ТРЕНІНГ-КУРС З ІНФОРМАЦІЙНОГО УПРАВЛІННЯ ПІДПРИЄМСТВАМИ ТА ПРОЄКТАМИ

Самостійна робота за змістовним модулем 6

6.1. Основні поняття і принципи тестування

Для створення програмних виробів застосовуються технології, які містять сукупність виробничих процесів, методів і засобів, призначених для розробки програм із заданими показниками якості. Будь-який виріб тестується на відповідність цим показникам.

Стосовно до програмного виробу тестування – це процес багаторазового виконання програми з метою виявлення помилок.

Основним методом виявлення помилок під час налагодження програм є їх тестування. При цьому витрати на тестування для виявлення помилок є найбільшими, досягають 30–40 % загальних витрат на розробку програм і значною мірою визначають якість створеного програмного продукту.

Особливостями тестування програмного виробу є [28]:

- відсутність еталона (програми), якому повинна відповідати програма, що тестується;
- висока складність програм і теоретична неможливість вичерпного тестування;
- практична неможливість створення єдиної методики тестування (формалізації процесу тестування) внаслідок різноманітності програмних виробів за їх складністю, функціональним призначенням, сферою використання і т. ін.

Для підвищення ефективності тестування розроблені різні методи систематичного і регламентованого тестування, враховуючі особливості програм, що створюються.

Основна мета тестування для виявлення помилок – визначення всіх відхилень результатів функціонування реальної програми від заданих еталонних значень. Ефективними є операції тестування, які виявляють максимальну кількість помилок у програмі, що обґрунтовує витрати на їх виконання.

Задачі тестування доцільно розділити на групи відповідно до трьох стадій тестування:

- тестування для виявлення помилок у програмі;
- тестування для діагностики і локалізації причин виявлених перекручень результатів;
- тестування для контролю виконаних коректувань програм і даних.

Після тестування для виявлення помилок застосовується тестування для їх діагностики і локалізації. Його цілі й методи дещо відрізняються від тих, які відповідають першій стадії тестування. На цій стадії найважливіше

завдання – точно встановити місце перекручень програми або даних, що стало причиною відхилення результатів від еталонних. Тобто визначити, яку частину програми необхідно коректувати. На цій стадії витрати виправдані, а тестування вважається ефективним, якщо воно приводить до повної локалізації помилки, що підлягає виправленню.

Після локалізації і усунення виявлених помилок застосовується контрольне тестування, яке має підтвердити правильність виконаного коректування програми і відсутність раніше виявлених помилок, а також відсутність повторних помилок, які можуть з'явитися під час коректування.

Домашинний контроль програми не гарантує її придатність для роботи, тому необхідний контроль програми за результатами її виконання на ЕОМ. Для кожного класу задач можуть існувати свої особливі способи контролю програм на ЕОМ. Розглянемо один метод контролю, який можна вважати універсальним для всіх класів задач, – метод контрольних тестів.

Тестом називають інформацію, що складається з вхідних даних, спеціально підібраних для програми, що налагоджується, і з відповідних їм еталонних результатів (не тільки остаточних, а й проміжних), що використовуються надалі для контролю правильності роботи програми.

Тестування базується на наступних загальних принципах [28]:

Принцип 1. Процес тестування більш ефективний, якщо проводиться не автором програми.

З визначення тестування як процесу ясно, що тестування тим ефективніше, чим більше помилок виявлено. При створенні програми неможливо свідомо залишати в ній помилки, це означає, що програміст заздалегідь налаштований на їх відсутність, тому підсвідомо не намагається їх знайти.

Це не означає, що програміст не може тестувати свою програму. Що стосується налагодження, то його ефективніше виконує сам автор програми.

Принцип 2. Тестовий набір даних повинен включати два компоненти: опис вхідних даних і опис точного і коректного результату, відповідного цьому набору вхідних даних.

Складність реалізації принципу полягає в тому, що під час тестування програми (модуля) необхідно для кожного вхідного набору даних вручну розрахувати очікуваний результат або знайти припустимий інтервал зміни вихідних даних, а це дуже трудомісткий процес навіть для невеликих програм.

Принцип 3. Необхідно ретельно вивчати результати застосування кожного тесту.

Значну частину всіх помилок можна виявити внаслідок перших тестових прогонів.

Принцип 4. Тести для неправильних і непередбачених вхідних даних повинні розроблятися так само ретельно, як для правильних, передбачених.

При обробці даних в програмі, що тестується, необхідно передбачити діагностику неприпустимих значень даних і повідомлення про причину неможливості їх обробки.

Принцип 5. Необхідно перевіряти не тільки, чи робить програма те, для чого вона призначена, але і чи не робить вона те, що не повинна робити.

Необхідно будь-яку програму перевірити на небажані побічні дії.

Принцип 6. Імовірність присутності не виявлених помилок у частині програми пропорційна до кількості помилок, уже виявлених у цій частині.

Ця властивість помилок групуватися пояснюється тим, що ті частини програми, де під час тестування виявлено більше помилок, або мають слабку ідеологію, або розроблялися програмістами більш низької кваліфікації. З цього принципу витікає практичний висновок: якщо в якій-небудь частині програми виявлено більше помилок, ніж в інших, то її необхідно тестувати більш ретельно.

6.2. Способи тестування

Контроль програми зводиться до того, щоб підібрати таку систему тестів, отримання правильних результатів для якої гарантувало б правильну роботу програми і для інших початкових даних із сфери, вказаної в технічному завданні. Для реалізації методу контрольних тестів мають бути виготовлені або заздалегідь відомі еталонні результати, на основі зіставлення яких з отриманими тестовими результатами можна було б зробити висновок про правильність роботи програми на даному тесті. Еталонні тестові результати для обчислювальних задач можна отримати, здійснюючи обчислення вручну, застосовуючи результати, здобуті раніше на інший ЕОМ, або по іншій програмі, або використовуючи відомі факти, властивості, фізичні закони.

Є три способи тестування: алгоритмічне, аналітичне, змістовне.

Алгоритмічне тестування застосовується для контролю етапів алгоритмізації та програмування. Програмісти проектують тести і готують еталонні результати на етапі алгоритмізації, а використовують їх на етапі налагодження.

Функціональне або аналітичне тестування використовується для контролю вибраного методу розв’язування задачі, правильності його роботи у заданих режимах із установленими діапазонами даних. Тести проектують і створюють на етапі проектування після вибору методу, а використовують їх на останньому етапі налагодження або для аналізу результатів пробного розрахунку; в ході тестування застосовуються ще й якісні оцінки результатів.

Змістовне тестування використовують для перевірки правильності постановки задачі, принципи якої формулюються в технічному завданні. Для контролю застосовуються, як правило, якісні оцінки і статистичні характеристики програми, фізичне значення отриманих результатів і т. ін. У проведенні змістовного тестування активну участь повинні брати замовники або майбутні користувачі програми.

Змістовні і аналітичні тести перевіряють правильність роботи програми загалом або великих її частин, в той час як алгоритмічні тести перевіряють роботу окремих блоків або операторів програми. Алгоритмічні тести, крім

встановлення наявності помилок у програмі, мають давати певну інформацію також про їх місцезнаходження у програмі, тобто локалізувати помилки на наступному етапі. Розробляючи систему тестів, треба намагатися, щоб успішний прогін її на ПК доводив відсутність помилок у програмі (або окремому її блоці), хоч виготовлення і прогін усіх тестів, необхідних для доказу, може зробити етап контролю досить довгим. Тому при розробці системи тестів постає задача мінімізації кількості необхідних тестових результатів, машинного часу і зусиль програміста. Здебільшого в разі використання методу тестування можна говорити про відсутність помилок лише для невеликих блоків (модулів) програми, а для цілої програми обмежуватися ймовірністю відсутності помилок у програмі.

Методи тестування спрямовані на виявлення максимального числа помилок у найбільш важливих режимах функціонування програм при обмежених ресурсах.

Розглянемо детальніше методи тестування, що послідовно застосовуються: статичний, детермінований, стохастичний і в реальному масштабі часу.

Статичне тестування – найбільш формалізоване і автоматизовуване, базується на правилах структурної побудови програм і обробки даних. Перевірка виконання цих правил проводиться без виконання об'єктного коду програми формальним аналізом тексту програми мовою програмування. Статичне тестування реалізовується ручними методами тестування програм, написаних за вимогами структурного і модульного програмування. Це зробило програми зручними для читання і дало змогу провести тестування окремих модулів вручну без використання ЕОМ.

Використання ручних методів тестування досить ефективне, вони сприяють істотному збільшенню продуктивності і підвищенню надійності програм, дозволяють раніше виявити помилки, а отже зменшити вартість виправлення. У разі ручних методів тестування ймовірність того, що при виправленні помилок не вносяться нові помилки, набагато вища.

Основні методи ручного тестування: інспекції початкового тексту і наскрізний перегляд. Оператори і операнди тексту програми аналізуються в символьному вигляді, тому цей метод тестування іноді називають символічним тестуванням.

Детерміноване тестування є найбільш трудомістким і деталізованим, воно вимагає багаторазового виконання програми на ЕОМ з використанням визначених, спеціально підібраних тестових наборів даних. При детермінованому тестуванні контролюється кожна комбінація вхідних даних і відповідні їй результати для виявлення відхилень від еталонних значень, а також кожне затвердження у специфікації тестованої програми.

Детермінований метод тестування внаслідок трудомісткості можливо застосовувати для окремих модулів у процесі збирання програми або для невеликих і нескладних програмних комплексів.

При тестуванні програмного виробу неможливо перебрати всі комбінації початкових даних і проконтролювати результати функціонування

на кожній з них, тому для комплексного тестування програмного виробу застосовується стохастичне тестування.

Стохастичне тестування передбачає використання як початкових даних множини випадкових величин з відповідними розподілами, а для порівняння отриманих результатів використовуються ті самі розподіли випадкових величин.

Стохастичне тестування застосовується, здебільшого, для виявлення помилок, а для діагностики і локалізації помилок необхідно використати детерміноване тестування з конкретними значеннями початкових даних зі сфери вже отриманих випадкових величин. Стохастичне тестування легше інших піддається автоматизації використанням генераторів випадкових величин.

Тестування в реальному масштабі часу є обов’язковим для програмних виробів, призначених для роботи в системах реального часу. У процесі такого тестування перевіряються результати обробки початкових даних з урахуванням часу їх надходження, тривалості і пріоритетності обробки, динаміки використання пам’яті і взаємодії з іншими програмами. Якщо виявлено відхилення результатів виконання програм від очікуваних, то для локалізації помилок фіксують час і переходять до детермінованого тестування.

6.3. Правила тестування

Досвід багатьох програмістів дозволив сформулювати деякі загальні правила тестування:

Прохід ділянок. Кожна лінійна ділянка програми повинна бути обов’язково пройдена при виконанні, принаймні, одного тесту. Оскільки це правило використовується для поблокового контролю, постає питання про автоматизацію обліку пройдених (і непройдених) ділянок програми.

За допомогою спеціально підібраних тестових вхідних даних можна перевірити роботу окремих блоків надійніше, ніж з реальними даними, які отримують блоки від сусідніх блоків, оскільки такі умовні дані дають змогу обійти виконання (і контроль) протестованих раніше ділянок або блоків. Такий незалежний контроль роботи окремих блоків буває можливий, якщо вони володіють властивістю модульності, яка закладається ще під час розробки алгоритму. Якщо виконання якоїсь ділянки змінює порядок виконання або характер роботи інших ділянок, може знадобитися багаторазова перевірка ділянок програми, і навіть перегляд всіх гілок програми; багаторазова перевірка потрібна, зокрема, і для ділянок, що містять змінні з індексами.

Точність перевірки. Контроль арифметичних блоків (як і інших блоків) проводиться шляхом порівняння результатів, отриманих при виконанні блоку, з еталонними результатами. Для арифметичних результатів додатково визначають точність, з якою необхідно порівнювати еталонні і тестові результати, щоб можна було пересвідчитися у правильності роботи блоку.

Складність полягає в тому, що величини, які входять у певний арифметичний вираз, що перевіряється у блоці, залежно від співвідношення їхніх значень і характеру операцій, що виконуються з ними, вкладають різний внесок у результат.

Щоб бути упевненим в тому, що правильний числовий результат, отриманий на ПК, свідчить про правильність програми, необхідно стежити за проміжними результатами обчислень, які не повинні виходити за певний діапазон, що встановлюється залежно від точності обчислень еталонних результатів. Виконання такої вимоги може призвести до необхідності багаторазової перевірки виразу для різних діапазонів даних.

Мінімальність обчислень. Коли тривалість роботи контрольованої програми залежить від яких-небудь параметрів, то під час контролю їх потрібно вибирати такими, щоб вони мінімізували кількість обчислень. До таких параметрів, наприклад, можна віднести крок або відрізок інтегрування, порядок матриці або кількість елементів вектора, довжину символьних рядків, точність для ітераційних обчислень і т. ін. Значення початкових даних треба вибирати такими, щоб виготовлення еталонних результатів вручну було достатньо легким. Наприклад, спочатку можна взяти дані цілочисельними або такими, щоб при перевірці виразів деякі їх доданки, вже перевірені раніше, оберталися на нуль.

Достовірність еталонів. Треба звернути увагу і на достовірність процесу отримання еталонних результатів. Вони повинні обчислюватися не самим програмістом, а кимось іншим, щоб одні й ті самі помилки у завданні на програмування або в алгоритмі не проникли і в програму, і в еталонні результати. Якщо тести готує сам програміст, то є небезпека мимовільної підгонки обчислюваних значень під бажані, отримані раніше на машині. Як еталонні результати можна використати і дані, отримані під час прокручування програми.

Планування. Основою ефективного тестування є його плановість і систематичність. Тільки діючи за планом, що враховує структуру і особливості розроблюваної програми, можна сподіватися швидко виявити всі (або майже всі) помилки у програмі і переконливо продемонструвати замовникові правильність тестованої програми.

У разі планового підходу перевіряється блок за блоком (для алгоритмічного тестування), режим роботи програми за режимом (для функціонального і змістовного контролю). Плануються як загальний підхід до контролю програми і основні його принципи (стратегія контролю), так і особливості реалізації вибраних принципів, наприклад розв’язується задача мінімізації кількості тестів (тактика контролю). Таким чином, заздалегідь встановлюється, що необхідно проконтролювати і як це краще виконати.

Детерміноване тестування, або тестування на певних вхідних значеннях, ґрунтується на двох підходах: структурне тестування (СТ) і функціональне тестування (ФТ).

Структурне тестування (стратегія «білого ящика») передбачає відповідно до логіки програми побудову таких вхідних наборів даних, які під

час багаторазового виконання програми на ЕОМ забезпечили б виконання максимально можливої кількості маршрутів, всіх логічних розгалужень, циклів, усіх операторів, всіх умов (і їх комбінацій) і т. д.

Під час побудови тестових наборів даних за принципом «білого ящика» керуються такими критеріями: покриття операторів, покриття вузлів розгалуження, покриття умов, комбінаторне покриття умов.

Це означає, що кожний оператор і кожний вузол і розгалуження мають бути виконані хоча б один раз, перевірені всі умови та їх комбінації.

Функціональне тестування (стратегія «чорного ящика» або тестування за «входом-виходом») повністю абстрагується від логіки програми: передбачається, що програма – це «чорний ящик», а тестові дані вибираються на основі аналізу вхідних функціональних специфікацій програми.

До стратегії «чорного ящика» належать методи: еквівалентного розбиття; аналізу граничних значень; функціональних діаграм.

Метод еквівалентного розбиття полягає в тому, що виділяються класи еквівалентності шляхом аналізу вхідної умови і розбиттям її на дві або більше групи. Для будь-якої умови існують правильний клас еквівалентності (що представляє правильні вхідні дані програми) і неправильний, тобто помилкові вхідні значення. На основі класів еквівалентності будуються тестові набори. Причому для правильних класів еквівалентності кожний тест повинен покривати якомога більше правильних класів еквівалентності.

Для кожного неправильного класу еквівалентності будується принаймні один тестовий набір.

Аналіз граничних значень передбачає перевірку ситуацій, що виникають на межах і поблизу меж еквівалентного розбиття. Наприклад, якщо правильна область значень є –1,0 до +1,0, то треба передбачити тести –1,0, 1,0, –1,001 і 1,001.

Метод функціональних діаграм полягає в тому, що створюється набір тестів, що визначаються функціональними задачами та їх зв’язками в програмному комплексі. Ці тести мають забезпечити перевірку якості рішення функціональних задач, сформульованих у технічному завданні, перевірку стійкості функціонування за наявності аномалій на вході у програму, тобто в разі аномалій у зовнішньому середовищі. Це особливо важливо під час тестування великих програмних комплексів, коли помилки розв’язування однієї функціональної задачі можуть заблокувати виконання всіх програмних модулів, пов’язаних з нею функціонально.

Упорядкування при тестуванні. Для тестування застосовуються методи, що передбачають упорядкування та систематизацію тестів за різними стратегіями і параметрами, і методи неврегульованого тестування. У разі неврегульованого тестування початкові дані, що імітують зовнішнє середовище, випадковим чином генеруються у всьому діапазоні можливої зміни параметрів, проводиться випадковий перебір значень у довільних комбінаціях різних величин. При цьому багато які значення початкових даних характеризуються малою ймовірністю виявлення помилок і не виправдовують витрат на виконання тестування. Крім того, можливо поява

даних, логічно суперечливих. Проте дані, найбільш важливі з позиції реального використання програм і можливості виявлення помилок, можуть виявитися не охопленими в процесі тестування. При реально існуючих обмеженнях на об’єми тестування його неврегульоване застосування виявляється малоефективним і майже не знаходить застосування.

Прагнення до раціонального використання обмежених ресурсів приводить до систематизації процесу і методів тестування. Методи впорядкованого тестування базуються на виділенні чинників і параметрів, що дають змогу ефективно розподіляти ресурси тестування з урахуванням їх впливу на якість програм. Систематизація може значно змінюватися залежно від етапів тестування, однак можна виділити декілька загальних принципів, на базі яких будуються основні методи тестування. Для упорядкування операцій тестування використовується інформація про структуру програми і процес обробки інформації, про характер зміни і взаємозв’язку змінних, про найбільш вірогідні і важливі комбінації початкових даних, про характеристики помилок і ймовірність їх виявлення і т. ін. Таким чином обмежені ресурси тестування використовуються передусім для виявлення найбільш небезпечних помилок в найбільш важливих режимах функціонування програм. З цією метою послідовно застосовуються методи тестування: статичний, детермінований, стохастичний і в реальному масштабі часу.

У реальних умовах на вхід програми можуть потрапити сильно перекручені або помилкові дані. Програми повинні зберігати свою працездатність після надходження таких даних або відновлювати її при подальшому надходженні даних, що змінюються в заданих межах. Тому тестування необхідно провести не тільки в разі коректних початкових даних, а й у разі перекручених. У зв’язку з цим впорядкований підхід до тестування може базуватися на критеріях досягнення максимальної коректності або максимальної надійності програм.