МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК, ФІЗИКИ ТА МАТЕМАТИКИ
КАФЕДРА ІНФОРМАТИКИ, ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ЕКОНОМІЧНОЇ КІБЕРНЕТИКИ

ТЕСТУВАННЯ САЙТУ ПРАЦЕВЛАШТУВАННЯ СТУДЕНТІВ ХДУ

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти «бакалавр»

Виконала: студентка 4 курсу
Спеціальності 121 Інженерія
програмного забезпечення
Освітньо-професійної програми
«Інженерія програмного забезпечення»
першого (бакалаврського) рівня освіти
Кузьменко Катерина Олександрівна
Керівники: старший викладач
Черненко Ірина Євгенівна,
кандидат фізико-математичних наук,
доцент Єрмолаєв Вадим Анатолійович
Рецензент: ФОП Федянін Павло
Костянтинович

Херсон – 2021

ЗМІСТ

ВСТУП

Актуальність дослідження. Контроль якості програмного забезпечення є невід'ємною частиною створення успішного програмного продукту. Адже будь-яка людина може допустити помилку під час виконання своєї роботи, наприклад, програміст під час створення коду може допускати синтаксичні помилки. Крім того, на стадії формування технічного завдання можуть бути допущені структурні та системні помилки. Також може бути обрано неефективні, або помилкові алгоритми обробки, передавання та зберігання даних.

Тестування має виявити і усунути можливості збоїв, пов'язаних із помилками користувачів. Саме тому, будь-який програмний продукт перед використанням має пройти стадію попереднього тестування з метою виявлення помилок, вдосконалення інтерфейсу, алгоритмів обробки та зберігання даних.

Важливим аспектом при тестуванні програмного забезпечення є щільність покриття тестами, як готового продукту, так і коду. Для цього існує безліч підходів, видів та цілей тестування. У покращенні якості тестування та щільності покриття продукту тестами велику роль відіграє автоматизація тестування. Проте не для всіх програмних продуктів вона є необхідною. У будь-якому разі, тестування програмного продукту (виявлення та виправлення помилок, контроль) є невід'ємною частиною якісного програмного забезпечення.

Метою кваліфікаційної роботи є виявлення помилок, здійснення автоматизованого та ручного тестування сайту «Працевлаштування студентів ХДУ».

Об’єктом дослідження кваліфікаційної роботи є методи автоматизованого та ручного тестування для виявлення та усунення помилок при реалізації програмного забезпечення.

Предметом дослідження кваліфікаційної роботи є інструменти автоматизованого та ручного тестування для виявлення та усунення помилок при реалізації сайту «Працевлаштування студентів ХДУ».

Для досягнення мети були сформульовані наступні **задачі**:

1. Дослідити методи тестування.
2. Обрати найефективніші способи тестування сайтів.
3. Визначити недоліки та переваги автоматизованого та ручного видів тестування.
4. Протестувати сайт «Працевлаштування студентів ХДУ».
5. Скласти тестову документацію.
6. Прослідкувати за усуненням виявлених помилок та провести повторне тестування.

Структурно кваліфікаційна робота складається: з вступу, двох розділів, висновків та списку використаних джерел.

РОЗДІЛ 1.
Аналіз предметної області

1.1. Основні поняття тестування програмного забезпечення

Тестування програмного забезпечення – це метод перевірки на відповідності фактичного результату програмного продукту очікуваним вимогам, що також необхідно щоб переконатися, що продукт не містить дефектів. Передбачає виконання попередньо визначених алгоритмів з використанням ручних або автоматизованих інструментів для оцінки одного або декілька властивостей. [1]

Метою тестування програмного забезпечення є виявлення помилок, пропусків, неточностей або відсутності вимог, заданих на етапі програмування продукту.

Якщо у програмному забезпеченні є помилки або дефекти, вони можуть бути знайдені на ранньому етапі виробництва продукту і виправлені до початку його реального використання. Правильно протестований програмний продукт забезпечує надійність, безпеку, та високу продуктивність.[6]

Переваги тестування програмного забезпечення:

- Безпека (люди чекають перевірені програмні продукти, тестування допоможе позбутися ризиків витоку інформації).
- Якість продукту (головна мета будь-якого проєкту принести свою користь користувачам, тестування гарантує, що продукт буде виконаний відповідно до специфікацій, бізнес-вимог та виконує свої функції).
- Зручність користувачів (UI / UX Testing забезпечує кращий користувацький досвід та враження).

Основи тестування програмного забезпечення:

- Тестування ПЗ визначається, як діяльність, що направлена на перевірку фактичних результатів очікуваним та відсутність дефектів в програмній системі;
- Тестування важливе, адже помилки можуть бути не лише неприємними при використанні, а й небезпечними для людини.
- Важливими причинами використання тестування програмного забезпечення є: економічність, безпека, якість продукту та задоволення користувача.
- Зазвичай тестування ділиться на дві категорії : функціональне та нефункціональне.
- Основними стратегіями при розробці програмного забезпечення є: модульне, інтеграційне та системне тестування.

У тестуванні можна визначити декілька різних процесів, такі терміни, як тестування, доведення, відлагодження, контроль і тест, часто використовують як синоніми, що для різних людей мають різний сенс. Стандартних, загальноприйнятих визначень цих термінів немає, спроба сформулювати їх була прийнята на симпозиумі по тестуванню програм.

Тестування (testing) - процес виконання програми або її частини з метою знайти помилки.

Доказ (proof) - спроба знайти помилку в програмі, що не відноситься до зовнішнього для програми оточення. Докази можуть розглядатись, як форма тестування, хоча вони і не вимагають прямого виконання програми.

Верифікація (verification) - спроба знайти помилку, виконуючи програму в тестовому, або модельованому оточенні.

Валідація (validation) - спроба знайти помилку, при виконанні програми в заданому реальному оточенні.

Сертифікація (certification) - авторитетне підтвердження правильності програми, аналогічне атестації електротехнічних приладів

Underwriters Laboratories. При тестуванні з метою атестації виконується порівняння з деякими заздалегідь визначеним стандартом.

Відлагодження (debugging) не є різновидом тестування. Хоча слова «відлагодження» і «тестування» часто використовують як синоніми, під ними розуміють різні види діяльності. [2]

Тестування - діяльність, спрямована на знаходження помилок; відлагодження спрямоване на встановлення точного походження вже відомої помилки, а згодом - на виправлення цієї помилки. Саме ці два виду діяльності пов'язані - результати тестування є вихідним даними для відлагодження.

Тестування модуля або автоматизоване тестування (module testing, unit testing) - контроль окремого програмного модуля, зазвичай в ізольованому оточенні (ізольованого від усіх інших модулів).

Інтеграційне тестування (integration testing) - контроль інтеграції між частинами системи (модулями, компонентами, підсистемами).

Тестування зовнішніх функцій (external function testing) - контроль зовнішньої поведінки системи, визначеного зовнішніми специфікаціями. Комплексне тестування (system testing) - контроль і / або випробування системи стосовно вихідних цілям.

Комплексне тестування є процесом контролю, якщо виконується в модельованому середовищі, і процесом тестування, якщо виконується в реальному оточенні. [3]

Тестування приемлемості (acceptance testing) - перевірка співвідношення програми вимогам користувача або замовника програмного продукту. Тестування установки (installation testing) - перевірка відповідності кожного конкретного варіанту встановлення системи з метою виявити будь-яку помилку, що виникає в процесі налаштування системи. Відношення між цими типами тестів і проєктної документації, на якій базується тест.

1.2. Класифікація видів та рівнів тестування програмного забезпечення

Усі види робіт, що проводить тестувальник умовно можна поділити на два типи :

1. ***Функціональне тестування*** – тестування програмного забезпечення головна ціль якого, це перевірка реалізованості, функціональних вимог системи, можливість програми в заданих критеріях вирішувати покладені на нього задачі. Вимоги містять:
    a. Захищеність;
    b. Відповідність стандартам;
    c. Здатність до взаємодії;
    d. Точність;
    e. Функціональна придатність;
2. ***Нефункціональне тестування програмного забезпечення*** – в першу чергу перевірку на відповідність нефункціональним вимогам:
    a. Зручність (в основному це оцінка зручності інтерфейсу для користувача);
    b. Масштабованість (Можливість роботи додатку при різних навантаженнях)
    c. Безпека (Захист даних користувача, захист даних додатку, стійкість до хакерських атак);
    d. Можливість сумісності додатку для різних платформ та під різне оточення;
    e. Надійність (Поведінка системи при різних непередбачених ситуаціях, можливість розробки нестандартних ситуацій користувача);

Види тестування за ступенем ізольованості частин компонентів програмного забезпечення:

- Компонентне (модульне) тестування;
- Інтеграційне тестування;
- Системне тестування;
- Приймальне;

Рисунок 1.1. – Рівні тестування ПЗ

Модульне (компонентне або юніт) тестування – тести, головна мета яких перевірити кожний модуль або компонент системи окремо один від одного. Ціль полягає в тому, щоб перевірити, що кожна одиниця програмного коду працює належним чином. Цей вид тестування виконується розробником на етапі програмування продукту.

В моделях розробки SDLC, STLC, V Model модульне тестування – це перший рівень тестування, що виконується перед інтеграційним тестуванням. Компонентне тестування – це метод тестування білого ящика.

Відсутність модульного тестування при написанні коду значно збільшує рівень дефектів у подальшому (інтеграційному, системному, приймальному) тестуванні.

Основні функції модульного тестування:

1. Модульні тести дозволяють виправити помилки на ранніх етапах розробки програмного продукту.
2. Допомагає розробникам більш детально вивчити та розуміти кодову базу проєкту і дозволяє швидко та легко вносити зміни в продукт.
3. Хороші юніт-тести можуть слугувати проєктною документацією.
4. Модульні тести допомагають з міграцією коду.

Переваги модульного тестування:

- Розробники, що бажають знати, які функції та можливості представляє собою модуль і я к його використовувати, можуть переглянути модульні тести, щоб отримати загальне представлення що до API модуля.
- Модульні тестування дозволяє програмісту легко виконувати рефакторинг коду на етапі регресивного тестування та переконатися, що модуль все ще функціонує коректно. Процедура полягає написанні контрольних прикладів для всіх функцій та методів, щоб у випадку помилки після введення змін, це можна біло швидко ідентифікувати та виправити.
- Можна тестувати частини проєкту. не чекаючи готового програмного продукту.

Недоліки модульного тестування:

- Не будуть виявлені всі помилки. Неможливо оцінити усі шляхи виконання навіть у зовсім звичайних програмах.

- Модульне тестування більш орієнтоване на одиницю коду, саме тому не зможе ідентифікувати системного рівня, а також ті, що пов’язані з інтеграцією.

Використання модульного тестування буду більш ефективним по поєднанні його з іншими видами тестування.

Інтеграційне тестування – це тип тестування при якому програмні модулі поєднуються логічно і тестуються як група. Як правило, програмний продукт складається з декількох програмних моделей, написаних різними програмістами.

Метою цього тестування є виявлення багів при взаємодії між програмними модулями, в першу чергу направлене на перевірку обміну даних між самими модулями.

Методи тестування за ступенем знання тестувальної системи можна класифікувати на:

- Техніка «Чорного ящика» (Black box testing);
- Техніка «Білого ящика» (White box or «glass-box» testing);
- Техніка «Сірого ящика» (Grey box);

При тестуванні методикою «чорного ящика», тестувальник має доступ до готового програмного продукту лише через інтерфейси що і замовник або користувач продукту, тобто без доступу до програмного коду. Тобто ця техніка базується на аналізі вимог та специфікацій або тестування поведінки. [4]

Згідно до ISTQB (некомерційна організація, що займається питаннями розвитку сфери тестування програмного забезпечення, також однойменна програма, що дозволяє спеціалістам цієї сфери отримувати міжнародний сертифікат) тестування чорного ящика це:

- Як нефункціональне так і функціональне тестування, що не потребує знання внутрішньої побудові системи або компонента;

- Тест-дизайн, побудований на основі методу чорного ящика – процедура написання або вибору тест-кейсів проаналізувавши нефункціональної або функціональні специфікації окремого компонента або цілої системи без поглиблених знань їх побудови;

Переваги техніки чорного ящика:

- Тестування з точки зору фінального користувача, що допоможе знайти неточності та протиріччя у вимогах замовника;
- Тестувальник може не знати мови програмування та мінімальні аспекти реалізації програмного продукту;
- Початок роботи, наприклад написання тестів, можна починати уже після того як специфікація буду готовою;
- Тестування може проводитися спеціалістами не пов’язаними з розробкою, що допомагає уникнути упередженого ставлення;

Недоліки методу чорного ящика

- Тестується дуже обмежена кількість шляхів виконання програми;
- Без чітко написаної специфікації дуже складно написати ефективні тест-кейси;
- Деякі тести можуть виявитися надлишковими, якщо вони вже були проведені програмістом на рівні модульного тестування.

Повною протилежністю техніки чорного ящика є тестування методом білого ящика.

Тестування методом білого ящика (що також називають прозорим, скляним, відкритим) – метод тестування програмного забезпечення, що базується на тому, що внутрішня структура, побудова або реалізація системи відомі нам, як тестувальнику. Ми обираємо вхідні данні спираючись на знання коду, який буде їх обробляти. Точно так мі знаємо і результат, що маємо отримати. Знання усіх аспектів реалізації системи, обов’язкова умова цієї техніки.

Згідно до ISTQB тестування білого ящика це:

- Тестування, що базується на аналізі внутрішньої структури системи;
- Тест дизайн, оснований на техніці білого ящика – це процедура написання або вибору тест-кейсів на основі аналізу внутрішньої побудови системи або компоненту;

Переваги методу «білого ящика»:

- Тестування може проходити на різних етапах, немає необхідності чекати інтерфейсу користувача;
- Можна провести більш детальне тестування с покриттям більшої кількості шляхів виконання програми;

Недоліки методу «білого ящика»:

- Для виконання тестування білого ящика необхідна велика кількість спеціальних знань;
- При використані автоматизованого тестування на цьому рівні підтримка тестових скриптів може стати значною проблемою, якщо програма постійно змінюється;

Критерій	Black box testing	White box testing
Визначення	Тестування, як функціональне, так і нефункціональне, не передбачає знання внутрішньої побудови системи або компонента.	Тестування, основане на внутрішній структурі компонента або системи.
Рівні до яких може бути використана дана техніка	В основному: • Системне тестування • Приймальне	В основному: • Юніт-тестування • Інтеграційн

	тестування	е тестування
Хто виконує	Як правило, тестувальники	Як правило, розробники
Знання програмування	Не потрібно	Необхідно
Знання реалізації	Не потрібно	Необхідно
Основа для тест-кейсів	Специфікація, вимоги	Проєктна документація.

Таблиця 1.1. – Таблиця порівняння методів

Тестування методом сірого ящика – метод тестування , який передбачає комбінацію двох попередніх методів. Тобто внутрішня побудова програми відома нам лише частково. Наприклад, доступ до внутрішньої структурі та алгоритмам роботі програмного забезпечення для написання максимально ефективних тест-кейсів, проте саме тестування провадиться за допомогою техніки чорного ящика, а саме з позиції користувача. Цю техніку також називають методом напівпрозорого ящика.

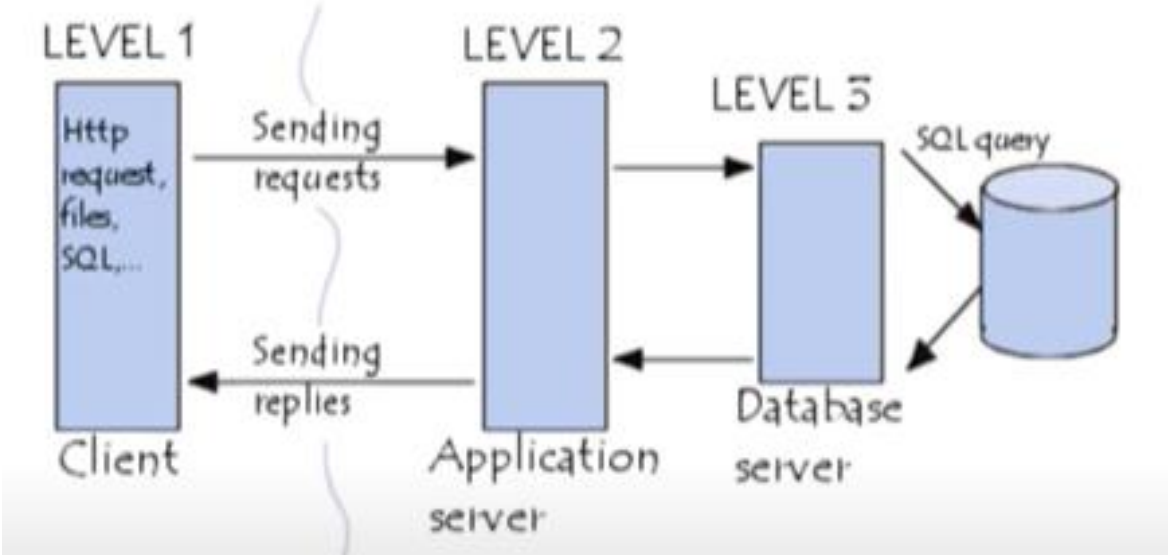
Техніка сірого ящика може бути використана на різних рівнях тестування – від модульного до системного, але головним чином застосовується на інтеграційному рівні для перевірки взаємодії різних моделей програми.

1.3. Клієнт-серверна архітектура

Загалом усі веб-додатки в Інтернеті побудовані за визначеною архітектурою, тобто у нас є клієнтська частина та серверна. На даному зображенні 1.1 спершу є перший рівень ньому знаходиться клієнтська частина, це саме те що відображається у браузері, як для користувача, так і для тестувальника. На рівнях два та три мі бачимо серверну частину, також її наживають бєкендом (back end), у ній відбувається вся логіка роботи додатку[5].

Натискаючи на кнопки на сайті клієнтська частина починає відправляє http запит на сервер, Також цей процес часто називають спілкуванням інтерфейсу програми з сервером.

Протокол передачі Гіпертексту (http)- протокол прикладного рівня для розподілених, спільних інформаційних систем. Це загальний, платформно-незалежний, об'єктно-орієнтований протокол, який може використовуватися в багатьох задачах, таких як сервера імен і розподілені системи управління об'єктами, з розширення методів запиту. Можливість HTTP - це друк і обговорення подання даних, що дозволяє



будувати системи незалежно від переданих даних .[6]

Рисунок 1.2. – Приклад клієнт-серверної архітектури.

Http метод - це саме та дія яку ми хочемо зробити над ресурсом сервері. Їх досить багато:

- Http CONNECT;
- Http PUT;
- Http TRACE;
- Http GET;
- Http POST;
- Http DELETE;
- Http HEAD;
- Http PATCH; [3]

Проте за основні можна виділити чотири з них, а саме :

- Http PUT (замінює всі поточні представлення ресурсу даними запиту);
- Http GET (запрошує представлення ресурсу, запити з використанням цього методу можуть тільки отримувати дані);
- Http POST (використовується для відправки сутностей до певного ресурсу);
- Http DELETE (видаляє вказаний ресурс);[7]

Переваги клієнт-серверної архітектури:

- Відсутність дублювання коду програми-сервера програмами-клієнтами.
- Так як всі обчислення виконуються на сервері, то вимоги до комп'ютерів, на яких встановлено клієнт, знижуються.
- Всі дані зберігаються на сервері, який, як правило, захищений набагато краще за більшість клієнтів. На сервері простіше організувати контроль повноважень, щоб вирішувати доступ до даних тільки клієнтам з відповідними правами доступу.[8]

Недоліки клієнт-серверної архітектури:

- Непрацездатність сервера може зробити непрацездатною всю мережу. Непрацездатним слід вважати сервер, продуктивності якого не вистачає на обслуговування всіх клієнтів, а також сервер, що знаходиться на ремонті, профілактиці.
- Підтримка роботи даної системи вимагає окремого фахівця - системного адміністратора [9].

РОЗДІЛ 2.
Методика тестування

2.1. Тестування API

На перших стадіях розробки програмного забезпечення створюється API (Application Programming Interface) або програмний інтерфейс системи, що слугує зв’язком однієї програми с іншою. API дозволяє напряму обмінюватись інформацією, уникаючи інтерфейс взаємодії з користувачем.

API може бути внутрішнім, тобто компоненті програми використовуються всередині системи і зв’язані між собою, або відкритим, що дозволяє отримувати або інтегрувати інформацію зовнішнім користувачам або програмам. Коректний API повинен бути побудованим з використанням одного и того самого стандарту.[11]

Тестування API проводять для виявлення помилок у взаємодії між системами або між її модулями, спираючись саме на бізнес-логіку проєкту. Для цього використовують різні спеціальні інструменти, за допомогою яких можна виправляти данні у запиті та перевіряти їх точність на виході. Одним із таких інструментів є Postman, що має такі переваги :

- Складати та відправляти запити;
- Зберігати запити в колекцію або папки;
- Задавати параметри запитам;
- Створювати різне оточення для однакових запитів;
- Використовувати, як автоматизовані тести колекцію запитів;

Основні складові Postman це колекція (Collection) на верхньому рівні та запит(Request) на нижньому. Більша частина роботи це створення колекції та опис API за допомогою запитів. Основною частиною колекції є запит, що створюється у спеціальному конструкторі, що є головним полем взаємодії. Колекція станове файл всього проєкту по тестуванню API. Вона об’єднує в собі усі зв’язані між собою запити.[12]

Функціонал Postman дозволяє якісно та швидко протестувати різні сервіси та додатки. Його значними перевагами стали колекція, запити та зручний формат введення параметрів, наявність відправки методами get, post, put та інші.

2.2. Структура тестового випадку (тест-кейсу)

Тест-кейс – набір вхідних даних, передумов виконання, очікуваних результатів та постумов виконання, розроблений для визначеної цілі або текстової умови, наприклад виконання конкретного шляху програми для перевірки відповідності до вимог.[15]

Обов’язкові атрибути тест-кейсу та їх вимоги:

1. Номер тест-кейсу – унікальний ідентифікатор тест-кейсу.
2. Заголовок – короткий та лаконічний опис суті перевірки:
    a. повинен бути чітким, лаконічним, зрозумілим та однозначно характеризувати суть тест-кейсу;
    b. не може містити виконувані кроки та результати.
3. Передумова – опис дій, що необхідно виконати або урахувати, які мають безпосереднє відношення до перевірки:
    a. може містити повну інформацію по стан системи або компонента, що необхідна для початку виконання кроків тест-кейсу;

b. може містити посилання на інформаційні ресурси, що необхідно вивчити перед проходженням тест-кейсу (інструкція, опис системи і т. п.);
c. не може містити данні для авторизації, ця інформація повинна бути внесена в інструкцію, а посилання в передумову;
d. не може містити кроки перевірки, очікувальний або фактичний результат.

4. Кроки перевірки – описання послідовності дій, що необхідно виконати:

a. повинні бути чіткими, зрозумілими, лаконічними та послідовними;
b. уникати зайвої деталізації кроків;
c. не має бути прописні статистичні дані та прикладів, для виключення ефекту пестициду.

5. Очікуваний результат – перевірка, що встановлює те, що мі очікуємо отримати після перевірки:

a. повинен бути для кожного кроку перевірки;
b. повинен бути коротко и зрозуміло описаний стан системи або компонента, що наступає після виконання відповідного кроку;
c. не повинно бути надлишкового опису.

6. Фактичний результат – перевірка, яку ми отримали в ході реального тестування.

7. Додаткові матеріали – фотографії або скриншоти.

8. Загальні вимоги до тест-кейсів:

a. мова опису тест-кейсів повинна бути зрозумілою широкому колу користувачів;
b. тест-кейс повинен бути максимально незалежним один від одного і не мати посилань на інші;

с. тест-кейси формуються у функціональні блоки за їх змістом;

Для більш детального розуміння, можна навести такий приклад тест-кейсу:

Номер	1	
Заголовок	Відправка повідомлення через форму для зворотного зв’язку «Контакти»	
Передумова	Відкрита сторінка сайту Є доступ до пошти адміністратора	
Крок	Очікувальний результат	Фактичний результат
В нижньому меню сайту натиснути на посилання «Контакти»	Відкрилась сторінка «Контакти»	
Ввести в значення «Ваше ім’я», що складається з латинських літер, кирилиці	В полі «Ваше ім’я» відображаються введені данні	
В поле «e-mail» ввести коректну електронну пошту	В поле «e-mail» відображаються введений email	
В поле «Повідомлення» ввести значення, що складається з	В полі «Повідомлення» відображаються введені	

латинських літер, кирилиці	данні	
Під заповненою формою натиснути кнопку «Відправити»	З’являться текст «Дякуємо, ваше повідомлення було відправлено». Всі поля були очищені	
Перевірити пошту адміністратора	На пошту адміністратора прийшло повідомлення, відправлене через форму зворотного контакту, що містить повідомлення введене в кроках 1-5	

Таблиця 2.1 – Приклад тест-кейсу

2.3. Тест план

Тест план- це документ, що входить в список текстової документації згідно до стандарту ISO/IEC/IEEE 29119 він представляє собою шаблон. При створенні тест плану можна також використовувати такі шаблони, як RUP. [16]

У вимогах ISO 9001 до документації написано, що вона повинна бути прописана у тому обсязі, що буде достатньою для впевненості в тому, що всі процеси виконуються так, як заплановано. Під час створення документації необхідно враховувати всі вимоги усіх членів команди та замовника, стандарти компанії особливості проєкту и т. д.

При написанні тест-плану мі починаємо думати про стратегію тестування проєкту, про ті проблеми, з якими доведеться зустрітися. Виникає багато питань, що потрібно з’ясувати між членами команди.

Будь-який тест план повинен:

- Пройти внутрішнє ревю між всіма членами команди;
- Фінальна версія повинна бути доступна кожному та в будь-який момент;
- Бути простим у розумінні, конкретно визначеним для кожного;
- Актуальним та оновлюватися по мірі необхідності;
- Логічно пов’язаний з іншою документацією по даному проєкту;

2.4. Управління дефектами в тестуванні програмного забезпечення

Баг / помилка – це несправність, що є наслідком помилки кодування, а дефект – це відхилення від початкових бізнес-вимог. Обидва терміну часто плутають, адже вони мають схожий характер. Але все ж таки це недоліки, що мають бути виправлені в найкоротші строки.

Коли тестувальник виконує тестовий випадок, він може стикнутися з результатом, що протирічить очікувальному результату. Саме ця зміна в результатах тесту і називається дефектом програмного забезпечення.

Повідомляючи розробнику про знайдену помилку, повинен бути складений документ, а саме звіт про помилку що має містити таку інформацію:

- Defect_ID – унікальний ідентифікатор номер для дефекту;

- Опис дефекту – детальний опис дефекту, включаючи інформацію про модуль, в якому знайдено цей дефект;
- Версія – версія додатку, у якому було знайдено дефект;
- Кроки – детальні кроки разом із скриншотами, за допомогою яких можна відтворити дефект;
- Дата появи дефекту;
- Посилання – на документ, вимоги, дизайн, архітектура, щоб краще зрозуміти дефект;
- Знайдено – ім’я / ідентифікатор тестувальника, що виявив дефект;
- Стан – стан дефект, виправлений, у розробці і т. д.;
- Виправлено – ім’я / ID розробника, що виправив даний дефект;
- Дата закриття дефекту;
- Серйозність, яка описує вплив дефекту на додаток;
- Пріоритет, пов'язаний з тип, як швидко треба виправляти дефект. Пріоритет може бути високий, середній та низький, що залежить від впливу, при якому дефект повинен бути виправлений.

Управління дефектами – це систематичний процес виявлення і усунення помилок. Цикл управління дефектами містить наступні етапи зображення 1.3.:

1. Виявлення дефект;
2. Категоризація дефект;
3. Усунення помилки розробниками;
4. Перевірка тестами;
5. Закриття помилки;
6. Звіт про дефект в кінці проєкту;

Рисунок 2.1. – Цикл управління дефектами.

Відкриття

На етапі виявлення проєктні групи повинні знайти як умога більше дефектів, перед тим як кліент зможе іх знайти. Дефект вважається аиявленим та його статус змінюється на прийнятий, коли він признан розробником.

Категорії помилок

Класифікація дефектів допомагає розробникам програмного забезпечення визначити приорітет своїх задач в першу чергу усунути дефекти, що загрожують роботоздатності продукту.

Критичний – дефект має бути усунени негайно, адже він може призвести до великих проблем з пробуктом. Наприклад, якщо на сайті банку не працює функція входу в систему, так як це одна з основних функцій, саме це буде серйозною помилкою.

Високий – дефект негативно впливає на основні функції проєкту, наприклад знужуючи продуктивність сайту.

Середній – дефект вносить мінімальні відхилення від вимог до продукту. Наприклад, сайт некорентно відображається на мобільних присроях.

Низький – минімальний нефункціональний вплив на продукт. Наприклад деякі посилання не відкриваються.

Вирішення

Після того як дефекти пряйнті і класифіковані, можна виконати наступні кроки для їх вирішення:

- Назначення: проблеми відправлені розробниуи чи іншому технічному сспеціалісту для виправлення і змінено статус на очікуючий.
- Графік уснення: сторона розробника бере на себе відповідальність на цьому єтапі, вони складають графік усунення цих дефектів залежно від їх пріорітету;
- Виправлення: пока група розробників займаєтся усуненням дефекту, диспечер тестів слідкує за цим процесом знідно графіку.
- Повідомити про рішення: отримати звіт про усунення помилок від розробників, коли все дефекти буде усунено.

Верифікація

Після того як команда розробників виправила дефект і повідомила про це, команда тестування перевіряє, чи дійсно виправленя все заявлені помилки.

Закриття

Після усунення і перевірки дефекту його статус змінюється на закритий. Якщо він не усунутий, відправляється повторне повідомлення до команди розробників, щоб вони повторно перевірили дефект.

Складання звіту

Завершується процес скраданням звіту, щоб повідомити про поточну ситуацію з дефектами та отримати зворотний зв’язок.

Система звіту про знайдені помилки (баг-репорт)

Баг або дефект репорт – це документ, що описує ситуацію або послідовність дій, яка приводить до некоректної робити об’єкту тестування.

Різні системи менеджменту дефектів, надають різні поля для заповнення та різні структури написання дефектів.

Заголовок	
Короткий опис	Короткий опис проблеми, що вказує на причину и тип помилкової ситуації.
Проєкт	Назва проєкту
Компонент додатку	Назва частини або функції трестованого продукту
Номер версії	Версія в якій була знайдена помилка
Серйозність	Найбільш розповсюджена п’ятирівнева система градації серйозності дефекту: • S1 Блокуючий • S2 Критичний • S3 Значний • S4 Незначний • S5 Тривіальний
Пріоритет	Пріоритет дефекту: • P1 Високий • P2 Середній • P3 Низький
Статус	Статус багу залежить від кого життєвого циклу.
Автор	Людина, що створила баг-репорт.
Призначення	Ім’я розробника, який повинен виправити помилку.
Оточення	Інформація про оточення, в якому було

	знайдено дефект: операційна система, сервіс пак, версія браузера і т.д.
Опис	
Кроки для відтворення	Кроки відтворивши які можна знайти помилку.
Фактичний результат	Результат, що отримано після проходження кроків
Очікуваний результат	Результат, що повинен бути за відсутності дефекту
Доповнення	
Додаткові файли	Файл з логами, скриншот або інший документ, що допоможе дізнатись причину помилки або її спосіб вирішення.

Таблиця 2.2 – Приклад звіту про помилку

2.5. Автоматизоване тестування

Автоматизоване тестування – це метод тестування програмного забезпечення, що виконужться з використанням спеціальних програмних засобів, які в свою чергу необхідні для виконання набору тестових прикладів. [5]

Програмне забезпечення для автоматизації тестування також може вводити тестові дані в тестове отчення порівнювати очікуваний результат з фактичним та створювати звіти про тести.

Послідовні цикли розробки, собливо в великих компаніях (Google, Facebook, Microsoft і т.д.) потребують багаторазового виконання одного і того ж самого набору тестів. Використовуючи інструмент автоматизації тестування, можна записати цей набір тестів і використовувати його за необхідністю. Мета автоматизованого

тестування – зменшити кількість тестових прикладів, що необхідно виконувати вручну.

Автоматизація це гврний спосіп підвищіти ефективність, а також охоплення та швидкість тестування програмного забезпечення, коли необхідно перевіряти одні й тіж самі тестові сценярії.

- Ручне тестування всіх робочих процесів, полей та негативних сценаріїв потребує багато часу та ресурсів.
- Складно тестувати багатомовні сайти вручну.
- Не потребує втручання людини.
- Збільшує швидкість виконання тестів.
- Допомагає пркащити покриття тестами.

Випадки в яких автоматизоване тестування поркщить процес тестування:

- Високі ризикита помилки недопустиі.
- Тестові сценарії регулярно повторюються.
- Тестові сценарії громіткі та велики для виконання вручну.

Наступна категорія не підходить для автоматизації:

- Сценарії тестування, вимоги до яких часто змінюються.
- Тестові приклади виконються на разовій основі.
- Нові тестові приклади, що не виконувались хоча б один раз вручну.

Процес автоматизайії включає такі кроки (Рисунок 1.4):

Рисунок 2.2. - Кроки для автоматизації процесу тестування.

Вибір інстурментів значно залежить від технології на якій побудовано додаток, що тестується. Наприклад, QTP не підтримує Informatica. Тим чином вони несуміні для тестування.

Об'єм автоматизаціє – це область тестованого додатку, що буде автоматизована. Її допомагють взначити наступні пункти:

- Сценарії з великим об'ємом даних.
- Загальні функції додатку.
- Технічна здатність.
- Частота повторного використання тестів.
- Складність тестових випадків.

На етапі планування, проєктування та розробки ми складаємо стратегію та план автоматизації, що містить наступні деталі:

- Обрані інструменти автоматизації.
- Конструктор каркаса та його особливості.

- Елементи, що входять та виходять за рамки автоматизованого тестування.
- Піднотовка стендів автоматизації.
- Графік та тимчасова шкала сценаріїв виконання.
- Результати тестування автоматизації.

На етапі виконання тесту, виконуються сценарії автоматизації. Сценяріям необхідно ввести тестові дані, перед її запуском. Після викоання вони предствляють звіт, що до виконаної роботи.

На етапі обслуговування автоматизованого тестування проводяться перевірки того, як працюють нові функції додані в програмне забезпечення. Перевірки виконуються коли додаються нові сценарії автоматизації і їх потрібно перевірити , щоб повисити ефективність з кожним наступним циклом випуску.[17]

Для автоматизованого тестування програмного забезпечення використовують 4 типи фреймворків:

1. Платформа автоматизації на очнові даних.
2. Фреймворк автоматизації на оччнові ключових слів.
3. Модульна платформа автоматизації.
4. Гібридне оточення автоматизіції.

Для отримання максимальної ефективності від автоматизованого тестування, необхідно дотримуватися наступних правил:

- Обсяг автоматизації необхідно детально визначити до початку проєкту. Це дозволить переконатися, що очікування від автоматизації будуть виправдані.
- Визначте правильний інструмент автоматизації: інструмент не повинен вибиратися на підставі його популярності, він повинен відповідати вимогам автоматизації на конкретному проєкті.
- Виберіть відповідний фреймворк.

- Стандарти створення сценаріїв. При написанні сценаріїв для автоматизації необхідно дотримуватися стандартів. Ось деякі з них:
    - створені єдині скрипти, коментарі і відступи коду;
    - розробіть правила найменування тестових сценаріїв;
    - застосовуйте необхідні документи, якщо, наприклад, складно зрозуміти проходження тестового сценарію без скріншота і / або специфікації.
- Визначте метрики і стежте за ними. Успіх автоматизації не можна визначити лише шляхом порівняння витрачених зусиль, на той чи інший вид тестування. Ось основні показники:
    - відсоток виявлених дефектів;
    - час, необхідний для тестування автоматизації випуску кожного нового циклу;
    - мінімальний час необхідний для випуску;
    - поліпшення продуктивності.

2.6. Ручне тестування

Ручне тестування програмного забезпечення - це процес перевірки ПО, що виконується фахівцями вручну. Це означає, що для його проведення не використовуються будь-які спеціальні автоматизовані засоби.

Програмне забезпечення перевіряється інженерами з тестування, які беруть на себе роль кінцевих користувачів, моделюють ситуації відповідно до тестовим сценаріям і фіксують результат. Завдання ручного тестування програмного забезпечення - виявити будь-яку поведінку, що відрізняється від очікуваного користувачем. Це важливий етап забезпечення якості ПЗ, який спрямований на ретельне дослідження програмного коду та виявлення помилок в роботі систем.[22]

Ручне тестування може проводитися в рамках регресійного (тестування різних змін), інтеграційного (взаємодія з іншими системами і ПО) і при системному функціональному тестуванні.

Переваги ручного тестування:

- Оперативність (проєкт по ручному тестуванні досить швидко можна запустити);
- Гнучкість (ручне тестування дозволяє змінювати і актуалізувати тести в разі потреби);
- Адаптація (легко адаптується до динамічних змін тестованої системи);
- Людський фактор (ручне тестування повністю імітує використання системи кінцевим користувачем);
- Зворотній зв'язок (дозволяє отримати користувальницький фідбек, особливо на разових і нетривалих проєктах);

Основні етапи ручного тестування прогрмного забезпечення:

1. Підготовчий:
   a. Аналіз документації (технічні вимоги, специфікації, бізнес-вимоги і т.д.);
   b. Розробка та узгодження плану ручного тестування, тест-кейсів, термінів, кількості ітерацій та інше;
   c. Оцінка можливих ризиків, визначення меж проєкту;
2. Основний:
   a. Виконання ручного тестування на основі специфікацій вимог і згідно заздалегідь підготовленими тестовими сценаріями;
   b. Фіксація виявлених дефектів в системі відслідковування помилок;
   c. Виправлення помилок та повторне тестування;
3. Заключний:
   a. Розробка та узгодження звітів про проведене тестування;

b. Надання рекомендацій щодо впровадження ПО

Інструменти ручного тестування

Оскільки ручне тестування програмного забезпечення досить гнучке, воно допускає використання великої кількості різноманітних інструментів.

Управління тестуванням зазвичай ведеться в спеціалізованих системах, на зразок HP ALM, IBM Rational Quality Manager, MS Team Foundation Server, TestRail, TestLink, Jira та Redmine.

Для пошуку, конвертації і порівняння файлів можуть використовуватися Notepad ++, Intype або PSPad. Серед файлових менеджерів популярністю користуються Total Commander, trolCommander, Free Commander і Far Manager. З XML-редакторів часто використовуються Altova XML Spy, Xsemmel і XMLPad.

З інструментів для створення скріншотів, відео, скрінкасти і анімації (gif) можна виділити Snagit, ScreenHunter, Monosnap, Snipping Tool, GreenShot, Recordit, CamStudio, Jing, LICEcap і Ashampoo Snap. Для порівняння зображень і інших графічних файлів інженери з ручного тестування часто використовують FastStone Image Viewer, ImageDupeless і ImageDiscerner.

2.7. Практична частина

Основними видами тестування веб-додатків можна виділити:

- Тестування функціональності;
- Тестування зручності використання (UI);
- Тестування сумісності;

Складено тест-план відповідно до основних етапів тестування сайту працевлаштування студентів ХДУ

Під час ***тестування функціональності*** було перевірено усі наявні посилання (внутрішні, вихідні, та на інші елементи розміщені всередині сторінки), а також посилання на бази даних.

Перевірено форми, що використовуються для отримання інформації від користувача та для взаємодії з ним (зображення 2.3.). А саме правильність роботи валідації у кожному полі форми, значення, що використовуються за замовчуванням, перевірка форм, обов'язкових для заповнення. На цьому етапі тестування було виявлено ряд помилок, пов'язаних з заповненням форм (наприклад, при виборі дати народження є можливість брати дати з майбутнього). Баг-репорт цього дефекту було направлено до розробника.

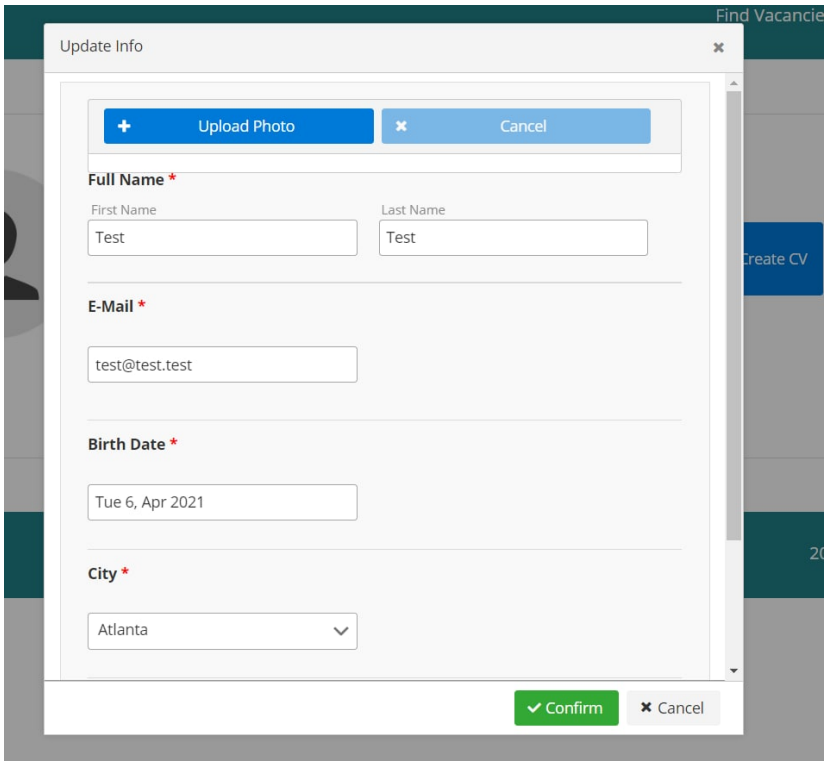


Рисунок 2.3. – Форми для реєстрації користувача

Взаємодія веб-додатку з базою даних, а саме перевірка даних на цілісність, проведення тестування сайту на помилки при редагуванні, видаленні, зміні форм або інших діях, що мають відношення до бази даних. Перевірено, що всі запити до бази виконуються коректно, оновлюються та відправляються належним чином.

Тестування зручності використання (зображення 2.3.)показало, що у сайту досить проста навігація (засоби для перегляду сторінок). Проте оцінити зовнішній вигляд сайту в цілому неможливо, адже фронт-енд ще не має свого кінцевого варіанту.

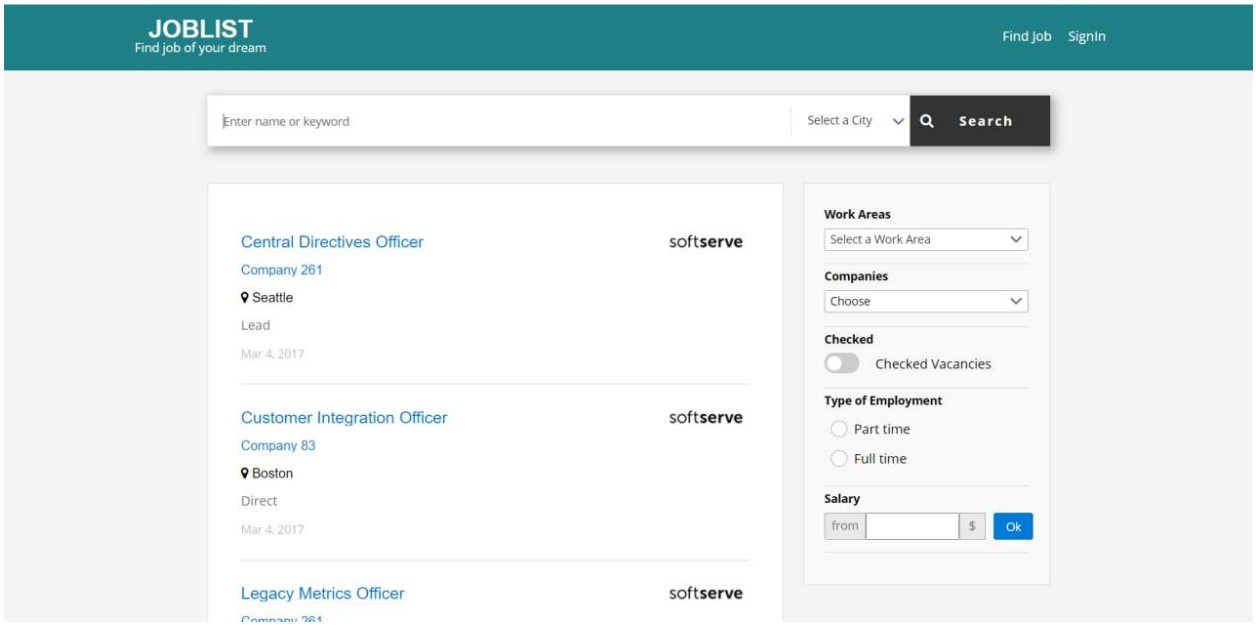


Рисунок 2.4. – Головна сторінка сайту

Також було протестовано API, для перевірки правильності сервера.

Перевірена сумісність з різними конфігураціями та переметами різноманітних браузерів. Перевірена робота веб-додатку в браузерах Firefox, Google Chrome, Safari, Opera різних версій.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розглянуто види та рівні тестування програмного забезпечення, клієнт-серверну архітектуру побудови веб-додатків. Визначено переваги та недоліки ручного та автоматизованого тестування. Обрано найбільш ефективний спосіб тестування у рамках поставленої задачі – це ручне тестування програмного забезпечення, адже воно дозволяє оцінити ефективність робити різних систем з точки зору користувача, на відміну від автоматизованого, яке показує лише працездатність системи. Також вивчено структуру створення тест-кейсів, що дозволило реалізувати тест-кейси відповідно до вимог програмного забезпечення.

Протягом другої частини обрано техніку тестування для сайту у рамках університету «Працевлаштування студентів ХДУ». Створено тест-кейси для більшості користувацьких сценаріїв. Протягом процесу тестування було виявлено ряд дефектів пов’язаних з функціоналом сайту, а саме формою реєстрації, полями для пошуку та формою створення вакансії. Складена текстова документація відповідно до знайдених помилок та дефектів додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Протокол передачи гипертекста http/1.1 [Електронний ресурс] – Режим доступу : http://lib.ru/WEBMASTER/rfc2068/

2. PATCH Method for HTTP [Електронний ресурс] – Режим доступу : https://tools.ietf.org/html/rfc5789#section-2

3. Methods for HTTP [Електронний ресурс] – Режим доступу : https://tools.ietf.org/html/rfc7231#section-4

4. Software testing training and stoftware testing services. [Электронный ресуср]. Режим доступу: http://www.rbcs-us.com/

5. Дастін Е., Рєшка Д., Пол Д. \ Ї. Молодцова, М. Павлов. Автоматизоване тестування программного забеспечення. Видавництво «ЛОРИ», 2003 – 435 с.

6. Котляров В.П., Коликова Т.В. Основи тестування программного забезпечення., 2006. – 321 с.

7. Библиотека MSDN. Источник информации для разработчиков, использующих средства, продукты, технологии и службы корпорации Майкрософт. [Электронный ресурс]. – Режим доступу: http://msdn.microsoft.com

8. Поскребышев Р.С., Тарасов В.Г. API на основе SOAP и REST // Молодые ученые - ускорению научно-технического прогресса в XXI веке. Ижевск: ИННОВА, 2016. С. 404-410.

9. Акинин Ю.Р., Барабанов А.В., Гребенникова Н.И. Быстрое создание REST API сервиса на основе облачных технологий Azure // Вестник Воронежского государственного технического университета. 2012. №12-1. С. 66-68.

10. . POSTMAN URL [Електронний ресурс] – Режим доступу: https://www.getpostman.com/ .

11. Чеглаков А.Л. Композиция WEB-сервисов на основе архитектуры REST // Инновационная наука. 2016. №12-2 . С. 118-120.

12. С.В. Бирюков АНАЛИЗ СТРАТЕГИЙ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ // Секция математического обеспечения и применения ЭВМ. - 2010. - №УДК 004.4. - С. 59-63.

13. Канер С., Фолк Д., Кек Нгуен Е.. Тестирование программного обеспечения. – Киев: ДиаСофт, 2000. – 544 с.

14. Бейзер Б. Тестирование “черного ящика”. Технология функционального тестирования программного обеспечения. – СПб.: Питер, 2004. – 318 с

15. Куликов С. П Тестирование программного обеспечения. Базовый курс. EPAM Systems, 2015–2021. - 298 с.

16. Блэк Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование. - СПб.: Лори, 2006. с. 576.

17. Винниченко И. Автоматизация процессов тестирования. - СПб.: Питер, 2005. с. 203.

18. Липаев В.В. Отладка сложных программ. - М.: Энергоатомиздат, 1993. с. 235.

19. Тамре Л.Введение в тестирование программного обеспечения. - СПб.: Вильямс, 2003. с. 386.

20. Шнейдерман Б. Психология программирования. - М.: Радио и связь, 1984. с. 512.

21. Липаев В.В. Надёжность программных средств. - М.: СИНТЕГ, 1998. с. 240.

22. Смагин В.А., Солдатенко В.С., Кузнецов В.В. Моделирование и обеспечение надёжности программных средств АСУ. - СПб.: ВИКУ им. А.Ф. Можайского, 1999. с. 49.

23. Соммервилл И. Инженерия программного обеспечения. - СПб.: Вильямс, 2002. с. 624.

24. Чеглаков А.Л. Композиция WEB-сервисов на основе архитектуры REST // Инновационная наука. 2016. №12-2 . С. 118-120.