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

МЕТОДИ ТЕСТУВАННЯ ПРОГРАМНИХ ПРОДУКТІВ НА ПРИКЛАДІ WEB-СЕРВІСУ KSU ON-LINE

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти "бакалавр"

Виконав: студент 4 курсу
Спеціальності 121 Інженерія
програмного забезпечення
Освітньо-професійної програми
«Інженерія програмного забезпечення»
першого (бакалаврського) рівня освіти
Болокан Владислав Анатолійович
Керівники кандидат технічних наук,
доцент
Шишко Людмила Станіславівна
кандидат фізико-математичних наук,
доцент
Єрмолаєв Вадим Анатолійович
Рецензент кандидат фізико-
математичних наук, доцент
Котова Ольга Володимирівна

Херсон – 2020

ЗМІСТ

ВСТУП

При розробці сучасного програмного забезпечення значну частину часу і зусиль займає процес тестування. Найчастіше він зводиться до аналізу поведінки програми на спеціально підібраному наборі тестів. Оскільки перебір всіх можливих ситуацій неможливий на практиці, намагаються вибрати невелике число тестів, таких щоб з поведінки програми на них можна було судити про її поведінку в цілому. Використовуються різні метрики покриття коду для кількісної оцінки якості набору тестів. Якісним вважається набір тестів, що задовольняє деякому критерію повноти. За частую критерій повноти визначається за допомогою обраної метрики покриття коду. Автоматизація процесу побудови набору тестів дозволить істотно скоротити витрати на тестування. Існує безліч підходів до автоматизованого побудови набору тестів. Однак лише деякі з них мають програмну реалізацію, наприклад Microsoft Pex. Всі ці реалізації націлені на модульне тестування, при якому покриття фрагментів програми відбувається незалежно один від одного.

Актуальність теми. Тестування дозволяє підтримувати програмний продукт чи сервіс в працездатному стані і уникати великих витрат на ліквідацію проблем пов’язаних з виходу із ладу основних функцій.

Мета дослідження: реалізація автоматизованого та ручного тестування веб-сервісу.

Відповідно до мети дослідження визначено такі ***завдання***:

1. Проаналізувати існуючі методи тестування.
2. Розглянути детально методи тестування.
3. Знайти найбільш ефективні інструменти для тестування даного веб-сервісу.

4. Протестувати веб-сервіс.
5. Прослідкувати запобігання дефектів у роботі.

Об'єкт дослідження: системи автоматизованого та навантажувального тестування.

***Предмет дослідження*:** Процес тестування систем веб-сервісу

Методи дослідження включають в себе:

- тестування програмного забезпечення;
- визначення якості продукту;
- верифікація програмного забезпечення;
- автоматизація тест-кейсів.

Практичне значення одержаних результатів полягає у залученні досліджуваного матеріалу для підготовки до тестування веб-систем. Матеріали дипломної роботи можуть бути використані при написанні курсових робіт, рефератів та інших наукових розвідок.

Структура дослідження. Дипломна робота складається з вступу, двох розділів, висновків та списку використаних джерел.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Місце тестування в структурі розробки ПЗ

Тестування це процес з допомогою якого виконується перевірка програми на відповідність між очікуваним результатом та отриманим. Найчастіше використовуються ітеративні процеси розробки ПЗ, зокрема, технологія **RUP** - Rational Unified Process. На Рис. 1.1 можна побачити цикл RUP. Їх роль не зводиться просто до виявлення помилок у можливій повноті і якомога раніше. Вони повинні брати участь в загальному процесі виявлення та усунення найбільш істотних ризиків проєкту. Для цього на кожну ітерацію визначається методи та мета тестування. У свою чергу, кожен виявлений дефект повинен пройти через свій життєвий цикл.[1]

Рис. 1.1. RUP

Тестування проходить циклами, кожен має своє завдання.

Ітераційна модель життєвого циклу не починається з повної специфікації вимог. У цій моделі розробка починається з конкретизації та реалізації лише частини програмного забезпечення, яке потім переглядається з метою визначення подальших вимог. Більше того, в ітераційній моделі ітераційний процес починається з простої реалізації невеликого набору програмних вимог, що ітеративно розширює версії, що розвиваються, до тих пір, поки повна система не буде впроваджена та готова до розгортання. Кожен випуск «Iterative Model» розробляється у конкретний та фіксований часовий період, який називається ітерацією. [2]

За кореляцією різних завдань, ітерації групуються у фази. На першому етапі **(Початок)** зазвичай проводять аналіз; на другому(**Розробка) -** проєктування і випробування ключових проєктних рішень; на третьому **(Побудова)** - найбільш велику частку задач розробки та тестування; а на заключному **(Передача)** - вирішуються завдання тестування і передачі системи замовнику.

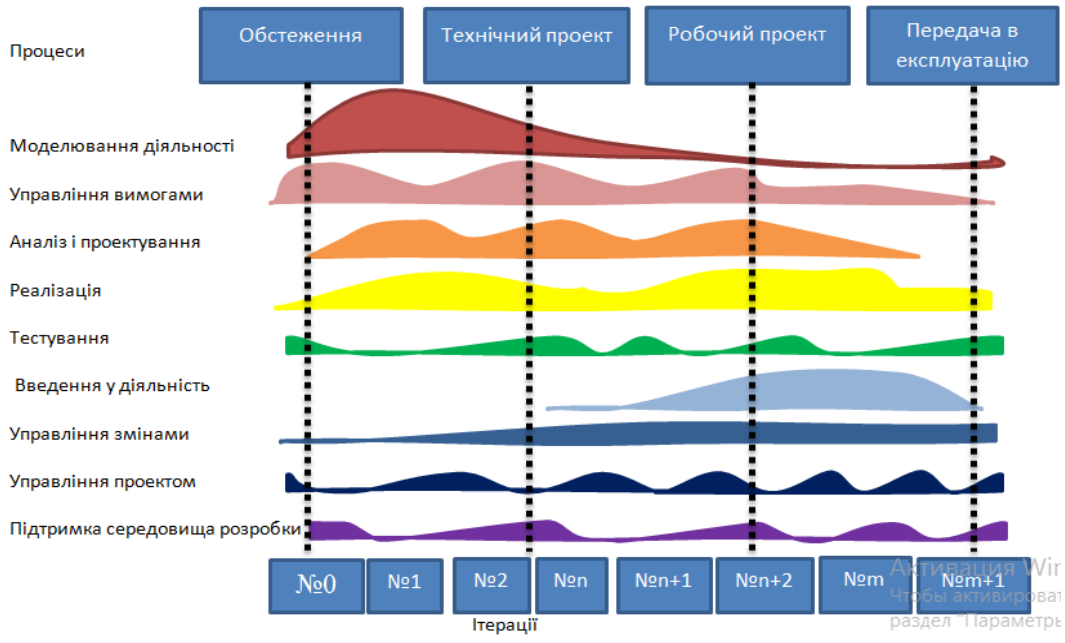

Рис. 1.2. Життєвий циклу програмного продукту

Структура фірми - розробника програмного забезпечення відображає етапи життєвого циклу програмного засобу. Той чи інший підрозділ виконує одну або декілька фаз життєвого циклу ПЗ.

Аналітичний відділ виконує зазвичай такі завдання:

- визначення ідеологічного і функціонального напрямку розвитку програмного продукту; здійснення попередньої перевірки;
- визначення функціональних можливостей системи;
- визначення (спільно з розробниками) технічних вимог до системи;
- характеристика бізнес-процесів предметної області в поняттях, зручних для розробників;
- оформлення поставлених задач і специфікацій на внесення змін до програмного засобу при зміні законодавства, вимог замовників або виявленні нових можливостей;
- перевірка ходу реалізації потенціалу програмних продуктів.

Відділ документації зазвичай не виділяють як самостійний, а тому часто він є частиною іншого відділу, наприклад, аналітичного. До завдань відділу входять складання технічної документації для кінцевого користувача, аналіз змін в ПЗ і поновлення документації.

Відділ розробки це ключовий відділ. Якщо іншими часто можна знехтувати, то без відділу розробки не обійтись. До завдань відділу належать:

- визначення (спільно з аналітиками) технічних вимог до системи;
- реалізація основних функцій ПЗ;
- реалізація нових можливостей ПЗ;
- усунення виявлених при тестуванні недоліків;

- адаптація ПЗ для роботи за різних умов (зміна СУБД, мови програмування, операційної системи та ін.)
- оптимізація ПЗ (підвищення швидкості роботи, надійності та ін.).

Відділ тестування. До завдань відділу входять:

- перевірка якості ПЗ;
- підготовка тестової документації (плани тестування та ін.);
- пошук та локалізація помилок у роботі ПЗ;
- контроль відповідності документації ПЗ стандартам і реально реалізованим функціям;
- участь в розробці та впровадженні засобів контролю якості;
- автоматизація тестування;
- оцінка роботи ПЗ на різних програмно-апаратних платформах.

Іноді на відділ тестування також покладаються збірка та випуск ПЗ (зазвичай це завдання відділу розробки) [3].

Усі підрозділи взаємодіють між собою, результати взаємодії впорядковують і представляють виробничі технологічні процеси. Технологічні процеси, звичайно, регламентовані внутрішньо корпоративними правилами; в сукупності представляють собою цикл виробництва ПЗ.

Таких циклів всередині компанії може бути велика кількість. Як зразок розберемо схему взаємодії відділу тестування ПЗ з іншими відділами за умови виявлення помилки у роботі програмного забезпечення на Рис.1.3.

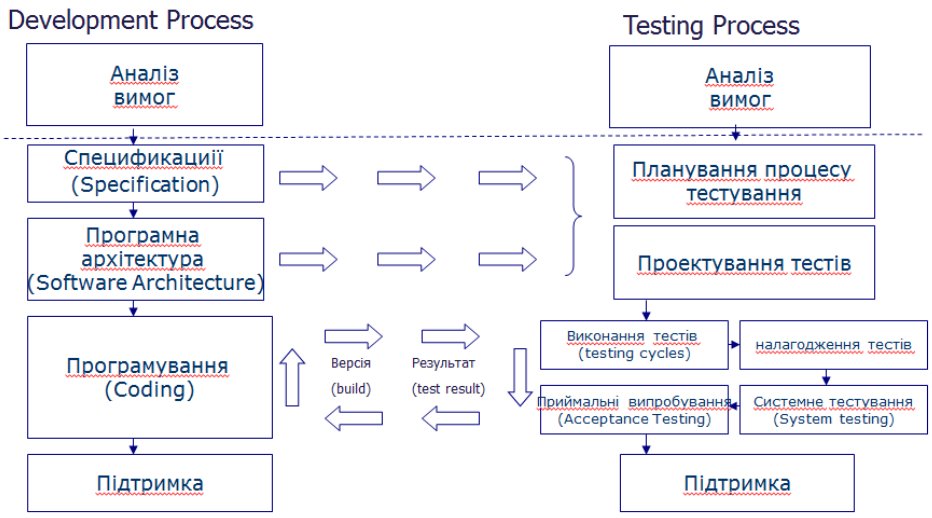

Рис. 1.3. Схема взаємодії відділу тестування програмного забезпечення з іншими відділами

1.2. Види та рівні тестування

Види тестування

Існує класифікація видів тестування, яка ґрунтується на тому рівні деталізації робіт проєкту, на котрий вона направлена. На малюнку 1.4 схематично показані рівні тестування. Ці ж види тестування можна пов'язати з етапом, на якому вони здійснюються.

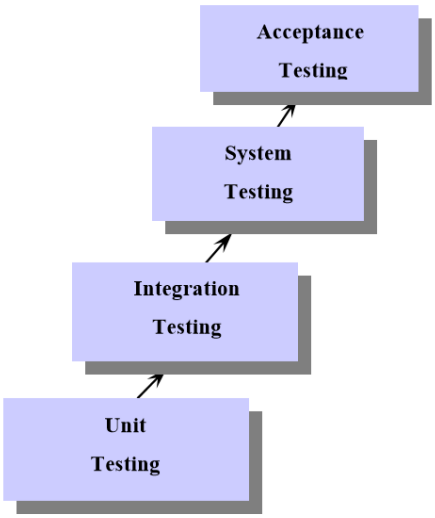

Рис. 1.4. Рівні тестування

Модульне тестування або (Unit-testing) - рівень тестування, на якому перевіряються найменші елементи, такі як клас або функція. На даному рівні використовуються методи «білого ящика». У більшості проєктів модульне тестування виконується розробниками.

Інтеграційне тестування (Integration testing) - рівень тестування, на якому найменші модулі ПЗ об'єднують і тестують групами. Найчастіше, integration testing проводять після виправлення помилок, знайдених під час модульного[4].

Системне тестування (System testing) – це рівень тестування ПЗ, який є обов’язковим для перевірки відповідності системи оригінальним вимогам до повної інтегрованої системи. Тестування системи відноситься до методів тестування "чорної скриньки", не вимагає знання внутрішнього пристрою системи. Тестування системи здійснюється за допомогою зовнішніх програмних інтерфейсів і пов'язане з перевіркою користувальницького інтерфейсу (або через інтерфейс користувача), що проводиться шляхом моделювання дій користувача на елементах цього інтерфейсу.

Приймальне тестування (Acceptance testing) - є тестуванням готового продукту кінцевими користувачами на реальному оточенні, в якому буде функціонувати тестований додаток. Приймальні тести розробляються користувачами, зазвичай, у вигляді сценаріїв. Для того, щоб знайти більше помилок рекомендується планувати не тільки системне тестування і приймальне, але і модульне і інтеграційне [5].

Статичне тестування (Static testing) - тестування, в ході якого тестована програма (код) не виконується (запускається). Аналіз програми здійснюється на основі кінцевого вихідного коду, який вираховується вручну, або аналізується за допомогою спеціальних інструментів.

Приклади статичного тестування:

- огляди (Reviews); інспекції (Inspections);
- наскрізні перегляди (Walkthroughs);
- аудити (Audits).

Динамічне тестування (Dynamic testing) - тестування, в ході якого тестована програма (код) виконується (запускається). Альфа-тестування - тестування в процесі розробки. Бета-тестування - тестування виконується користувачами (end-users). До того як випускається програмне забезпечення, воно має пройти стадії альфа (або внутрішнього пробного використання) та бета (пробного використання продукту із залученням зовнішніх вибраних користувачів) версій. Звіти про знайдені помилки, що надходять від вибраних користувачів поданих версій продукту, обробляються відповідно до визначених процедур, які включають тести (будь-якого рівня), які проводяться фахівцями групи розробки. Такий вид тестування заздалегідь не може бути спланований.

«Смок-тест» (Smoke Testing, «димове тестування») в тестуванні це мінімальний набір тестів на виявлення явних помилок. Димовий тест зазвичай виконується програмістом; програму не має сенсу віддавати на глибше тестування якщо вона не проходить цей тест [6].

Види тестування:

Функціональне тестування (Functional testing) - мета даного тестування полягає у тому, щоб переконатися у якісному та повному функціонуванні об'єкта тестування:

- кожна функціональна вимога транслюється в тест-кейси (використовуючи техніки «чорного ящика») для того, щоб перевірити, що система функціонує в точності, як і описано в специфікації (функціональних вимогах до системи);
- перевіряється, чи всі функціональні вимоги дійсно запрограмовані / реалізовані.

Тестування продуктивності (Perfomance testing) - тестування, яке виконується із метою виявлення, як швидко працює частина системи або система у цілому під певним навантаженням. Також може використовуватись для перевірки та підтвердження атрибутів якості системи, таких як надійність масштабованість та споживання програмним засобом ресурсів.

Існує особливий вид тестів, коли робиться спроба досягнення кількісних границь, обумовлених характеристиками системи і її оточення:

- продемонструвати, що при заданих умовах система задовольняє критеріям продуктивності;
- виміряти, яка частина системи є причиною «поганої» продуктивності системи;
- виміряти час реакції на дію користувача, час відгуку на запит, і тощо.[7].

Тестування навантаження (Load testing) - це найлегша форма для тестування продуктивності. Зазвичай тестування навантаження проводиться для оцінки поведінки самої програми під заданим навантаженням. Цим навантаженням може бути, кількість одночасно працюючих користувачів програмного засобу, що здійснюють вказану кожному кількість транзакцій за відрізок часу. Поданий вищу тип тестування зазвичай дозволяє отримати середній час відгуку усіх важливих транзакцій. У разі спостереження за мережею, базою даних або сервером додатків.[8]

Тестування стабільності (або Stability testing) проводиться завжди з метою переконатися, що додаток витримує виділене навантаження протягом тривалого відрізку часу. При проведенні цього виду тестування здійснюється спостереження за споживанням програмним засобом ліміту пам'яті, для того щоб виявити потенційні

витоки. Таке тестування може виявити деградацію продуктивності програмного засобу, яке виражається у більшій мірі у зниженні швидкості обробки інформації або збільшенні часу відповіді програмного засобу після тривалої.

Тестування зручності використання (Usability testing) - дослідження, яке виконується з метою визначення, чи зручний деякий штучний об'єкт (такий як веб-сторінка, призначений для користувача інтерфейс чи пристрій) для його передбачуваного застосування. Таким чином, перевірка ергономічності вимірює ергономічність систем або об'єкта. Перевірка ергономічності програмного засобу зосереджена на вибраному об'єкті або невеликому наборі таких об'єктів.[9]

Тестування інтерфейсу користувача (UI testing) - тестування графічного інтерфейсу користувача для того, щоб запевнитись, що він відповідає прийнятим стандартам і їх вимогам. Перевіряється, як додаток обробляє введення з клавіатури і «мишки» і як відображаються елементи графічного інтерфейсу (текст, кнопки, меню, списки та інші елементи).[10]

Тестування безпеки (security testing) - оцінка уразливості програмного забезпечення до шкідливих атак. Комп'ютерні системи нерідко є цілю незаконного проникнення в систему. Під таким проникненням говориться про великий діапазон дій: спроби групи невідомих проникнути в систему з невідомими цілями, помста службовців, злом шахраями для наживи. Тестування безпеки виявляє фактичну реакцію програмного засобу та його захисних механізмів, вбудованих у нього, на проникнення. Під час процесу такого тестування безпеки випробувач відіграє роль хакера.

Хакеру дозволено все:

- дізнатися пароль за допомогою будь-яких зовнішніх засобів;
- атака на систему за допомогою утиліт, які аналізують захист;

- пригнічення, приголомшення системи (в надії, що вона відмовить, або відмовиться обслуговувати інших користувачів);
- цілеспрямоване введення хакером помилок в надії потрапити в систему в ході її оновлення;
- перегляд даних в надії знайти ключ або доступ для входу в систему.[11]

Тестування локалізації (Localization testing) - багатогранна річ, що має на увазі перевірку безлічі аспектів, пов'язаних з адаптацією продукту для користувачів з інших країн. Наприклад, тестування локалізації для користувачів з Японії може полягати в перевірці того, чи не покаже система помилку, якщо цей користувач на сайті знайомств введе розповідь про себе символами Kanji, а не англійським шрифтом.

Тестування сумісності (або Compatibility testing) - є видом нефункціонального тестування, метою якого є перевірка коректної роботи програмного засобу в певному оточенні. Оточення може включати в себе такі елементи:

- мережеві пристрої;
- апаратна платформа;
- системне програмне забезпечення (брандмауер, веб-сервер, антивірус, ...);
- периферія (CD / DVD-приводи, принтери, веб-камери, ...);
- операційна система (Windows, Unix, MacOS, ...);
- браузери (Internet Explorer, Safari, Firefox, Opera, Yandex, Chrome, Safari).
- бази даних (Oracle,Realm, MS SQL, MySQL, ...);[12]

1.3. План тестування

Тест план (або Test Plan) - являє собою документ, який описує велику кількість роботи з напрямку тестування, починаючи з стратегії, опису об'єкта, розкладу, критеріїв та початку закінчення тестування, до необхідного в процесі тестування роботи самого обладнання, знань та оцінки ризиків із варіантами їх вирішення.

"Тест план містить:

1) перелік рекомендованих тестових ресурсів;

2) перелік функцій та підсистем, які підлягають тестуванню;

3) стратегію тестування:

- аналіз функцій та підсистем для виявлення слабких сторін, що вимагають вичерпного тестування, тобто функціональних можливостей, де найбільш імовірно виникає дефект;
- визначення стратегію вибору вхідних даних для тестування. Оскільки у реальних додатках набір вхідних даних програмного продукту є нескінченним, вибір остаточного підмножини для тестування є складним завданням. Для її вирішення можна застосувати методи, щоб покрити класи вхідних та вихідних даних, аналіз екстремальних значень, охоплення випадків використання тощо. Обрана стратегія тестування має бути обґрунтована та задокументована;
- визначення необхідності автоматизації процесу тестування. У той же час слід обґрунтувати рішення щодо використання існуючого або створення нової автоматичної системи тестування, а також оцінку вартості для створення нової системи або впровадження існуючої.

4) графік (або розклад) циклів тестування;

5) вказівку конкретних параметрів апаратного та програмного оточення;

6) визначення тестових метрик, які необхідно складати і проводити аналіз, таких як покриття коду, обсяг тестового коду, покриття набору вимог, кількість і рівень серйозності дефектів.

Тест план повинен відповідати на такі питання:

1) **що тестувати**? Опис об'єкта тестування: додатки, системи, обладнання та ін.;

2) **як тестувати**? Стратегія тестування: методології, види тестування та їх застосування по відношенню до об'єкту тестування, пріоритети, автоматизація та інші .;

3) **коли тестувати**? Послідовність робіт: фази, цикли тестування, процедура тестування - підготовка або (Test Preparation), тестування або (Testing), аналіз результатів (Test Result Analisys) в запланованих фазах розробки;

4) **де будете тестувати**?

- оточення системи - її опис;
- необхідне обладнання для тестування та програмні засоби.

5) **хто буде тестувати**?

- ролі та обов'язки;
- імена і прізвища.

6) **критерії початку тестування**:

- готовність тест кейсів;
- готовність оточення;
- закінченість розробки необхідного функціоналу;
- виконання юніт-тестів;
- білд побудований і задовольняє певним критеріям

7) **критерії закінчення тестування**:

- результати тестування задовольняють певним критеріям;
- вимоги до кількості відкритих багів виконані (визначаються заздалегідь).

8) **критерії передачі системи для приймального тестування**:

- приймальні випробування - де зберігаються і коли виконуються;
- процедура приймання.

9) **які ризики існую і як ми ними буде керувати**? Ризики та їх дозвіл

10) **метрики і звіти**:

- продуктові метрики - хто збирає, з якою періодичністю;
- звіти - хто створює, кому відправляє, і тощо.

11) **розклад білдів**.

Відповівши в тест плані на ці запитання, можливо вважати, що є документ з плануванням тестування. Щоб документ набув більш серйозний вигляд, пропонується доповнити його такими пунктами:

- оточення тестової системи;
- опис програмно-апаратних засобів;
- необхідні для тестування програмні засоби та обладнання (тестовий стенд та його конфігурація, програмні продукти для автоматизованого тестування та ін.);
- ризики та варіанти їх вирішення."[13]

РОЗДІЛ 2

МОДЕЛЬ І РОЗРОБКА ТЕСТУВАННЯ СЕРВІСУ

2.1. Огляд програм для навантажувального тестування

«Віддаючи веб-сервер в повсякденну експлуатацію, потрібно бути впевненим, що він витримає плановану навантаження. Тільки створивши умови, наближені до бойових, можна оцінити, чи достатня потужність системи, правильно встановлені такі додатки, які беруть участь у створенні веб-контенту, і інші чинники, що впливають на роботу веб-сервера. У цій ситуації на допомогу прийдуть спеціальні інструменти.»[14]

OpenSTA (Рис. 2.1) – більше ніж додаток для тестів, це відкрита архітектура, проєктована навколо відкритих стандартів. Проєкт створений в 2001 році групою компаній CYRANO, яка підтримувала комерційну версію продукту, але CYRANO розпався, і зараз OpenSTA поширюється як додаток з відкритим кодом під ліцензією GNU GPL. Для роботи вимагає Microsoft Data Access Components (MDAC), який можна завантажити з сайту корпорації.[15]

Поточний інструментарій дозволяє провести навантажувальний випробування HTTP / HTTPS сервісів, хоча його архітектура здатна на більше. OpenSTA дозволяє створювати тестові сценарії на спеціалізованій мові SCL (Script Control Language Всі параметри запиту піддаються редагуванню, можлива підстановка змінних. Реалізована можливість організації розподіленого тестування, що підвищує реалістичність, або коли з одного PC не виходить навантажити потужний сервер. Кожна з машин такої системи може виконувати свою групу завдань, а repository host здійснює збір і зберігання результатів.

Після установки на кожній тестовій системі запускається сервер імен, робота якого обов'язкова. Результати тестування, які включають час відгуків, кількість переданих байт в секунду, коди відповіді для кожного запиту і кількість помилок виводяться у вигляді таблиць і графіків. Використання великого числа фільтрів дозволяє відібрати необхідні результати. Результат можна експортувати в CSV-файл. Можливості щодо виведення звітів дещо обмежені, але по посиланнях на сайті можна знайти скрипти і плагіни, що спрощують аналіз отриманої інформації.

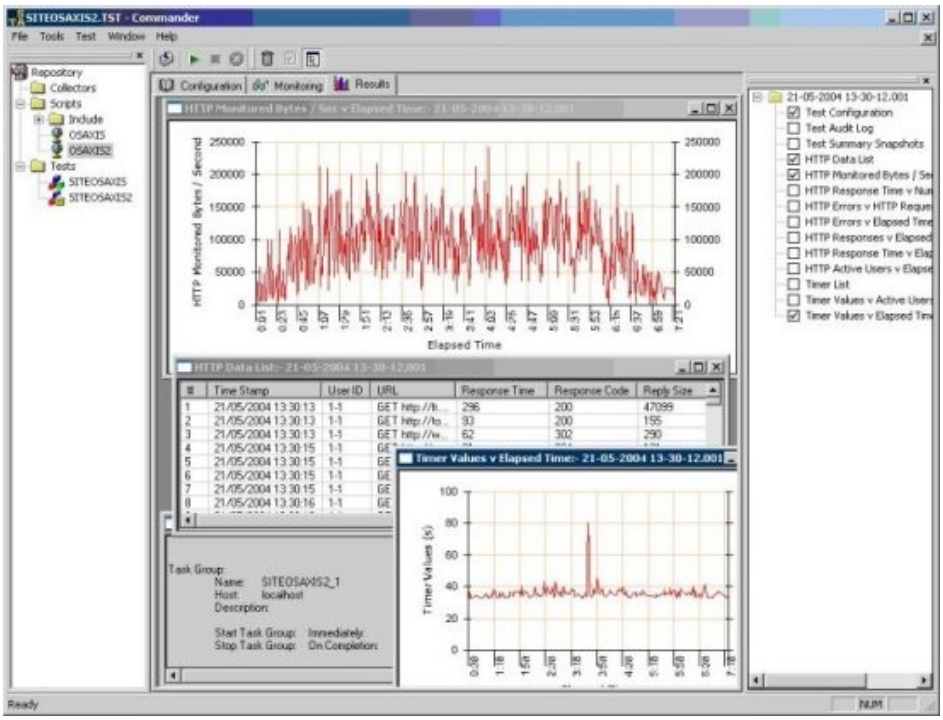

Рис. 2.1. OpenSTA

Apache Jmeter, Рис 2.2 - це єдиний інструмент для навантажувального тестування, який з однієї сторони безкоштовний і open source, а з іншої досить розвинений і має можливість створення тестового навантаження одночасно з декількох PC.[16]

З чого складається тест

При створенні тесту JMeter пропонує кілька типів компонент:

1) samplers - основні елементи, які безпосередньо спілкуються з тестованим додатком, наприклад http sampler для звернення до веб-додатку;

2) logic controllers - елементи, що дозволяють групувати інші елементи в цикли, групи паралельного запуску;

3) assertions - елементи, що виконують контроль.

З їх допомогою ви можете перевірити текст, який ви очікуєте на веб-сторінці або, наприклад, вказати, що ви очікуєте відповідь від сервера не більше ніж 2 секунди. Якщо який-небудь Assert не буде задоволений, тест буде мати негативний результат.

Як виглядає звіт

Що буде міститися в звіті, користувач налаштовує сам. Зручно те, що до звіту можна включати таблиці та графіки.

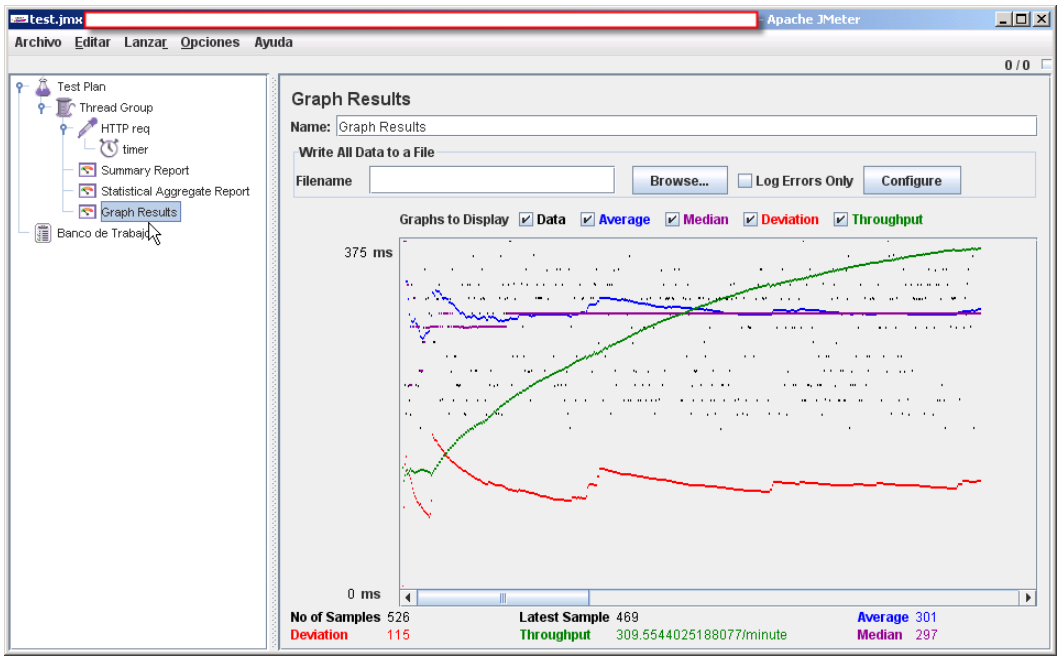

Рис. 2.2. JMeter

Чи можна розширювати JMeter .

Jmeter є зручним інструментом для запуску та моніторингу тестів, а також для перегляду звітів. Тому якщо користувачеві потрібні особливі види тестів, які він може написати на Java, то все що йому потрібно, це написати код самого тесту. Далі його потрібно оформити як Sampler, після чого JMeter допоможе йому зробити всю іншу роботу по організації, конфігурації і виконання тестів (в тому числі і з декількох комп'ютерів).

WAPT (Web Application Testing , Рис. 2.3) дозволяє випробувати стійкість веб та інших додатків, що використовують веб-інтерфейс, до реальних навантажень.

«Розробляється новосибірської компанією SoftLogica LLC. Це одна з найпростіших у використанні програм огляду. Для проведення простого тесту навіть не потрібно заглядати в документацію, інтерфейс простий, але не локалізований. Для перевірки WAPT може створювати безліч віртуальних користувачів, кожен з індивідуальними параметрами. Підтримується кілька видів аутентифікації і кукі. Сценарій дозволяє змінювати затримки між запитами і динамічно генерувати деякі випробувальні параметри, максимально імітуючи таким чином поведінку реальних користувачів. У запит можуть бути підставлені різні варіанти HTTP-заголовка, в налаштуваннях можна вказати кодування сторінок. Параметри User-Agent, X-Forwarded-For, IP вказуються в настройках сценарію. Значення параметрів запиту можуть бути розраховані кількома способами, в тому числі, визначені відповіддю сервера на попередній запит, використовуючи змінні і функції».[17]

Підтримується робота з захищеним протоколом HTTPS (і всі типи проксі-серверів). Створені сценарії, які зберігаються в файлі XML-формату, можна використовувати повторно. Крім стандартних Performance і Stress, в списку присутні кілька інших тестів, що

дозволяють визначити максимальну кількість користувачів і тестувати сервер під навантаженням протягом довгого періоду.

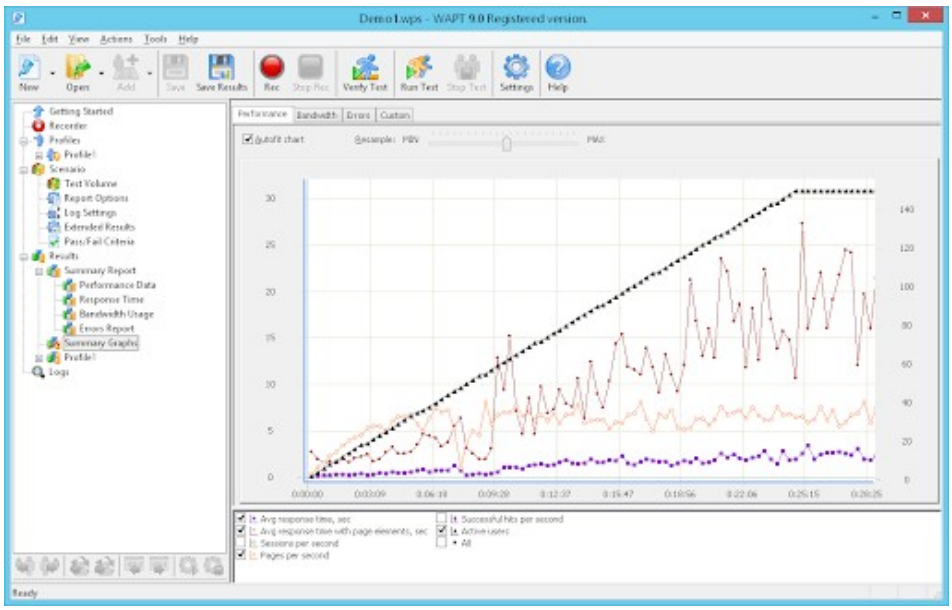

Рис. 2.3. WAPT

NeoLoad, Рис 2.4 - система, що дозволяє провести тестування навантаження веб додатків.

Написана на Java. У звіті можна отримати детальну інформацію по кожному завантаженому файлі. NeoLoad вельми зручний для оцінки роботи окремих компонентів (AJAX, PHP, ASP, CGI, Flash, аплетів і ін.). Можлива установка часу затримки між запитами (thinktime) глобально і індивідуально для кожної сторінки. Тестування проводиться як з використанням дуже зручної графічної оболонки, так і за допомогою командного рядка (використовуючи заздалегідь підготовлений XML-файл). Підтримує роботу з протоколом HTTPS, з HTTP і HTTPS проксі, basic веб-аутентифікацію і cookies, автоматично визначаючи дані під час запису сценарію, і потім програє під час тесту. Для роботи з різними профілями для реєстрації користувачів можуть бути використані змінні. При проведенні тесту можна задіяти додаткові монітори (SNMP, WebLogic, WebSphere, RSTAT і Windows, Linux, Solaris), що дозволяють контролювати і параметри системи, на якій працює веб-сервер.[18] За

допомогою NeoLoad можна проводити і розподілені тести. Один з комп'ютерів є контролером, на інші встановлюються генератори навантаження (loadGenerator). Контролер розподіляє навантаження між loadGenerator і збирає статистику. Дуже зручно реалізована робота з віртуальними користувачами. Користувачі мають індивідуальні настройки, потім вони об'єднуються в Populations (повинна бути створена як мінімум одна Populations), в Populations можна задати загальну поведінку (наприклад, 40% користувачів популяції відвідують динамічні ресурси, 20% читають новини). Віртуальні користувачі можуть мати індивідуальний IP-адреса, смугу пропускання і свій сценарій тесту. Використовуючи утиліти навантажувального тестування, можна отримати інформацію про роботу Веб сервіс, вжити необхідних заходів щодо усунення виявлених недоліків і гарантувати необхідну продуктивність.

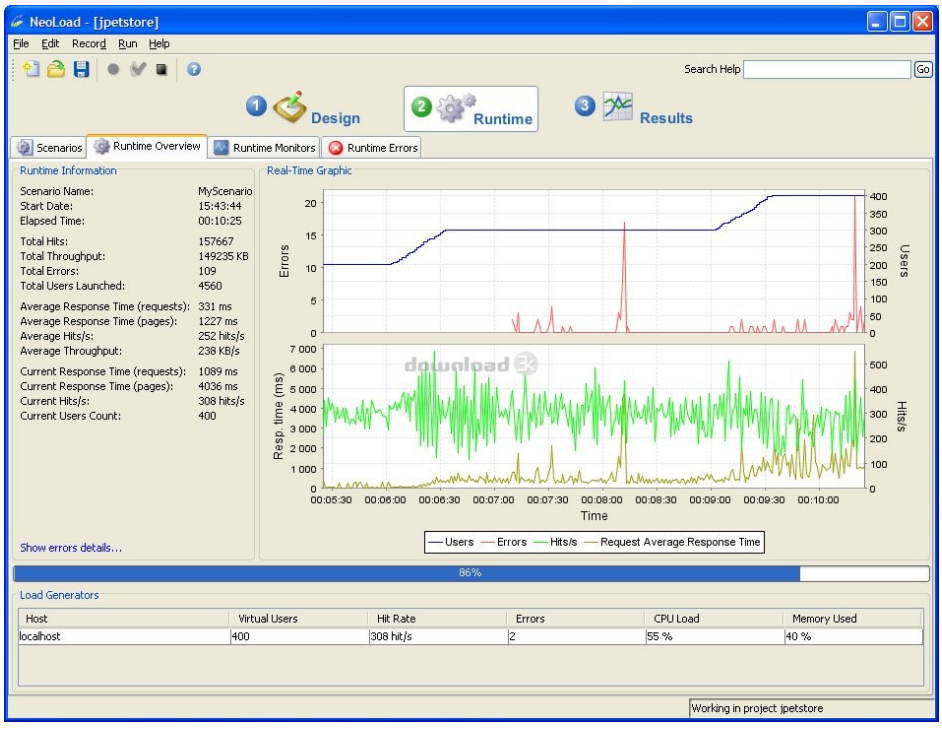

Рис. 2.4 NeoLoad

2.2. Тестовий випадок (або Test case).Види. Структура.

Тестовий випадок (або Test Case) – це:

− мінімальний елемент тестування (всього одна верифікація \ перевірка);

− сукупність конкретних умов, кроків і атрибутів, необхідних для перевірки тестування реалізації певної функції або їх частин;

− опис певних дій і умов, які необхідні для того, щоб виявити той чи інший баг.

Під терміном тест кейсом розуміється структура виду: Action > Expectd Results > Test Results. У таблиці 2.1 показаний приклад test case. [19]

Таблиця 2.1

Приклад тестового випадку

Action	**Expected Result**	**Test Result**
Open web page "Login"	Login web page is opened	Passed...

Види тестових випадків

Тест кейси діляться за очікуваним результатом на позитивні та негативні:

- позитивний тест кейс бере тільки валідні дані і перевіряє, що програмний продукт вірно здійснює функцію, що викликається;
- негативний тест кейс використовує як вірні так і не вірні данні (повинен бути хоча б 1 некоректний параметр) і ставить за перевірку виняткових ситуацій, тобто(спрацьовування валідаторів),

також перевіряє, щоб функція яка викликається програмним засобом не виконувалася при спрацьовуванні такого валідатора.

Структура тестових випадків (або Test Case Structure)

Кожен тест кейс має складатися з трьох частин. У таблиці 7.2 показані ці частини.

Таблиця 2.2

Структура Test case

Pre conditions	Список послідовних дій, які призводять програмний засіб до такого придатного для проведення перевірки стану. Або список таких умов, виконання яких говорить про те, що система знаходиться в оптимальному для проведення тесту стані.
Test case description	Список таких дій, які переводять про з програмний засіб з одного стану в інший, для того щоб отримати результат, на підставі якого, зробити висновок про задоволенні реалізації, поставленим замовником вимогам
Post conditions	Список дій, які переводять систему у початковий стан (до проведення тесту - initial state)

Приклад тест кейса

do N1, verify D1

do N2, verify D2

do N3, verify D3

В наведеному прикладі відображена кінцева перевірка - D3. Це означає, що вона є ключовою. Отже, N1 і N2 - це дії призводять систему до готовності. А D1 і D2 - умови того, що система знаходиться в стані придатному для тестування. Таким чином в таблиці 2.3 показано умова тест кейса.

Таблиця 2.3

Умова тест кейса

Action	Expected Result	Test Result (is passed/is failed/is blocked)
Pre Conditions		
do N1	verify D1	
do N2	verify D2	
Test Case. Description		
do N3	verify D3	
Post Conditions		

Post Conditions не описані в умовах, але якщо це потрібно то треба повернути систему в початковий стан. (Наприклад, видалили запис).

Потрібно відповісти на питання: "Чому розбиття такого виду зручно використовувати?". Відповідь в таблиці 2.4: кінцева перевірка одна, тобто в разі якщо тест провалений (test failed) буде відразу ясно через що. Якщо виявляться провальними перевірки D1 або D2, то тест case буде заблокований (test will be blocked), тому, що функцію неможливо привести придатний для тестування стан, але це не означає, що тестова функції не працює.

Таблиця 2.4

Один з варіантів результату

Action	Expected Result	Test Result (is passed/is failed/is blocked)
Pre conditions		
do N1	verify D1	is passed
do N2	verify D2	is failed
Test Case Description		
do N3	verify D3	is blocked
Post conditions		

2.3. Атрибути тест кейса

У таблиці 2.5 представлені часто використовувані атрибути тест кейса.

Таблиця 2.5

Атрибути тест кейса

Атрибут тест кейса	Опис
Test Case ID	Унікальне значення в межах не тільки документа, але і всієї фірми
Test Case Priority	Пріоритет. Вимірюється від 1 до n 1 - найвищий n - найнижчий (для не дуже великих проєктів раціонально вибирати n = 4)

Продовження табл.2.5

IDEA	Опис ідеї, що перевіряється тест кейсом
SETUP and ADDITIONAL INFO	Вхідні параметри
Revision History	Історія редагування.
Execution Part	"Здійснена частина тест кейса.

	Приклад формату: Action - список команд EXPECTED RESULT - або очікуваний результат TEST RESULT - (blocked, passed, failed)"[2]

Приклад Тест кейса: нехай для тестування перегляду публікацій на сторінці необхідно виконати наступний алгоритм:

1) Відкрити сервіс

2) Перейти на Факультет

3) Вибрати Кафедру

4) Знайти Курс

5) Перейти до дисципліни

Очікуваний результат: "Сторінка з інформацією завантажена "

У таблиці 2.6 можна побачити, як для даного прикладу буде виглядати тест кейс. Для проходження одного і того ж сценарію на N наборах тестових даних створюється N тестів. Цьому правилу необхідно слідувати. Таким чином, щоб перейти на іншу публікацію, або перегляну інформацію, необхідно скопіювати вміст тест кейса VV12345, наприклад, в тест кейс VV12346 і переписати тільки вхідні параметри. Такий вид тест кейса називають керованим даними (data-driven).

Отже, в будь-якому разі доведеться переписувати весь тест кейс, щоб заново протестувати первинне завдання.

Таблиця 2.6

Тест кейс для прикладу

Test Case ID/Priority	VV1234567890	1
1) Відкрити сервіс		

2) Перейти на Факультети		
Revision History		
Created on 20/01/2020 by Tester	Новий тест кейс	
Execution Part		
ACTION	EXPECTED RESULT	TEST RESULT
1) Вибрати Кафедру «» 2) Знайти Курс «3» 3)Перейти до дисципліни «Інформатика»	Відкрилась сторінка яка нам потрібна	

Якщо ж розбити задачу на кілька тест кейсів, то можна переписати тест кейс так, як вказано в таблиці 2.6.

У таблиці 2.7 вказано спрощений варіант тест кейса .

Таблиця 2.7.

Спрощення тест кейсу

Test Case ID/Priority	VV12345	1

Продовження табл.2.7

IDEA: Пошук SETUP and ADDITIONAL INFO: Аккаунт: user / password E-mail: studhub@gmail.com		
Revision History		
Created on 20/01/2020 by Tester	Новий тест кейс	
Modified on 20/01/2020 by Tester	Спрощення тест кейсу	
Execution Part		
ACTION	EXPECTED RESULT	TEST RESULT
1) увійти до систему; 2) знайти потрібну	Сторінка з потрібною нам інформацією	

сторінку;		

2.4. Структура баг репорту

Різні системи відслідковування помилок, пропонують різні поля для заповнення та відмінні структури опису помилок.

У таблиці 2.9 вказано шаблон баг репорту.

Таблиця 2.8

Шаблон баг репорту

Шапка(Hat)	
Короткий опис (або Summary)	Короткий опис існуючої проблеми, вказує на причину та тип помилкової ситуації.
Проєкт (або Project)	Назва продукту тестування

Продовження табл. 2.8

Компонент додатку (або Component)	Назва функції або частини продукту тестування
Номер версії (або Version)	Версія на якій була зафіксована помилка
Серйозність(або Severity)	Поширена п'ятирівнева система градації серйозності дефекту: • p1 Блокуючий (або Blocker); • p2 Критичний (або Critical); • p3 Значний (або Major); • p4 Незначний (або Minor); • p5 Тривіальний (або Trivial).
Пріоритет (або Priority)	Пріоритети дефекту: • k1 Високий (High priority);

	• k2 Середній (Medium priority); • k3 Низький (Low priority). (Докладніше можна дізнатись в розділі Серйозність і пріоритет помилки)
Автор(або Author)	
Назначено на (або Assigned To)	
Оточення	
OS/Service-pack / Browser та Version/…	Інформація про оточення, у якому знайшли баг або помилку: OS/Service-pack / Browser та Version/…- ім'я та версія браузера

Продовження табл. 2.8

Опис	
Кроки відтворення багу(або Steps to Reproduce)	Кроки, відтворюючі ситуацію, яка призвела до помилки.
Фактичний результат процесу(або Result)	Результат, отриманий після проходження кроків до відтворення
Очікуваний результат процесу(або Expected Result)	Очікуваний результат
Доповнення	
Приєднаний файл (або Attachment)	Файл з логами процесу, Screenshot або інший документ, який проясняє причину помилки та може вказати на спосіб вирішення проблеми

2.5. Написання баг репорту

Баг репорт є технічним документом в зв'язку з яким мова опису проблеми має бути технічною. Також має використовуватися правильна термінологія при створені назв елементів призначеного для користувача інтерфейсу (PopUp, Menu, DataPicker, TextBox, Label, ViewController, Combobox, DropDown), дій користувача (Gest, click, hold, move, touch) і отриманих результатах (View is opened, crash report, info view).[20]

У таблиці 2.10 представлені основні поля баг репорту і роль працівника, відповідального за заповнення даного поля.

Таблиця 2.9

Заповнення полів баг репорт

Шапка	
Короткий опис (або Summary)	Автор репорту помилки(зазвичай тестувальник)
Проєкт (або Project)	Автор репорту помилки (зазвичай тестувальник)
Компонент додатку (або Component)	Автор репорту помилки (зазвичай тестувальник)
Номер версії (або Version)	Автор репорту помилки (зазвичай тестувальник) цей атрибут може бути змінений менеджером
Серйозність(або Severity)	Менеджер проєкту або відповідальний менеджер за розробку компоненту, до якого написаний баг репорт
Пріоритет (або Priority)	Автор репорту помилки (зазвичай це Тестувальник), але багато систем баг трекінгу виставляють статус за замовчуванням

Автор(Author)	Встановлюється за замовчуванням, якщо немає, то вказується ім'я автора баг репорт
Назначено на (Assigned To)	Менеджер проєкту або менеджер відповідальний за розробку компонента, на який написаний баг репорт

Продовження табл. 2.9

ОС/Сервіс Пак тощо/ Браузер+версія/…	Автор баг репорту(зазвичай тестувальник)
Кроки відтворення(Steps to Reproduce)	Автор баг репорту(зазвичай тестувальник)
Фактичний результат (Result)	Автор баг репорту(зазвичай тестувальник)
Очікуваний результат (Expected Result)	Автор баг репорту(зазвичай тестувальник)
Приєднаний файл (Attachment)	Автор баг репорту (зазвичай це Тестувальник), а також будь-який член командної групи, який вважає, що прикріплені дані допоможуть у виправленні бага

2.6. Автоматизоване навантажувальне тестування

Щоб обговорювати підходи до тестування навантаження і проблеми, які вирішуються з його допомогою, потрібно почати з основ - термінології .

1) віртуальний користувач (Virtual User) - програмний процес, циклічно виконує пені дії. Приклад простий користувач, який хоче скористатися системою;[21]

2) ітерація (Iteration) є одиним повтором виконуваної в циклі операції. Приклад ітерації вхід в систему -> проведення аналізу -> отримання результату аналізу -> вихід з системи;[22]

3) інтенсивність (Operation Intensity) - часте виконання операції в одиницю часу, в тестовому скрипті задається інтервалом часу між ітераціями. Приклад 3 користувача, які входять в систему через хвилину, після того, як зайшов попередній користувач;[23]

4) навантаження (Loading) виконання операцій на ресурсі. Приклад - загальна кількість виконаних операцій в системі трьома користувачами за певний проміжок часу;[24]

5) продуктивність (Performance) - кількість операцій за виділений період часу. Якщо система не може виконати певної кількість операцій за даний період часу, то вона починає гальмувати, що означає брак продуктивності;

6) масштабованість додатків (Application Scalability) - пропорційне зростання продуктивності при збільшенні навантаження;[25]

7) профіль навантаження (Performance Profile) - це набір операцій з заданими інтенсивностями, отриманий на основі збору статистичних даних або певний шляхом аналізу вимог до тестованої системі. Також називається сценарієм (скриптом);[26]

8) навантажувальної – виконання певних дій з великою кількістю повторів і багатьма юзерами.

Потрібно розглянути як ці сутності пов'язані між собою. Висловивши інтенсивність через інтервал часу між ітераціями, можна побачити, що зростання інтенсивності виконуваних операцій це скорочення інтервалу часу. Зростання навантаження пропорційне

зростанню інтенсивності. Також, що при збільшенні інтенсивності зростає продуктивність. При цьому збільшується ступінь використання (завантаженості) ресурсів. З якогось моменту зростання продуктивності припиняється (а навантаження може продовжувати рости), відбувається насичення і потім деградація системи. На додаток можна помітити що при тестуванні зміна інтенсивності операцій може підкорятися якогось закону (наприклад, Пуассона) або бути рівномірним протягом усього тесту.

2.7. Автоматизоване функціональне тестування

Автоматизоване тестування ПЗ (Automation Testing) – це процес в якому відбувається верифікація ПЗ, в котрому основні функції та кроки, такі як запуск програмного засобу, його ініціалізація, його виконання, його аналіз і виведення результату, також виконуються автоматично за допомогою інструментів для автоматизованого тестування" .[27]

З автоматизацією тестування, як і з іншими вузько направленими IT - направленнями, пов'язано багато не правдивих уявлень. Для того, щоб уникнути застосування автоматизації який є неефективним, слід обходити недоліки.

Переваги автоматизації:

- повторюваність - тест виконується автоматично, «людський фактор» відсутній.
- швидке виконання коду. Економія часу на перевірку проблемних місць.
- скорочення витрат на проведення ручних тестів.
- детальний менеджмент звітів.
- можливість витратити час раціонально під час прогону тестів.

Недоліки автоматизації тестування:

- повторюваність одноманітність сценарію виконання повністю виключає «людський фактор».
- великі витрати на розробку якщо розробляти унікальну програму, але використовуючи доступні фреймворки можливо компенсувати недолік.[28]

Далі розглянемо Selenium WebDriver - це програмна бібліотека для управління браузерами. WebDriver представляє одночасно драйвери для різних браузерів та бібліотек для різних програм виконуючи всю свою роботу автоматизовано.[29]

Модуль selenium.webdriver надає весь функціонал WebDriver'a. Наданий момент WebDriver підтримує реалізації Firefox, Chrome, IE і Remote. Клас Keys забезпечує взаємодію з командами клавіатури, такими як RETURN, F1, ALT і т.д .[30]

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
```

Клас тесту успадкований від unittest.TestCase. Спадкування

```
class Registr(unittest.TestCase):
```

setUp - це частина ініціалізації, цей метод буде викликатися перед кожним методом тесту, який ви збираєтеся написати всередині класу тесту. Тут ми створюємо елемент класу Chrome WebDriver.

```
def setUp(self):
    self.driver = webdriver.Chrome("C:\Driver\chromedriver.exe")
```

Далі описаний метод нашого тесту. Метод тесту завжди повинен починатися з фрази test. Перший рядок методу створює локальну посилання на об'єкт драйвера, створений методом setUp.

```
def test_registration(self):
    driver = self.driver
```

WebDriver надає ряд способів отримання елементів з допомогою методів find_element_by_ *. Для прикладу, елемент введення тексту input

може бути знайдений по його атрибуту name методом find_element_by_name. Знаходимо усі поля вводу

```
name_field = driver.find_element_by_name("username")
password_field = driver.find_element_by_name("password")
email_registration = driver.find_element_by_name("email")
email2_registration = driver.find_element_by_name("email2")
firstname = driver.find_element_by_name("firstname")
lastname = driver.find_element_by_name("lastname")
```

Після цього ми посилаємо натискання клавіш (аналогічно введенню клавіш з клавіатури). Спеціальні команди можуть бути передані за допомогою класу Keys імпортованого з selenium.webdriver.common.keys:

```
name_field.send_keys("cstuser")
password_field.send_keys("7732465q")
email_registration.send_keys("dojnegeorn@mailinator.com")
email2_registration.send_keys("dojnegeorn@mailinator.com")
firstname.send_keys("JOTARO")
lastname.send_keys("DIO")
submitbutton.send_keys(Keys.ENTER)
```

Після відповіді сторінки, ви отримаєте результат, якщо такий очікується. Щоб упевнитися, що ми отримали якийсь результат, додамо твердження:

```
assert "No results found." not in driver.page_source
```

Метод tearDown буде викликаний після кожного методу тесту. Це метод для дій чистки. У поточному методі реалізовано закриття вікна браузера.

```
def tearDown(self):
    self.driver.close()
```

Завершальний код - це стандартна вставка коду для запуску набору тестів [Порівняння __name__ з "__main__" означає, що модуль (файл програми) запущений як окрема програма («main» (англ.) - «основна», «головна») (а не імпортовано з іншого модуля)

```
if __name__ == "__main__":
    unittest.main()
```

На даному етапі тестування ми розглянули автоматизоване ПО яке допомагає скоротити час на виконання типових завдань. Можна зробити припущення що дане ПО має в собі достатній набір функцій для реалізації функціонального тестування.

ВИСНОВКИ

В ході виконання роботи було вивчено процес тестування програмного забезпечення в ІТ-компаніях. Були розглянуті більшість видів дефектів в ПЗ, причини виникнення та способи їх відстеження за допомогою систем баг-трекінгу. Також вивчені способи створення і застосування тест кейсів для чорного ящика.

Виходячи з вище викладеного матеріалу і протестованих систем Ksu-online та StudHub, можна зробити такі висновки:

- Програми та способи тестування які були розглянуті можна практично застосовувати до будь-якого проєкту.
- Вони детально відображають картину працездатності веб-сервісу чи іншого додатку.
- Тестування є нормою в сучасному світі та використовується всіма ІТ компаніями, допомагає скоротити витрати на доопрацювання ПЗ до того, як воно піде в реліз, забезпечує знешкодження помилок на стадії створення специфікацій та розробки.
- Особливо треба приділити увагу таким методам, як навантажувальне тестування, тестування безпеки та зручності.
- ПЗ, використане для написання програми автоматичного тестування, придатне до використання і має в собі повний набір функціоналу для реалізації виконання певних тест-кейсів, які являються шаблонними, без присутності «людського фактора».
- Даний матеріал можна використовувати як для навчальних цілей, так і для успішного проходження технічної співбесіди для влаштування на роботу в якості QA.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rational Unified Process [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Rational_Unified_Process.

2. Ітераційна модель життєвого циклу [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/5203612/page:6/.

3. Про тестинг. [Електронний ресурс]. Режим доступа: http://www.protesting.ru/

4. Software testing training and stoftware testing services. [Електронний ресурс]. Режим доступа: http://www.rbcs-us.com/

5. Приймальне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://internetdevels.ua/blog/user-acceptance-testing.

6. Смок-тест [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/badoo/blog/316874/.

7. Performance-testing-what-is-it [Електронний ресурс] – Режим доступу до ресурсу: https://www.quality-assurance-group.com/performance-testing-what-is-it/.

8. Нефункціональне тестування ПЗ [Електронний ресурс] – Режим доступу до ресурсу: http://qalearning.com.ua/theory/lectures/material/nonfunctional-testing/.

9. Usability testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.ipsos.com/ua-ua/testuvanna-zrucnosti-vikoristanna.

10. Тестування інтерфейсу користувача. Електронна підтримка та сучасні інформаційні технології у інтерфейсах користувача

[Електронний ресурс] – Режим доступу до ресурсу: https://msn.khnu.km.ua/pluginfile.php.

11. Тестування безпеки (security testing) [Електронний ресурс] – Режим доступу до ресурсу: https://www.quality-assurance-group.com/testuvannya-bezpeky-vydy-vrazlyvostej/.

12. Compatibility testing [Електронний ресурс] – Режим доступу до ресурсу: https://internetdevels.ua/blog/using-browserstack-for-cross-browser-and-cross-platform-compatibility-testing.

13. Тест план [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.com.ua/baza-znaniy/test-plan/.

14. Під граничним навантаженням огляд програм навантажувального тестування веб-серверів [Електронний ресурс] – Режим доступу до ресурсу: 1. https://ua.waykun.com/articles/pid-granichnim-navantazhennjam-ogljad-program.php.

15. What is OpenSTA [Електронний ресурс] – Режим доступу до ресурсу: http://opensta.org/.

16. QA [Електронний ресурс] – Режим доступу до ресурсу: https:// www.quality-assurance-group.com/jmeter-navantazhuvaty-sajty-po-max/.

17. WAPT [Електронний ресурс] – Режим доступу до ресурсу: https://ua.waykun.com/articles/pid-granichnim-navantazhennjam-ogljad-program.php.

18. NeoLoad [Електронний ресурс] – Режим доступу до ресурсу: http://treatface.ru/brands/neotys/neoload-po-nagruzochnogo-testirovaniya-setevykh-prilozheniy/.

19. Тестовий випадок [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.com.ua/baza-znaniy/test-case/.

20. Баг репорт [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/156099/.

21. Virtual User [Електронний ресурс] – Режим доступу до ресурсу: https://community.microfocus.com/t5/Silk-Performer-Knowledge-Base/What-is-the-definition-of-a-Virtual-User/ta-p/1748082.

22. Ітерація [Електронний ресурс] – Режим доступу до ресурсу: https://searchsoftwarequality.techtarget.com/definition/iteration.

23. Operation Intensity [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Roofline_model.

24. Loading [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/load-testing-tutorial.html.

25. Application Scalability [Електронний ресурс] – Режим доступу до ресурсу: https://dzone.com/articles/application-scalability-how-to-do-efficient-scalin.

26. Performance Profile [Електронний ресурс] – Режим доступу до ресурсу: https://software-testing.ru/library/testing/performance-testing/55-2008-10-13-19-04-11.

27. Automation Testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/automation-testing-tutorial-1/.

28. Переваги та недоліки automation Testing [Електронний ресурс] – Режим доступу до ресурсу: https://qaat.ru/preimushhestva-i-nedostatki-avtomatizirovannogo-testirovaniya/.

29. Selenium [Електронний ресурс] – Режим доступу до ресурсу: https://www.selenium.dev/.

30. Selenium-python [Електронний ресурс] – Режим доступу до ресурсу: https://selenium-python.readthedocs.io/index.html.

ДОДАТКИ

Додаток А

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО
ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Болокан Владислав Анатолійович,
учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

23.04.2020		Владислав Болокан
(дата)	(підпис)	(ім'я, прізвище)