МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

***Циклова комісія програмування***

**Реферат**

з курсу «Основи інформатики, програмування та алгоритмічні мови»

На тему: “Тестування програмного забезпечення”

Студента групи: **2П-19**

**Виконав:** Гапченко Володимир

Викладач: Марченко С.В

Черкаси-2020

Зміст:

1. Вступ

2. Види тестування

3. Автоматизоване тестування

4. Висновок

5. Література

1. 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 починається 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 невідомо, 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 знаємо, 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 замінилася 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.

Сформулюємо основоположний висновок:

- Якщо ваша мета - показати відсутність помилок, то ви їх знайдете не надто багато.

- Якщо ж ваша мета - показати наявність помилок, ви знайдете значну їх частину.

2. 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 специфіковані 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.

- 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. 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 (Reliability achievement and evaluation):

Допомагаючи ідентифікувати причини збоїв, тестування увазі і підвищення надійності програмних систем. Випадково генеруються сценарії тестування можуть застосовуватися для статичної оцінки надійності. Обидві цілі - підвищення і оцінка надійності - можуть досягатися при використанні моделей підвищення надійності.

- Регресійне тестування (Regression testing):

Визначення успішності регресійних тестів (IEEE 610-90 Standard 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 (Performance 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):

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 непередбаченої 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.

- 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.

3. Автоматизоване тестування використовує програмні засоби для виконання тестів і перевірки коректності результатів виконання, що спрощує тестування і скорочує його тривалість. Головна перевага автоматизованого тестування полягає в можливості повторного прогону тестів без участі людини.

Традиційний і найбільш популярний серед розробників спосіб полягає в організації автоматизації тестування на рівні коду. Даний підхід буде детально розглянуто при описі модульного тестування. Автоматизоване тестування на рівні коду часто критикують за неможливість тестування користувальницького інтерфейсу програми. Однак прихильники TDD показали, що при правильному використанні патернів сімейства MVC (Model-View-Controller) можливо організувати програмну імітацію дій користувача без використання GUI (Graphical User Interface). Цей підхід дозволяє організувати тестування обробників дій користувача, залишаючи не покритий тестами лише частину, що відноситься до безпосереднього відображенню даних.

Другий спосіб автоматизації тестування полягає в імітації дій користувача з використанням спеціальних інструментальних засобів (GUI-тестування). Даний вид тестування відноситься до тестування методом «чорного ящика».

Існують чотири покоління інструментів і технік, призначених для організації GUI-тестування:

- Утиліти запису і відтворення (capture/playback tools) записують дії під час ручного тестування. Надалі вони дозволяють відтворити раніше записані дії без участі людини, значно збільшуючи продуктивність і усуваючи повторення одноманітних дій. Основним недоліком інструментальних засобів даного покоління є те, що будь-яка зміна розташування візуальних елементів програми призводить до необхідності повторної записи ручних тестів.

- Сценарій (scripting) - форма автоматизації тестування з використанням спеціалізованих скриптових мов. Мова повинен підтримувати емуляцію дій користувача та отримання результатів дій. Розробкою тестів займаються програмісти, які працюють окремо від тестувальників, безпосередньо запускають те6сти. Зміни в тестованому ПО вимагають внесення виправлень і у відповідних скриптах.

3. Data-driven testing - методологія автоматизації тестування, заснована на використанні в скриптах параметрів виконання тестів. Параметри, що задають логіку роботи тестів (наприклад, вхідні значення та очікувані результати), знаходяться в деякому зовнішньому сховищі. Подібний підхід дозволяє організувати виконання сценаріїв з різними наборами вхідних параметрів і підвищити гнучкість тестування.

При використанні keyword-based testing створюється спеціалізований словник ключових слів, що описують системні події (наприклад, «Logon User»). З кожним ключовим словом пов'язані необхідні параметри (наприклад: «UserID», «Password») та очікувані результати. Для кожного ключового слова має бути задане опис. Даний підхід дозволяє писати функціональні тести практично на природній мові, не вимагаючи від тестувальника навичок програмування.

Автоматизовані тести, як правило, є регресійний (від лат. Regression - рух назад), тобто спрямованими на виявлення помилок у вже протестованих ділянках вихідного коду при внесенні змін.

Однією з головних проблем автоматизованого тестування є його трудомісткість: незважаючи на те, що воно дозволяє усунути частина рутинних операцій і прискорити виконання тестів, великі ресурси можуть витрачатися на оновлення самих тестів. Проте подібні інвестиції в більшості випадків виправдані, так як ручне тестування вимагає набагато більше ресурсів.

5. Висновок

Покриття коду є важливою метрикою для забезпечення якості тестованого програми, особливо якщо мова про проекти зі складною логікою і великим обсягом коду. Аналіз покриття коду виконується за допомогою спеціального інструментарію, який дозволяє простежити в які рядки, гілки і т.д. коду, були входження під час роботи Автотест. Найбільш відомі інструменти для проведення вимірювання покриття коду: AQTime, Bounds Checker, Bullseye Coverage, Coverage Meter, Clover, NCover, IBM Rational PurifyPlus, Intel Compiler, Intel Code Coverage Tool Prototype, JetBrains. За допомогою аналізу покриття коду можна оцінити густину покриття авто-тестами виконуваного коду тестованого програми (можна відповісти на питання який обсяг тестування ми (наші Автотест) виконуємо?). При детальному аналізі результатів покриття коду Автотест можна оцінити покриття окремих компонентів системи (тобто можна відповісти на питання: що і в якому обсязі ми тестуємо ?, в яких місцях потрібно оптимізувати покриття ?, які місця системи не перевіряються тестами? І т. д.). Таким чином, знаючи дану метрику, стане ясно для яких тестових випадків потрібно створити нові тести, або прибрати дублюючі тести. Дані заходи допоможуть збільшити значення метрики Code Coverage, що в свою чергу має підвищити якість коду і якість тестованого програми в цілому. Природно, чим вище показник даної метрики - тим краще, проте вже добре якщо у вас покриті тестами найбільш складні і важливі фрагменти коду.

6. Література:

1. Борзов Ю. В., Уртанс Г. Б., Шимаров В. А. Выбор путей программы

для построения тестов УСиМ. – 2013. – с. 29–36.

2. Брукс Ф. Мифический человеко-месяц или как создаются

программные системы СПб.: Символ-Плюс, 2015. – 304 с.

3. Канер С., Фолк Дж., Тестирование программного

обеспечения К: ДиаСофт, 2013 – 544 с.

4. Котляров В.П. Основы тестирования программного обеспечения

Интернет-университет информационных технологий; 2012.

5. Майерс Г., Баджетт Т., Сандлер К. Искусство тестирования

программ, 3-е издание. – М.: «Диалектика», 2012. – 272 с.

6. Макгрегор Дж, Сайкс Д. Тестирование объектно-ориентированного

программного обеспечения К: Диасофт, 2015. – 432