Управління проектами

ДЕНЬ 5

Розробка (конструювання, Construction)

ПРОДОВЖЕННЯ...

Кодування

Практика конструювання програмного забезпечення показує активне застосування таких міркувань і технік:

- техніки створення вихідного коду, який легко зрозуміти, на основі використання угод про іменування, форматування і структурування коду;
- використання класів, типів перерахування, змінних, іменованих констант та інших виразних сутностей;
- організація вихідного тексту (у вирази, шаблони, класи, пакети / модулі та інші структури)
- використання структур управління;
- обробка помилкових умов і виняткових ситуацій
- запобігання можливих проломів в безпеці (наприклад, переповнення буферу або вихід за межі індексу в масиві)
- використання ресурсів на основі застосування механізмів виключення (з розгляду) та порядку доступу до паралельно використовуваних ресурсів (наприклад, на основі блокування даних, використання потоків і їх синхронізації і т.п.)
- документування коду

Кодування.2

Інші техніки

- «захищене» кодування
- кодування за прикладами
- автоматична генерація коду

Повторне використання

Завдання, пов'язані з повторним використанням у процесі конструювання та тестування, включають:

- вибір одиниць (units), даних тестових процедур, даних, отриманих в результаті тестів, і самих тестів, що підлягають повторному використанню
- оцінку потенціалу повторного використання коду і тестів
- відстеження інформації та створення звітності по повторному використанню у новому коді, тестових процедурах і даних, отриманих в результаті тестів

Якість конструювання

Існує ряд технік, призначених для забезпечення якості коду, виконуваних під час його конструювання. Основні техніки забезпечення якості, що використовуються в процесі конструювання, включають:

модульне (unit) і інтеграційне (integration) тестування

розробка з первинністю тестів (test-driven development - тести пишуться до конструювання коду)

покрокове кодування (діяльність з конструювання коду розбивається на дрібні кроки, тільки після тестування результатів яких здійснюється перехід до наступного кроку кодування; відомий також як ітеративне кодування з тестуванням)

використання тверджень (assertion)

налагодження (в звичному розумінні - debugging)

огляди і оцінки (review)

Інтеграція

Одна з ключових активностей, здійснюваних у процесі конструювання, - інтеграція окремо сконструйованих операцій (процедур), класів, компонентів і підсистем (модулів)

До інтеграційних питань конструювання відносяться:

- планування послідовності, в якій інтегруються компоненти;
- забезпечення підтримки створення проміжних версій програмного забезпечення;
- завдання "глибини" тестування (зокрема, на основі критеріїв "прийнятної" якості) та інших робіт з забезпечення якості компонент, які інтегруються в подальшому;
- нарешті, визначення етапних точок проекту, коли будуть тестуватися проміжні версії програмної системи, що конструюється

Неперервна інтеграція

Неперервна інтеграція (CI – Continuous Integration) – це процес підтримки постійної готовності програмної системи у робочому стані

Забезпечується щоденними (або частіше) побудовами розроблюваної програмної системи у поточному стані

Тестування

Основні поняття

Тестування (software testing) - діяльність, виконувана для оцінки та вдосконалення програмного забезпечення

Ця діяльність, у загальному випадку, базується на виявленні дефектів і проблем у програмних системах.

Тестування програмних систем складається з динамічної верифікації поведінки програм на кінцевому (обмеженому) наборі тестів (set of test cases), обраних відповідним чином із зазвичай виконуваних дій прикладної області і забезпечують перевірку відповідності очікуваній поведінці системи

Основні поняття.2

Більшість встановлених результатів теорії тестування – негативні

Це означає те, що тестування програми може використовуватися для демонстрації наявності дефектів, але ніколи не покаже їх відсутність

Основна причина цього в тому, що повне (всеосяжне) тестування недосяжно для реального програмного

Рівні і цілі тестування

Рівень тестування визначає "над чим" виконуються тести: над окремим модулем, групою модулів або системою, в цілому

Є 3 рівні тестування:

- Модульне тестування
- Інтеграційне тестування
- Системне тестування

Модульне тестування (Unit testing)

Цей рівень тестування дозволяє перевірити функціонування окремо взятого елемента системи

Що вважати елементом - модулем системи - визначається контекстом

Інтеграційне тестування (Integration testing)

Даний рівень тестування є процесом перевірки взаємодії між програмними компонентами / модулями

Класичні стратегії інтеграційного тестування - "зверху-вниз" і "знизу-вгору" - використовуються для традиційних, ієрархічно структурованих систем і їх складно застосовувати, наприклад, до тестування слабкозв'язаних систем, побудованих у сервісно-орієнтованій архітектурі (SOA)

Інтеграційне тестування (Integration testing).2

Сучасні стратегії більшою мірою залежать від архітектури тестованої системи і будуються на основі ідентифікації функціональних "потоків" (наприклад, потоків операцій і даних)

Інтеграційне тестування – діяльність, яка проводиться постійно, що передбачає роботу на досить високому рівні абстракції

Найбільш успішна практика інтеграційного тестування базується на інкрементному підході, що дозволяє уникнути проблем проведення разових тестів, пов'язаних з тестуванням результатів чергового тривалого етапу робіт, коли кількість виявлених дефектів призводить до серйозної переробки коду

Негативний досвід випуску і тестування тільки великих релізів називають "big bang" - "великий вибух"

Системне тестування (System testing)

Системне тестування охоплює усю систему

Більшість функціональних збоїв повинна бути ідентифікована ще на рівні модульних та інтеграційних тестів

У свою чергу, системне тестування, зазвичай фокусується на нефункціональних вимогах - безпеки, продуктивності, точності, надійності тощо

На цьому рівні також тестуються інтерфейси до зовнішніх застосувань, апаратного забезпечення, операційного середовища

Цілі тестування

Можна виділити наступні, найбільш поширені і обгрунтовані цілі (а, відповідно, - види) тестування.

Головні:

- Приймальне тестування
- Функціональні тести / тести відповідності
- Регресійне тестування
- Навантажувальне тестування

Цілі тестування.2

Додаткові

- Інсталяційне тестування
- Альфа-і бета-тестування
- Досягнення та оцінка надійності
- Тестування продуктивності
- Порівняльне тестування
- Відновлювальні тести
- Конфігураційне тестування
- Тестування зручності та простоти використання

Розподіл на головні та додаткові види тестування є, взагалі, умовним та залежить від природи системи, серійності та інших характеристик

Види, зазначені вище як головні, стосуються унікальних та дрібносерійних систем.

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 "що приймає" програмну систему, або наявні 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

Функціональні тести / тести відповідності (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 характеристик поведінки системи

Регресійне тестування (Regression testing)

Визначення успішності регресійних тестів 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 у пошуку компромісу між наявними ресурсами і необхідністю проведення таких тестів після внесення кожної зміни

В певній мірі, завдання полягає в тому, щоб визначити критерії "масштабів" змін, з досягненням яких необхідно проводити регресійні тести.

Навантажувальне тестування (Stress testing)

Спеціалізовані тести перевірки задоволення специфічних вимог, що пред'являються до параметрів продуктивності

Тести проводять з метою досягнення реальних (досяжних) можливостей продуктивності системи з підвищенням навантаження, аж до досягнення запланованих характеристик і далі, з відстеженням поведінки на всьому протязі підвищення завантаження системи

Інсталяційне 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 з залученням відібраних зовнішніх користувачів) версій

Звіти про помилки, що надходять від користувачів цих версій продукту, обробляються відповідно до певних процедур, що включають підтверджуючі тести (будь-якого рівня), які проводяться фахівцями групи розробки

Даний вид тестування не може бути заздалегідь спланований

Досягнення та оцінка надійності (Reliability achievement and evaluation)

Допомагає ідентифікувати причини збоїв та сприяє підвищенню надійності програмних систем

Випадково генеруються сценарії тестування, які можуть застосовуватися для статистичної оцінки надійності

Обидві цілі - підвищення та оцінка надійності - можуть досягатися при використанні моделей підвищення надійності

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

Порівняльне тестування (Back-to-back testing)

Одиничний набір тестів, що дозволяють порівняти дві версії системи

Відновлювальні тести (Recovery testing)

Мета - перевірка можливостей рестарту системи в разі 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 засвоїти, включаючи не тільки функціональну складову - саму систему, але й її документацію
- Наскільки ефективно користувач може виконувати завдання, автоматизація яких здійснюється з використанням даної системи
- Наскільки добре система застрахована (з точки зору потенційних збоїв) від помилок користувача

Техніки тестування (Test Techniques)

Техніки, що базуються на інтуїції і досвіді інженера (Based on the software engineer's intuition and experience)

Техніки, що базуються на специфікації (Specification-based techniques)

Техніки, орієнтовані на код (Code-based techniques)

Тестування, орієнтоване на дефекти (Fault-based techniques)

Техніки, що базуються на умовах використання (Usage-based techniques)

Техніки, що базуються на інтуїції і досвіді інженера

Спеціалізоване тестування (Ad hoc testing)

Можливо, техніка, яка найбільш широко практикується

Тести грунтуються на досвіді, інтуїції і знаннях інженера, що розглядає проблему з точки зору наявних раніше аналогій

Даний вид тестування може бути корисним для ідентифікації тих тестів, які не охоплюються більш формалізованими техніками

Дослідницьке тестування (Exploratory testing)

Таке тестування визначається як одночасне навчання, проектування тесту та його виконання

Даний вид тестування заздалегідь не визначається в плані тестування і такі тести створюються, виконуються і модифікуються динамічно, по мірі необхідності

Ефективність дослідницьких тестів безпосередньо залежить від знань інженера, сформованих на основі:

- поведінки тестованого продукту в процесі проведення тестування
- ступеня знайомства з додатком, платформою, типами можливих збоїв і дефектів
- ризиками, асоційованими з конкретним продуктом
- тощо

Техніки, що базуються на специфікації

Еквівалентний поділ <програми> (Equivalence partitioning)

Розглянута область розділяється на колекцію наборів або еквівалентних класів, які вважаються еквівалентними з точки зору розглянутих зв'язків і характеристик <специфікації>

Репрезентативний набір тестів (іноді - тільки один тест) формується з тестів еквівалентних класів (або наборів класів)

Аналіз граничних значень (Boundary-value analysis)

Тести будуються з орієнтацією на використання тих величин, які визначають граничні характеристики тестованої системи

Розширенням цієї техніки є тести оцінки живучості (robustness testing) системи, що проводяться з величинами, що виходять за рамки специфікованих меж значень

Таблиці прийняття рішень (Decision table)

Такі таблиці представляють логічні зв'язки між умовами (можуть розглядатися в якості "входів") і діями (можуть розглядатися як "виходи")

Набір тестів будується послідовним розглядом усіх можливих крос-зв'язків в такій таблиці

Тести на основі скінченного автомата (Finite-state machine-based)

Будуються як комбінація тестів для всіх станів і переходів між станами, представлених у відповідній моделі (переходів і станів застосування)

Тестування на основі формальної специфікації (Testing from formal specification)

Для специфікацій, визначених з використанням формальної мови, можливо автоматично створювати і тести для перевірки функціональних вимог

У ряді випадків ці тести можуть будуватись на основі моделі, що є частиною специфікації, та не використовує формальної мови опису

Випадкове тестування (Random testing)

Тести генеруються випадковим чином за списком заданого набору специфікованих характеристик

Техніки, орієнтовані на код

Тести, що базуються на блок-схемі (Control-flow-based criteria)

Набір тестів будується виходячи з покриття всіх умов і блоків блок-схеми

В якійсь мірі нагадує тести на основі скінченного автомата

Відмінність - в джерелі набору тестів

Максимальна віддача від тестів на основі блок-схеми виходить тоді, коли тести покривають різні шляхи блок-схеми - по-суті, сценарії потоків робіт (поведінки) тестованої системи

Адекватність таких тестів оцінюється як відсоток покриття всіх можливих шляхів блок-схеми

Тести на основі потоків даних (Data-flow-based criteria)

У даних тестах відстежується повний життєвий цикл величин (змінних) - з моменту народження (визначення), та протягом всього використання, аж до знищення (невизначеності)

У реальній практиці використовуються нестроге тестування такого виду, орієнтоване, наприклад, тільки на перевірку завдання початкових значень всіх змінних або всіх входжень змінних в код, з точки зору їх використання

Тестування, орієнтоване на дефекти

Припущення помилок (Error guessing)

Спрямовані на виявлення найбільш ймовірних помилок, які передбачаються, наприклад, в результаті аналізу ризиків

Тестування мутацій (Mutation testing)

Мутація - невелика зміна тестованої програми, що сталося за рахунок часткових синтаксичних змін коду (зокрема, рефакторинга)

Відповідні тести запускаються для оригінального і всіх "мутованих" варіантів тестованої програми

Техніка фокусується на можливості, за допомогою тестів, визначати відмінності між мутантами і вихідним варіантом коду

Якщо таку відмінність встановлено, мутанта "вбивають", а тест вважається успішним