



**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ імені ІВАНА ПУЛЮЯ

Кафедра: "Програмної інженерії"

**М Е Т О Д И Ч Н І РЕКОМЕНДАЦІЇ**

до виконання лабораторних робіт

з дисципліни

**Якість програмного забезпечення**

**та тестування**

для студентів спеціальності
121 – Інженерія програмного забезпечення

ТЕРНОПІЛЬ 2020

Методичні рекомендації до виконання лабораторних робіт з дисципліни „Якість програмного забезпечення та тестування ” студентами усіх форм навчання для спеціальності 121 – «Інженерія програмного забезпечення» /Упор.: Кінах Я.І., Бойко І.В., Максимів Т.Б. - Тернопіль: ТНТУ, 2020-22 с. .

 Методичні рекомендації призначені для лабораторних робіт студентами, що навчаються за спеціальністю 121 – «Інженерія програмного забезпечення», та містять інформацію про структуру дисципліни та лабораторні роботи.

Упорядники: к.т.н., доцент кафедри програмної інженерії Кінах Я.І.,

к.ф.-м.н., доцент кафедри програмної інженерії Бойко І.В.,

асистент Максимів Т.Б.

Розглянуто на засіданні кафедри програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя,
протокол № 1 від 31.08.2020р.

**Зміст**

**Вступ…………………………………………………………………………………4**

**Лабораторна робота №1………………………………………….………………5**

**Лабораторна робота №2…………………………………………………………8**

**Лабораторна робота №3……………………………………….….……………..10**

**Лабораторна робота №4……………………………………….…….…………..13**

**Лабораторна робота №5………………………………….……..………………15**

**Варіанти завдань лабораторних робіт………….………………….…………..16**

**Список використаних джерел…………………………………………………..22**

**Вступ**

Мета лабораторного курсу полягає в оволодінні сучасними методами, практичними навичками та основними застосуваннями оцінки якості програмного забезпечення; основними підходи до тестування програм; правилами взаємодії користувача з аплікацією та об’єктами.

Основним завданням є оволодіння побудовою алгоритмічних методик розв’язку інженерних задач оцінки якості інформації та тестування програм на етапах проектування сучасних програмних засобів.

За результатами вивчення дисципліни студент повинен продемонструвати такі результати навчання:

* мати навички командної розробки, погодження, оформлення і випуску програмної документації для тестування;
* вміти застосовувати методи компонентної розробки тестів програмного забезпечення;
* знати та вміти застосовувати інформаційні технології оцінки якості ПЗ, зберігання та паредачі даних;
* знати та вміти застосовувати методи верифікації та валідації програмного забезпечення;
* знати підходи щодо оцінки та забезпечення якості програмного забезпечення;
* знати, аналізувати, вибирати, кваліфіковано застосовувати засоби тестування відповідно до розв’язуваних прикладних завдань та створюваних програмних систем.

Вивчення навчальної дисципліни передбачає формування та розвиток у студентів **компетентностей**:

**загальних:**

* здатність до пошуку, оброблення та аналізу інформації з різних джерел;
* здатність працювати в команді;
* здатність діяти на основі етичних міркувань;
* прагнення до збереження навколишнього середовища;
* здатність діяти соціально відповідально та свідомо.

**спеціальних (фахових, предметних):**

* здатність формулювати та забезпечувати вимоги щодо якості програмного забезпечення у відповідності з вимогами замовника, технічним завданням та стандартами;
* здатність дотримуватися специфікацій, стандартів, правил і рекомендацій в професійній галузі при реалізації процесів життєвого циклу;
* володіння знаннями про інформаційні моделі даних, здатність створювати програмне забезпечення для зберігання, видобування та опрацювання даних;
* здатність застосовувати фундаментальні і міждисциплінарні знання для успішного розв’язання завдань інженерії програмного забезпечення.

**Лабораторна робота №1**

**Тема:** основи Software Quality Control. Аналіз та перевірка вимог до проекту.

**Мета:** аналіз вимог до програмного забезпечення.

**Теоретичні відомості:**

Software Development Process – структура накладена на розробку ПЗ. Процеси розробки ПЗ складаються з великої кількості етапів. Нижче наведені основні активності, які розташовані в послідовності «водоспадного» процесу Waterfall process. Послідовність етапів в інших процесах може бути змінена.

Аналіз вимог (Requirements Analysis)

Складання вимог, яким повинен відповідати розроблювальний продукт, першим кроком до створення необхідного ПЗ. Доки замовник, можливо, вважає, що він знає, що повинне робити необхідне ПЗ, може знадобитися знання й досвід у розробці ПЗ, щоб розпізнати недостатність, двозначність і протиріччя у вимогах.

Специфікації (Specification)

Ця активність точно описує ПЗ, яке потрібно розробити в строгій математичній формі. У реальності самі вдалі специфікації пишуться на зрозумілі й точно врегульовані додатки, які були успішно розроблені в минулому. Специфікації найбільш важливі для замовників які хочуть переконатися в стабільності майбутнього продукту.

Програмна архітектура (Software Architecture)

Архітектура ПЗ є абстрактним представленням розроблювального додатка. Основна мета даного етапу - переконатися що система буде відповідати вимогам, а також майбутні вимоги зможуть бути реалізовані. Ще одною важливою метою на даному етапі є розгляд всіх інтерфейсів міжрозроблювальним продуктом та іншим ПЗ, а також визначення необхідних апаратних засобів і ОС для успішної розробки.

Програмування (Coding)

На даному етапі створюють дизайни (design у цьому випадку - це фаза в розробці ПЗ, на якій функціональні специфікації транслюються в текст програми) і реалізують їх за допомогою певних технологій і мови програмування. Спрощення програмування за допомогою дизайнів є очевидною частиною розробки ПЗ, проте складання дизайнів не повинне бути більшою частиною даного етапу.

Тестування (Testing)

Тестування частин системи, особливо в тих місцях де код двох та більше розроблювачів повинен працювати разом.

Документування (Documentation)

Важлива частина (якій найчастіше не приділяють значення) - документування внутрішніх дизайнів ПЗ для подальшої підтримки і розширення. Документація - найважливіший етап для замовника.

Підтримка (Maintenance)

Підтримка й розширення ПЗ разом зі знайденими проблемами або новими вимогами можуть вимагати набагато більше часу, ніж розробка самого ПЗ. Не тільки додавання коду, що не був написаний відповідно до дизайну, а й визначення, як ПЗ працює з різних сторін після закінчення розробки, може потребувати значних зусиль зі сторони інженерів. Близько 2/3 роботи всіх інженерів - підтримка, проте така статистика може ввести в оману. Лише мала частина підтримки це виправлення помилок. Більша частина - це додавання нових можливостей системи.

Аналіз вимог

Процес статичного тестування починається з визначення й аналізу вимог до системи. Вивчаються й уточнюються:

– існуючі матеріали й методичні питання (технічне завдання на розробку, робітник і технічний проект та ін.);

–основні використовувані поняття;

–терміни й визначення;

– очікувані функціональні вимоги до системи, такі як вимоги до інтерфейсу, де описують входи, одержувані із зовнішніх систем, і виходи, що направляються в зовнішні системи, чи накладаються на ці інтерфейси якісь обмеження, даним описують вхідні й вихідні дані системи, формат даних, їх зберігання, продуктивності описують проблеми масштабування й синхронізації, наприклад, скільки користувачів одночасно повинна обслуговувати система;

– вимоги до користувачів і людського фактора (хто буде працювати із програмним забезпеченням, облік необхідного рівня зручності й простоти використання);

– вимоги до фізичних засобів тестування (операційна система під керуванням якої виконується програмний продукт, і обчислювальна платформа, на якій експлуатується система);

– вимоги до безпеки (доступ до програмного продукту й керування даними),

документації (визначається в якому виді вона повинна бути й чи повинна бути взагалі);

– вимоги до усунення несправностей (реакція системи на несправності);

– вимоги до супроводження (визначається, як виробляється усунення проблем, виявлених у системі).

Зазвичай визначають більша десятка параметрів для аналізу вимоги на валідність.

Розглянемо 5 основних параметрів, які використовуються найчастіше.

CORRECT Вимога є **коректною**, якщо вона визначає тільки правдиві факти, є точною і не повинна містити жаргонів чи розмовних висловів.

*// Tax for car rent in USA shall be …*

*(ціна податку за оренду автомобіля в США залежить від штату, тому визначати її у вимогах є некоректно);*

UNAMBIGUOUS Вимога є **однозначна**, якщо всі слова чи терміни мають тільки одну інтерпретацію.

*//The system shall be able to provide historical reports*

*(що мається на увазі під історичними репортами: чи мається на увазі репорти з життя додатку чи репорти певних історичних подій);*

CONSISTENT Вимога є **послідовна**, якщо немає ніяких конфліктних термінологій; суперечливих дій; неможливих чи несумісних комбінацій/дій.

//*The page layout contains with Email and Password fields. When user* *submit form with incorrect Username and Password, system displays appropriate error message*

*(Поле називається Email чи все таки Username?);*

COMPLETE Вимога є **повна**, якщо вона містить всю потрібну інформацію для розуміння. Всі умови, які можуть статися, повинні бути вказані.

* *вимога не повинна містити загальних слів, як: TBD (to be defined), etc;*
* *вимога не повинна містити посилання на якийсь інший документ чи джерело;*
* *вимога не повинна містити ніяких пропущених характеристик, наприклад, потрібно зменшити час загрузки сайту, але не вказано який поточний час загрузки;*
* *вимога не повинна містити пропущених параметрів, наприклад, користувач має змогу зареєструватися на сайті, але не вказано які саме поля потрібні для реєстрації;*
* *вимога повинна повністю розкрити суть функціоналу, наприклад, має бути забезпечене кросбраузерне тестування, але не вказано, які саме браузери потрібно використовувати для цього;*

VERIFIABLE Вимога має бути **виконувана**, якщо актуальний результат виконання можна перевірити.

*// It should provide the user a fast response*

 *(потрібно конкретизувати швидкість відповіді);*

**Хід роботи:**

1. Вибрати варіант згідно з списком студентів у підгрупі.
2. Проаналізувати кожну з наведених у варіанті вимог.
3. Протестувати кожну з вимог на коректність, однозначність, послідовність, повноту, виконуваність.
4. До кожного параметру поставити статус перевірки: passed або failed.
5. Якщо вимога не пройшла перевірку – дати короткий опис проблеми.
6. Оформити звіт.

**Контрольні питання:**

1. Що таке тестування? Яка його ціль?
2. Назвіть основні етапи Software Development Process.
3. Опишіть стадії тестування.
4. Що таке «Аналіз вимог» і що відбувається в період аналізу вимог

на проекті?

1. Назвіть 5 основних параметрів аналізу вимоги на валідність.
2. Дайте визначення до кожного з 5 основних параметрів аналізу вимоги на валідність?

**Лабораторна робота №2**

**Тема:** функціональне тестування проектів.

**Мета:** позитивні та негативні тест-кейси для функціонального тестування.

**Теоретичні відомості:**

Тест-кейс - це мінімальний компонент тесту, як правило, він націлений тільки на один елемент об'єкта тестування. Чим менше у тест-кейса покриття функціональності, тим чіткіше область пошуку причини у випадку знайденої помилки. Якщо ж у тест-кейсі розписаний цілий сценарій з десятків дій, то ускладнюється, по-перше, його проходження, по-друге, визначення причини збою, якщо результат негативний; також є велика ймовірність одержати помилковий негативний результат через помилку в іншому, другорядному для даного тест-кейса вузлі, залученому в цей сценарій. Однак у деяких випадках тест-кейси навмисно робляться об'ємними й складними, якщо того вимагає специфікація об'єкта тестування.

Позитивне тестування – це тестування на дозволених вхідних даних або сценаріях, які відповідають нормальній (штатній, очікуваній) поведінці системи на допустимих значеннях (happy path).

Основною метою позитивного тестування є перевірка того, що за допомогою системи можна робити те, для чого вона створювалася.

Негативне тестування – це тестування свідомо помилкових вхідних даних і позаштатних сценаріїв, що виходять за інтервали коректності; перевірка реакції системи на виняткові ситуації (exception path, позамежні стани) – перевіряється обробка винятків, виведення повідомлень про помилки, зрозумілість і коректність цих повідомлень.

Основною метою “негативного” тестування є перевірка стійкості системи до впливів різного роду, валідація невірного набору даних, перевірка обробки виняткових ситуацій (як в реалізації самих програмних алгоритмів, так і в логіці бізнес-правил).

* тест-плані кожний тест-кейс складається мінімум із трьох частин:

–заголовка (Summary);

–кроків для виконання (Steps to reproduce);

–очікуваного результату (Expected result).

Summary

//*назва кейсу,* *повинна точно і коротко описати,* *що перевіряється цим* *кейсом*

Приклад: Перевірити, що система перенаправляє користувача на головну сторінку, якщо всі обов’язкові поля заповнені і “Login” кнопка активована на сторінці входу в систему.

Preconditions

*//стан в якому має перебувати система, щоб почати виконання кейсу – є необов’язковим*

Приклад: користувач знаходиться на сторінці входу

Steps to reproduce

//*кроки,* *які потрібно виконати в кейсі*

Приклад:

1. Ввести ім’я користувача в "Username" поле
2. Ввести пароль у “Password” поле
3. Натиснути “Login” кнопку

Expected Result

//*як повинна вести себе система після проходження кроків кейсу*

Приклад:

1. Система перенаправляє користувача на Головну сторінку сайту
2. Ім’я користувача відображається в хедері

Postconditions

//с*тан,* *в який потрібно перевести систему після закінчення виконання* *кейсу – є необов’язковим*

Приклад: вийти із системи

**Хід роботи:**

1. Вибрати власну написану студентом програму чи будь-який десктопний додаток з коротким описом та скріншотами.
2. Скласти 7 тест-кейсів для покриття тестуванням програми/додатку.
3. Декілька з проведених тест кейсів мають бути негативними.

**Контрольні питання:**

1. Що таке Test Case?
2. З яких частин складається Test Case?
3. Що таке тестові кроки й результати та з чого вони складаються?
4. Що таке позитивне і негативне тестування?

**Лабораторна робота № 3**

**Тема:** системи відслідковування багів.

**Мета:** звітування про дефекти (bug reporting) програмного забезпечення

Баґ або Баг (англ. bug — жук) — сленгове слово, рівнозначне помилці, ваді або дефекту програмі. Термін зазвичай використовується у відношенні до помилок, котрі виявляються на стадії роботи програми, на відміну від помилок проектування чи синтаксичних помилок. «Баґи» локалізуються та виправляються в процесі тестування та доробки програми.

За легендою, 9 вересня 1945 року вчені Гарвардського університету, що тестували обчислювальну машину Mark II Aiken Relay Calculator, знайшли метелика, що застряг між контактами електромеханічного реле й Грейс Хоппер вперше вжила цей термін. Знайдена комаха була вклеєна до технічного щоденника, з супроводжувальним надписом: «First actual case of bug being found» (англ. «перший випадок у практиці, коли було знайдено комаху»).

Цей цікавий факт став передумовою для вживання слова «баг» у значенні «помилка».

Дефектом вважається:

– нездатність системи або компонента системи виконувати необхідні функції в рамках заданих вимог до продуктивності.

– помилка в програмі, яка примушує цю програму виконувати ненавмисні або непередбачені дії.

– розбіжність між реальним станом чи результатом роботи програми і очікуваними результатми або станом.

Для фіксування та виправлення помилок пишуться звіти (bug reports).

Баг Репорт (Bug Report) — це документ, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Actual result), із зазначенням причин та очікуваного результату (Expected Result).

Для написання звітів існує багато систем спостереження (Bug tracking systems). Хоча немає жодних збитків у використанні листів Excel для запису і відстеження багів, але оскільки масштаби проектів, кількість тестових циклів, кількість зацікавлених людей зростає, стає зрозуміло, що потрібно набагато сильніший механізм.

Bug tracking system - це система, що є програмним додатком, який призначений для допомоги у роботі тестерам та програмістам відслідковувати дефекти у їхній роботі.

Для програмного забезпечення для відстеження помилок необхідно мати:

* об'єкт репорту - заповнення репорту полями, які дозволять надавати інформацію про помилку, середовищі, модулі, ступені тяжкості з можливістю додавати скріншоти;
* можливість призначення багу – призначення створеного багу девелоперу чи відповідальній особі;
* життєвий цикл багу - робочий процес життя багу (new, deferred, assigned, in progress, fixed, need more information, not an issue, reopened, verified, closed);
* звіти - графіки чи діаграми;
* зберігання та пошук – додавання до багу ідентифікатора для полегшення пошуку.

До найпопулярніших систем належать системи JIRA (з додатком Zephyr), Mantis Bug Tracker, BugZilla, Redmine та TRAC. Jira з додатками Confluence та Zephyr – є найкращою та найпоширенішою системою відслідковування багів. Confluence – засіб для створення/редагування вимог до продукту, Zephyr – засіб для зберігання тест кейсів.



Рис. 1 – JIRA - Bug Tracking System

* + баг репорті вказуються наступні поля:
* Summary – це короткий опис проблеми, що має бути достатньо конкретний для уявлення та причину проблеми. Опис – Preconditions (стан, в якому має перебувати система, щоб можна було відтворити баг), Steps to reproduce (кроки, щоб відтворити баг), Notes (додаткова інформація для девелоперів), Expected Result, Actual Result.
* Тип репорту – баг, поліпшення/нововведення у систему…
* Серйозність (Severity) - це атрибут, що характеризує вплив дефекту на працездатність програми. Виставляється тестувальником.

S.1 – blocker – Блокуюча помилка, що приводить додаток в неробочий стан, в результаті якого подальша робота з системою або її ключовими функціями стає неможлива. Рішення проблеми необхідно для подальшого функціонування системи.

S.2 – critical – Критична помилка, неправильно працює ключова бізнес логіка, діра в системі безпеки, проблема, яка призвела до тимчасового падіння сервера або приводить в неробочий стан деяку частину системи, без можливості вирішення проблеми, використовуючи інші вхідні точки. Рішення проблеми необхідно для подальшої роботи з ключовими функціями системи.

S.3 – major – Значна помилка, частина основної бізнес логіки не функціонує належним чином. Помилка не критична або є можливість обхідного шляху.

S.4 – low – Незначна помилка, що не порушує бізнес логіку частини програми, проблема призначеного для користувача інтерфейсу.

* Пріоритет (Priority) - це атрибут, який вказує на черговість виконання завдання або усунення дефекту. Можна сказати, що це інструмент менеджера з планування робіт. Чим вище пріоритет, тим швидше потрібно виправити дефект. Виставляється менеджером або замовником.

P.1 – виправити негайно.

Р.2. – виправити у найближчий час.

Р.3. – виправити до кінця закінчення етапу.

Р.4. – помилку було б чудово виправити.

Наприклад, заведено дефект, що на головній сторінці сайту назва компанії написана неправильно. Серйозність (Severity) для такого багу буде S.4, адже назва сайту не впливає на функціональність сайту, але Пріоритет (Priority) буде P.1, адже для бізнесу/замовника неправильно написана назва компанії на головній сторінці може призвести до втрати репутації компанії.

* Reporter – той, хто створив баг репорт.
* Assigner – той, кому адресований баг репорт (менеджер, тімлід, девелопер).
* Додатки – скріни або відео записи знайденого дефекту.

Наприклад, для створення скрінів чи відео відтворень дефекту часто

використовують додатки Snagit, Screencast (Jing). Скріни чи відео додають прямо в баг репорт.

Приклад: написання звіту про тестування олівця використовуючи функціональне тестування.

|  |  |
| --- | --- |
| Summary | Олівець моделі Skizzo HP ламається при писанні по картону |
| Severity | S.2 //*так як олівець ламається тільки при написання по картону,* *тому це* |
|  | *не є блокером для нас* |
| Priority | P.2 |  |
| Description | Preconditions: |
|  | - | Олівець і картон готові для писання |
|  | Steps to reproduce: |
|  | 1. | Взяти олівець в руку |
|  | 2. | Провести лінію по картону |
| Expected Result | 1. | Олівець залишає лінію на картоні |
|  | 2. | Олівець не ламається |
| Actual Result | 1. | Олівець ламається при писанні по картоні |

**Хід роботи:**

1. Ознайомтесь з текстом лабораторної роботи.
2. Провести функціональне тестування написаної студентом програми/вебсайту і при знаходженні дефектів завести баг репорти (*див.* *приклад зверху*). Додайте скріни відтворення дефекту до баг репортів.
3. Вибрати 5 тест кейсів з лабораторної роботи №2; уявіть ситуацію, що вибрані тест кейси не пройшли перевірку (функціонал, який перевіряється цими тест кейсами є несправним); створіть до цих тест кейсів баг репорти.
4. Оформити звіт із висновком.

**Контрольні запитання:**

1. Що таке дефект ПЗ?
2. Які етапи контролю якістю включає тестування?
3. Що таке звіт тестувальника (Bug report)
4. Які системи спостереження Вам відомі?
5. З яких обов’язкових складових елементів складається звіт?

**Лабораторна робота № 4**

**Тема:** навантажувальне тестування.

**Мета:** виконати навантажувальне тестування із використанням спеціалізованого програмного забезпечення.

**Теоретичні відомості:**

Тестування продуктивності(Performance Testing)–це тестування,якепроводиться з ціллю визначення, як швидко працює програма або її частина під деяким навантаженням. Зазвичай таким тестуванням перевіряється максимальна кількість користувачів сайту, час відповіді серверу, час відображення зображень.

В тестуванні продуктивності розрізняють наступні напрямки: навантажувальне, стресове, тестування стабільності, конфігураційне.

Навантажувальне тестування(Load Testing)–зазвичай проводитьсядля того, щоб оцінити поведінку програми(додатка) із заданим очікуваним навантаженням. Цим навантаженням може бути, наприклад, кількість користувачів, які будуть одночасно працювати з програмою. Такий вид тестування дозволяє отримати час відгуку всіх найважливіших бізнес-транзакцій.

Стресове тестування(Stress Testing)зазвичай використовується длявстановлення межі пропускної здатності програми. Цей тип тестування проводиться для визначення надійності системи під час екстремальних або непропорційних навантаженнях і відповідає на питання про достатню продуктивність системи у випадку, якщо поточне навантаження значно перевищить очікуваний максимум.

Тестування стабільності(Stability Testing)проводиться з метоюпереконатися в тому, що програма витримає очікуване навантаження протягом тривалого часу. При проведенні цього виду тестування здійснюється спостереження за споживанням програмою пам'яті, щоб виявити потенційні втрати. Крім цього, таке тестування виявляє деградацію продуктивності, що виражається в зниженні швидкості обробки інформації та збільшенням часу відгуку програми після тривалої роботи порівняно з початком тесту.

Конфігураційне тестування(Configuration Testing)—в цьому випадкузамість того щоб тестувати продуктивність системи з точки зору створеного навантаження, тестується ефект впливу на продуктивність змін в конфігурації. Вдалим прикладом такого тестування можуть бути експерименти з різними методами балансування навантаження. Конфігураційне тестування може бути поєднано з навантажувальним, стресовим або тестування стабільності.

**Хід роботи:**

1. Виконати навантажувальне тестування написаної студентом програми/вебсайту чи будь-якого іншого сайту з використанням спеціалізованого програмного забезпечення для перевірки навантаження на сайт. Додати короткий опис сайту.

1. Проаналізувати наданий програмою звіт про навантаження на сайт.
2. Оформити звіт до лабораторної роботи з висновками.

**Контрольні питання:**

1. Що таке тестування продуктивності? Які види тестування продуктивності ви знаєте?
2. Що таке навантажувальне тестування? Для чого виконується навантажувальне тестування?
3. Які програми ви знаєте для проведення навантажувального тестування?

**Лабораторна робота №5**

**Тема:** автоматизоване тестування.

**Мета:** сформувати позитивні та негативні тест-кейси для автоматизованого тестування з використанням спеціалізованого програмного забезпечення.

**Теоретичні відомості:**

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



Рис. 2 - Вікно програми Selenium IDE

Selenium IDE - це інтегроване середовище розробки скриптів Selenium. Вона реалізована як розширення Firefox і дозволяє записувати, редагувати та виправляти помилки. IDE Selenium - це не тільки інструмент запису: це повна IDE. Можна вибрати, чи використовувати його можливості запису, або ви можете редагувати свої скрипти вручну. Для користуванням Selenium IDE потрібно додати розширення для Mozilla Firefox браузера. Для ознайомлення роботи з Selenium IDE і його основних складових можна скористатися <http://toolsqa.com/selenium-ide-tutorial/>сайтом.

**Хід роботи:**

1. Записати якнайбільше автоматизованих позитивних і негативних тестів з використанням Selenium IDE для написаного студентом веб-сайту чи логін форми будь-якого іншого сайту.
2. Оформити звіт з висновками; для захисту потрібно показати виконання автоматизованих тест-кейсів на ноутбуці.

**Контрольні питання:**

1. Що таке автоматизоване тестування?
2. Охатактезуйте ПЗ Selenium IDE?

**Варіанти завдань лабораторних робіт**

**Варіант 1**

1. The tool shall easily allow connecting additional components, including generator, measuring device and more non-standard devices.
2. System allows to enter digits to username field: 1, 2, 3 and so on. Charge numbers should be validated on-line against the master corporate charge number list, if possible.
3. User can select end date.
4. The system shall process all mouse clicks very fast to ensure user’s do not have to wait.
5. On a launched Movement page user can see exactly the same information design details and prices that he choose (title, description, end date).
6. System should have a natural language interface. System shall be developed in one months.

**Варіант 2**

1. The system shall be implemented using ASP.
2. The System shall make it easy to find the required article.
3. For outbound and inbound flights, the user shall be able to compare flight prices from other, nearby airports.

The outbound and return flights shall be sorted by the smallest number of stops.

1. By clicking "Next Step" user can move to "Movement Launch" page.
2. System send appropriate email when user gonna change his/her username.
3. If a client approves the order and edits the order or postpones it, the system ought to give a request to pay the order.
4. All requirements specified in the Vision document shall be implemented and tested.

**Варіант 3**

1. Dates shall be displayed in the mm/dd/yyyy format. Dates shall be displayed in the dd/mm/yyyy format.
2. A destination country does not need to be displayed for flights within the Ukraine.
3. The search facility should allow the user to find a reservation based on Last Name, Date, etc.
4. If the customer opts out from receiving the email notification, customer profile should be updated to receive emails.
5. User can add additional color to current product.
6. To close New Ticket modal, cause close button is clicked.
7. The system shall validate and accept Credit Cards and cashier’s checks.

**Варіант 4**

1. For overseas flights, the system shall display a destination country.
2. Validate within the Phone Number box as 10 digits. Dashes, etc.
3. All configuration settings of the System shall be easily upgradeable in future releases.
4. As a user, I can change product.
5. User can edit Movement information on Manage Movement page.
6. The system shall store user information i Oracle database including First/Last Name, Address, and Social Security Number.
7. Sometimes the user will enter Airport Code, which the system will understand, but sometimes the closest city may replace it, so the user does not need to know what the airport code is, and it will still be understood by the system.

**Варіант 5**

1. The system shall identify the airport based on either an Airport Code or a City Name.
2. On the “Stored Flight” screen, the user can only view one record.
3. Please see User Story: ID 1.2.3 for details’.
4. As a user, I can upload image.
5. The application shall the high level of usability.
6. In the date picker, make sure current date is highlighted so user knows what today's date is.
7. Delay is determined by seconds.

We shall use minutes for describing of delays.

**Варіант 6**

1. System shall request user email addresses from social network.
2. The system shall not accept passwords shorter than 6 characters.
3. Icons for social networks on homepage to be determined by developers.
4. For users in the U.S., dates shall be displayed in the mm/dd/yyyy format. For users in France, dates shall be displayed in the dd/mm/yyyy format.
5. The system response time should be reduced twice.
6. I can upload image.
7. User can't set price lower than minimum, info message "Whoops! The price you entered must be higher than the minimum selling price." will appear.

**Варіант 7**

1. The system interface will be localization ready.
2. The System shall make it easy to find the required article.
3. User can see confirmation message "Your password was changed" when fill out correct info in "Current Password", "New Password" text fields and clicked button "Save Password". Confirmation email is sent.
4. The tool shall easily allow connecting additional components, including generator, measuring device and more non-standard devices.
5. User can delete colors.
6. The product shall provide status messages at regular intervals not less than every 60 seconds.
7. The airport code shall be entered or no.

**Варіант 8**

1. The system shall not accept username longer than 48 characters.
2. User should be able to run search.
3. Not only current user can add additional color to current product.
4. You cannot open new warranty when at least one warranty has already been existed.
5. The product shall switch between displaying and hiding non-printing characters instantaneously.
6. User see "Incorrect password" modal when incorrect password was typed into "Current Password" text field and “Save” button is activated.
7. System shall request user email addresses from social network by clicking Find button on My Account page.

**Варіант 9**

1. The system should never crash.
2. The system shall resist concurrent usage by many users.
3. User can see "Incorrect password" when incorrect password was typed into "Current Password" text field.
4. If a client approves the order and edits the order or postpones it, the system ought to give a request to pay the order.
5. Payment by PayPal shall be available.

Only credit card payments shall be accepted.

1. On selected image user should see “X” button which delete current image.
2. The system shall not accept passwords longer than 15 characters.

**Варіант 10**

1. "Login / Signup" window preserve entered info in text fields if window was closed.
2. User can change password by logging to his account and selecting "Account" button in main menu.
3. Customer can delete image background on My Account page by clicking Delete button on Settings section.
4. As a system, I can relocate image.
5. The system shall be implemented using ASP.
6. The list of available flights shall include flight numbers, departure time, and arrival time for every leg of a fligh.
7. All configuration settings of the System shall be easily upgradeable in future releases.

**Варіант 11**

1. The system shall validate and accept Credit Cards and cashier’s checks.
2. It should be sorted by price.
3. Entering incorrect credentials will bring user to Login page and info message will show "Email and password are invalid. Please try again".
4. By entering correct email and clicking "Submit" button on Reset password page user will receive email with password reset link.
5. Clicking on reset password link in email will open "Reset Your Password" page with text fields "New Password", "Confirm new password" and button "Save".
6. The application shall the high level of usability.
7. The system shall have a natural language interface that will understand commands given in English language.

**Варіант 12**

1. The system shall process all mouse clicks very fast to ensure user’s do not have to wait.
2. The system shall store user information in Oracle database including First/Last Name, Address, and Social Security Number.
3. We believe user get Welcome email message after registration on our site.
4. User can see "Must be 6 characters long" on Registration page when password is less than 6 characters was typed in.
5. Clicking "Save" button user will be logged in and taken to dashboard page.
6. "Your email" text field on Registration page is in focus by default.
7. The HTML Parser shall produce an HTML markup error report which allows quick resolution of errors when used by HTML novices.

**Список використаних джерел**

1. Cem Kaner Testing Computer Software (1st ed.). N.p.: TAB Professional & Reference Books. December 1987. [ISBN](https://en.wikipedia.org/wiki/ISBN_%28identifier%29) [0-8306-9563-X](https://en.wikipedia.org/wiki/Special%3ABookSources/0-8306-9563-X).
2. [Testing Computer Software](https://archive.org/details/testingcomputers00kane) (2nd ed.). N.p.: International Thomson Computer Press. 1993. [ISBN](https://en.wikipedia.org/wiki/ISBN_%28identifier%29) [0-442-01361-2](https://en.wikipedia.org/wiki/Special%3ABookSources/0-442-01361-2). (with coauthors Jack Falk and Hung Q. Nguyen)
3. Cem Kaner Bad Software: What To Do When Software Fails. New York: [John Wiley & Sons](https://en.wikipedia.org/wiki/John_Wiley_%26_Sons). 28 September 1998. [ISBN](https://en.wikipedia.org/wiki/ISBN_%28identifier%29) [0-471-31826-4](https://en.wikipedia.org/wiki/Special%3ABookSources/0-471-31826-4). (with coauthor David L. Pels)
4. Cem Kaner Testing Computer Software (2nd (softback) ed.). New York: Wiley. 12 April 1999 [1993]. [ISBN](https://en.wikipedia.org/wiki/ISBN_%28identifier%29) [0-471-35846-0](https://en.wikipedia.org/wiki/Special%3ABookSources/0-471-35846-0). (with coauthors Jack Falk and Hung Q. Nguyen) (Received the Award of Excellence from the [Society for Technical Communication](https://en.wikipedia.org/wiki/Society_for_Technical_Communication), Northern California Technical Publications Competition.)
5. Cem Kaner Lessons Learned in Software Testing: A Context-driven Approach. New York: Wiley. 15 December 2001. [ISBN](https://en.wikipedia.org/wiki/ISBN_%28identifier%29) [0-471-08112-4](https://en.wikipedia.org/wiki/Special%3ABookSources/0-471-08112-4). [LCC](https://en.wikipedia.org/wiki/LCC_%28identifier%29) [QA76.76.T48 K34 2001](https://catalog.loc.gov/vwebv/search?searchCode=CALL%2B&searchArg=QA76.76.T48+K34+2001&searchType=1&recCount=25). (with coauthors James Bach and Bret Pettichord and editor Margaret Eldridge)
6. Cem Kaner The Domain Testing Workbook. Context Driven Press. 30 October 2013. [ISBN](https://en.wikipedia.org/wiki/ISBN_%28identifier%29) [978-0989811903](https://en.wikipedia.org/wiki/Special%3ABookSources/978-0989811903). (with coauthors Sowmya Padmanabhan and Douglas Hoffman)
7. Cem Kaner Foundations of Software Testing Workbook. Context Driven Press. 30 December 2013. [ISBN](https://en.wikipedia.org/wiki/ISBN_%28identifier%29) [978-0989811927](https://en.wikipedia.org/wiki/Special%3ABookSources/978-0989811927). (with coauthor Rebecca L. Fiedler)
8. Бабенко Л.П., Лавріщева К.М. Основи програмної інженерії.– Навч. посібник.–К.: Знання, 2001. –269 с.
9. Лаврищева Е.М., Грищенко В.Н. Области знаний программной инженерии – SWEBOK и подход к обучению этой дисциплине// Управляющие системы и машины.–2005. – №1.– С.38–54.
10. Pfleeger S.L. Software Engineering. Theory and practice. – Printice Hall: Upper Saddenle River, New Jersey, 1998. – 576 p.
11. Jacobson I. Object-Oriented Software Engineering. A use Case Driven Approach, Revised Printing. – New York: Addison-Wesley Publ. Co., 1994. – 529 p.
12. Лаврищева Е.М. Методы программирования. Теория, инженерия, практика. – К.: Наук. думка, 2006.–450с.
13. Основы инженерии качества программных систем / Ф.И.Андон, Г.И.Коваль, Т.М. Коротун, Е.М.Лаврищева, В.Ю. Суслов  – К.: Академпериодика.– 2007. – 678 с.
14. Лаврищева Е.М.,  Коваль Г.И.,  Коротун Т.М. Подход к управлению качеством программных систем обработки данных // Кибернетика и системный анализ.– 2006.–№ 5.–С.174–185.