Міністерство освіти та науки України

Черкаський державний бізнес-коледж

Кафедра інженерії та програмного забезпечення

РЕФЕРАТ

На тему: Методи тестування програмного забезпечення

З предмета: Алгоритми мови

Виконали:

Студенти 1 та 2 курсів

Групи 3П-20 та 2П-19

Крутько Артем Олександрович та

Снесар Сергій Михайлович

Перевірив:

Марченко С.В.

Черкаси - 2020

ЗМІСТ

Вступ

1. Тестування ПЗ

1.1.Класифікація видів тестування

2. Види і рівні тестування

2.1. Види тестування

2.2. Рівні тестування

3. Методи білого, чорного та сірого ящика

3.1 Метод «білого ящика»

3.2 Метод «чорного ящика»

3.3 Метод «сірого ящика»

Висновок

Вступ

Історія тестування програмного забезпечення відображає еволюцію розробки самого програмного забезпечення. Протягом тривалого часу розробка програмного забезпечення приділяла основну увагу великомасштабним науковим програмам, а також програмам міністерства оборони, пов'язаним з системами корпоративних баз даних, які проектувалися на базі універсальної ЕОМ або мінікомп'ютера. Тестові сценарії записувалися на папір. З їх допомогою перевірялися цільові потоки управління, обчислення складних алгоритмів та маніпулювання даними. Остаточний набір тестових процедур міг ефективно протестувати всю систему повністю. Тестування зазвичай починалося лише після завершення плану-графіка проекту і виконувалося тим же персоналом.

«Тестування - процес, що підтверджує правильність програми і демонструє, що помилок у програмі немає.» Основний недолік подібного визначення полягає в тому, що воно абсолютно неправильно; фактично це майже визначення антоніма слова «тестування». Люди з деяким досвідом програмування вже, ймовірно, розуміє, що неможливо продемонструвати відсутність помилок у програмі. Тому визначення описує нездійсненне завдання, а так як тестування найчастіше все ж виконується з успіхом, принаймні з деяким успіхом, то таке визначення логічно некоректно. Правильне визначення тестування таке: Тестування - процес виконання програми з наміром знайти помилки.

Неможливо гарантувати відсутність помилок в нетривіальною програмі; в кращому випадку можна спробувати показати наявність помилок. Якщо програма правильно поводиться для солідного набору тестів, немає підставі стверджувати, що в ній немає помилок; з усією визначеністю можна лише стверджувати, що не відомо, коли ця програма не працює. Звичайно  якщо є причини вважати даний набір тестів здатним з великою ймовірністю виявити всі можливі помилки, то можна говорити про деяке рівні впевненості у правильності програми, що встановлюється цими тестами.

Більшість людей, поставивши мету (наприклад, показати, що помилок немає), орієнтується у своїй діяльності на досягнення цієї мети. Тестувальник підсвідомо не дозволить собі діяти проти мети, т. Е. Підготувати тест, який виявив би одну з решти в програмі помилок. Оскільки ми всі визнаємо, що досконалість у проектуванні та кодуванні будь-якої програми недосяжно і тому кожна програма містить деяку кількість помилок, найбільш плідним застосуванням тестування буде знайти деякі з них. Якщо ми хочемо добитися цього і уникнути психологічного бар'єру, що заважає нам діяти проти поставленої мети, наша мета повинна полягати в тому, щоб знайти якомога більше помилок.

Поява персональних комп'ютерів сприяло стандартизації цієї галузі, оскільки програми стали спочатку створюватися для роботи із загальною операційною системою. Впровадження персональних комп'ютерів відкрило нову еру і призвело до швидкого і бурхливому зростанню комерційних розробок. Комерційні програми жорстко боролися за першість і виживання. Користувачі комп'ютерів брали вижило програмне забезпечення як стандарти defacto. Пакетна обробка замінялася системами, що працюють в реальному часі.

Тестування систем реального часу зажадало іншого підходу до проектування тестування через те, що робочі потоки могли викликатися в будь-якому порядку. Ця особливість призвела до появи величезної кількості процедур тестування, здатних підтримати нескінченне число перестановок і сполучень.

1.Тестування ПЗ

1.1.Класифікація видів тестування

Тестування ПЗ - це процес його дослідження з метою отримання інформації про якість. Метою тестування є виявлення дефектів у ПЗ. За допомогою тестування не можна довести відсутність дефектів і коректність функціонування аналізованої програми. Тестування складних програмних продуктів є творчим процесом, що не сводящимся до слідування строгим і чітким процедурам.

Тестування програмного забезпечення охоплює цілий ряд видів діяльності, дуже аналогічний послідовності процесів розробки програмного забезпечення. Сюди входять постановка задачі для тесту, проектування, написання тестів, тестування тестів і, нарешті, виконання тестів і вивчення результатів тестування. Вирішальну роль відіграє проектування тесту.

Склад і зміст документації, супутньої процесу тестування, визначається закордонним стандартом IEEE 829-2008 Standard for Software Test Documentation.

Існує кілька підстав, за якими прийнято виробляти класифікацію видів тестування.

. По об'єкту тестування

· Функціональне тестування (functional testing)

· Навантажувальне тестування (performance/load/stress testing)

· Тестування зручності використання (usability testing)

· Тестування інтерфейсу користувача (UI testing)

· Тестування безпеки (security testing)

· Тестування локалізації (localization testing)

· Тестування сумісності (compatibility testing)

. За знанням про тестованої системі

· Тестування методом «чорного ящика» (black box)

· Тестування методом «білого ящика» (white box)

1

· Тестування методом «сірого ящика» (grey box)

· За рівнем автоматизації

· Ручне тестування (manual testing)

· Автоматизоване тестування (automated testing)

. За ступенем ізольованості

. Модульне тестування (unit testing)

· Інтеграційне тестування (integration testing)

· Системне тестування (system testing)

. За рівнем готовності

· Альфа-тестування (alpha testing)

· Бета-тестування (beta testing)

· Приймально-здавальні випробування (acceptance testing)

1.2. Функціональне тестування та тестування якості

Функціональне тестування проводиться для перевірки виконання системою функціональних вимог.

Навантажувальне тестування проводиться для аналізу роботи системи при різних рівнях навантаження (великі обсяги даних або кількість користувачів). За допомогою навантажувального тестування можна експериментально визначити вимоги до ресурсів, масштабованість і надійність створеної системи. З погляду замовника системи, тестування навантаження є одним із способів перевірки роботи системи в умовах, наближених до реальних.

Основними показниками продуктивності інформаційної системи, вимірюваними в ході навантажувального тестування, є:

· Час відгуку (час виконання операції)

· Число операцій, які виконуються в одиницю часу (наприклад, transactions per second, TPS).

2

Основним результатом навантажувального тестування є вимірювання продуктивності інформаційної системи, які можуть бути використані для локалізації вузьких місць і подальшої оптимізації. У процесі навантажувального тестування може бути побудована «крива деградації» - графік, що показує залежність продуктивності системи (наприклад, в одиницях часу відгуку) від робочого навантаження (наприклад, від числа віртуальних користувачів).

Стресовий (stress) тестування проводиться в умовах недостатніх системних ресурсів і дозволяє оцінити рівень надійності роботи системи під навантаженням.

Тестування зручності використання має на меті оцінити прийнятність користувальницького інтерфейсу додатку (час, витрачений на досягнення мети, отриманий результат, легкість доступу до потрібної інформации, інтерпретація відповідей системи і т.д.)

Щоб охопити всі аспекти зручності використання, поряд з фахівцями щодо забезпечення якості в цьому вигляді тестування можуть брати участь фахівці з маркетингу і психологи, а також майбутні користувачі системи. У процесі тестування користувачам під керівництвом модератора пропонується вирішити з використанням системи ряд завдань. Для подальшого аналізу результатів тестування необхідна чітка фіксація всіх подій, що відбуваються: використання клавіатури і миші, вираз обличчя респондента, переходи між екранами, мова модератора і респондента.

Тестування інтерфейсу користувача (UI testing) передбачає перевірку відповідності ПЗ вимогам до графічного інтерфейсу користувача. Розрізняють такі види тестування графічного інтерфейсу користувача:

· Тестування на відповідність стандартам графічних інтерфейсів;

· Тестування з різними дозволами екрану;

· Тестування локалізованих версій: перевірка довжини назв елементів інтерфейсу і т.п .;

· Тестування графічного інтерфейсу користувача на різних цільових пристроях (для мобільних додатків, можливо з використанням емуляторів).

У ході тестування безпеки (security testing) проводиться оцінка уразливості системи по відношенню до атак. Тестування безпеки перевіряє фактичну реакцію захисних механізмів, вбудованих в систему, на спроби їх злому і обходу.

3

У ході тестування безпеки випробувач грає роль потенційного порушника і намагається перевірити наступні аспекти безпеки системи:

· Тестування механізмів контролю доступу - допомагає виявити дефекти, в результаті яких користувачі можуть отримувати несанкціонований доступ до об'єктів і функцій програми;

· Тестування авторизації користувачів - виявляє дефекти, пов'язані з авторизацією окремих користувачів і г8рупп користувачів і з перевіркою їх автентичності;

· Тестування процедур перевірки коректності введення - має на меті виявлення помилок в процедурах перевірки даних, що надходять в систему ззовні;

· Тестування криптографічних механізмів захисту - використовується для виявлення дефектів, пов'язаних з шифруванням і розшифрування даних, використанням цифрових підписів і перевіркою цілісності даних;

· Тестування правильності обробки помилок - включає в себе перевірку таких аспектів, як вивід на екран фрагментів коду за ощибки, вплив помилок на роботу всього програми, аналіз помилок в коді їх обробки;

4

2.Види і рівні тестування

2.1. Види тестування

Виділяють три рівні тестування: модульне, інтеграційне і системне.

Модульне тестування - тестування, що має метою перевірити працездатність окремих модулів (функції або класу). Модульне тестування зазвичай виконується незалежно для кожного програмного модуля і є, мабуть, найбільш поширеним видом тестування, особливо для систем малих і середніх розмірів. В якості критерію повноти використовується відсоток покриття тестами ключових елементів модуля (оператори, гілки логічних умов і т.д.). Стандарт IEEE 1008-1987 визначає зміст фаз процесу модульного тестування.

Модульні тести перевіряють, що певні дії на модуль призводять до бажаного результату. Як правило. Модульні тести створюються з використанням методу «білого ящика». При наявності залежностей модуля, що тестується від інших модулів замість них використовуються так звані mock-об'єкти, що надають фіктивну реалізацію їх інтерфейсів. З використанням mock-об'єктів можуть бути протестовані такі аспекти функціонування, які неможливо перевірити з використанням реальних залежних модулів. Існують спеціальні бібліотеки (наприклад, Moq), що спрощують завдання створення mock-об'єктів. У роботі (Месарош Дж. «Шаблони тестування xUnit. Рефакторинг коду тестів») описуються найбільш вдалі підходи до організації модульного тестування.

Модульне тестування дозволяє програмістам безпечно проводити рефакторинг, будучи впевненими, що змінений модуль раніше працює коректно. Модульні тести можуть використовуватися в якості специфікації, так як при тестуванні фактично перевіряється їх очікуване поведінка.

Для більшості популярних мов програмування високого рівня існують інструменти і бібліотеки модульного тестування (наприклад, інструменти сімейства xUnit: NUnit, JUnit, CppUnit).

Інтеграційне тестування (integration testing) - одна з фаз тестування ПЗ, при якому окремі програмні модулі об'єднуються і тестуються в комплексі. Зазвичай інтеграційне тестування проводиться після модульного тестування і передує системному тестуванню. Метою даного виду тестування є знаходження проблем взаємодії модулів взаємодії модулів (компонент, підсистем). При наявності резерву часу на дане стадії тестування ведеться ітераційно, з поступовим підключенням подальших підсистем.

5

Тестування виконується через інтерфейс модулів з використанням методу «чорного ящика».

Існує декілька стратегій проведення інтеграційного тестування, які грунтуються на знаннях про архітектуру системи.

Монолітне тестування припускає, що окремі компоненти системи серйозного тестування не проходили. Система перевіряється вся в цілому після розробки всіх модулів. Цей підхід не слід плутати з системним тестуванням. Незважаючи на те що при монолітному тестуванні перевіряється робота всієї системи в цілому, основне завдання цього тестування - визначити проблеми взаємодії окремих модулів системи. Основні недоліки монолітного тестування полягають в складності виявлення джерел помилок, а також в складності автоматизації даного виду тестування. На практиці найчастіше в різних частинах проекту застосовуються всі розглянуті в попередньому розділі методи в сукупності. Кожен модуль тестують у міру готовності окремо, а потім включають у вже готову композицію. Для одних частин тестування виходить низхідним, для інших - висхідним.

Раніше розглядався підхід до організації інтеграційної тестування на основі застосування систем безперервної інтеграції, використання яких дозволяє швидко виявляти проблеми взаємодії модулів.

Системне тестування - це тестування повною, інтегрованої системи з метою перевірки її відповідності системним вимогам і показникам якості.

2.2. Рівні тестування

· Приймальне тестування (Acceptance/qualification testing).

Перевіряє поведінку системи на предмет задоволення вимог замовника. Це можливо в тому випадку, якщо замовник бере на себе відповідальність, пов'язану з проведенням таких робіт, як сторона «приймаюча» програмну систему, або спеціфіровани типові завдання, успішна перевірка (тестування) яких дозволяє говорити про задоволення вимог замовника.

Такі тести можуть проводитися як із залученням розробників системи, так і без них.

· Установче тестування (Installation testing).

З назви випливає, що дані тести проводяться з метою перевірки процедури інсталяції системи в цільовому оточенні.

6

· Альфа- і бета-тестування (Alpha and Beta testing).

Перед тим, як випускається програмне забезпечення, як мінімум, воно повинно проходити стадії альфа (внутрішнє пробне використання) і бета (пробне використання із залученням відібраних зовнішніх користувачів) версій. Звіти про помилки, що надходять від користувачів цих версій продукту, обробляються відповідно до певними процедурами, що включають підтверджуючі тести (будь-якого рівня), що проводяться фахівцями групи розробки. Даний вид тестування не може бути заздалегідь спланований.

Альфа-тестування - використання незавершеною (альфа) версії ПЗ, в якій реалізована не вся функціональність, запланована для даної версії продукту. При альфа-тестуванні проводиться імітація реальної роботи з системою (виконується штатними розробниками) або реальна робота з системою потенційних користувачів (замовника) з метою виявлення помилок у роботі реалізованих модулів і функцій для їх подальшого усунення перед бета-тестуванням. Найчастіше альфа-тестування проводиться на ранній стадії розробки продукту, але в деяких випадках може застосовуватися для закінченого продукту в якості внутрішнього приймального тестування. Іноді альфа-тестування виконується під відладчиком або з використанням оточення, яке допомагає швидко виявляти знайдені помилки. Виявлені помилки можуть бути передані тестувальникам для додаткового дослідження.

Бета-тестування - інтенсивне використання майже готової версії ПЗ з метою виявлення максимального числа помилок в його роботі для їх подальшого усунення перед остаточним виходом (релізом) продукту на ринок, до масового споживача. У деяких випадках виконується поширення версії з обмеженнями (по функціональності або часу роботи) для деякої групи осіб, з тим, щоб переконатися, що продукт містить достатню мала кількість помилок. Іноді бета-тестування виконується для того, щоб отримати зворотній зв'язок про продукт від його майбутніх користувачів (їх називають бета-тестерами). Бета-версія не є фінальною версією продукту, тому розробник не гарантує повної відсутності помилок, які можуть порушити роботу комп'ютера і привести до втрати даних.

При розробці замовленого ПЗ фазу альфа- і бета-тестування замінюють проектні випробування. Під час цих випробувань замовник засвідчується, що система працює відповідно до його потребами (як зафіксованими в технічному завданні на систему, так і не зафіксованими). Замовник може проводити такі випробування самостійно, виконуючи заздалегідь підготовлені тести системи, або проводити їх спільно з представниками виконавця.

7

У цьому випадку тестові приклади також готуються розробниками, наприклад на основі тестових прикладів, що використовувалися на етапі системного тестування.

Завершуються проектні випробування або підписанням акта приймання, або видачею замовником додаткових вимог до системи, які повинні бути виправлені до приймання системи. Після усунення всіх недоліків системи проектні випробування повторюються (можливо, за скороченою програмою). Після успішного підписання акта система надходить в експлуатацію замовнику.

· Функціональні тести/тести відповідності (Conformance testing/Functional testing/Correctness testing)

Ці тести можуть називатися по різному, проте, їх суть проста - перевірка відповідності системи, висунутим до неї вимогам, описаним на рівні специфікації поведінкових характеристик.

Досягнення та оцінка надійності (Reability achievement and evaluation)

· Регресійне тестування (Regression testing)

Визначення успішності регресійних тестів (IEEE 610-90 Standart Glossary of Software Engineering Terminology) говорить: «повторне вибіркове тестування системи або компонент для перевірки зроблених модифікацій не повинно призводити до непередбачуваних ефектів». На практиці це означає, що якщо система успішно проходила тести до внесення модифікацій, вона повинна їх проходити і після внесення таких. Основна проблема регресійного тестування полягає в пошуку компромісу між наявними ресурсами і необхідністю проведення таких тестів в міру внесення кожної зміни. Певною мірою, завдання полягає в тому, щоб визначити критерії «масштабів» змін, з досягненням яких необхідно проводити регресійні тести.

· Тестування продуктивності (Perfomance testing)

Спеціалізовані тести перевірки задоволення специфічних вимог, що пред'являються до параметрів продуктивності. Існує особливий підвид таких тестів, коли робиться спроба досягнення кількісних меж, обумовлених характеристиками самої системи та її операційного оточення.

· Навантажувальне тестування (Stress testing)

8

Необхідно розуміти відмінності між розглянутим вище тестуванням продуктивності з метою досягнення її реальних (досяжних) можливостей продуктивності та виконанням програмної системи з підвищенням навантаження, аж до досягнення запланованих характеристик і далі, з відстеженням поведінки на всьому протязі підвищення завантаження системи.

· Порівняльне тестування (Back-to-back testing)

Одиничний набір тестів, що дозволяють порівняти дві версії системи.

· Відновлювальні тести (Recovery testing)

Мета - перевірка можливостей рестарту системи у випадку непередбачуваної катастрофи (disaster), що впливає на функціонування операційного середовища, в якій виконується система.

· Конфігураційне тестування (Configuration testing)

У випадках, якщо програмне забезпечення створюється для використання різними користувачами (в термінах «ролей»), даний вид тестування спрямований на перевірку поведінки і працездатності системи в різних конфігураціях.

· Тестування зручності та простоти використання (Usability testing)

Мета - перевірити, наскільки легко кінцевий користувач системи може її освоїти, включаючи не тільки функціональну складову - саму систему, але і її документацію; наскільки ефективно користувач може виконувати завдання, автоматизація яких здійснюватися з використанням даної системи; нарешті, наскільки добре система застрахована (з погляду потенційних збоїв) від помилок користувача.

· Розробка, керована тестуванням (Test-driven development)

По суті, це не стільки техніка тестування, скільки стиль організації процесу розробки, життєвого циклу, коли тести є невід'ємною частиною вимог (і відповідних специфікацій) замість того, щоб розглядатися незалежної діяльністю з перевірки задоволення вимог програмною системою.

9

3. Методи білого, чорного та сірого ящика.

3.1 Метод «білого ящика».

При тестуванні методом «білого ящика» розробник тесту має доступ до вихідного тестируемому кодом і може компонувати з ним код тестів. Така ситуація типова для модульного тестування, при якому тестуються тільки окремі частини системи. При тестуванні методом «білого ящика» може використовуватися знання про внутрішній устрій перевіряється ПЗ, у тому числі і для організації перевірки обробки допустимих, граничних і некоректних даних. Крім того, даний вид тестування дозволяє оцінити рівень покриття коду тестами.

Термін білий ящик означає, що при розробці тестових випадків тестувальники використовують будь-які доступні відомості про внутрішню структуру чи коді. Технології, застосовувані під час тестування білого ящика raquo ;, зазвичай називають технологіями статичного тестування.

Цей метод не ставить мети виявлення синтаксичних помилок, так як дефекти такого роду звичайно виявляє компілятор. Методи білого ящика спрямовані на локалізацію помилок, які складніше виявити, знайти і зафіксувати. З їх допомогою можна виявити логічні помилки і перевірити ступінь покриття тестами.

Тестові процедури, пов'язані з використанням стратегії білого ящика, використовують керуючу логіку процедур. Вони надають ряд послуг, в тому числі:

· Дають гарантію того, що всі незалежні шляхи в модулі перевірені принаймні один раз.

· Перевіряють всі логічні рішення на предмет того, істини вони чи хибні.

· Виконують всі цикли всередині операційних кордонів і з використанням граничних значень.

· Досліджують структури внутрішніх даних із цілі перевірки їх достовірності.

Тестування допомогою білого ящика, як правило, включає в себе стратегію модульного тестування, при якому тестування ведеться на модульному або функціональному рівні та роботи з тестування направлені на дослідження внутрішнього устрою модуля.

10

Даний тип тестування називають також модульним тестуванням, тестуванням прозорого ящика (clear box) або прозорим (translucent) тестуванням, оскільки співробітники, які проводять тестування, мають доступ до програмного коду і можуть бачити роботу програми зсередини. Даний підхід до тестування відомий також як структурний підхід.

На цьому рівні тестування перевіряється керуюча логіка, що виявляється на модульному рівні. Тестові драйвери використовуються для того, щоб всі шляхи в даному модулі були перевірені хоча б один раз, всі логічні рішення розглянуті у всіляких умовах, цикли були виконані з використанням верхніх і нижніх меж і роконтроліровани структури внутрішніх даних.

Методи тестування на основі стратегії білого ящика:

· Ввід невірних значень. При введенні невірних значень тестувальник змушує коди повернення показувати помилки і дивиться на реакцію коду. Це хороший спосіб моделювання певних подій, наприклад переповнення диска, нестачі пам'яті й т.д. Популярним методом є заміна alloc () функцією, яка повертає значення NULL в 10% випадків з метою з'ясування, скільки збоїв буде в результаті. Такий підхід ще називають тестуванням помилкових вхідних даних. При та...

кому тестуванні перевіряється обробка як вірних, так і невірних вхідних даних. Тестувальники можуть вибрати, які значення перевіряють діапазон вхідних/вихідних параметрів, а також значення, що виходять за межу діапазону.

· Модульне тестування. При створенні коду кожного модуля програмного продукту проводиться модульне тестування для перевірки того, що код працює вірно і коректно реалізує архітектуру. При модульному тестуванні новий код перевіряється на відповідність докладного опису архітектури; обстежуються шляху в коді, встановлюється, що екрани, спадаючі меню і повідомлення належним чином відформатовані; перевіряються діапазон і тип даних, що вводяться, а також те, що кожен блок коду, коли потрібно, генерує виключення і повертає помилки (Еггог returns). Тестування кожного модуля програмного продукту проводиться для того, щоб перевірити коректність алгоритмів і логіки і те, що програмний модуль задовольняє пропонованим вимогам і забезпечує необхідну функціональність. За підсумками модульного тестування фіксуються помилки, що відносяться до логіки програми, перевантаження і виходу з діапазону, часу роботи і витоку пам'яті.

· Тестування обробки помилок. При використанні цього методу зізнається, що нереально на практиці перевірити кожне можливе умова виникнення помилки

11

З цієї причини програма обробки помилок може згладити наслідки при виникненні несподіваних помилок.

Тестувальник зобов'язаний переконатися в тому, що додаток належним чином видає повідомлення про помилку. Так, додаток, що повідомляє про системну помилку, що виникла через проміжного програмного забезпечення представляє невелику цінність, як для кінцевого користувача, так і для тестувальника.

· Витік пам'яті. При тестуванні витоку пам'яті додаток досліджується з метою виявлення ситуацій, при яких додаток не звільняє виділену пам'ять, внаслідок чого знижується продуктивність або виникає тупикова ситуація. Дана технологія застосовується як для тестування версії додатка, так і для тестування готового програмного продукту. Можливе застосування інструментів тестування. Вони можуть стежити за використанням пам'яті програми протягом декількох годин або навіть днів, щоб перевірити, чи буде зростати обсяг використовуваної пам'яті. З їх допомогою можна також виявити ті оператори програми, які не звільняють виділену пам'ять.

· Комплексне тестування. Метою комплексного тестування є перевірка того, що кожен модуль програмного продукту коректно узгоджується з іншими модулями продукту. При комплексному тестуванні може використовуватися технологія обробки зверху вниз і знизу вгору, при якій кожен модуль, що є листом в дереві системи, інтегрується з наступним модулем нижчого або більш високого рівня, поки не буде створено дерево програмного продукту. Ця технологія тестування спрямована на перевірку не тільки тих параметрів, які передаються між двома компонентами, але й на перевірку глобальних параметрів і у випадку об'єктно-орієнтованого додатки, всіх класів верхнього рівня.

· Тестування ланцюжків. Тестування ланцюжків передбачає перевірку групи модулів, що складають функцію програмного продукту. Ці дії відомі ще як модульне тестування, з його допомогою забезпечується адекватне тестування компонентів системи. Дане тестування виявляє, чи достатньо надійно працюють модулі для того, щоб утворити єдиний модуль, і чи видає модуль програмного продукту точні і що погодяться.

· Дослідження покриття. При виборі інструмента для дослідження покриття важливо, щоб група тестування проаналізувала тип покриття, необхідний для програми. Дослідження покриття можна провести за допомогою різних технологій. Метод покриття операторів часто називають С1, що також означає покриття вузлів. Ці виміри показують, чи був перевірений кожен виконуваний оператор. Даний метод тестування зазвичай використовує програму протоколювання (profiler) продуктивності.

12

· Покриття рішень. Метод покриття рішень спрямований на визначення (у відсотковому співвідношенні) всіх можливих результатів рішень, які були перевірені за допомогою комплекту тестових процедур. Метод покриття рішень іноді відносять до покриттю гілок і називають С2. Він вимагає: щоб кожна точка входу і виходу в програмі була досягнута хоча б один раз, щоб всі можливі умови для рішень в програмі були перевірені не менше одного разу і щоб кожне рішення в програмі хоча б раз було протестовано при використанні всіх можливих результатів.

· Покриття умов. Покриття умов схоже на покриття рішень. Воно спрямоване на перевірку точності істинних або помилкових результатів кожного логічного виразу. Цей метод включає в себе тести, які перевіряють вираження незалежно один від одного. Результати цих перевірок аналогічні тим, що одержують при застосуванні методу покриття рішень, за винятком того, що метод пок риття рішень більш чутливий до керуючої логіці програми.

5.2 Тестування методом «чорного ящика»

При тестуванні методом «чорного ящика» тестувальник має доступ до ПЗ тільки через ті ж інтерфейси, що і замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншого комп'ютера або іншому процесу підключитися до системи для тестування. Як правило, тестування «чорного ящика» ведеться з використанням специфікацій чи інших документів, що описують вимоги до системи. У даному вигляді тестування намагаються забезпечити покриття вимог і вхідних даних.

Тестування на основі стратегії чорного ящика можливе лише за наявності встановлених відкритих інтерфейсів, таких як інтерфейс користувача або програмний інтерфейс додатку (API). Якщо тестування на основі стратегії білого ящика досліджує внутрішню роботу програми, то методи тестування чорного ящика порівнюють поведінку додатки з відповідними вимогами. Крім того, ці методи зазвичай спрямовані на виявлення трьох основних видів помилок: функціональності, підтримуваної програмним продуктом; вироблених обчислень; допустимого діапазону або області дії значень даних, які можуть бути опрацьовані програмним продуктом. На цьому рівні тестувальники не досліджують внутрішню роботу компонентів програмного продукту, тим не менше вони перевіряються неявно.

13

Група тестування вивчає вхідні і вихідні дані програмного продукту. У цьому ракурсі тестування за допомогою методів чорного ящика розглядається як синонім

тестування на рівні системи, хоча методи чорного ящика можуть також застосовуватися під час модульного або компонентного тестування.

При тестуванні методами чорного ящика важлива участь користувачів, оскільки саме вони найкраще знають, яких результатів слід очікувати від бізнес-функцій. Ключем до успішного завершення системного тестування є коректність даних. Тому на фазі створення даних для тестування вкрай важливо, щоб кінцеві користувачі надали якомога більше вхідних даних.

Тестування за допомогою методів чорного ящика спрямоване на одержання множин вхідних даних, які найбільш повно перевіряють всі функціональні вимоги системи. Це не альтернатива тестуванню за методом білого ящика. Цей тип тестування націлений на пошук помилок, які відносяться до цілого ряду категорій, серед них:

v Невірна або пропущена функціональність

v Помилки інтерфейсу

v Проблеми зручності використання

v Методи тестування на основі Автоматизовані інструменти

v Помилки в структурах даних або помилки доступу до зовнішніх баз даних

v Проблеми зниження продуктивності та інші помилки продуктивності

v Помилки завантаження

v Помилки багатокористувацького доступу

v Помилки ініціалізації та завершення

v Проблеми збереження резервних копій і здатності до відновлення роботи

v Проблеми безпеки

v Методи тестування на основі стратегії чорного ящика

Рішення цих помилок може бути знайдено за допомогою таких методів тестування:

14

Еквівалентна розбиття. Повне тестування вхідних даних, як правило, нездійсненно. Тому слід проводити тестування з використанням підмножини вхідних даних.

При тестуванні помилок, пов'язаних з виходом за межі області допустимих значень, застосовують три основних типи еквівалентних класів: значення всередині межі діапазону, за межею діапазону і на кордоні. Виправдовує себе практика створення тестових процедур, які перевіряють граничні випадки плюс/мінус один щоб уникнути пропуску помилок «на одиницю більше» або «на одиницю менше». Крім розроблення тестових процедур, що використовують сильно структуровані класи еквівалентності, група тестування повинна провести дослідницьке тестування. Тестові процедури, при виконанні яких видаються очікувані результати, називаються правильними тестами. Тестові процедури, проведення яких має призвести до помилки, носять назву неправильних тестів.

? Аналіз граничних значень. Аналіз граничних значень можна застосувати як на структурному, так і на функціональному рівні тестування. Межі визначають дані трьох типів: правильні, неправильні і лежать на границі. Тестування кордонів використовує значення, що лежать всередині або на кордоні (наприклад, крайні точки), і максимальні/мінімальні значення (наприклад, довжини полів). При такому дослідженні завжди повинні враховуватися значення на одиницю більше і менше граничного. При тестуванні за межами кордону використовується репрезентативний зразок даних, що виходять за межу, тобто невірні значення.

Діаграми при чинно-наслідкових зв'язків. Складання діаграм причинно-наслідкових зв'язків - це метод, що дає чітке уявлення про логічних умовах і відповідних діях. Метод передбачає чотири етапи. Перший етап полягає в складанні переліку причин (умов введення) і наслідків (дій) для модуля й у присвоєнні ідентифікатора кожному модулю. На другому етапі розробляється діаграма причинно-наслідкових зв'язків. На третьому етапі діаграма перетворюється в таблицю рішень. Четвертий етап включає в себе встановлення причин і наслідків в процесі читання специфікації функцій. Кожній причини і слідству присвоюється власний ідентифікатор. Причини перераховуються в стовпчику з лівого боку аркуша паперу, а слідства - з правого. Потім причини і наслідки з'єднуються лініями так, щоб були відображені наявні між ними відповідності. На діаграмі проставляються булеві вирази, які об'єднують дві або більше причин, пов'язаних зі слідством. Далі правила таблиці рішень перетворюються в тестові процедури.

15

Системне тестування. Термін «системне тестування» часто вживається до?? до синонім «тестування за допомогою методів чорного ящика», оскільки під час системного тестування група тестування розглядає в основному «зовнішня поведінка» додатка.

Системне тестування включає в себе кілька підтипів тестування, у тому числі функціональне, регресійне, безпеки, перевантажень, продуктивності, зручності використання, випадкове, цілісності даних, перетворення даних, збереження резервних копій і здатності до відновлення, готовності до роботи, приймально-здавальні випробування і альфа/бета тестування.

Функціональне тестування. Функціональне тестування перевіряє системне додаток відносно функціональних вимог з метою виявлення невідповідності вимогам кінцевого користувача. Для більшості програм тестування програмного продукту даний метод тестування є головним. Його основне завдання - оцінка того, чи працює додаток відповідно до ставляться.

Регресійне тестування. Сенс проведення тестування полягає у виявленні дефектів, їх документуванні та відстеженні аж до усунення. Тестувальник повинен бути впевнений в тому, що заходи, прийняті для усунення знайдених помилок, які не породять у свою чергу нових помилок в інших областях системи. Регресійне тестування дозволяє з'ясувати, чи не з'явилися які-небудь помилки в результаті ліквідації вже виявлених помилок. Саме для регресійного тестування застосування інструментів автоматизованого тестування дає найбільшу віддачу. Всі створені раніше скрипти можна використовувати знову для підтвердження того, що в результаті змін, внесених при усуненні помилки, чи не з'явилися нові дефекти. Ця мета легко досяжна, оскільки скрипти можна виконувати без ручного втручання і використовувати стільки разів, скільки необхідно для виявлення помилок.

? Тестування безпеки. Тестування безпеки включає в себе перевірку роботи механізмів доступу до системи і до даних. Для цього придумують тестові процедури, які намагаються подолати захист системи. Тестувальник перевіряє ступінь безпеки й обмеження доступу, виявляючи таким чином відповідність встановленим вимогам до безпеки і всім застосовуваним правилам з безпеки системи.

? Тестування перевантажень. При тестуванні перевантажень виконується перевірка системи без врахування обмежень архітектури з цілі виявлення технічних обмежень системи. Ці тести проводяться на піку обробки транзакцій і при безперервному завантаженні великого обсягу даних. Тестування перевантажень вимірює пропускну спроможність системи та її еластичність (resiliency) на всіх апаратних платформах.

16

Цей метод передбачає одночасне звернення з боку багатьох користувачів до певних функцій системи, причому деякі вводять значення, що виходять за межі норми. Від системи потрібна обробка величезної кількості даних або виконання великого числа функціональних запитів протягом короткого періоду часу.

? Тестування продуктивності. Тести продуктивності перевіряють, чи задовольняє системне додаток вимогам по продуктивності. Застосовуючи тестування продуктивності, можна заміряти і скласти звіти за такими показниками, як швидкість передачі вхідних і вихідних даних, загальне число дій по введенню і виведенню даних, середній час, що витрачається базою даних на відгук на запит, і інтенсивність використання центрального процесора. Як правило, для автоматичної перевірки ступеня продуктивності, проведеної в рамках тестування продуктивності, використовуються ті ж інструменти, що і при тестуванні перевантажень.

? Тестування зручності використання. Тести зручності використання спрямовані на підтвердження простоти застосування системи і того, що користувальницький інтерфейс виглядає привабливо. Такі тести враховують людський фактор у роботі системи. Тестувальника потрібно оцінити програму з точки зору кінцевого користувача.

lign="justify">. 3 Тестування методом «сірого ящика»

При тестуванні методом «сірого ящика» розробник тесту має доступ до вихідного коду, але при безпосередньому виконанні тестів доступ до коду, як правило, не потрібно. Тестування проводиться так само, як і в методі «чорного ящика», однак для побудови тестів використовуються знання про внутрішній устрій програми.

17

Висновок

Тестова діяльність, що пов'язана з аналізом результатів розробки програмного забезпечення, називається статичним тестуванням. Воно передбачає перевірку програмних кодів, контроль та перевірку програми без запуску на комп'ютері. Тестова діяльність, що передбачає експлуатацію програмного продукту, називається динамічним тестуванням. Динамічне та статичне тестування доповнюють одне одного.

На етапі статичного тестування перевіряється вся документація, отримана як результат життєвого циклу програми. Це і технічне завдання, і специфікація, і вихідний текст програми на мові програмування. Вся документація аналізується на предмет дотримання стандартів програмування. У результаті статичної перевірки встановлюється, наскільки програма відповідає заданим критеріям та вимогам замовника. Усунення неточностей та помилок у документації — запорука того, що створюваний програмний засіб має високу якість.

Динамічні методи застосовуються в процесі безпосереднього виконання програми. Коректність програмного засобу перевіряється на безлічі тестів або наборів підготовлених вхідних даних. При прогоні кожного тесту збираються та аналізуються дані про відмови та збої в роботі програми.