Лекція 2.
Основні поняття тестування

Зміст

Цільова настанова: Розглянуто підходи до обгрунтування істинності формул і програм та їх зв'язок з тестуванням. Представлені на конкретних прикладах поняття налагодження і тестування. Розглянуто питання організації тестування. На прикладах пояснені методи пошуку помилок і процедура тестування. Розглянуто фази тестування, основні проблеми тестування та поставлено завдання вибору кінцевого набору тестів.

Ключові слова: істинність формул, налагодження, тестування, організація тестування, методи пошуку помилок, процедура тестування, оракул, дамп, точка зупинки, фази тестування, проблеми тестування, набір тестів, прогін тестів, керуючий граф, шлях, гілка, вибір кінцевого набору тестів .

2.1. Концепція тестування

Програма - це аналог формули в звичайній математиці.

Формула для функції f отриманої суперпозицією функційf_1, $f_{2,...,}$ f_n — вираз, що описує цю суперпозицію:

$$f = f_1 * f_2 * \ldots * f_n.$$

Якщо аналог $f_1, f_2,\ldots, f_n$ — оператори мови програмування, то їх формула - програма.

Існує два методи обгрунтування істинності формул:

1. *Формальний підхід* або *доказ* застосовується, коли з ис Ходна формул-аксіом за допомогою формальних процедур (пра вил виведення) виводяться шукані формули і затвердження (теоре ми). Висновок здійснюється шляхом переходу від одних формул до інших за суворими правилами, які дозволяють звести процеду ру переходу від формули до формули до послідовності тексто вих підстановок:

$$A**3 = A*A*A$$
$$A*A*A = A \rightarrow R,\ A*R \rightarrow R,\ A*R \rightarrow R.$$

Перевага формального підходу полягає в тому, що з його допомогою вдається уникати звернень до нескінченної області значень, і на кожному кроці докази оперувати тільки кінцевим безліччю символів.

2. *Інтерпретаційний* підхід застосовується, коли здійснюється підстановка констант у формули, а потім - інтерпретація фор - мул як осмислених тверджень в елементах множин кін - ретних значень. Істинність інтерпретованих формул прове - ряется на кінцевих множинах можливих значень. Складність підходу полягає в тому, що на кінцевих множинах комбінації можливих значень для реалізації вичерпної перевірки, можуть виявитися досить, великі.

Інтерпретаційний підхід використовується при експери - ної перевірки відповідності програми своєї специфікації. Застосування інтерпретаційного підходу в формі експериментів над виконуваною програмою становить суть налагодження і тестування.

2.2. Основна термінологія

Відладка (debug, debugging) - процес пошуку, локалізації і ІСПР - тичних помилок в програмі [9].

Термін «налагодження» у вітчизняній літературі використовується двояко: для позначення активності з пошуку помилок (власне тестірова - ня), по знаходженню причин їх появи і виправлення, або активно - сті по локалізації та виправлення помилок.

Тестування забезпечує виявлення (констатацію наявності) фак - тов розбіжностей з вимогами (помилок).

Як правило, на фазі тестування здійснюється і виправлення ідентифікованих помилок, що включає локалізацію помилок, знахо - ждение причин помилок і відповідне коригування програми тестованого програми (Application (AUT) або Implementation Under Testing (IUT)).

Якщо програма не містить синтаксичних помилок (пройшла трансляцію) і може бути виконана на комп'ютері, вона обов'язково обчислює будь-яку функцію, яка здійснює відображення вхід - них даних у вихідні. Це означає,

що комп'ютер на своїх ресурсах доопределять частково певну програмою функцію до тоталь - ної визначеності. Отже, судити про правильність чи непра - ності результатів виконання програми можна, тільки порівнюючи специфікацію бажаної функції з результатами її обчислення, що і здійснюється в процесі тестування.

Судити про правильність чи неправильність результатів виконан - ня програми можна тільки порівнюючи специфікацію бажаної функції з результатами її обчислення.

Приклад пошуку та виправлення помилки

Налагодження забезпечує локалізацію помилок, пошук причин помилок і відповідне коригування програми. Так, наприклад, якийсь метод обчислює неотрицательную ступінь n числа x.

Якщо викликати метод Power з від'ємним значенням ступеня n Power (2, -l), то отримаємо некоректний результат - 2. Исправим метод так, щоб помилкове значення параметра (неприпустиме за специфікацією значення) ідентифікувалися спеціальним повідомленням, а повертається результат дорівнював 1.

Якщо викликати скоригований метод PowerNonNeg (2, -1) з отрица - тельним значенням параметра ступеня, то повідомлення про помилку буде видано автоматично.

Судити про правильність чи неправильність результатів виконання програми можна тільки порівнюючи специфікацію бажаної функції з результатами її обчислення.

Тестування поділяють на такі види:

- статичне,
- динамічне.

Статичне тестування виявляє формальними методами аналізу без виконан - нання програми, що тестується невірні конструкції або невірні від - носіння об'єктів програми (помилки формального завдання) за допомогою на - гою спеціальних інструментів контролю коду - CodeCheker.

Динамічне тестування (власне тестування) здійснюва - ет виявлення помилок тільки на виконується програмою за допомогою спеціальних інструментів автоматизації тестування - Testbed [9] або Testbench.

2.3. Організація тестування

Тестування здійснюється на заданому заздалегідь безлічі вхід - них даних X і безлічі передбачуваних результатів Y - (X, Y), кото - риє задають графік бажаного ефекту. Крім того, зафіксована про - цедура Оракул (oracle), яка визначає, чи відповідають вихідні дані - Y в (обчислені за вхідними даними - X) бажаним результа - там - Y, тобто чи належить кожна обчислена точка (x, y в) графіку бажаної функції (X, Y).

Оракул дає висновок про факт появи неправильної пари (x, y в) і нічого не говорить про те, яким чином вона була обчислена або який правильний алгоритм - він тільки порівнює обчислені і ж - гавкотом результати. Оракулом

може бути навіть замовник або програм - мист, що виробляє відповідні обчислення в розумі, оскільки Ора - кулу потрібен якийсь альтернативний спосіб отримання функції (X, Y) для обчислення еталонних значень Y.

Приклад порівняння словесного опису пункту специфікації з результатом виконання фрагмента коду

Пункт специфікації: Метод Power повинен приймати вхідні пара - метри:
x - ціле число, що зводиться в ступінь,
n - невід'ємні порядок ступеня.
Метод повинен повертати обчислене значення x n.
Виконуємо метод з наступними параметрами: Power (2,2).
Перевірка результату виконання можлива, коли результат обчислити - ня заздалегідь відомий - 4. Якщо результат виконання 2 + 2 = 4, то він зі - відповідає специфікації.
В процесі тестування Оракул послідовно отримує елементи безлічі (X, Y) і відповідні їм результати обчислень (X, Y b) для ідентифікації фактів розбіжностей (test incident).
При виявленні (x, y в) (X, Y) запускається процедура виправлення помилки, яка полягає в уважному аналізі (перегляді) про - токола проміжних обчислень, що призвели до (x, y в), за допомогою сле - дмуть методів:

1) Виконання програми «в умі» (deskchecking).
2) Вставка операторів протоколювання (друку) проміжних результатів (logging).

Приклад вставки операторів протоколювання проміжних результатів

Можна виводити проміжні значення змінних при виконан - ванні програми. Цей метод відноситься до найбільш популярним засобам автоматизації налагодження програмістів минулих десятиліть. В на - варте час він відомий як метод впровадження «агентів» в текст налагоджують програму.

3) Покрокове виконання програми (single-step running)..

Приклад покрокового виконання програми

При покроковому виконанні програми код виконується рядок за рядком. У середовищі Microsoft Visual Studio. NET можливі наступні команди покрокового виконання:

• Step Into - якщо виконувана рядок коду містить виклик функ - ції, процедури або методу, то відбувається виклик, і програма ос - новлюють на першій сходинці викликається функції, процедури або методу.

• Step Over - якщо виконувана рядок коду містить виклик функ - ції, процедури або методу, то відбувається виклик і виконання всієї функції і програма зупиняється на першій сходинці після ви - зувати функції.

• Step Out - призначена для виходу з функції в зухвалу функцію. Ця команда продовжить виконання функції і остано - віт виконання на першій сходинці після викликається функції.

Покрокове виконання досі є потужним методом автоном - ного тестування і налагодження невеликих програм.

4) Виконання з замовленими зупинками (breakpoints), аналізом трас (traces)

або станів пам'яті - дампов (dump).

Приклад виконання програми із замовленими контрольними точками і аналізом трас і дампов

Контрольна точка (breakpoint) - точка програми, яка при її досягненні посилає отладчику сигнал. За цим сигналом або тимчасово призупиняється виконання налагоджують програму, або запускається програма «агент», яка фіксує стан заздалегідь визначених змінних або областей в даний момент.

Коли виконання в контрольній точці припиняється, налагоджують програму переходить в режим зупинки (break mode). Вхід в режим зупинки не переріває і не закінчує виконання програми і дозволяє аналізувати стан окремих змінних або структур даних. Повернення з режиму break mode в режим виконання може статися в будь-який момент за бажанням користувача.

Коли в контрольній точці викликається програма «агент», вона теж призупиняє виконання налагоджують програму, але тільки на час, необхідний для фіксації стану обраних змінних або структур даних в спеціальному електронному журналі - log-файл, після чого відбувається автоматичне повернення в режим виконання.

Траса - це «збережений шлях» на керуючому графові програми, тобто зафіксовані в журналі записи про станах змінних в заданих точках в ході виконання програми.

Дамп - область пам'яті, стан якої фіксується у контрольній точці у вигляді єдиного масиву або декількох пов'язаних масивів. При аналізі, який здійснюється після виконання траси в режимі off - line, стану дампа структуруються, і виділені області або поля порівнюються з станами, передбаченими специфікацією. Наприклад, при моделюванні поведінки керуючих програм контролерів у вигляді дампа фіксуються області загальних і спеціальних регістрів, або цілі області оперативної пам'яті, стану якої визначає алгоритм управління зовнішнім середовищем.

5) Реверсивний (зворотне) виконання (reversible execution).

Зворотне виконання програми можливе за умови сохране - ня на кожному кроці програми всіх значень змінних або перебуваючи - ний програми для відповідної траси. Тоді піднімаючись від ко - кінцевих точки траси до будь-якої іншої, можна по кроках зробити обчислюва - лення станів, рухаючись від слідства до причини, від станів на ви - Під час перетворювача даних до станів на його вході. Природно, та - кі можливості ми отримуємо в режимі off - line аналізу при фіксації в log-файл всій історії виконання траси.

Приклад зворотного виконання для програми обчислення ступеня числа x

Нехай, в якійсь програмі фіксуються значення всіх змінних після виконання кожного оператора. Знаючи структуру керуючого графа програми і маючи значення всіх змінних після виконання кожного оператора, можна здійснити зворотне виконання (наприклад, в розумі), підставляючи значення змін - них в оператори і рухаючись знизу-вгору, починаючи з останнього.

Отже, в процесі тестування порівняння проміжних резуль - татів з отриманими незалежно еталонними результатами дозволяє знайти причини і місце помилки, виправити текст програми, провисання - ти повторну трансляцію і настройку на виконання і продовжити тестування.

Тестування закінчується, коли виповнилося або «пройшло» (pass) успішно достатню кількість тестів відповідно до обраний - вим критерієм тестування.

Тестування - це:

• Процес виконання ПО системи або компонента в умовах аналізу або записи отриманих результатів з метою перевірки (оцінки) деяких властивостей тестованого об'єкта [9].

• Процес аналізу пункту вимог до ПЗ з метою фіксації раз - лічій між існуючим станом ПО і необхідним (що свідчить про прояв помилки) при експериментальній перевірці відповідного пункту вимог [9].

• Контрольоване виконання програми на кінцевому множест - ве тестових даних і аналіз результатів цього виконання для пошуку помилок.

2.4. Специфікація програми

Візьмемо якусь програму, яка обчислює ступінь числа x.

- На вхід програма приймає два параметри: x - число, n - ступінь. Результат обчислення виводиться на консоль.
- Значення числа і ступеня повинні бути цілими.
- Значення числа, що зводиться в ступінь, повинні лежати в діапазоні - [0 .. 999].
- Значення ступеня повинні лежати в діапазоні - [1 ... 100].
- Якщо числа, що подаються на вхід, лежать за межами зазначених діапа Зонов, то повинен видаватися повідомлення про помилку.

2.5. Розробка тестів

Визначимо області еквівалентності вхідних параметрів.

Для x - числа, що зводиться в ступінь, визначимо класи можли - них значень:

1) x < 0 (помилкове);
2) x> 999 (помилкове);
3) x - не числом (помилкове);
4) 0 ≤ x ≤ 999 (коректне).

Для n - ступеня числа:

5) n <1 (помилкове);
6) n> 99 (помилкове);
7) n - не числом (помилкове);
8) 1 ≤ n ≤ 100 (коректне).

2.6. Аналіз тестових випадків

1. Рівні введення: (x = 2, n = 3).
2. Очікуваний результат: The power n of x is 8.
3. Рівні введення: {(x ~ -1, n - 2), (x - 1000, n = 5)}.
4. Очікуваний результат: Error: x must be in [0. ..999].
5. {(x = 100, n - 0), (x = 100, n - 200)}.
6. очікуваний результат: Error: n must be in [1 ... 100].
7. Рівні введення: (x = ADS n = ASD).
8. очікуваний результат: Error: Please enter a numeric argument.
9. Перевірка на граничні значення:
   5.1 вхідні значення: (x = 999 п = 1).
   Очікуваний результат: The power п of x is 999.
   5.2 вхідні значення: x = On = 100.
   Очікуваний результат: The power п of x is 0.

2.7. Виконання тестових випадків

Запустимо програму з заданими значеннями аргументів.

2.8. Оцінка результатів виконання програми на тестах

В процесі тестування Оракул послідовно отримує елементи безлічі (X, Y) і відповідні їм результати обчислень YB. У процесі тестування проводиться оцінка результатів виконання шляхом порівняння одержуваного результату з очікуваним.

2.9. Три етапи тестування

Реалізація тестування поділяється на три етапи:

• *Створення тестового набору* (test suite) шляхом ручного розробки або ав томатически генерації для конкретної середовища тестірова - ня (testing environment).

• *Прогін* програми на тестах, керований тестовим монітором (test monitor, test driver [9]) з отриманням про - токола результатів тестування (test log).

• *Оцінка результатів* виконання програми на наборі тестів з метою прийняття рішення про продовження або зупинці тестують - вання.

Основна проблема тестування - визначення достатності мно - жества тестів для істинності висновку про правильності реалізації програм - ми, а також знаходження безлічі тестів, що володіє цією властивістю.

2.10. Керуючий граф програми

Керуючий граф програми (УГП) - граф G (V, A), де V (V 1, ..., V т) - множин вершин (операторів), A (Ai, ..., A n) - безліч дуг (управ - ний), що з'єднують оператори-вершини.

Шлях - послідовність вершин і дуг УГП, в якій будь-яка ду - га виходить

з вершини і приходить в вершину V j.

Гілка - шлях (V 1, V 2, ..., V n), де V 1 - або перший, або умовний оператор програми, V k - або умовний оператор, або оператор ви - ходу з про грами, а всі інші оператори - безумовні. Шляхи, що розрізняються хоча б числом перехожих - дений циклу - різні шляхи, тому число шляхів у програмі може бути необмежена. Гілки - лінійні ділянки програми, їх конеч - ве число.

Існують реалізовані і нереалізовані шляхи в програмі, в Нерео - лізуемие шляху в звичайних умовах потрапити не можна, але при збоях навіть нереал зуемое шлях може реалізуватися.

2.11. Основні проблеми тестування

• Тестування програми на всіх вхідних значеннях неможливо.
• Неможливо тестування і на всіх шляхах.
• Отже, треба відбирати кінцевий набір тестів, дозволяю - щий перевірити програму на основі наших інтуїтивних предста - тичних.

Вимога до тестів - програма на будь-якому з них повинна останав - Ліван, тобто не зациклюватися. Чи можна заздалегідь гарантувати останов на будь-якому тесті? У теорії алгоритмів доведено, що не існує загального методу для вирішення цього питання, а також питання, чи досягне про - грами на даному тесті заздалегідь фіксованого оператора.

Завдання про вибір кінцевого набору тестів (X, Y) для перевірки програм - ми в загальному випадку нерозв'язна.

Тому для вирішення практичних завдань залишається шукати окремі випадки вирішення цього завдання.

Для демонстрації і закріплення теоретичних знань розроблений практикум, що містить:

• опис лабораторних робіт;
• методичні вказівки по проведенню самостійної роботи студентів;
• рекомендації з підготовки комп'ютерної лабораторії до проведення лабораторних робіт.

В рамках практикуму студенти освоюють різні підходи до розробки тестів і тестування та умови їх застосування.

Практикум представлений у формі тренінгу, в якому розглянуті наступні теми:

• Розробка документації на тестируемую систему і її оточення: опис вимог (Requirement Specification) і специфікацій розробника (High Level Design).
• Планування тестування.
• Практикум модульного тестування.
• Практикум інтеграційного тестування.
• Практикум системного тестування.
• Ручне тестування і тестові процедури.
• Автоматизоване тестування на основі скриптів.
• Автоматизоване тестування на основі Msc-діаграм і генерація тестів.
• Засоби підтримки автоматизації тестування.

Контрольні запитання

1. Дайте визначення поняттю «Програма».
2. У чому сенс формального підходу до тестування?
3. У чому сенс інтерпретаційного підходу до тестування?
4. Дайте визначення поняттю «Тестування».
5. Дайте визначення поняттю «Налаштування».
6. Поясніть сенс тестування.
7. Поясніть сенс статичного тестування.
8. Яка організація тестування?
9. Що означає виконання програми «в умі»?
10. Дайте визначення поняттю «траса».
11. Дайте визначення поняттю «дамп».
12. Дайте визначення поняттю «шлях».
13. Поясніть поняття «гілка».
14. Що таке реалізовані і реалізуються шляху в програмі?