрати по програмному забезпеченню і аналогу включають вартість підготовки даних і вартість роботи комп'ютера (за час дії програми):

$$E_n = E_{1n} + E_{2n} \tag{4.10}$$

де E_n - одноразові експлуатаційні витрати на ПЗ (аналог), грн.;

E_{1n} - вартість підготовки даних для експлуатації ПЗ (аналогу), грн.;

E_{2n} - вартість роботи комп'ютера для виконання проектного рішення (аналогу), грн.

Річні експлуатаційні витрати B_{en} визначаються за формулою:

$$B_{en} = E_n * N_n \tag{4.11}$$

де N_n - періодичність експлуатації ПЗ (аналогу), раз/рік.

Вартість підготовки даних для роботи на комп'ютері визначається за формулою:

$$E_{1n} = \sum_{l=1}^{n} n_i t_i c_i \,, \tag{4.12}$$

де i - категорії працівників, які приймають участь у підготовці даних (i=1,2,...n);

n_i - кількість працівників *i*-ої категорії, осіб.;

t_i - трудомісткість роботи співробітників *i*-ої категорії по підготовці даних, год.;

c_i - середнього годинна ставка працівника *i*-ої категорії з врахуванням додаткової заробітної плати, що знаходиться із співвідношення:

$$c_i = \frac{c_i^0(1+b)}{m} \tag{4.13}$$

де c_i^0 - основна місячна заробітна плата працівника *i*-ої категорії, грн.;

b - коефіцієнт, який враховує додаткову заробітну плату (приймемо 0,57);

m - кількість робочих годин у місяці, год.

Для роботи з даними як для проектного рішення так і аналогу потрібен один працівник, основна місячна заробітна плата якого складає: c^0 = 1400 грн. тоді:

$$c_1 = \frac{1400(1+0{,}57)}{20*6} = 19{,}32 \text{ грн/год} \tag{4.14}$$

Таблиця 4.7- Розрахунок витрат на підготовку даних та реалізацію проектного рішення в комп'ютерній системі

№	Час роботи співробітників, год.	Середньо - годинна заробітна плата, грн./год.	Витрати , грн.
	Проектне рішення		
1	1	18,32	18,32
	Аналог		
1	1,5	18,32	27,48

Витрати на експлуатацію комп'ютера визначається за формулою:

$$E_{2n} = t * S_{МГ} \quad (4.15)$$

де t - витрати машинного часу для реалізації проектного рішення (аналогу), год.;

$S_{МГ}$ - вартість однієї години роботи комп'ютера, грн./год.

$$E_{2n} = 1 * 3 = 3 \text{ грн.}; E_{2a}=1,5 * 3 = 4,5 \text{ грн.} \quad (4.16)$$

$$E_n=18,32 + 3=21,32 \text{ грн.}; E_a=27,48 + 4,5 = 31,98 \text{ грн.} \quad (4.17)$$

$$B_{en}= 21,32 * 150 = 3198 \text{ грн.}; B_{ea}= 31,98 * 150 = 4797 \text{ грн.} \quad (4.18)$$

4.3 Розрахунок ціни споживання проектного рішення

Ціна споживання - це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

$$Ц_{С(П)} = Ц_П + B_{(E)NPV} \quad (4.19)$$

де $Ц_П$ - ціна придбання проектного рішення, грн.:

$$Ц_П = K(1 + \frac{П_Р}{100}) + K_0 + K_к \quad (4.20)$$

де K - кошторисна вартість;

$П_Р$ - рентабельність;

$К_0$ - витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті, грн.;

$К_к$ - витрати на доукомплектування технічних засобів на об'єкті, грн.;

$$Ц_Д = 5733,785 \cdot (1 + 0,3) = 5905,80 \text{ (грн.)}. \quad (4.21)$$

Вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

$$B_{enpv} = \sum_{t=0}^{T} \frac{B_{e\Pi}}{(1+R)^t} \quad (4.22)$$

де B_{en} - річні експлуатаційні витрати, грн.;

T - строк служби проектного рішення, років;

R - річна ставка проценту банку.

$$B_{e\text{пр}v} = \sum_{t=1}^{5} \frac{3198}{(1+0,02)^t} = 15676,47 \text{ грн.} \quad (4.23)$$

$$B_{e\text{пр}v} = \sum_{t=1}^{5} \frac{4797}{(1+0,02)^t} = 23514,71 \text{грн.} \quad (4.24)$$

Тоді ціна споживання проектного рішення дорівнюватиме:

$$Ц_{\text{сп}} = 5905,8 + 15676,47 = 21582,27 \text{ грн.} \quad (4.25)$$

Аналогічно визначається ціна споживання для аналогу:

$$Ц_{са} = 4500,0 + 23514,71 = 28014,71 \text{ грн.} \quad (4.26)$$

4.4 Визначення показників економічної ефективності

Економічний ефект в сфері проектування рішення:

$$E_{ПР} = Ц_{П} - Ц_{А} \quad (4.27)$$

$$E_{ПР} = 5905,8 - 4500,0 = 1405,8 \text{ грн.} \quad (4.28)$$

Річний економічний ефект в сфері експлуатації:

$$E_{КС} = B_{ЕА} - B_{ЕП} \quad (4.29)$$

$$E_{кс} = 4797 - 3198 = 1599 \text{ грн.} \quad (4.30)$$

Додатковий економічний ефект у сфері експлуатації:

$$\Delta E_{екс} = \sum_{t=1}^{T} E_{екс}(1+R)^{T-t} \quad (4.31)$$

$$\Delta E_{екс} = \sum_{t=1}^{5} 1599(1+0,02)^{5-1} = 8654,04 \text{ грн.} \quad (4.32)$$

Сумарний ефект складає:

$$E = E_{пр} + \Delta E_{екс} = 1405,8 + 8654,04 = 10059,\ 84 \text{ грн.} \quad (4.33)$$

Таблиця 4.8 - Показники економічної ефективності проектного рішення

№	Найменування	Одиниці вимірювання	Значення показників	
			Базовий варіант	
1	Капітальні вкладення	грн.	-	1
2	Ціна придбання	грн.	4500,0	2
3	Річні експлуатаційні витрати	грн.	23514,71	3
4	Ціна споживання	грн.	28014,71	4
5	Економічний ефект в сфері проектування	грн.	-	5
6	Економічний ефект в сфері експлуатації	грн.	-	6
7	Додатковий ефект в сфері експлуатації	грн.	-	7
8	Сумарний ефект	грн.	10059, 84	

З даного розділу можна побачити, що сумарна вартість даного програмного рішення є відносно досить низькою. Вартість впровадження є також низькою. Саме тому було обрано даний підхід до вирішення проблеми.

ВИСНОВКИ

Відповідно до постановлених задач було реалізовано наступні пункти:

1. На основі аналізу, проведеного під час проходження переддипломної практики було сформовано план та архітектуру системи, які було згодом розроблено.

2. Для більш повної реалізації розроблюваної системи тестування проведено аналіз універсального та спеціалізованого апаратного та програмного забезпечення комп'ютерних систем і мереж, яке використовується організацією. Сформовано рекомендації, щодо покращення існуючих в організації технологічних процесів створення та використання комп'ютерних систем і мереж, їх програмного забезпечення, а саме:

- рекомендовано розробити тести, які б покривали абсолютно усю систему;
- рекомендовано оновити використовувані організацією системи, а саме SQL Server, Visual Studio та TeamCity;
- рекомендовано провести глобальний рефакторинг існуючого коду та архітектури проектних рішень, для більш ефективного використання простору та ресурсів системи.

3. Розроблено деяку кількість модульних тестів, які проводять порівняння результатів та станів окремих модулів системи. Всі розроблені тести використовують фреймворк для розробки юніт-тестів NUnit. Для запуску тестів було використано менеджер Nunit Runner.

Результатом виконання завдання дипломної роботи стали автоматизовані модульні тести, розроблені засобами фреймворку NUnit. Тести розроблено для веб-додатку на основі технології ASP.NET.

4. Короткий опис основних аспектів економічної сторони розробленого продукту описано в розділі 4. В даному розділі коротко описано витрати на виплати учасникам дипломного проекту, витрати на

матеріали для виконання поставленого завдання, витрати на комунальні послуги, непередбачувані витрати тощо. Також, в четвертому розділі було коротко описано особливості процесу розробки модулю тестування, а саме економічні аспекти даного процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Иванов Л. В. Факторный анализ в социальных исследованиях / Л. В. Иванов, В. С. Смирнов. – М. : Наука, 1996. – 352 с.

2. Власов П. К. Психология менеджмента / П. К. Власов, А. В. Лепницкий, И. М. Лущихина [и др.]. – Х. :Гуманит. центр, 2007. – 300 с.

3. Харман Г. Современный факторный анализ – М. : Статистика, 1972. – 489 с.

4. Классификация и кластер / ред. Дж. Вэн Райзин. – М. : Мир, 1980. – 389 с.

5. Петров В. С. Применение методов кластерного анализа при обработке данных экспертного опроса / В. С. Петров // Изв. АН СССР. Сер.: Техн. кибернетика. – 1985. – Т. 202, № 3. – С. 15–18.

6. Zhang Z. Experimental research on the localized electrochemical micro-machining / Zhang, Zhu // Електрохімія – Russian Journal of Electrochemistry. – 2008. – Т 44 – № 8. – С. 926–930.

7. Тишков В. Т. Кластерный анализ в социальных исследованиях / В. Т. Тишков // Вестн. Харьк. политехн. ин та. Сер.: Техн. кибернетика и ее приложения. – 1990. – № 260, вып. 10. – С. 5–7.

8. Лобанова Л. С. Оптимізація обчислень в інтегральному методі найменших квадратів наближення функцій однієї та двох змінних / Л. С. Лобанова, В. О. Пасічник, О. О. Черняк // Вісник НТУ «ХПІ». – 2010. – № 9. – С. 57–62.

9. Иванов Л. В. Новый алгоритм классификации объектов, описываемых качественными признаками / Л. В. Иванов, В. С. Петров // Статистический анализ социально-экономических данных / ред. Р. В. Сидоров. – К. : Наук.думка. – 1997. – С. 57–65.

10. Иванов Л. В. Статистический подход к обработке социоло¬гических данных / Л. В. Иванов // Труды междунар. конф. «Социология и математика». – Т. 2. – Х. : НТУ «ХПИ», 2006. – С. 5–9.

11. Гамаюн І. П. Оцінювання міри схожості між об’єктам, що характеризуються кількісними і номінальними ознаками / І. П. Гамаюн, О. М. Безменова // Інформаційні технології: наука, техніка, технологія, освіта, здоров’я. Тези доповідей XXI міжнародної науково-практичної конференції. Ч. 1 (29–31 травня 2013 р., Харків). – Х. : НТУ «ХПІ», 2013. – С. 9.

12. Національний технічний університет «Харківський політехнічний інститут». Вісник НТУ «ХПІ» / Національний технічний університет «Харківський політехнічний інститут». – НТУ «ХПІ», 2014. – Режим доступу : http://vestnik.kpi.kharkov.ua. – Дата звертання : 30 березня 2014.

13. Азаренков В. И. Аналитическое решение уравнения тепло-проводности в задачах анализа и синтеза температурных полей радиоэлектронной аппаратуры :дис. канд. техн. наук : 01.05.02 / Азаренков – Х., 2015. – 225 с.

14. Северин В. П. Моделі та методи оптимізації показників якості систем автоматичного керування енергоблоку атомної електростанції : автореф. дис. на здобуття наук. ступеня д-ра техн. наук : спец. 05.13.07 «Автоматизація технологічних процесів» / В. П. Северин. – Х., 2007. – 35 с.

15. Моджтаба Д. Х. С. Багатокритеріальний синтез інтелектуальних систем керування енергоблоків АЕС генетичними алгоритмами : автореф. дис. на здобуття наук. Ступеня канд. техн. наук : спец. 05.13.07 «Автоматизація технологічних процесів» / Джафарі Хенджані Сайд Моджтаба. – Х., 2010. – 20 с.

16. ДСТУ 3582–97. Скорочення слів в українській мові у бібліографічному описі. – К. : Держстандарт України, 1998. –27 с.

17. ГОСТ 8.586.5–2005. Методика выполнения измерений расхода и количества жидкостей и газов с помощью сужающих устройств. – М. :Стандартинформ, 2007. – 10 с.

18. Палкин М. В. Пат. 2187888, Российская Федерация. Способ ориентирования по крену летательного аппарата с оптической головкой самонаведения / М. В. Палкин. – 2006 р.

19. Чугаева В. 2187888, Российская Федерация. Приемо-передающее устройство / В. И. Чугаева. – 2002 р.

20. Основи інформатики: Методичні вказівки до лабораторних робіт: У 2 ч. / Укл .: І.В. Юрченко.- Чернівці: Рута, 2000.- 79 с.

21. Система управління базами Даних Microsoft Access: Методичні вказівки до лабораторних робіт / Укл .: В.С. Сікора, І.В. Юрченко.- Чернівці: Рута, 2002.- 40 с.

22. Комп'ютерні мережі: Методичні вказівки до лабораторних робіт / Укл .: В.С. Сікора, І.В. Юрченко.- Чернівці: Рута, 2002.- 43 с.

23. Операційна система Microsoft Windows: Методичні вказівки до лабораторних робіт / Укл .: В.С. Сікора, І.В. Юрченко.- Чернівці: Рута, 2003.- 48 с.

24. Текстовий редактор Microsoft Word: Методичні вказівки до лабораторних робіт / Укл .: В.С. Сікора, І.В. Юрченко.- Чернівці: Рута, 2003.- 56 с.

25. Семчук А.Р., Юрченко І.В. Економічна інформатика. Навчальний посібнік.- Чернівці: МВІЦ "Місто", 2008.- 426 с.

26. Симонович С., Євсєєв Г. Практична інформатика: універсальний курс.- М .: АСТ-ПРЕСС; Інфорком-Пресс, 1999.- 480 с.

27. Руденко В.Д. та ін. Базовий курс інформатики; за заг. ред. В.Ю.Бікова: [Навч. посіб.]. - К .: Вид. група BHV. - Кн. 1: Основи інформатики. - 2005. - 320 с.

28. Руденко В.Д. та ін. Базовий курс інформатики; за заг. ред. В.Ю.Бікова: [Навч. посіб.]. - К .: Вид. група BHV. - Кн. 2: Інформаційні технології. - 2006. - 368 с .: іл.

29. Методичні рекомендації до виконання дипломного проекту з освітньо-кваліфікаційного рівня «Бакалавр» напряму підготовки 6.050102 «Комп'ютерна інженерія» фахового спрямування «Комп'ютерні системи та мережі» / О.М. Березький, Л.О. Дубчак, Р.Р. Трембач, Г.М. Мельник, Ю.М. Батько, С.В. Івасьєв / Під ред. О.М. Березького. – Тернопіль: ТНЕУ, 2016. – 60 с.

30. Методичні вказівки до написання техніко-економічного розділу для дипломних проектів на здобуття освітньо-кваліфікаційного рівня «Бакалавр» напряму підготовки 6.050102 «Комп'ютерна інженерія» / І.Р. Паздрій. – Тернопіль: ТНЕУ, 2015. – 36 с.