Лабораторна робота №4
Тестування програмного забезпечення

I. Підготовка до лабораторної роботи

Ознайомитись з теоретичною та практичною частиною наведеною нижче для виконання даної лабораторної роботи

II. Теоретичні відомості

1. Вступ

Життєвий цикл програмного забезпечення (SDLC – Software Development Life Cycle) – період часу, який починається з моменту прийняття рішення про необхідність створення програмного продукту і закінчується в момент його повного вилучення з експлуатації. Цей цикл – процес побудови і розвитку програмного забезпечення.

Етап 1 – Планування (Planning). На цій фазі клієнт пояснює основні деталі і концепції проекту, оговорюється необхідний ресурс, час і бюджет, що необхідний для розробки.

Етап 2 – Аналіз вимог (Requirements analysis). Ця фаза розрахована для підготовки набору вимог. Потім йде етап узгодження вимог. Як результат ми маємо отримати узгоджений документ з вимогами.

Етап 3 – Дизайн і розробка (Design & Development). На цій фазі визначаються основні концепції дизайну програмного забезпечення. Після узгодження дизайну починається безпосередньо розроблення продукту.

Етап 4 – Впровадження (Implementation). Включає в себе програмування і отримання кінцевого продукту (бібліотеки, білди, документація).

Етап 5 – Тестування (Testing). На цій фазі проводиться перевірка на відповідність вимогам і підтвердження того, що продукт розроблений згідно з ними.

Етап 6 – Оцінка (Evaluation). На фазі оцінки (або пререлізу) продукт оцінюється замовником і вносяться останні уточнення.

Етап 7 – Реліз (Release). Заключна фаза розробки, враховуються уточнення, що зроблені замовником на фазі оцінки. Підготовка продукту в «коробці».

Етап 8 – Підтримка (Support). Фаза технічної підтримки продукту.

Тестування ПЗ (Software testing) – перевірка відповідності між реальною і очікуваною поведінкою програми.

Тестування – це процес дослідження ПЗ з метою виявлення помилок і перевірки якості.

У більш широкому сенсі: Тестування – це одна з технік контролю якості, що включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) та аналізу отриманих результатів (Test Analysis).

Тестування так само можна описати як процес верифікації та валідації того чи іншого програмного продукту, щоб дізнатися на скільки точно він задовольняє всім встановленим вимогам.

Верифікація (Verification – узгодження) – це процес оцінки системи або її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу (чи виконуються наші цілі, терміни, завдання, по розробці проекту, визначені на початку поточної фази.)

Валідація (Validation – затвердження) – це визначення відповідності ПЗ очікуванням і потребам користувача, вимогам до системи.

Приклад:

Замовник хоче щось. Узгоджується ТЗ, з якого ви розумієте що повинні зробити “синій автомобіль”. Ви починаєте процес виробництва, і в процесі перевіряєте те, що виходить: Чи автомобіль це? Чи він синій? Це – верифікація.

А потім ви приходите до замовника і говорите – ось дивіться, вийшов синій автомобіль, приймаєте роботу? А він каже: «хлопці, ви не так зрозуміли, я хотів рожевий трактор, переробити

швидко». Або навпаки каже: «так, це саме те що я хотів, швидше підписуйте акт здачі-приймання». Це – валідація.

Цілі і завдання процесу тестування

Основною метою процесу тестування – є доказ того, що результат розробки відповідає пред'явленим до нього вимогам.

Основна завданням тестування ПЗ: отримання інформації про статус готовності заявленої функціональності системи або програми.

Необхідні і достатні умови для проведення тестування

Необхідними умовами істинності твердження А називаються умови, без дотримання яких А не може бути істинним.

Достатніми називаються такі умови, за наявності (виконанні, дотриманні) яких твердження А є істинним.

Необхідні умови:

• Наявність об'єкта тестування, доступного для проведення випробувань.

• Наявність виконавця (ів) (людина або машина, або комбінація людина + машина)

Достатні умови:

• Наявність об'єкта тестування, доступного для проведення випробувань

• Наявність виконавця (ів) (людина або машина, або комбінація людина + машина)

• Наявність плану тестування

• Наявність тест кейсів / тестів

• Наявність звіту, що підтверджує виконання завдань і досягнення цілей, з тестування об'єкта

План Тестування (Test Plan) – це документ, що описує весь обсяг робіт з тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх дозволу .

Тестовий випадок (Test Case) – це артефакт, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується або її частини.

Основні атрибути Test Case:

1) ID (номер),

2) Name (ім'я),

3) Preconditions (умови і параметри),

4) Steps (кроки до відтворення),

5) Expected result (очікуваний результат),

6) Actual result (фактичний результат),

7) Postconditions (постумови)

Тест дизайн (Test Design) – це етап процесу тестування ПЗ, на якому проектуються і створюються тестові випадки (тест кейси), відповідно з визначеними раніше критеріями якості і цілями тестування.

Баг/Дефект Репорт (Bug Report) – це документ, що описує ситуацію або послідовність дій, що призвела до некоректної роботи об'єкта тестування, із зазначенням причин і очікуваного результату.

Тестове Покриття (Test Coverage) – це одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами.

Специфікація Тест Кейсів (Test Case Specification) – це рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття

Час Проходження Тест Кейса (Test Case Pass Time) – це час від початку проходження кроків тест кейса до отримання результату тесту.

Проблемні тестові випадки:

– Які залежні один від одного (наприклад, очікується, що частина кроків виконана в попередньому test case, є посилання на інші test cases)

– З нечітким формулюванням кроків

– З нечітким формулюванням ідеї або очікуваного результату

Test cases можуть бути:

– Позитивні – використовуються тільки коректні дані і перевіряють, чи правильно додаток виконує функцію;

– Негативні – використовуються як коректні, так і некоректні дані (мінімум 1 некоректний параметр) і ставить за мету перевірку виняткових ситуацій (спрацьовування валідаторів), а також перевіряє, що функція не виконується при спрацьовування валідатора.

Основні стани тест кейсу:

- Created (створений)
- Modified (змінений)
- Retired (більше не дійсний)

Основні стани проходження тест кейсу:

- No run (не запущено)
- Passed (пройдено)
- Failed (помилка)
- Blocked (заблоковано)
- Not Completed (не завершено)

2. Особливості вимог програмного забезпечення

Вимоги (Requirements) до програмного забезпечення – сукупність тверджень щодо атрибутів, властивостей, або якостей програмної системи, що підлягає реалізації:

- Одиничність – Вимога описує одну і тільки одну річ.
- Завершеність – Вимога повністю визначена в одному місці і вся необхідна інформація присутня.
- Послідовність – Вимога не суперечить іншим вимогам і повністю відповідає зовнішній документації.
- Атомарність – Вимога не може бути розбита на ряд більш детальних вимог без втрати завершеності.
- Відстежування – Вимога повністю або частково відповідає діловим потребам як заявлено зацікавленими особами і задокументовано.
- Актуальність – Вимога не стала застарілою з часом.
- Здійснимість – Вимога може бути реалізовано в межах проекту.
- Недвозначність – Вимога коротко визначена без звернення до технічного жаргону та інших прихованих формулювань. Вона виражає об’єктивні факти, можлива одна і тільки одна інтерпретація. Визначення не містить нечітких фраз. Використання негативних тверджень заборонено.
- Обов’язковість – Вимога представляє певну характеристику, відсутність якої призведе до неповноцінності рішення, яка не може бути проігнорована.
- Верифікованість – Реалізованість вимоги може бути визначена через один з чотирьох можливих методів: огляд, демонстрація, тест чи аналіз..

3. Методи та фази тестування

Методи тестування:

1. Білий ящик (WhiteBox) – цей метод заснований на тому, що розробник тесту має доступ до коду програм і може писати код, який пов’язаний з бібліотеками ПЗ, що тестується.
2. Чорний ящик (Black Box) – цей метод заснований на тому, що тестувальник має доступ до ПЗ тільки через ті ж інтерфейси, що і замовник або користувач, або зовнішні інтерфейси, що дозволяють іншому комп’ютеру або іншому процесу підключитися до системи для тестування.
3. Сірий ящик (Grey Box) – поєднує елементи двох попередніх підходів.

Фази тестування:

1. Створення тестового набору (Test Suit) для конкретного середовища тестування (Testing Environment)
2. Прогон програми на тестах з отриманням протоколу результатів тестування (Test Log)

3. Оцінка результатів виконання програми на наборі тестів з метою прийняття рішення про продовження або зупинку тестування.

4. Класи еквівалентності (Equivalence class)

Підхід полягає в наступному: вхідні/вихідні дані розбиваються на класи еквівалентності, за принципом, що програма веде себе однаково з кожним представником окремого класу. Таким чином, немає необхідності тестувати всі можливі вхідні дані, необхідно перевірити по окремо взятому представнику класу.

Клас еквівалентності – це набір значень змінної, який вважається еквівалентним.

Тестові сценарії еквівалентні, якщо:

- Вони тестують одне і те ж;
- Якщо один з них знаходить помилку, то й інші виявлять її;
- Якщо один з них не знаходить помилку, то й інші не виявлять її.

Еквівалентне розбиття: Розробка тестів методом чорного ящика, в якому тестові сценарії створюються для перевірки елементів еквівалентної області. Як правило, тестові сценарії розробляються для покриття кожній області як мінімум один раз.

Приклад:

Припустимо, ми тестуємо Інтернет-магазин, який продає олівці. У замовленні необхідно вказати кількість олівців (максимум для замовлення – 1000 штук). Залежно від замовленої кількості олівців змінюється вартість:

1 – 100 – 10 грн. за олівець;

101 – 200 – 9 грн. за олівець;

201 – 300 – 8 грн. за олівець і т.д.

З кожною новою сотнею, ціна зменшується на гривню.

Якщо тестувати «в лоб», то, щоб перевірити всі можливі варіанти обробки замовленої кількості олівців, потрібно написати дуже багато тестів (згадуємо, що можна замовити аж 1000 штук), а потім ще все це і протестувати. Спробуємо застосувати розбиття на класи еквівалентності. Очевидно, що наші вхідні дані можна розділити на наступні класи еквівалентності:

Невалідне значення:> 1000 штук;

Невалідне значення: <= 0;

Валідне значення: від 1 до 100;

Валідне значення: від 101 до 200;

Валідне значення: від 201 до 300;

Валідне значення: від 301 до 400;

Валідне значення: від 401 до 500;

Валідне значення: від 501 до 600;
Валідне значення: від 601 до 700;
Валідне значення: від 701 до 800;
Валідне значення: від 801 до 900;
Валідне значення: від 901 до 1000.

На основі цих класів ми і складемо тестові сценарії. Отже, якщо взяти по одному представнику з кожного класу, то отримуємо 12 тестів.

5. Багтрекінгові системи

Система відстеження помилок (багтрекінгова система, bugtracking system) – програма, розроблена з метою допомоги розробникам ПЗ враховувати і контролювати помилки (баги), знайдені в програмах, побажаннях користувачів, а також стежити за процесом усунення цих помилок і виконання або невиконання побажань.

Головний компонент такої системи – база даних, що містить відомості про виявлені дефекти:

- Номер (ідентифікатор об'єкта)
- Дата і час, коли був виявлений дефект
- Хто повідомив про дефект
- Хто відповідальний за усунення дефекту
- Короткий опис проблеми (обов'язкове поле)
- Критичність дефекту (обов'язкове поле) і пріоритет рішення
- Опис кроків для виявлення дефекту (відтворення неправильної поведінки програми) (обов'язкове поле)
- Очікуваний результат (обов'язкове поле)
- Фактичний результат (обов'язкове поле)
- Обговорення можливих рішень і їх наслідків
- Статус дефекту
- Версія продукту, в якій дефект був знайдений
- Версія продукту, в якій дефект виправлений
- Скріншот

Серйозність (Severity) – це атрибут, що характеризує вплив дефекту на працездатність програми:

1. Блокуюча (Blocker) – розробка або використання продукту неможливо. Необхідно негайне виправлення проблеми.
2. Критична (Critical) – серйозні проблеми (не працює критичний блок функціоналу, спостерігаються серйозні помилки, пов'язані з втратою даних тощо).
3. Значна (Major) – проблема, пов'язана з важливим функціоналом продукту.
4. Незначна (Minor) – проблема пов'язана з другорядним функціоналом продукту або є легкий обхідний шлях для цієї проблеми.
5. Тривіальна (Trivial) – проблема косметичного рівня («недопрацьований» інтерфейс: друкарські помилки, різнобій з кольорами)

Пріоритет (Priority) – це атрибут, який вказує на черговість виконання завдання або усунення дефекту. Чим вище пріоритет, тим швидше потрібно виправити дефект.

1. Високий (High) – помилка повинна бути виправлена якомога швидше, тому що її наявність є критичною для проекту.
2. Середній (Medium) – помилка повинна бути виправлена, її наявність не є критичною, але вимагає обов'язкового рішення.
3. Низький (Low) – помилка повинна бути виправлена. Її наявність не є критичною, і не вимагає термінового вирішення.

Порядок виправлення помилок з їх пріоритетами: High -> Medium -> Low

Життєвий цикл дефекту

Система відслідковування помилок використовує один з варіантів «життєвого циклу» помилки, стадія якого визначається поточним станом, або статусом, в якому знаходиться помилка.

Типовий цикл дефекту (Defect cycle):

1. Новий (New) – дефект зареєстрований тестувальником.
2. Призначено (Open) – призначений відповідальний за виправлення дефекту.
3. Вирішений (Fixed) – дефект переходить назад в сферу відповідальності тестувальника. Як правило супроводжується резолюцією, наприклад:
    a. Виправлено (виправлення включені у версію таку-то),
    b. Дубль (повторює дефект, вже знаходиться в роботі),
    c. НЕ виправлено (працює відповідно до специфікації, має занадто низький пріоритет, виправлення відкладено до наступної версії і т.п.)
4. Відхилений (Rejected) – запит додаткової інформації про умови, в яких дефект проявляється, або не згоду з тим, що виявлена поведінка системи дійсно являється дефектом.
5. Відкрито повторно (Reopen) – дефект знову знайдений в іншій версії.
6. Закритий (Closed) – використовується системою для закриття вирішених завдань.

Принцип: «Де? Що? Коли?»

Де?: У якому місці інтерфейсу користувача або архітектури програмного продукту знаходиться проблема.

Що?: Що відбувається або не відбувається згідно специфікації або вашому уявленню про нормальну роботу програмного продукту. При цьому вказуйте на наявність або відсутність об'єкта проблеми, а не на його утримання (його вказують в описі).

Коли?: У який момент роботи програмного продукту, по настанню якої події або за яких умов проблема проявляється.

6. Функціональне тестування (Functional Testing). Тестування безпеки (Security and Access Control Testing). Тестування взаємодії (Interoperability Testing)

Всі види тестування програмного забезпечення, залежно від переслідуваних цілей, можна умовно розділити на наступні групи:

• Функціональні (Functional testing)
• Нефункціональні (Non-functional testing)
• Пов'язані зі змінами (Regression testing)

Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component / Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing).

Функціональні види тестування розглядають зовнішню поведінку системи. Далі перераховані одні з найпоширеніших видів функціональних тестів:

- Функціональне тестування (Functional testing)
- Тестування безпеки (Security and Access Control Testing)
- Тестування взаємодії (Interoperability Testing)

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, "Як" система працює. Далі перераховані основні види нефункціональних тестів:

- Всі види тестування продуктивності:
    - тестування навантаження (Performance and Load Testing)
    - стресове тестування (Stress Testing)
    - тестування стабільності або надійності (Stability / Reliability Testing)
    - об'ємне тестування (Volume Testing)
- Тестування установки (Installation testing)
- Тестування зручності користування (Usability Testing)
- Тестування на відмову і відновлення (Failover and Recovery Testing)
- Конфігураційне тестування (Configuration Testing)
- Тестування, пов'язане зі змінами

Після проведення необхідних змін, таких як виправлення бага / дефекту, програмне забезпечення повинне бути перетестоване для підтвердження того факту, що проблема була дійсно вирішена. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту:

- Димове тестування (Smoke Testing)
- Регресійне тестування (Regression Testing)
- Тестування збірки (Build Verification Test)
- Санітарне тестування або перевірка узгодженості / справності (Sanity Testing)

Тестування функціональності може проводиться у двох аспектах:

• вимоги

• бізнес-процеси

Тестування в аспекті «вимоги» використовує специфікацію функціональних вимог до системи як основу для дизайну тестових випадків (Test Cases). У цьому випадку необхідно зробити список того, що буде тестуватися, а що ні, пріоритезувати вимоги на основі ризиків (якщо це не зроблено в документі з вимогами), а на основі цього пріоритезувати тестові сценарії (test cases). Це дозволить сфокусуватися і не упустити при тестуванні найбільш важливий функціонал.

Тестування в сенсі «бізнес-процеси» використовує знання цих самих бізнес-процесів, які описують сценарії щоденного використання системи. У цьому випадку тестові сценарії (test scripts), як правило, ґрунтуються на випадках використання системи (use cases).

Переваги функціонального тестування:

- імітує фактичне використання системи;

Недоліки функціонального тестування:

- можливість упущення логічних помилок у програмному забезпеченні;
- ймовірність надмірного тестування.

Досить поширеною є автоматизація функціонального тестування.

Тестування безпеки (Security and Access Control Testing) – це стратегія тестування, що використовується для перевірки безпеки системи, а також для аналізу ризиків, пов’язаних із забезпеченням цілісного підходу до захисту додатків, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних.

Загальна стратегія безпеки ґрунтується на трьох основних принципах: конфіденційність, цілісність, доступність.

Конфіденційність – це приховування певних ресурсів або інформації. Під конфіденційністю можна розуміти обмеження доступу до ресурсу деякої категорії користувачів, або іншими словами, за яких умов користувач авторизований отримати доступ до даного ресурсу.

Цілісність – Існує два основних критерії при визначенні поняття цілісності:

Довіра – Очікується, що ресурс буде змінений тільки відповідним способом певною групою користувачів.

Пошкодження і відновлення – у разі коли дані пошкоджуються або неправильно змінюються авторизованим або авторизованим користувачем, потрібновизначити наскільки важливою є процедура відновлення даних.

Доступність являє собою вимоги про те, що ресурси повинні бути доступні авторизованому користувачеві, внутрішньому об’єкту або пристрою. Як правило, чим більш критичний ресурс тим вище рівень доступності.

Тестування взаємодії (Interoperability Testing) – це функціональне тестування, що перевіряє здатність програми взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності (compatibility testing) і інтеграційне тестування (integration testing).

7. Нефункціональне тестування

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, “Як” система працює. Далі перераховані основні види нефункціональних тестів:

- Всі види тестування продуктивності (Performance testing):
  - тестування навантаження (Load Testing)
  - стресове тестування (Stress Testing)
  - тестування стабільності або надійності (Stability / Reliability Testing) об'ємне тестування (Volume Testing)
  - Тестування установки (Installation testing)
  - Тестування зручності користування (Usability Testing)
  - Тестування на відмову і відновлення (Failover and Recovery Testing)
  - Конфігураційне тестування (Configuration Testing)

Тестування навантаження або тестування продуктивності – це автоматизоване тестування, що імітує роботу певної кількості бізнес користувачів на ресурсі.

Завданням тестування продуктивності є визначення масштабованості програми під навантаженням, при цьому відбувається:

- вимір часу виконання обраних операцій при певних інтенсивностях виконання цих операцій
- визначення кількості користувачів, що одночасно працюють з додатком
- визначення меж прийнятної продуктивності при збільшенні навантаження (при збільшенні інтенсивності виконання цих операцій)
- дослідження продуктивності на високих, граничних, стресових навантаженнях

Стресове тестування дозволяє перевірити наскільки додаток і система в цілому працездатні в умовах стресу і також оцінити здатність системи до регенерації, тобто до повернення до нормального стану після припинення впливу стресу. Стресом в даному контексті може бути підвищення інтенсивності виконання операцій до дуже високих значень або аварійна зміна конфігурації сервера.

Завданням тестування стабільності (надійності) є перевірка працездатності програми при тривалому (багатогодинному) тестуванні з середнім рівнем навантаження. Часи виконання операцій можуть грати в даному вигляді тестування другорядну роль. При цьому на перше місце виходить відсутність втрат пам'яті, перезапусків серверів під навантаженням та інших аспектів, що впливають саме на стабільність роботи.

Завданням об'ємного тестування є отримання оцінки продуктивності при збільшенні обсягів даних в базі даних програми, при цьому відбувається:

- вимір часу виконання обраних операцій при певних интенсивностях виконання цих операцій;
- може проводитися визначення кількості користувачів, що одночасно працюють з додатком.

Тестування Установки (Installation Testing) направлено на перевірку успішної інсталяції і настройки, а також оновлення або видалення програмного забезпечення.

Для того щоб додаток був популярним, йому мало бути функціональним – він має бути ще й зручним. Тестування зручності користування (Usability Testing) – це метод тестування, спрямований на встановлення ступеня зручності використання, зрозумілості та привабливості для користувачів розроблюваного продукту в контексті заданих умов.

Тестування зручності користування дає оцінку рівня зручності використання програми за наступними пунктами:

- продуктивність, ефективність (efficiency) – скільки часу і кроків знадобиться користувачеві для завершення основних завдань програми, наприклад, розміщення новини, реєстрації, покупка і т.д. (менше – краще);
- правильність (accuracy) – скільки помилок зробив користувач під час роботи з додатком (менше – краще);
- активізація в пам'яті (recall) – як багато користувач пам'ятає про роботу програми після призупинення роботи з ним на тривалий період часу (повторне виконання операцій після перерви повинно проходити швидше ніж у нового користувача);
- емоційна реакція (emotional response) – як користувач себе почуває після завершення завдання – розгублений, випробував стрес? Порекомендує користувач систему своїм друзям? (позитивна реакція – краще).

Тестування на відмову і відновлення (Failover and Recovery Testing) перевіряє тестований продукт з точки зору здатності протистояти і успішно відновлюватися після можливих збоїв, що виникли у зв'язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв'язку (наприклад, відмова мережі). Метою даного виду тестування є перевірка систем відновлення (або дублюючих основний функціонал систем), які, у разі виникнення збоїв, забезпечать збереження і цілісність даних тестованого продукту. Методика подібного тестування полягає в симулюванні різних умов збою і наступному вивченні та оцінці реакції захисних систем. У процесі подібних перевірок з'ясовується, чи була досягнута необхідна ступінь відновлення системи після виникнення збою.

Конфігураційне тестування (Configuration Testing) – називається тестування сумісності продукту, що випускається (програмне забезпечення) з різним апаратним і програмним засобами. Основні цілі – визначення оптимальної конфігурації і перевірка сумісності програми з необхідним оточенням (обладнанням, ОС і т.д.).

8. Види тестування, пов'язані зі змінами. Кросбраузерність

Після проведення необхідних змін, таких як виправлення бага / дефекту, програмне забезпечення повинне бути перетестоване для підтвердження того факту, що проблема була дійсно вирішена. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту:

- Регресійне тестування (Regression Testing)
- Димове тестування (Smoke Testing)
- Санітарне тестування або перевірка узгодженості / справності (Sanity Testing)
- Тестування збірки (Build Verification Test)

Регресійне тестування (Regression Testing) – це вид тестування спрямований на перевірку змін, зроблених в додатку або середовищі (лагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб сервер або сервер додатки), для підтвердження того факту, що існуюча раніше функціональність працює як і раніше.

Регресійними можуть бути як функціональні, так і нефункціональні тести.

Як правило, для регресійного тестування використовуються тест кейси, написані на ранніх стадіях розробки і тестування. Це дає гарантію того, що зміни в новій версії програми не пошкодили вже існуючу функціональність.

Рекомендується робити автоматизацію регресійних тестів, для прискорення подальшого процесу тестування і виявлення дефектів на ранніх стадіях розробки програмного забезпечення. Сам по собі термін "Регресійне тестування", залежно від контексту використання може мати різний зміст. Сем Канер, наприклад, описав 3 основних типи регресійного тестування:

- Регресія багів (Bug regression) – спроба довести, що виправлена помилка насправді не виправлена.
- Регресія старих багів (Old bugs regression) – спроба довести, що недавня зміна коду чи даних зламала виправлення старих помилок, тобто старі баги стали знову відтворюватися.
- Регресія побічного ефекту (Side effect regression) – спроба довести, що недавня зміна коду чи даних зламала інші частини продукту.

Димове тестування (Smoke Testing)

Поняття пішло з інженерної середовища: "При введенні в експлуатацію нового обладнання вважалося, що тестування пройшло вдало, якщо з установки не пішов дим." В області ж тестування програмного забезпечення, воно застосовується для поверхневої перевірки всіх модулів програми на предмет працездатності і наявності швидкого знаходження критичних і блокуючих дефектів.

Санітарне тестування (Sanity Testing)

Це вузьконаправлене тестування, достатнє для доказу того, що конкретна функція працює згідно заявленим в специфікації вимогам. Є підмножиною регресійного тестування. Використовується для визначення працездатності певної частини програми після змін вироблених в ній або навколишньому середовищі. Зазвичай виконується вручну.

Тестування збірки (Build Verification Test)

Тестування спрямоване на визначення відповідності, випущеної версії, критеріям якості для початку тестування. За своїми цілями є аналогом димового тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Вглиб воно може проникати далі, залежно від вимог до якості випущеної версії.

Відмінність санітарного тестування від димового (Sanity vs Smoke testing)

У деяких джерелах помилково вважають, що санітарне та димове тестування – це одне і теж. Ми ж вважаємо, що ці види тестування мають "вектори руху", що спрямовані в різні боки. На відміну від димового (Smoke testing), санітарне тестування (Sanity testing) направлено вглиб функції, що перевіряється, в той час як димове направлено вшир, для покриття тестами якомога більшого функціоналу в найкоротші терміни.

Кросбраузерність (Cross-browser) – властивість сайту відображатися і працювати у всіх популярних браузерах ідентично. Під ідентичністю розуміється відсутність недоліків верстки і здатність відображати матеріал з однаковим ступенем читабельності.

Тестування сайту на кроссбраузерность необхідно проводити:

У різних браузерах (сімейство Mozila, Internet Explorer, Opera, Safari, Мобільні браузери)

При різних розширеннях екрану (зазвичай 640 * 480, 800 * 600, 1024 * 768, 1280 * 800, 1280 * 1024, 1366 * 768)

В різних операційних системах (Mac OS, Linux, Win)

III. Завдання на лабораторну роботу

1. Зареєструйтеся у TestRail (або іншій аналогічній системі): https://www.gurock.com/testrail/
2. Напишіть 5 Test Case на п'ять будь-яких функцій для подальшої перевірки продукту.
3. Зареєструйся у Jira (або іншій аналогічній системі).
4. Заповніть баг репорти на підставі тест кейсів.

IV. Контрольні питання

1. Що таке тестування програмного забезпечення?
2. Які види тестування програмного забезпечення ви знаєте?
3. Які основні принципи тестування програмного забезпечення?
4. Що таке багтрекінгова система?
5. Які основні функції багтрекінгової системи?
6. Які типи багів можна відстежувати в багтрекінговій системі?
7. Які є переваги використання багтрекінгової системи для проектів розробки ПЗ?
8. Які принципи керування процесом відстеження багів в багтрекінговій системі ви знаєте?
9. Які види тестування програмного забезпечення ви знаєте?
10. Що таке модульне тестування і для чого воно використовується?
11. Які підходи використовуються для тестування взаємодії програмного забезпечення зі зовнішніми системами?
12. Які основні принципи проведення тестування регресії?
13. Які методики тестування використовуються для перевірки відповідності вимогам функціональності програмного забезпечення?