Лекція 5.
Дефекти та помилки

Зміст

Цільова настанова: Розглянуто термінологія щодо дефектів та помилок, класифікацію дефектів, життєвий цикл дефекту.

Ключові слова: дефект, помилка, класифікація дефектів, життєвий цикл дефекту.

1. Термінологія щодо дефектів та помилок

Визначення з ISTQB:

Прорахунок (mistake): Див помилку;

Перешкода (bug): Див. Дефект;

Недолік (fault): Див. дефект;

Помилка (error): Дія людини, що призводить до неправильного результату;

Дефект (defect): Вада в компоненті або системі, що може призвести компонент або систему до неможливості виконати необхідну функцію, наприклад, неправильний оператор або визначення даних. Дефект, виявлений під час виконання, може призвести до відмови компонента або системи;

Відмова (failure): Відхилення компонента або системи від очікуваного виконання, експлуатації або результату.

Неофіційні джерела показують ширшу картину:

Помилка (Error) виникає через прорахунок (Mistake) у написанні коду розробником;

Дефект (Defect) це прихований недолік ПЗ, що виник через помилку в написанні коду;

Коли дефект (Defect) виявляється тестувальником, він називається багом (Bug);

Якщо тестувальники упустили дефект і знайшов користувач, це збій (Failure);

Якщо програма не виконує свою функцію, це відмова (Fault).

То що таке баг на практиці? Коли маємо ситуацію “1 вимога = 1 тест-кейс”, питання відпадає саме собою - тест-кейс не минуло, отже вимога реалізовано неправильно , отже баг.

Але зазвичай варіантів значно більше:

працювало, але раптом перестало;

працює, але неправильно;

реалізація не відповідає опису та у завданні у явному вигляді не зафіксовано коригування;

потрібно змінити назву кнопки/сторінки/розділу, тому що в них є помилка або "Скасувати скасування" (класика!);

помилки в принципі (легко може мати різний пріоритет залежно від цілей та завдань проекту);

після збереження інформація не з'являється на сторінці, навіть якщо консолі 200 ОК;

не всі вказані при збереженні поля відображаються на сторінці, але поля незмінно відображаються під час редагування;

при натисканні на кнопку “ВИДАЛИТИ ЗАГАЛЬНО ВСІ ДАНІ КЛІЄНТА” немає модального вікна з підтвердженням Так /Ні, та й зробити це може будь-який користувач без авторизації, який знайшов посилання;

по переходу за прямим посиланням на послугу не перевіряється який користувач зараз авторизований і таким чином можна подивитися чужі профілі або деталі послуг, якщо підібрано валідний id ;

можна cURL'ом замовити послугу іншому клієнту або в Elements через DevTools змінити вартість у кошику (не провертайте такі сценарії не на своїх робочих проектах);

інформація стирчить за межами свого блоку або "нашаровується" на інший (ж-ж-ж-жуть, але на деяких проектах цим можна легко знехтувати);

сторінка дуже довго відкривається, ну дуже довго - секунд 30 на стабільному інтернеті (розлючений клієнт гарантований);

система робить щось, що вона не повинна робити згідно з початковим задумом. Наприклад, закриття облікового запису не тільки переводить його в статус "Закрито", але й повертає клієнту всі гроші, які він приніс проекту за весь час співпраці за вже надані послуги (о-о-ой!);

незручно користуватися. Наприклад, щоб подивитись деталі послуги клієнта, потрібно зайти на три вкладки вглиб облікового запису, а дивитися потрібно 2-3 рази на день. Або незручно копіювати інформацію зі сторінки, а з робочих питань це потрібно робити кілька разів на день – це баг інтерфейсу і він має бути виправлений.

При цьому часто може виникнути споконвічне питання "баг чи фіча?", коли баг-репорт заводити не потрібно. Це фіча- реквест , якщо:

потрібно змінити назву кнопки/сторінки/розділу, тому що є відчуття, що воно не відбиває дійсності;

фічу зробили, але після використання видно, що є простір для суттєвих покращень. Наприклад, за послугою не вистачає моніторингу або статистичних даних щодо використання, а за перевитрату може стягуватися додаткова плата – клієнт точно буде нещасливим у невіданні;

знаєте , як покращити ту чи іншу частину системи, щоб було зручніше. Наприклад, меню необґрунтовано займає 30% ширини екрану, а корисна інформація тулиться на 70%, що залишилися;

Користувач регулярно робить рутинні монотонні дії, які можна автоматизувати. Наприклад, копіювати однотипну інформацію з 12 сторінок пагінації, коли просте вивантаження вирішило б проблему;

винаходите велосипед з діючих фіч продукту, щоб досягти бажаного результату;

на сторінці не вистачає якоїсь інформації чи можливості її додати;

на сторінці не вистачає фільтрів та пагінації, коли інформації багато і важко знайти потрібне або відображення 1000+ елементів суттєво позначається на швидкості завантаження сторінки;

Користувач веде додаткову звітність у блокноті/ екселе , коли проблему можна вирішити виведенням ID на сторінці та кількома фільтрами.

Добре, якщо в команді є UX/UI дизайнер, а якщо ні? Тестувальнику варто розрізняти що в дизайні баг, який може призвести до сумних наслідків, а що запит на покращення, який зробить взаємодію користувачів з системою більш гладкою та зручною, але може бути реалізований пізніше.

2. Класифікація дефектів

Дефекти можна класифікувати по-різному. Для організації важливо дотримуватися єдиної схеми класифікації та застосовувати її до всіх проектів. Деякі дефекти можна віднести до кількох класів чи категорій. Через цю проблему розробники, тестувальники та співробітники SQA повинні бути максимально послідовними при записі даних про дефекти.

Класи дефектів:

Дефекти вимог та специфікацій (Requirements and Specifications Defects): Початок життєвого циклу програмного забезпечення важливий для забезпечення високої якості програмного забезпечення, що розробляється. Дефекти, введені на ранніх етапах, дуже важко усунути на пізніших етапах. Оскільки багато документів з вимогами написані з використанням подання природною мовою, вони можуть стати:

Двозначними;

Суперечливими;

Незрозумілими;

Надмірними;

Неточними.

Деякі специфічні дефекти вимог / специфікацій:

Дефекти функціонального опису: Загальний опис того, що робить продукт і як він повинен поводитися (входи / виходи), неправильно, двозначно і / або неповно;

Дефекти функцій: описуються як відмінні характеристики програмного компонента чи системи. Дефекти функцій пов'язані з відсутністю, неправильним, неповним чи непотрібним описом функцій;

Дефекти взаємодії функцій: це відбувається через неправильний опис того, як функції повинні взаємодіяти один з одним;

Дефекти опису інтерфейсів: це дефекти, які виникають в описі взаємодії цільового програмного забезпечення із зовнішнім програмним забезпеченням, обладнанням та користувачами.

Дефекти дизайну: дефекти дизайну виникають коли неправильно спроектовані: Системні компоненти, Взаємодія між компонентами системи, Взаємодія між компонентами та зовнішнім програмним/апаратним забезпеченням або користувачами. Вони включають дефекти в конструкції алгоритмів, управління, логіки, елементів даних, описів інтерфейсів модулів та описів зовнішнього програмного забезпечення / обладнання / інтерфейсу користувача.

До дефектів дизайну відносяться:

Алгоритмічні дефекти та дефекти обробки: це відбувається, коли етапи обробки в алгоритмі, описаному псевдокодом, неправильні;

Дефекти управління, логіки та послідовності: Дефекти управління виникають, коли логічний потік у псевдокоді невірний;

Дефекти даних: вони пов'язані з неправильним дизайном структур даних;

Дефекти опису інтерфейсів модулів: ці дефекти виникають через неправильне або непослідовне використання типів параметрів, неправильної кількості параметрів або неправильного порядку параметрів;

Дефекти функціонального опису: до дефектів цієї категорії належать неправильні, відсутні або неясні елементи дизайну;

Дефекти опису зовнішніх інтерфейсів: вони виникають через неправильні описи дизайну інтерфейсів з компонентами COTS, зовнішніми програмними системами, базами даних та апаратними пристроями.

Дефекти коду: Дефекти кодування виникають через помилки під час реалізації коду. Класи дефектів кодування аналогічні класам дефектів дизайну. Деякі дефекти кодування виникають через нерозуміння конструкцій мови програмування та недорозуміння з розробниками.

Алгоритмічні дефекти та дефекти обробки:

Неперевірені умови overflow and underflow;

Порівняння невідповідних типів даних;

Перетворення одного типу даних на інший;

неправильний порядок арифметичних операторів;

Неправильне використання або пропуск круглих дужок;

Втрата точності (Precision loss);

Неправильне використання символів.

Дефекти управління, логіки та послідовності: цей тип дефектів включає неправильне вираження операторів case, неправильне повторення циклів та пропущені шляхи;

Типографічні дефекти: в основному це синтаксичні помилки, наприклад, неправильне написання імені змінної, які зазвичай виявляються компілятором, self-reviews , or peer reviews;

Дефекти ініціалізації: цей тип дефектів виникає, коли оператори ініціалізації пропущені чи неправильні. Це може статися через нерозуміння або відсутність зв'язку між програмістами або програмістом і дизайнером, недбалості або нерозуміння середовища програмування;

Дефекти потоку даних: дефекти потоку даних виникають, коли код не дотримується необхідних умов потоку даних;

Дефекти даних: це вказує неправильна реалізація структур даних;

Дефекти інтерфейсу модуля: виникають через використання неправильних чи несумісних типів параметрів, неправильної кількості параметрів або неправильного порядку параметрів;

Дефекти документації коду: коли документація за кодом не визначає, що програма насправді робить, або є неповною або двозначною;

Зовнішнє обладнання, дефекти програмних інтерфейсів: ці дефекти виникають через проблеми, пов'язані з Системними викликами, Посиланнями на бази даних, Послідовністю введення/виведення, Використанням пам'яті, Використанням ресурсів, Обробкою переривань та винятків, Обміном даними з обладнанням, Протоколами, Форматами, Інтерфейси з файлами збірки, Тимчасовими послідовностями.

Дефекти тестування: Плани тестування, тестові набори, засоби тестування та процедури тестування можуть містити дефекти. Ці дефекти називають дефектами тестування. Дефекти у планах тестування найкраще виявляти за допомогою методів review .

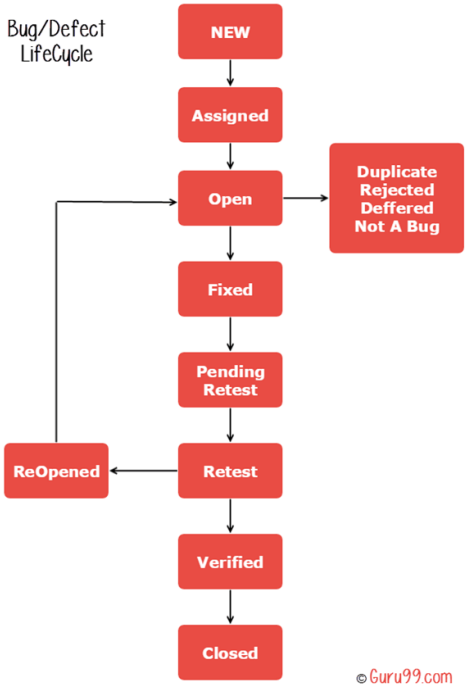
Дефекти тестової обв'язки: Для тестування програмного забезпечення на рівні модулів та інтеграції необхідно розробити допоміжний код. Це називається Test Harness або scaffolding code . Test Harness має бути ретельно спроектований, реалізований та протестований, оскільки це робочий продукт, і цей код можна повторно використовувати для розробки нових версій програмного забезпечення;

Дизайн тестового випадку та дефекти процедури тестування: сюди входять неправильні, неповні, відсутні, невідповідні тестові приклади та процедури тестування.

3. Життєвий цикл дефекту

Життєвий цикл дефекту (Defect/Bug Life)

Життєвий цикл дефекту - це уявлення різних станів дефекту, у яких він перебуває від початкового до кінцевого етапу свого існування. Він може змінюватись від компанії до компанії та налаштовуватися під процеси конкретного проекту**.**



Статуси дефекту :

Новий (New): коли новий дефект реєструється та публікується вперше;

Призначений (Assigned): після публікації бага тестувальником керівник тестувальника затверджує помилку та передає її команді розробників;

Відкритий (Open): розробник починає аналіз і працює над виправленням бага;

Виправлено (Fixed): розробник вніс необхідну зміну в код і перевірив його;

Чекає на повторне тестування (Pending retest): як тільки дефект буде виправлено, розробник надає тестувальнику конкретний код повторного тестування коду. Оскільки тестування програмного забезпечення залишається незавершеним з боку тестувальників, йому надається статус «очікує повторного тестування»;

Повторне тестування (Retest): на цьому етапі тестувальник виконує повторне тестування коду, щоб перевірити, чи виправлено дефект розробником;

Перевірений (Verified): тестувальник повторно тестує баг після його виправлення розробником. Якщо баг виправлений, то надається статус «перевірено»;

Перевідкритий (Reopen): якщо баг зберігається навіть після того, як розробник виправив баг, тестувальник змінює статус на «повторно відкритий». І знову баг проходить життєвий цикл.

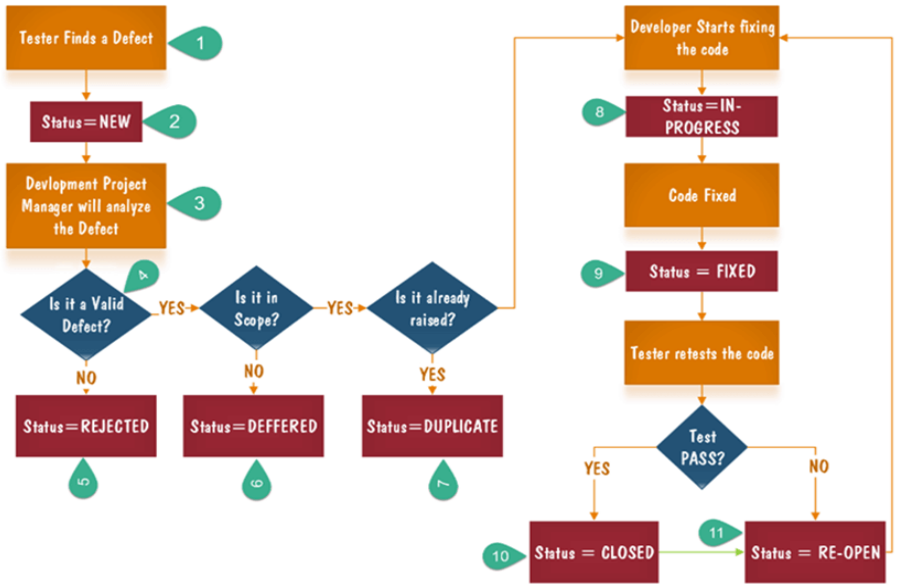
Закрито (Closed): якщо баг більше не існує, тестувальник надає статус «Закрито».

Дубль (Duplicate): якщо дефект повторюється двічі чи дефект відповідає тієї ж концепції помилки, статус змінюється на «дублювати».

Відхилений (Rejected): якщо розробник вважає, що дефект не є таким, він змінює статус на «відхилений»;

Відкладено (Deferred): якщо поточний баг не є пріоритетним і очікується, що він буде виправлений у наступному випуску, таким багам надається статус «Відкладено»;

Не є багом (Not a bug): якщо це не впливає на функціональність програми, то багу надається статус «Не є багом».



Витік дефектів та реліз бага (Bug Leakage & Bug Release)

Витік бага (Bug Leakage): виникає коли пропускається баг у білді , який вийшов у Production. Якщо баг був виявлений кінцевим користувачем або замовником, ми називаємо це витоком помилок.

Випуск бага (Bug release): Випуск програмного забезпечення в Production з деякими відомими багами. Ці відомі баги слід включити до приміток до випуску (release notes). Інший варіант - передача програмного забезпечення групі тестування з деякими відомими багами, серйозність та пріоритет яких невисокі. Ці помилки можна виправити перед випуском у Production.

Основна відмінність налагодження від тестування (Debugging Vs. Testing)

Після того, як розробник отримав баг-репорт, він починає виправляти баг. Але, перш ніж помилку виправити, потрібно відтворити її, зрозуміти, як вона відбувається і де її знайти в коді. Дебаг , буквально " de " + " bug " - це і є процес пошуку та усунення помилок у коді. Спеціальна debug- версія білда програми може мати розширений висновок для більш інформативних логів або будь-які інші модифікації для спрощення розуміння проблеми. Тактика налагодження може включати інтерактивне налагодження, аналіз потоку управління, модульне тестування, інтеграційне тестування, аналіз логів, моніторинг на рівні програми або системи, дампи пам'яті та профільування. Багато мов програмування та інструменти розробки програмного забезпечення також пропонують програми для допомоги у налагодженні, відомі як налагоджувачі/ дебагери.

Маскування дефектів (Defect masking)

Маскування дефектів (defect masking): Випадок, коли один дефект перешкоджає знаходженню іншого. (IEEE 610).

Прихований дефект (Latent defect)

Дефект, який є існуючим дефектом у системі, але ще не викликав збоїв, оскільки відповідний набір вхідних даних для його прояву не введено або його прояву заважає інший дефект (Defect masking).

Сортування дефектів (Bug triage)

пріоритету дефектів залежно від їх severity , ризику, повторюваності і т.д. Defect Triage Meeting. Така зустріч корисна в умовах обмежених ресурсів, коли потрібно розібратися з безліччю помилок і тим, які пріоритетні.

Поняття сортування прийшло з медицини, де це процес швидкого обстеження пацієнтів, доставлених до лікарні, щоб вирішити, які з них найбільш серйозно хворі та потребують лікування насамперед. У тестуванні ми використовуємо ту саму концепцію помилок, виявлених на етапі тестування.

Підсів недоліків (fault seeding)

Процес навмисного внесення дефектів на додаток до тих, що вже існують в компоненті або в системі, з метою відстеження рівня виявлення та усунення, а також оцінювання кількості дефектів, що залишилися в системі. Підсів недоліків зазвичай є частиною процесу тестування розробки та може застосовуватися будь-якому рівні тестування (компонентному, інтеграційному чи системному).

Контрольні запитання

1. Дайте визначення поняттю «дефект».
2. Дайте визначення поняттю «помилка».
3. Дайте визначення поняттю «дефекти функціонального опису».
4. Дайте визначення поняттю «дефекти функцій».
5. Дайте визначення поняттю «дефекти опису інтерфейсів».
6. Дайте визначення поняттю «дефекти дизайну».