Лекція 8.
Автоматизація тестування

Зміст

Цільова настанова: розглянемо структуру тестового набору для автоматичного прогону. Обговорюється структура інструментальної системи автоматизації тестування. Порівнюються витрати і ефективність різних методів тестування.

Ключові слова: автоматичний прогін тестів, структура тестового набору, інструментальна система автоматизації тестування, log-файл, витрати тестування, ефективність тестування.

Вступ

Автоматизоване тестування програмного забезпечення – частина процесу тестування на етапі контролю якості в процесі розробки програмного забезпечення. Воно використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити його процес.

Перші спроби «автоматизації» з'явилися в епоху операційних систем DOS і CP/M. Тоді вона полягала у видачі додатком команд через командний рядок і аналізі результатів.

Трохи пізніше додалися віддалені виклики через API – прикладний програмний інтерфейс) для роботи в мережі. Вперше про автоматизоване тестування згадується в книзі Фредеріка Брукса «Міфічний людино-місяць», де йдеться про перспективи використання модульного тестування. Але по-справжньому автоматизація тестування стала розвиватися тільки в 1980-х роках. Однією з головних проблем автоматизованого тестування є його трудомісткість.

Попри те, що автоматизоване тестування дозволяє усунути частину рутинних операцій і прискорити виконання тестів, великі ресурси можуть витрачатися на оновлення самих тестів.

Наприклад, при рефакторінгу часто буває необхідно оновити і модульні тести, і зміна коду тестів може зайняти стільки ж часу, скільки і зміна основного коду.

1. Структура тестового набору для автоматизованого прогону

Використання різних підходів до тестування визначається їхньою ефективністю стосовно щодо умов, обумовлених промисловим проектом. У реальних випадках робота групи тестування планується так, що розробка тестів починається з моменту узгодження вимог до програмного продукту і триває паралельно з розробкою дизайну й коду продукту.

Тестові (контрольні) випадки є базовими компонентами системи тестування. У найбільш узагальненій формі тестовий випадок (test case) є парою (вхідні дані, очікуваний результат), у якій вхідні дані – це опис даних, що подаються на вхід програмного продукту, котрий тестується, а очікуваний результат є описом вихідних даних, котрі програмний продукт повинен пред'явити у відповідь на відповідне введення. Вхідні дані та очікувані результати не обов'язково повинні бути простими величинами, подібно рядкам чи цілочисельним значенням; навпаки, їх складність нічим не обмежується. Часто поряд з командами користувача і даними, котрі підлягають обробці, у склад вхідних даних включається інформація про стан системи.

Очікувані результати – це не тільки такі легкі для сприйняття речі, як скажімо, лістинги результатів, звукові сигнали чи зміни у зображеннях, що відтворюються на екрані, але також зміни у самому програмному продукті, наприклад, оновлення вмісту бази даних чи зміна стану самої системи, котра в свою чергу впливає на обробку наступних наборів тестових даних. У результаті, до початку системного тестування створюються *тестові набори (test suite)*. Один набір може містити тисячі тестів. Тестові набори формуються з тестових випадків. Більшість тестових наборів у більшій чи меншій мірі організовані у визначеному порядку, котрий відображає властивості тестових випадків.

Наприклад одну частину тестових наборів можуть складати тестові випад-ки, котрі тестують можливості системи, а інша частина містить тестові випадки, призначені для поглибленої перевірки звичайної роботи системи у рамках конкретних можливостей. Якщо програмний продукт вдало справляється з усіма тестовими випадками, то ми кажемо, що продукт «вдало пройшов випробування на тестовому наборі». Великий набір тестів забезпечує всебічну перевірку функціональності й гарантує якість продукту, але виконання такої кількості тестів на етапі системного тестування є проблемою. Її розв'язання лежить в області автоматизації тестування, тобто в автоматизації розробки.

Розглянемо приклад, у якому наведена структура тесту, структура комплексу тестування та структура тестуючого модуля. Особливістю структури кожного з тестованих модулів M_i є запуск тестованої програми P_i після того як кожний з модулів M_{ij}, що входять у контекст модуля M_i, відтестовано. У цьому випадку запуск тестуючого модуля забезпечує рекурсивний спуск до програм тестування модулів нижнього рівня, а потім виконує тестування вищих рівнів в умовах відтестованості нижчих.

Структура програми Р тесту

Завантаження тесту (X, Y *)
Запуск модуля, що тестується
Порівняння отриманих результатів Y з еталонними Y *
Структура тестованого комплексу
  ModF ← $ModF_1$
  $ModF_2$
  $ModF_3$ ← $ModF_{31}$
  $ModF_{32}$
Структура тестирующего модуля Мод TestModF:
  Mod $TestМодF_1$
  Mod $TestМодF_2$
  Mod $TestМодF_3$
  P TestМодF
  Мод $TestModF_1$:
  P $TestМодF_1$
  Мод $TestModF_2$:
  P $TestМодF_2$
  Мод $TestModF_3$:
  Mod $TestМодF_{31}$
  Mod $TestМодF_{32}$
  P $TestМодF_3$

Тестові набори подібної структури орієнтовані на автоматичне керування пропуском тестового набору в тестовому циклі. Важливою перевагою подібної організації є можливість регулювання нижнього рівня, того, до якого варто доходити в циклі тестування. У цьому випадку контекст редукованих у конкретному тестовому циклі модулів позначається як базовий, котрий не підлягає тестуванню. Наприклад, якщо контекст модуля Mod_3: (Mod_{31}, Mod_{32}) – позначений як базовий, то в результаті рекурсивний спуск торкнеться лише модулів Mod_1, Mod_2, Mod_3 і вищого модуля Mod. Описаний спосіб організації тестових наборів успішно застосовується в системах автоматизації тестування. Власне використання ефективної системи автоматизації тестування скорочує до мінімуму (наприклад, до однієї ночі) час пропуску тестів, без якого не-можливо підтвердити факт росту якості (зменшення числа помилок, що залишилися) продукту. Системне тестування здійснюється в рамках циклів тестування (періодів пропуску розробленого тестового набору над build розроблюваного додатка). Перед кожним циклом фіксується розроблений або виправлений build, на який заносяться виявлені в результаті тестового прогону помилки. Потім помилки виправляються, і на черговий цикл тестування пред'являється новий build.

Закінчення тестування збігається з експериментально підтвердженим висновком про досягнутий рівень якості щодо обраного критерію тестування або про зниження щільності не виявлених помилок до деякої заздалегідь визначеної величини. Можливість обмежити цикл тестування межею в одну добу або кілька годин підтримується винятково за рахунок засобів автоматизації тестування.

2. Генератори тестів

У деяких випадках для спрощення процедури тестування використовуються спеціальні інструментальні засоби, які автоматично генерують тестові приклади. Ці системи різняться за використанням методів генерації тестових прикладів, а одержувані тестові приклади розрізняються за областями застосовності.

Розрізняють такі способи генерації тестових прикладів:

- за формалізованими вимогами;
- випадковим чином;
- з програмного коду.

Перший спосіб генерації тестових прикладів прийнятний длятестування системи як «*чорного ящика*», але вимагає щоб тест-вимоги (або системні/функціональні вимоги) були підготовлені на спеціальнійформальній мові оформлення вимог, наприклад RDL (Requirements Definition Language). Потім за вимогами будуються тестові приклади, які перевіряють функціональність системи з точки зору вимог, тобто в цьому випадку досягається основна мета верифікації – перевірити, чи поводиться система відповідно до вимог.

На жаль, цей шлях досить трудомісткий і економія часу від автоматичної генерації тестів часто зводиться нанівець необхідністю у виділенні додаткового часу на переклад всіх вимог у формальну форму. У зв'язку з цим рекомендується застосовувати даний метод тільки для тестування систем, вимоги яких можуть бути порівняно легко формалізовані з використанням тієї чи іншої мови, наприклад, системи підтримки комунікаційних протоколів.

Другий метод генерації тестових прикладів – на основі випадкових даних. У цьому випадку не може йти й мови про систематизоване тестування та гарантії якості системи. Такий підхід може застосовуватися тільки в разі потреби перевірити поведінку системи при передачі в неї великої кількості невірних даних або визначити кількісні параметри поведінки системи під великим навантаженням.

Третій метод тестування заснований на аналізі вихідних текстівсистеми і побудови тестів, які виконують кожну логічну умову і кожен оператор системи. У результаті досягається дуже високий рівень покриття програмного коду. Однак в цьому випадку тести перевіряють не те, що система повинна робити відповідно до вимог, а те, як вона робить те, що вже запрограмовано. Перед тестувальником в цьому випадку стоїть завдання аналізу програмного коду системи на відповідність вимогам, що часто являє собою завдання не менш складне, ніж ручне написання тестів для перевірки вимог. Зазвичай рекомендується спочатку написати всі тести за вимогами, а потім, у разі потреби, скористатися генератором тестів з програмного коду. При цьому метою використання генератора буде не досягнення максимально можливого покриття за будь-яку ціну, а аналіз причин не покриття при виконанні тестів вимог, і корекції вимог.

3. Структура інструментальної системи автоматизації тестування

Виконання (прогін) тестового випадку – це сеанс роботи програмного забезпечення, у рамках котрого на вхід програмного продукту подаються набори даних, передбачені специфікацією тестового випадку, і фіксуються результати їх обробки, котрі потім порівнюються з очікуваними результатами, що вказані у тестовому випадку. Якщо фактичний результат відрізняється від очікуваного, це означає, що виявлена відмова, і в таких ситуаціях ми кажемо, що тестований програмний продукт «не пройшов випробування на заданому тестовому випадку». Якщо ж отриманий результат співпадає з результатом очікуваним у даному тестовому випадку то ми кажемо, що програмний продукт «пройшов випробування на заданому тестовому випадку». На рис. 1 представлена узагальнена структура системи автоматизації тестування, у якій створюється й зберігається така інформація:

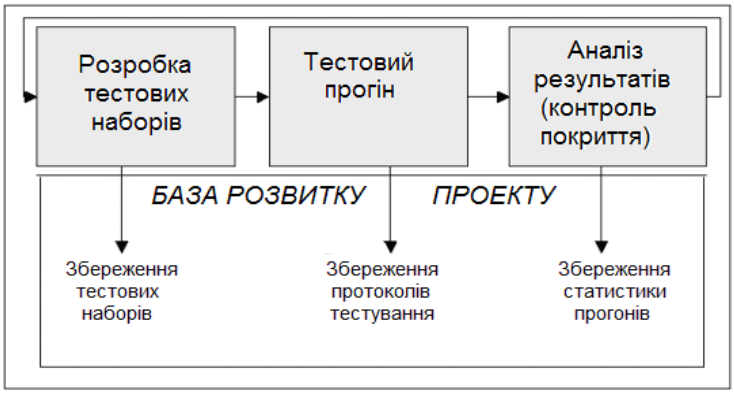

Рис. 1. Структура інструментальної системи автоматизації тесту-

1. Набір тестів, достатній для покриття додатка, що тестується відповідно до обраного критерію тестування – як результат ручної або автоматичноїрозробки (генерації) тестових наборів і драйвер/монітор пропускутестового набору.
2. Результати прогону тестового набору, зафіксовані в Log-файлі. Log-файл містить траси («протоколи»), що являють собою реалізовані при тестовому прогоні послідовності деяких подій (значень окремих змінних або їхніх сукупностей) і точки реалізації цих подій на графі програми. У складі трас можуть бути присутні послідовності явно й неявно заданих міток, що задають шляхи реалізації трас на керуючому графі програми, сукупності значень змінних на цих мітках, величини проміжних результатів, досягнутих на деяких мітках тощо.
3. Статистика тестового циклу, що містить:

– результати пропуску кожного тесту з тестового набору і їхньогопорівняння з еталонними величинами;
– факти, що послужили підставою для ухвалення рішення про продовження або закінчення тестування;
– критерій покриття й степінь його задоволення, досягнута в циклітестування.

Результатом аналізу кожного прогону є список проблем, у вигляді помилок і дефектів, що заноситься в базу розвитку проекту. Далі відбувається робота над помилками, де кожна піднята проблема ідентифікується, ставиться до відповідного модуля й розробника, отримує пріоритет й відслідковується, що забезпечує гарантію її вирішення (виправлення або віднесення до списку відомих проблем, вирішення яких по тим або іншим причинах відкладається) у наступних build.

Тестові набори також потребують супроводу. По мірі зміни вимог повинні змінюватись і тестові набори. Необхідно вносити зміни до тестових наборів у яких виявлені помилки. Якщо помилку виявили користувачі, то у тестовий набір повинні бути додані випадки, котрі дозволили б ліквідувати їху наступних версіях програмного продукту ще до розгортання на місцях.

Виправлений і зібраний для тестування build надходить на наступний цикл тестування, і цикл повторюється, поки необхідна якість програмного комплексу не буде досягнута. У цьому ітераційному процесі засоби автоматизації тестування забезпечують швидкий контроль результатів виправлення помилок і перевірку рівня якості, досягнутого в продукті.

4. Порівняння витрат і ефективності різних методів тестування

Інтенсивність виявлення помилок на одиницю витрат і надійність тісно зв’язані з часом тестування й, відповідно, з гарантією якості продукту (рис. 2А). Чим більше трудових затрат вкладається в процес тестування, тим менше помилок у продукті залишається непоміченими. Однак досконаліть в індустріальному програмуванні має межі, які насамперед пов’язані з витратами на одержання програмного продукту, а також з надлишком якості, що не вимагається замовником додатка. Знаходження оптимуму – дуже відповідальне завдання тестувальника й менеджера проекту.

Рух до зменшення числа помилок, що залишилися, або до якості продукту приводить до застосування різних методів налагодження й тестування в процесі створення продукту. На рис. 2 наведений витратний компонент тестування залежно від удосконалювання застосовуваного інструментарію й методів тестування. На практиці популярні такі методи тестування й налагодження, упорядковані за пов’язаними з їхнім застосуванням витратами:

1) статичні методи тестування;
2) модульне тестування;
3) інтеграційне тестування;
4) системне тестування;
5) тестування реального оточення й реального часу.

Залежність ефективності застосування перерахованих методів або їхньої здатності до виявлення відповідних класів помилок (С) представленана рис. 2С витратами (В). Графік показує, що згодом, у міру виявлення

складніших помилок і дефектів, ефективність низькозатратних методів падає разом з кількістю помилок, які вони виявляють.

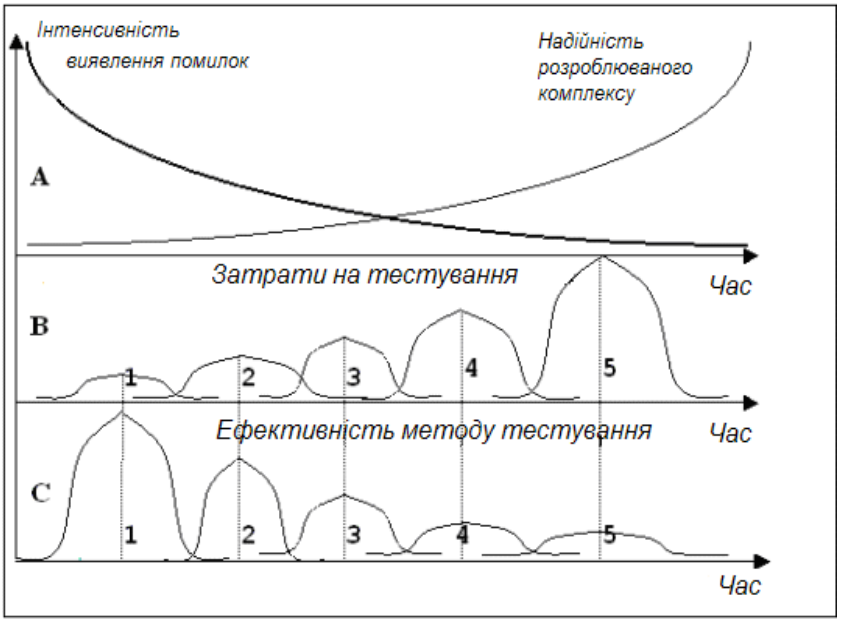

Рис. 2. Залежність ефективності застосування методів довиявлення відповідних класів помилок

Звідси слідує, що всі методи тестування не тільки мають право на існування, але й мають свою нішу, де вони добре виявляють помилки, тоді якпоза нішею їхня ефективність падає.

Тому необхідно поєднувати різні методий стратегії налагодження й тестування з метою забезпечення запланованої якості програмного продукту при обмежених витратах, що можливо при використанні процесу керування якістю програмного продукту.

5. Додатки автоматизованого тестування

Для автоматизації тестування існує велика кількість додатків.

Деякі з них:

- HP LoadRunner, HP QuickTest Professional, HP Quality Center;
- Segue SilkPerformer;
- IBM Rational FunctionalTester, IBM Rational PerformanceTester, IBM Rational TestStudio;
- SmartBear Software TestComplete.

Використання цих інструментів допомагає тестувальникам автоматизувати такі задачі:

- установка продукту;
- створення тестових даних;
- GUI взаємодія;
- визначення проблеми.

Однак автоматизовані тести не можуть повністю замінити ручне тестуван-ня. Автоматизація всіх випробувань – дуже дорогий процес, і тому

автоматичне тестування є лише доповненням ручного тестування. Найкращий варіант вико-ристання автоматизованих тестів – регресивне тестування.

Контрольні питання

1. Мета автоматизованого тестування програмного забезпечення.
2. Які структури тестового набору використовуються для автоматизованого прогону?
3. Спеціальні інструментальні засоби – генератори тестів.
4. Які ви знаєте методи генерації тестових прикладів?
5. Структура інструментальної системи автоматизації тестування.
6. Порівняння витрат і ефективності різних методів тестування.
7. Які популярні методи тестування й налагодження застосовуються на практиці?
8. Які додатки використовуються для автоматизованого тестування?