Сергей, [12.12.20 18:26]

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.

Сергей, [12.12.20 18:54]

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).

Сергей, [12.12.20 18:54]

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 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, технічному завданні на систему, так і не зафіксованими). Замовник може 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.