МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
«ОСНОВИ АНАЛІЗУ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»
ДЛЯ СТУДЕНТІВ ДЕННОЇ ФОРМИ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ 123 – «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ».

КРЕМЕНЧУК 2018

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Основи аналізу якості програмного забезпечення» для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерні системи та мережі»

Укладач к. т. н., доц. О. Г. Славко

Рецензент к. т. н., доц. В. М. Сидоренко

Кафедра комп'ютерних та інформаційних систем

Затверджено методичною радою КрНУ імені Михайла Остроградського
Протокол № 11 від 9 липня 2018 р.

Голова методичної ради _____ проф. В. В. Костін

ЗМІСТ

Вступ..... 4
1 Перелік лабораторних робіт..... 6
Лабораторна робота № 1 Опис і структура дефектів. Життєвий цикл дефектів..... 6
Лабораторна робота № 2 Тестова документація й артефакти тестування..... 10
Лабораторна робота № 3 Основний інструментарій для аналізу якості під час роботи з програмним забезпеченням..... 14
Лабораторна робота № 4 Основи тестування програмного забезпечення. Пошук дефектів у програмному застосунку ResumeBuilder..... 17
Лабораторна робота № 5 Основи тестування програмного забезпечення. Пошук дефектів у програмному застосунку ListBoxer..... 19
2 Критерії оцінювання якості виконання лабораторних робіт студентами..... 22
Список літератури..... 23

ВСТУП

Навчальна дисципліна є логічним продовженням навчального курсу «Прикладне програмування в комп'ютерних мережах», «Інженерія програмного забезпечення», «Комп'ютерні мережі», «Технології критичної програмної інженерії», «Організація баз даних». Предметом вивчення навчальної дисципліни є теорія і практика аналізу якості програмного забезпечення на базі сучасних технологій розробки програмного забезпечення, інструментів і систем баг-трекінгу та тест-управління.

У межах навчальної дисципліни розглядаються питання аналізу, забезпечення та контролю якості програмного забезпечення. Студенти набувають практичних навичок і здатності застосовувати технології та інструментальні засоби надання інформації про якість ПЗ кінцевому замовнику, підвищення якості ПЗ, запобігання появи дефектів, тестування програмного забезпечення.

Мета і завдання навчальної дисципліни: набуття студентами загальних теоретичних і практичних знань у галузі аналізу якості програмного забезпечення (ПЗ), які базуються на сучасних методологіях проектування програмних проектів і програмного забезпечення, організації функціонування та супроводження програмних проектів, а також контролю та забезпечення якості і тестування програмного забезпечення.

Місце навчальної дисципліни у навчальному процесі: навчальна дисципліна «Основи аналізу якості програмного забезпечення» тісно пов'язаний з такими дисциплінами, як «Прикладне програмування в комп'ютерних мережах», «Інженерія програмного забезпечення», «Комп'ютерні мережі», «Технології критичної програмної інженерії», «Організація баз даних» та іншими спеціальними дисциплінами навчального плану.

У результаті вивчення навчальної дисципліни студент повинен:

знати:

– основні моделі розробки програмного забезпечення;

– основні завдання та цілі забезпечення і контролю якості та тестування програмного забезпечення;

– рівні, типи і види тестування;

– тестові артефакти;

– життєвий цикл дефекту;

– основи вимоги до тестування та оцінювання якості програмного забезпечення, web-проектів і мобільних застосунків;

уміти:

– працювати із системами обліку дефектів (баг-трекінговими системами);

– працювати з Test Management системами;

– перевіряти функціональність, бізнес-логіку продукту, графічний інтерфейс, коректність виконання головних завдань продукту та зручність користувачів;

– працювати з техніками тест-дизайну;

– грамотно висловлювати свої думки в тексті, спілкуватися з різними типами людей (навіть тими, хто викликає неприязнь), планувати свій час.

1 ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1

Тема. Опис і структура дефектів. Життєвий цикл дефектів

Мета: ознайомлення та набуття практичних навичок формального опису дефектів програмного забезпечення в контексті життєвого циклу дефектів.

Короткі теоретичні відомості

Дефект (баг) – це невідповідність фактичного результату виконання програми очікуваному результату.

Дефекти виявляються на етапі тестування програмного забезпечення, коли тестер проводить порівняння отриманих результатів роботи програми (компонента або дизайну) з очікуваним результатом, описаним у специфікації вимог.

Error – помилка користувача, тобто він намагається використовувати програму іншим способом, наприклад, уводить літери в поля, де потрібно вводити цифри (вік, кількість товару тощо).

У якісній програмі передбачені такі ситуації і видаються повідомлення про помилку (error message), із червоним хрестиком.

Bug (defect) – помилка програміста (або дизайнера або того, хто бере участь у розробці), тобто якщо в програмі, що щось не так, як планувалося, і програма виходить з-під контролю. Наприклад, якщо не контролюється введення користувача, неправильні дані викликають краш чи інші проблеми в роботі програми. Або всередині програма побудована так, що спочатку не відповідає тому, що від неї очікується.

Failure – збій (причому не обов'язково апаратний) у роботі компонента, усієї програми або системи.

Тобто існують такі дефекти, які призводять до збоїв (A defect caused the failure) і існують такі, які не призводять, наприклад, UI-дефекти. Але апаратний збій, ніяк не пов'язаний з software, теж є failure.

Життєвий цикл дефекту

Життєвий цикл дефекту (bug workflow) – це послідовність етапів, які проходить баг на своєму шляху з моменту його створення до остаточного закриття. Для кращого сприйняття зображується у вигляді схеми з можливими статусами і діями, які призводять до зміни цих статусів (рис. 1).

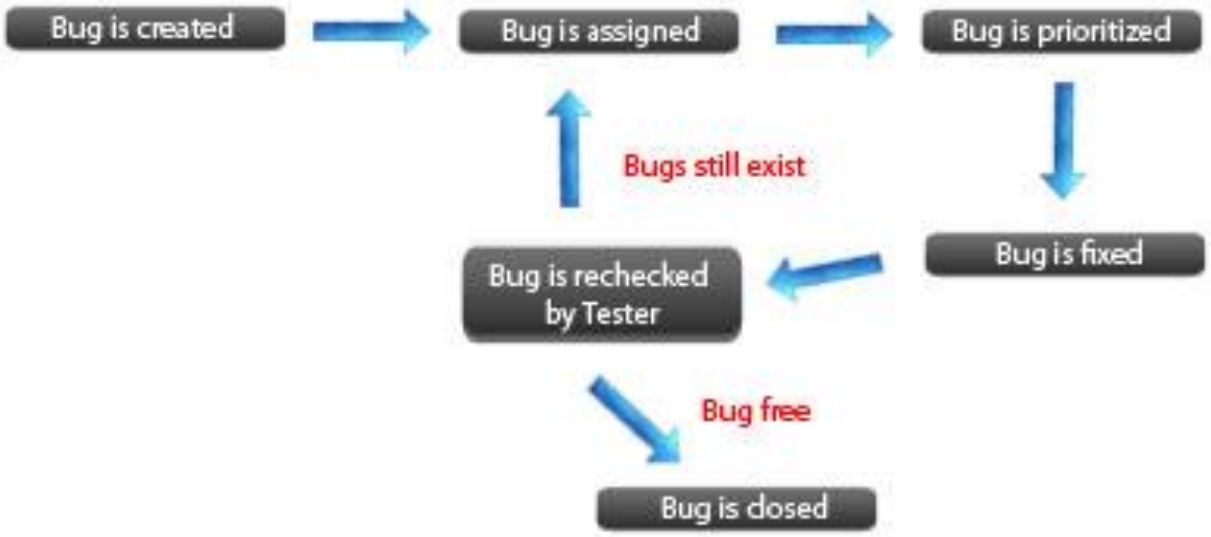

Рисунок 1 – Схема життєвого циклу дефекту

Найчастіше життєвий цикл бага пов’язаний з вибраною системою баг-трекінга, у якій за замовчуванням можуть бути настроєні статуси і переходи. Але також деякі системи керування дефектами дозволяють коригувати workflow бага у конкретному проекті, у разі такої необхідності.

Найпростіший життєвий цикл має такий вигляд:

Новий (статус присвоюється автоматично після внесення баг-репорт).

Відкритий (баг отримує цей статус після того, як була проведена його валідація керівником команди і ця помилка дійсно має бути виправлена).

Відхилений (присвоюється також після аналізу нового бага керівником команди в разі, якщо описана помилка вже раніше була внесена в систему (дублікат) або з якихось причин непотрібне її виправлення).

Виправлений (присвоюється фахівцем розробки після того, як помилка була усунена).

Повторно відкритий (у разі повторного виникнення помилки після її

попереднього виправлення).

Закритий (після остаточного виправлення бага і проведення додаткової перевірки).

Окрім вищевказаних основних статусів можуть також додатково використовуватися резолюції, наприклад:

– призначено на;

– потрібна додаткова інформація;

– у процесі виправлення;

– не може бути відтворений;

– на регресійне тестування.

На рис. 2 наведено схему типового workflow.

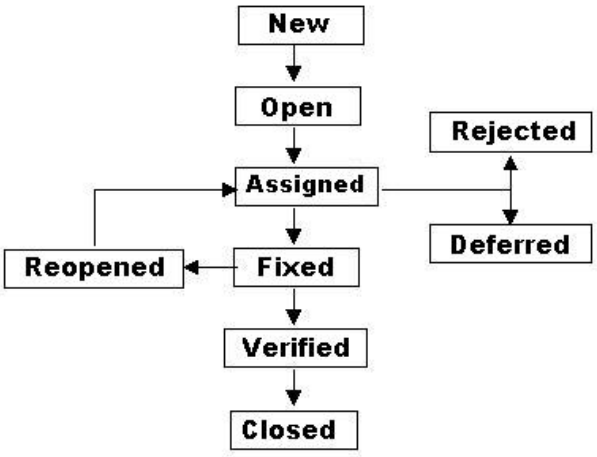

Рисунок 2 – Схема типового workflow бага

Порядок виконання роботи

1. Ознайомитися з матеріалом, викладеним у коротких теоретичних відомостях.

2. Побудувати детальну блок-схему типового життєвого циклу дефекту. Кожен блок має містити:

– опис задачі,

– указання, кому ця задача призначена;

– статус бага (за необхідності).

Для побудови блок-схеми використовувати такий алгоритм:

1. Тестер знаходить дефект.

2. Тестер оформляє звіт про дефект у баг-трекінгову систему.

3. Розробник перевіряє дефект (відтворюється він, чи ні) і присвоює йому один зі статусів:

«Дублікат» – подібний дефект уже існує у баг-трекінговій системі;

«Відхилено» – дефект не важкий;

«Відстрочка» – виправлення дефекту можна перенести в наступні версії програмного продукту;

«Не баг» – у функціонал програмного продукту не буде внесено ніяких змін;

«Відкритий» – розробник почав працювати з дефектом;

«Виправлений» – розробник вніс зміни в код і перевірив їх.

4. Тестер проводить повторне тестування дефекту.

5. Якщо дефект не відтворюється, тестер закриває його.

6. Якщо дефект відтворюється, тестер повертає його розробнику на виправлення і такий дефект проходить цей життєвий цикл ще раз.

Зміст звіту

1. Назва та мета роботи.

2. Методика проведення роботи з графічними результатами.

3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Надати визначення дефекту з точки зору забезпечення якості ПЗ.

2. Які типові статуси може набувати баг?

3. Які існують правила опису дефектів ПЗ?

4. Подати схему workflow для типової баг-трекінгової системи.

Література: [1–3].

Лабораторна робота № 2

Тема. Тестова документація й артефакти тестування

Мета: ознайомлення та набуття практичних навичок роботи з тестовою документацією та складання тестової документації відповідно до вимог.

Короткі теоретичні відомості

Розглянемо основну тестову документацію.

1. Test Case – це тестовий артефакт, суть якого полягає у виконанні певної кількості дій і/або умов, необхідних для перевірки певної функціональності програмної системи, що розробляється.

Структура даного артефакту полягає в наступному: що треба зробити (Action) – очікуваний результат (Expected result) – фактичний результат (Test result).

Безпосередньо сам тестовий випадок складається з 3 частин:

1. ***PreConditions*** (передумови) – або список кроків, які призводять систему в стан, придатний для тестування, або список перевірок умов того, що система вже знаходитися в необхідному стані.

2. ***Test Case Description*** (опис тестового випадку) – список дій, за допомогою яких здійснюється основна перевірка функціоналу (після якої і звіряється фактичний результат з очікуваним).

3. ***PostConditions*** (післяумови) – список дій, які повертають систему в початковий стан.

2. Testplan (план тестування) – документ, що описує весь обсяг робіт з тестування, починаючи з опису тестованих об’єктів, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, оцінки ризиків з варіантами їх вирішення.

Тест-план є важливою складовою будь-якого процесу тестування, оскільки містить усю необхідну інформацію, що описує цей процес. Але в більшості випадків Тест-план переважно має формальне значення, але, все ж, наявність надає багато переваг.

Залежно від конкретизації описуваних завдань, тест-план може мати два рівні деталізації: майстер тест-план і детальний тест-план.

Детальний тест-план містить завдання тестування для кожної команди, для кожного релізу або ітерації проекту. Детальний тест-план створюється або для декомпозованих частин проекту, або для невеликих проектів і складається з:

– переліку областей тестування з пріоритетами;

– стратегії тестування;

– переліку можливих ризиків;

– переліку необхідних ресурсів;

– плану виконання проекту.

Майстер тест-план створюється або для організації процесу тестування між декількома командами, які тестують один проект, але мають різні завдання, або для проекту, який складається з безлічі ітерацій, які пов'язує якась загальна інформація, повторення якої в кожному релізі займає надто багато часу. Майстер тест-план містить:

– загальну інформацію про проект (посилання на документацію, баг-трекер, і т.д.);

– положення, що описують процес тестування, закладу дефектів і т. д.;

– критерії готовності продукту до випуску.

Існують кілька шаблонів тест-планів (IEEE, RUP).

Баг репорт (bugreport) – це технічний документ, який містить повний опис бага з інформацією як про сам баг (короткий опис, серйозність, пріоритет і т. д.), так і про умови виникнення цього бага. Баг репорт повинен містити правильну, єдину термінологію, що описує елементи призначеного для користувача інтерфейсу та події цих елементів, що призводять до виникнення бага.

У загальному випадку, баг-репорт містить:

1) шапку:

– короткий опис (короткий опис проблеми);

– проект (назва поточного проекту);

– компонент додатка, у якому виник дефект;

– версія (версія білда, у якому знайдено баг);

– серйозність (градація ступеня впливу на додаток бага);

– пріоритет (черга виправлення бага);

– статус (відображає статус бага в своєму життєвому циклі);

– автор (автор баг репорт);

– призначення (хто повинен виправити дефект);

2) оточення:

– операційна система, розрядність, браузер, його версія і т. д.;

3) опис:

– кроки відтворення (опис шляху, який призводить до виникнення дефекту);

– фактичний результат (результат, який отримується після виконання всіх кроків відтворення);

– очікуваний результат (результат, який бути відповідно до вимог).

4. Додатки.

– приєднаний файл (логи, скріншоти, інші документи, які можуть допомогти відтворити проблему або розв’язати цю проблему).

Основні поля, присутність яких необхідна:

– короткий опис. Поле, де ви хочете розмістити весь сенс усього баг репорт. Найчастіше, у короткому описі лаконічно відповідають на 3 питання: «Де?», «Що?», «Коли?» (саме в такій послідовності, як би не хотілося змінити її за прикладом відомої всім гри);

– серйозність. Дефект або повністю зупиняє працездатність програми, або тільки частину функціональності, або інше;

– кроки до відтворення. Точний і зрозумілий опис усіх кроків, які призводять до появи дефекту, з урахуванням усіх необхідних вхідних даних;

– фактичний результат;

– очікуваний результат.

Порядок виконання роботи

1. Створити декілька тест-кейсів для будь-якого інтернет-магазину та будь-якого мобільного застосунку згідно табл. 1:

Таблиця 1 – Приклад опису тест-кейсу

Action	Expected Result	Test Result (passed/failed/blocked)
PreConditions		
do A1	verify B1	passed
do A2	verify B2	**failed**
Test Case Description:		
do A3	verify B3	**blocked**
PostConditions		

2. Написати декілька баг-репортів згідно зі структурою, наведеною в коротких теоретичних відомостях, для програми «ResumeBuilder», яку необхідно завантажити за цим посиланням:

https://www.portnov.com/files/ResumeBuilder_06_18_2012.zip.

3. Написати специфікацію вимог згідно зі стандартом IEEE 830 (структура SRS – Software requirements specification) до будь-якого власного курсового проекту, в якому розроблено програмне забезпечення.

Зміст звіту

1. Назва та мета роботи.
2. Методика проведення роботи з графічними результатами.
3. Письмові відповіді на контрольні питання.

Контрольні питання

1. Що таке матриця відповідності вимог?
2. Пояснити суть понять «верифікація» і «валідація».
3. Пояснити, у чому полягає різниця між QA,QC і тестуванням.

Література: [1, 6, 7].

Лабораторна робота № 3

Тема. Основний інструментарій для аналізу якості при роботі з програмним забезпеченням

Мета: набуття практичних навичок роботи з баг-трекінговими системам.

Короткі теоретичні відомості

На сьогодні Atlassian JIRA є одним з найвідоміших і популярних баг-трекерів. Окрім того, багато компаній використовують JIRA не тільки як баг-трекер, але і як систему керування проектами.

JIRA досить універсальна, щоб вирішувати велику кількість начебто не пов'язаних між собою завдань, і вона досить просто розширюється за рахунок розробки додаткових плагінів.

Проект в JIRA – це реєстр або контейнер для обліку «заявок».

JIRA веб-орієнтована і є java-застосунком, що за наявності відповідних знань може бути з легкістю змінено і доопрацьовано. Вихідний код можна отримати після придбання ліцензії. І у разі, коли необхідно розширити можливості системи, то скоріше знадобляться плагіни, які можна знайти на Marketplace (сховище плагінів для продуктів Atlassian).

Інтерфейс JIRA завантажується в браузері і вся основна робота з JIRA ведеться через браузер. Після закінчення встановлення буде запропоновано зареєструватися та активувати облікові записи користувачів.

Варіантами установки є установка на своєму сервері з використанням хостингу Atlassian. У цій лабораторній роботі буде використано другий спосіб.

Основою для роботи в JIRA є так звані дошки або board (набір інструментів).

Перша опція меню «Dashboards» – це робота з дошками. Тут можна створювати свої дошки, набирати туди різні набори інформації. Це зручно для управління процесом: видно кожну дію особи, яка зареєстрована в системі, можна управляти проектом, можна компонувати дошку для отримання інформації про поточний стан системи.

Другий пункт меню «Projects» – тут можна побачити поточні проекти (Current Projects), останні проекти (Recent Projects) і можна створити проекти або імпортувати якісь зовнішні проекти.

Третій пункт меню «Issues» (рішення) – те, що відбувається в системі.

Четвертий пункт меню «Boards» – створення та зміна дошок.

Проекти створюються адміністраторами, тімлідами, головними розробниками, які керуватимуть процесом.

Якщо натиснути **Create Project**, з'являється вікно вибору типу проекту.

Підрозділ **Software** пропонує створити проект, який заснований на розробці програмного забезпечення. Перший тип – це **Scrum**, другий – **Kanban** і найпростіший – **basic software development**.

Другий підрозділ – Business напрямок: просто керування проектами, управління процесом, управління завданнями.

Окрім того, тут можна імпортувати проект, якщо він є десь в іншій системі, можна розглянути його workflow.

Усі типи поставлених завдань проходитимуть через етапи Workflow. Етапи можна додавати і змінювати. Але в цілому цих основних етапів досить.

З Marketplace можна довантажити інші готові Workflow.

JIRA містить потужні механізми пошуку:

– пошук за всіма атрибутами;

– за виконавцями;

– різні фільтри;

– за темою та змістом;

– JQL – мова для складних вибірок.

Порядок виконання роботи

1. Перейти на сайт JIRA (https://www.atlassian.com/software/jira).

2. Вибрати варіант «Завантажити JIRA безкоштовно» («try it free»).

3. Зареєструватися на реальний email з яким-небудь доменом.

4. Підтвердити за допомогою email свій акаунт.

5. Вибрати опцію створення нового проекту та вибрати тип проекту (Scrum), натиснути «Ок».

6. Задати ім’я наприклад, Test Project (абревіатура проекту створюється автоматом). Проект буде додано до списку наявних проектів, тепер до нього можна додавати завдання і т. д.

7. Створити в цьому проекті декілька задач (issue):

– вибрати проект, тип задачі;

– сформулювати sammary (короткий опис) та опис (description), при цьому можна форматувати текст за необхідності та додавати додатки (можна прикріпити скрін, відео і т. д.);

– дати опис середовища (оточення, environment) – ті умови, за яких відтворюється цей баг, наприклад;

– натиснути Create (створити). При цьому задача додається до проекту та її можна переглянути, редагувати, видалити і т. д.

8. Створити спринт, указавши дату його завершення та додавши до нього задачі, створені в попередньому пункті (для цього послідовно необхідно натиснути Create Board –> Create Sprint –> Start Spring і виконати необхідні настроювання).

Контрольні питання

1. Як називається спеціальна стартова сторінка в JIRA?
2. Що відображає пункт меню «Projects» в JIRA?
3. Що відображається в JIRA, якщо зайти під обліковим записом?
4. Чи можна імпортувати проект, якщо він є десь в іншій системі?

Література: [4, 5, 8–11].

Лабораторна робота № 4

Тема. Основи тестування програмного забезпечення. Пошук дефектів у програмному застосунку ResumeBuilder

Мета: набуття практичних навичок застосування різних технік ручного тестування та опису знайдених дефектів у вигляді баг-репортів.

Короткі теоретичні відомості

Розглянемо типи тестування, які відрізняються знанням внутрішньої будови об’єкта тестування.

Black Box – відомо лише, як побудована система, що тестується.

Тестування методом «чорного ящика», також відоме як тестування, засноване на специфікації або тестування поведінки – техніка тестування, заснована на роботі виключно із зовнішніми інтерфейсами тестованої системи.

Згідно з ISTQB тестування «чорного ящика» – це:

– тестування, як функціональне, так і нефункціональне, яке не передбачає знання внутрішньої будови компонента або системи;

– тест-дизайн, заснований на техніці «чорного ящика» – процедура написання або вибору тест-кейсів на підставі аналізу функціональної або нефункціональної специфікації компонента або системи без знання її внутрішньої будови.

Тестована програма для тестувальника – чорний непрозорий ящик, змісту якого він не бачить. Метою цієї техніки є пошук помилок в таких категоріях:

– неправильно реалізовані або відсутні функції;

– помилки інтерфейсу;

– помилки у структурах даних або організації доступу до зовнішніх баз даних;

– помилки поведінки або недостатня продуктивність системи;

Отже, немає уявлення про структуру та внутрішню будову системи. Потрібно концентруватися на тому, що програма робить, а не на тому, як вона це робить.

White Box – відомі всі деталі реалізації програми, що тестується.

Тестування методом «білого ящика» (також: прозорого, відкритого, скляного ящика; засноване на коді або структурне тестування) – метод тестування програмного забезпечення, який передбачає, що внутрішня структура/пристрій/реалізація системи відомі тестувальникам. Вибираємо вхідні значення, ґрунтуючись на знанні коду, який їх оброблятиме. Також відомо, яким повинен бути результат цієї обробки. Знання всіх особливостей програми, що тестується та її реалізації – обов'язкові для цієї техніки. Тестування «білого ящика» – поглиблення під внутрішню будову системи, за межі її зовнішніх інтерфейсів.

Згідно з ISTQB тестування «білого ящика» – це:

– тестування, засноване на аналізі внутрішньої структури компонента або системи;

– тест-дизайн, заснований на техніці «білого ящика» – процедура написання або вибору тест-кейсів на підставі аналізу внутрішньої будови системи або компонента.

Grey Box

Summary: відомі тільки деякі особливості реалізації тестованої системи.

Тестування методом сірого ящика – метод тестування, який передбачає комбінацію White Box і Black Box підходів. Тобто внутрішній устрій програми відомий лише частково. Передбачається, наприклад, доступ до внутрішньої структури та алгоритмів роботи програмного забезпечення для написання максимально ефективних тест-кейсів, але саме тестування проводиться за допомогою техніки «чорного ящика», тобто, з позиції користувача. Цю техніку тестування також називають методом «напівпрозорого ящика».

Порядок виконання роботи

1. Завантажити десктопну програму «ResumeBuilder» за цим посиланням: https://www.portnov.com/files/ResumeBuilder_06_18_2012.zip.

2. Інсталювати програму ResumeBuilder на свій комп'ютер.

3. Провести тестування ResumeBuilder з використанням різних технік, знайти якомога більше багів.

4. Усі знайдені баги занести в JIRA у свій акаунт. Навести скріншоти створених у JIRA баг-репортів для програми «ResumeBuilder».

Контрольні питання

1. Перелічити переваги і недоліки тестування методом «білого ящика».

2. Перелічити переваги і недоліки тестування методом «чорного ящика».

3. Що таке юзабіліті-тестування? У чому полягає головна ідея цього методу тестування?

4. На підставі чого може проводитись функціональне тестування програмних застосунків?

Література: [12, 13].

Лабораторна робота № 5

Тема. Основи тестування програмного забезпечення. Пошук дефектів у програмному застосунку ListBoxer

Мета: набуття практичних навичок застосування різних технік ручного тестування та опису виявлених дефектів у вигляді баг-репортів.

Короткі теоретичні відомості

Функціональне тестування – це один з видів тестування, спрямованого на перевірку відповідностей функціональних вимог ПЗ його реальним характеристикам. Основним завданням функціонального тестування є підтвердження того, що програмний продукт, який розробляється, має весь функціонал, необхідний для замовника.

Залежно від мети, функціональне тестування може проводитися:

– на підставі функціональних вимог, зазначених у специфікації вимог.

При цьому для тестування створюються тестові випадки (testcases), складання яких ураховує пріоритетність функцій програмного забезпечення (ПЗ), які необхідно покрити тестами. Таким чином можна переконатися в тому, що всі функції розроблюваного продукту працюють коректно з використанням різних типів вхідних даних, їх комбінацій, кількості;

– на підставі бізнес-процесів, які має забезпечити додаток. У такому разі важливим є не так працездатність окремих функцій ПЗ, як коректність виконуваних операцій, з огляду на сценарії використання системи. Отже тестування в такому разі ґрунтуватиметься на варіантах використання системи (usecases).

Описані вище аспекти реалізуються за допомогою таких напрямів і рівнів тестування: модульне (компонентне); інтеграційне; системне; регресійне; приймальне.

Нефункціональне тестування – це тестування властивостей, які не належать до функціональності системи і визначаються нефункціональними вимогами, які характеризують продукт з таких сторін, :

– надійність (реакція системи на непередбачені ситуації);

– продуктивність (працездатність системи під різними навантаженнями);

– зручність (зручність роботи з додатком з огляду на користувача);

– масштабованість (вимоги до горизонтального або вертикального масштабування застосунку);

– безпека (захищеність призначених для користувача даних);

– портируваність (можливість перенесення застосунку на різні платформи).

Ці властивості системи можна досліджувати, використовуючи такі види тестування:

1. *Тестування установки* (Installation testing) – перевірка успішності установки додатка, його настроювання та видалення, що зменшує ризики втрати призначених для користувача даних, втрати працездатності програми та ін.

2. *Тестування зручності використання* (Usability testing) – характеризує систему щодо зручності використання кінцевим користувачем.

3. *Конфігураційне тестування* – дослідження працездатності програмної системи в умовах різних програмних конфігурацій.

4. *Тестування на відмову та відновлення* (Failover and Recovery Testing) – дослідження програмної системи щодо відновлення після помилок, збоїв. Оцінювання реакції захисних властивостей програми.

Порядок виконання роботи

1. Завантажити десктопну програму «**ListBoxer**» за цим посиланням:

https://onedrive.live.com/?authkey=%21ADXoHFFA7EDk3uw&cid=604400B879BA5CC1&id=604400B879BA5CC1%214335&parId=604400B879BA5CC1%214331&action=locate

2. Інсталювати програму ListBoxer на свій комп’ютер.

3. Занотувати у звіт короткий опис функціоналу цієї програми (повний опис принципу роботи ListBoxer, доступний за допомогою меню «Help» після запуску).

3. Провести тестування ListBoxer з використанням різних технік, знайти якомога більше багів.

4. Усі знайдені баги подати у вигляді баг-репортів у звіті.

Контрольні питання

1. У чому полягає зміст таких типів тестування, як статичне та динамічне тестування?

2. Які переваги модульного тестування?

3. Коли застосовується димове тестування?

4. Коли слід припинити тестування проекту, що тестується?

Література: [12, 13].

2 КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ СТУДЕНТАМИ

У 11-му семестрі студенти виконують 5 лабораторних робіт. Загальна кількість балів, яку отримують студенти за виконання лабораторних робіт, складає 25 балів – сума за захист виконаних лабораторних робіт (максимально по 5 балів на кожну лабораторну роботу).

Шкала оцінювання: національна та ECTS

<table>
<tr><td rowspan="2">Сума балів за всі види навчальної діяльності</td><td rowspan="2">Оцінка ECTS</td><td colspan="2">Оцінка за національною шкалою</td></tr>
<tr><td>Для екзамену, курсового проекту (роботи), практики</td><td>Для заліку</td></tr>
<tr><td>90–100</td><td>A</td><td>Відмінно</td><td rowspan="5">Зараховано</td></tr>
<tr><td>82–89</td><td>B</td><td rowspan="2">Добре</td></tr>
<tr><td>74–81</td><td>C</td></tr>
<tr><td>64–73</td><td>D</td><td rowspan="2">Задовільно</td></tr>
<tr><td>60–63</td><td>E</td></tr>
<tr><td>35–59</td><td>FX</td><td>Незадовільно з можливістю повторного складання</td><td>Не зараховано з можливістю повторного складання</td></tr>
<tr><td>0–34</td><td>F</td><td>Незадовільно з обов’язковим повторним вивченням навчальної дисципліни</td><td>Не зараховано з обов’язковим повторним вивченням навчальної дисципліни</td></tr>
</table>

СПИСОК ЛІТЕРАТУРИ

1. Майерс Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер. – М. : «Диалектика», 2012. – 272 с.
2. Криспин Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд / Л. Криспин, Дж. Грегори. – М. : «Вильямс», 2010. – 464 с.
3. Канер К. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / К. Канер, Дж. Фолк, Е. Нгуен. – Киев : ДиаСофт, 2001. – 544 с.
4. Калбертсон Р. Быстрое тестирование / Р. Калбертсон, К. Браун, Г. Кобб. – М. : «Вильямс», 2002. – 374 с.
5. Синицын С. В. Верификация программного обеспечения / С. В. Синицын, Н. Ю. Налютин. – М. : БИНОМ, 2008. – 368 с.
6. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер. – СПб. : Питер, 2004. – 320 с.
7. Кон М. Scrum: гибкая разработка ПО / М. Кон. – М. : «Вильямс», 2011. – 576 с.
8. Мартин Р. С. Быстрая разработка программ. Принципы, примеры, практика / Р. С. Мартин, Дж. В. Ньюкирк, Р. С. Косс. – М. : «Вильямс», 2004. – 752 с.
9. Савин Р. Тестирование Дот Ком или Пособие по жестокому обращению с багами в интернет-стартапах / Р. Савин. – М. : Дело, 2007. – 312 с.
10. Котляров В. П. Основы тестирования программного обеспечения / В. П. Котляров, Т. В. Коликова. – М. : Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2006. – 285 с.
11. Портал специалистов по тестированию и обеспечению качества ПО [Электронный ресурс]. – Режим доступа: http://software-testing.ru/.

12. Технические статьи [Электронный ресурс]. – Режим доступа: – http://training.qatestlab.com/front-page/blog/technical-articles/.
13. Куликов С.С. Тестирование программного обеспечения. Базовый курс. – Минск: Четыре четверти, 2017. – 312 с.

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Основи аналізу якості програмного забезпечення» для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерна інженерія»

Укладач к. т. н., доц. О. Г. Славко

Відповідальний за випуск зав. кафедри КІС А. В. Луговой

Підп. до др. _____. Формат 60×84 1/16. Папір тип. Друк ризографія.
Ум. друк. арк. ____. Наклад _____ прим. Зам. №_____. Безкоштовно.

Видавничий відділ
Кременчуцького національного університету
імені Михайла Остроградського
вул. Першотравнева, 20, м. Кременчук, 39600