ТЕМА 10. МЕТОДИ ТЕСТУВАННЯ. СПОСОБИ СКОРОЧЕННЯ КІЛЬКОСТІ ТЕСТОВИХ ВИПАДКІВ

1. Тест дизайн

Тест дизайн - один з початкових етапів тестування програмного забезпечення, етап планування і проектування тестів. Тест дизайн являє собою продумування і написання тестових випадків (testcase), відповідно до вимог проекту, згідно з критеріями якості майбутнього продукту і фінальними цілями тестування.

Цілі тест дизайну - забезпечити покриття функціоналу додатка тестами:

– тести повинні покривати весь функціонал;

– тестів має бути мінімально достатньо. Тест дизайн завдання:

– проаналізувати вимоги до продукту;

– оцінити ризики можливі при використанні продукту;

– написати достатню мінімальну кількість тестів;

– розмежувати тести на приймальні, критичні та інші.

Техніки тест дизайну це рекомендації, поради та правила за якими стоїть розробляти тест для проведення тестування програми. Це не зразки тестів, а тільки рекомендації до застосування. Зокрема різні інженери можуть працюючи під одним і тим же проектом створити різний набір тестів. Правильним буде вважатися той набір тестів, який за меншу кількість перевірок забезпечить більш повне покриття тестами.

Тест-дизайнер - це співробітник, в чиї обов'язки входить створення набору тестових випадків, що забезпечують оптимальне тестове покриття додатки .Тест-дизайнер повинен вибудувати процес тестування всіх найважливіших частин програмного продукту, використовуючи мінімально можливу кількість перевірок. У невеликих командах робота тест-дизайнера часто лягає на плечі пересічного тестувальника, в великих же компаніях функції тестування і тест-дизайну, як правило, чітко розділені між фахівцями.

В результаті ланцюжок тестування виглядає так:

– тест-аналітик виконує аналіз продукту, розбиває його на складові частини, розставляє пріоритети тестування і становить логічну карту додатки;

– тест-дизайнер на підставі інформації, отриманої від аналітика, приступає до розробки тестів;

– тестувальник проводить безпосередньо тестування з уже готовим тест-кейсів.

Лі Копланд (LeeCopeland) виділяє наступні техніки, що використовуються в тест-дизайні:

– Тестування Класами Еквівалентності (EquivalenceClassTesting).

– Тестування Граничних (кордонних) Значень (BoundaryValueTesting).

– Таблиця Прийняття Рішень (DecisionTableTesting).

– Тестування Станів і Переходів (State-TransitionTesting).

– Метод Парного Тестування (Pairwisetesting).

– Доменний аналіз (DomainAnalysisTesting).

– Тестування сценаріїв використання (UseCaseTesting).

Ресурс Про-Тестинг пропонує наступний перелік популярних технік тест-дизайну:

– Еквівалентний Поділ (EquivalencePartitioning - EP).

– Аналіз Граничних Значень (BoundaryValueAnalysis - BVA).

– Причина / Наслідок (Cause / Effect - CE).

– Передбачення помилки (ErrorGuessing - EG).

– Вичерпне тестування (ExhaustiveTesting - ET)

2. Класи еквівалентності

Клас еквівалентності (equivalenceclass) - одне або кілька значень введення, до яких програмне забезпечення застосовує однакову логіку.

Техніка аналізу класів еквівалентності - це техніка, при якій ми поділяємо функціонал (часто діапазон можливих значень, що вводяться) на групи еквівалентних за своїм впливом на систему значень. Такий поділ допомагає переконатися в правильному функціонуванні цілої системи - одного класу еквівалентності, перевіривши тільки один елемент цієї групи. Ця техніка полягає в розбитті всього набору тестів на класи еквівалентності з подальшим скороченням числа тестів.

Техніка рекомендує проведення тестів для всіх класів еквівалентності, хоча б по одному тесту для кожного класу. Техніка аналізу класів еквівалентності прагне не тільки скорочувати кількість тестів, але і зберігати прийнятне тестове покриття.

Ознаки еквівалентності тестів:

– спрямовані на пошук однієї і тієї ж помилки;

– якщо один з тестів виявляє помилку, інші швидше за все, теж її виявлять;

– якщо один з тестів не виявляє помилку, інші, швидше за все, теж її не виявлять;

– тести використовують схожі набори вхідних даних;

– для виконання тестів ми здійснюємо одні і ті ж операції;

– тести генерують однакові вихідні дані або призводять додаток в один і той же стан;

– всі тести призводять до спрацьовування одного і того ж блоку обробки помилок;

– жоден з тестів не призводить до спрацьовування блоку обробки помилок.

Алгоритм використання техніки аналізу класів еквівалентності:

1. Визначити класи еквівалентності.

Це головний крок техніки, тому що багато в чому від нього залежить ефективність її застосування.

2. Вибрати одного представника від кожного класу еквівалентності. На цьому етапі слід вибрати один тест з еквівалентного набору тестів.

3. Виконання тестів.

На цьому кроці слід виконати тести від кожного класу еквівалентності.

Якщо є час, можна протестувати ще кілька представників від кожного класу еквівалентності. Слід мати на увазі, при правильному визначенні класів еквівалентності додаткові тести швидше за все будуть надмірними і дадуть такий же результат.

На практиці класи еквівалентності обов'язкові при тестуванні всіляких форм і полів введення.

До плюсів техніки можна віднести відсіювання величезної кількості значень введення, використання яких просто безглуздо. До мінусів можна віднести неправильне використання техніки, через що є ризик втратити баги. Таким чином, техніка аналізу класів еквівалентності хоча і значно скорочує кількість тестів необхідних для перевірки функціоналу і час, з іншого боку в недосвідчений руках може стати інструментом який не тільки не допоможе знайти дефекти, але і складе хибне уявлення про покриття додатку тестами.

3. Метод кордонних (граничних) умов.

Ця техніка заснована на тому факті, що одним з найслабших місць будь-якого програмного продукту є область граничних значень.

Фактично, граничні значення - це ті місця, в яких один клас еквівалентності переходить в інший. Граничне тестування також може включати тести, що перевіряють поведінку системи на вхідних даних, що виходять за допустимий діапазон значень. При цьому система повинна певним (заздалегідь обумовленим) способом обробляти такі ситуації. Наприклад, за допомогою виняткової ситуації або повідомлення про помилку.

На кожен з кордонів створюється 3 тест-кейса:

– перший перевіряє значення кордону,

– другий - значення нижче кордону,

– третій - значення вище кордону.

Алгоритм використання техніки граничних значень:

1. Виділити класи еквівалентності.

Як і в попередній техніці, цей крок є дуже важливим і від того, наскільки правильним буде розбиття на класи еквівалентності, залежить ефективність тестів граничних значень.

2. Визначити граничні значення цих класів.

3. Визначити, до якого класу буде відноситися кожен кордон.

4. Провести тести по перевірці значення до кордону, на кордоні і відразу після кордону.

Кількість тестів для перевірки граничних значень буде дорівнювати кількості кордонів, помноженої на 3. Рекомендується перевіряти значення впритул до кордону. Наприклад, є діапазон цілих чисел, кордон знаходиться в числі 100. Таким чином, будемо проводити тести з числом 99 (до кордону), 100 (сам кордон), 101 (після кордону).

4. Таблиця Прийняття Рішень (DecisionTableTesting).

Таблиці рішень - це зручний інструмент для фіксування вимог і опису функціональності додатку. Таблицями дуже зручно описувати бізнес-логіку програми, і вони можуть служити відмінною основою для створення тест- кейсів. Таблиці рішень

описують логіку додатка, ґрунтуючись на умовах системи, що характеризують її стану. Кожна таблиця повинна описувати один стан системи.

Таблиця рішень - спосіб компактного представлення моделі зі складною логікою; інструмент для упорядкування складних бізнес вимог, які повинні бути реалізовані в продукті. Це взаємозв'язок між множиною умов і дій. У таблицях рішень представлений набір умов, одночасне виконання яких повинно привести до певного дії.

Таблиця прийняття рішень, як правило, поділяється на 4 квадранта:

Умови	**Варіанти виконання дій**
Дії	**Необхідність дій**

Умови - список можливих умов.

Варіанти виконання дій - комбінація з виконання і/або невиконання умов цього списку.

Дії - список можливих дій.

Необхідність дій - вказівка треба чи не треба виконувати відповідну дію для кожної з комбінацій умов.

Шаблон таблиці рішень має вигляд:

	Правило 1	**Правило 2**	**…**	**Правило k**
Умови				
Умова 1				
Умова 2				
…				
Умова m				
Дії				
Дія 1				
дія 2				
…				
Дія n				

Кожне правило містить набір умов, для яких виконуються певні дії. Матриця заповнюється можливими комбінаціями вхідних умов. При цьому кожна клітинка може мати одне з трьох значень:

- так,
- ні,
- неважливо (може бути позначено текстурною заливкою).

Використання значення "неважливо" допомагає скоротити кількість описуваних випадків, при цьому забезпечити той же рівень покриття, що і при описі всіх допустимих комбінацій умов.

Значення таблиці рішень можна отримати із використанням методу Причина/Наслідок (Cause / Effect - CE). Метод був розроблений фірмою IBM і являє собою оцінку причинно-наслідкових зв’язків, де вхідними даними є причини, а вихідними - наслідки. Метод функціональних діаграм є формальний, мову, на якому транслюються специфікації, написані на природній мові. Побудова тестів здійснюється в кілька етапів для зменшення громіздкості і трудомісткості, специфікація розробляється на робочій ділянці в кожній робочій ділянці, визначається причини, які характеризують окрема вхідна умова або визначають деякі залишкові дії програми. Кожній причини і наслідку присвоюється окремий номер. Причинно-наслідкові зв'язки відображаються у вигляді булевого відношення або графа з двома вузлами і дугою, де під дугою розуміються логічні умови «і», «або», «не». Отриманий граф і отримає назву функціональної діаграми. Обмежені і синтаксично неможливі комбінації причин і наслідки оформляються окремо, як додаток до функціональної діаграми і розглядаються самостійно. Алгоритм методу полягає в наступному:

1. Факторизація специфікації (розбиття її на окремі «робочі» ділянки).
2. Визначення причин та наслідків:
   - причина – окрема вхідна умова або клас еквівалентності даних умов;
   - наслідок – вихідна умова або перетворення системи;
3. Побудова булевого графа, який зв'язує причини і наслідки.
4. Встановлення обмежень на комбінації причин і наслідків (коментарі до діаграми).
5. Побудова таблиці істинності (таблиці рішень).
6. Перетворення кожного стовпця таблиці в тест – формування таблиці рішень. Кількість отриманих шляхів відповідає кількості тест-кейсів.

5. Тестування Станів і Переходів (State-TransitionTesting).

Система переходить в той чи інший стан в залежності від того, які операції над нею виконуються.

Стан (state, представлене у вигляді кола на діаграмі) - це стан додатку, в якому воно очікує одну або більше подій. Стан пам'ятає вхідні дані, отримані до цього, і показує, як додаток буде реагувати на отримані події. Події можуть викликати зміну стану і / або ініціювати дії.

Перехід (transition, представлено у вигляді стрілки на діаграмі) - це перетворення одного стану в інший, що відбувається за подією.

Подія (event, представлене ярликом над стрілкою) - це щось, що змушує додаток поміняти свій стан. Події можуть надходити ззовні додатку, через інтерфейс самого додатка. Сам додаток також може генерувати події (наприклад, подія «закінчився таймер»). Коли відбувається подія, додаток може поміняти (або не міняти) стан і виконати (або не виконати) дію. Події можуть мати параметри (наприклад, подія «Оплата» може мати параметри «Готівка», «Чек», «Прибуткова карта» або «Кредитна картка»).

Дія (action, представлено після «/» в ярлику над переходом) ініціюється зміною стану («надрукувати квиток», «показати на екрані» і ін.). Зазвичай дії створюють щось, що є вихідними даними системи. Дії виникають при переходах, самі по собі стани пасивні.

Точка входу позначається чорним кружком.

Точка виходу показується на діаграмі у вигляді мішені.

6. Тестування сценаріїввикористання (UseCaseTesting).

UseCase описує сценарій взаємодії двох і більше учасників (як правило - користувача і системи). Користувачем може виступати як людина, так і інша система. Для тестувальників UseCase є чудовою базою для формування тестових сценаріїв (тест-кейсів), так як вони описують, в якому контексті повинно проводитися кожну дію користувача. UseCase, за замовчуванням, є тестовими вимогами, так як в них завжди зазначена мета, якої потрібно досягти, і кроки, які треба для цього відтворити.

7. Передбачення помилки (ErrorGuessing - EG).

Це коли тест аналітик використовує свої знання системи і здатність до інтерпретації специфікації на предмет того, щоб "вгадати" за яких вхідних умовах система може видати помилку. Наприклад, специфікація каже: "користувач повинен ввести код". Тест аналітик, буде думати: "Що, якщо я не введу код?", "Що, якщо я введу неправильний код?", і так далі. Це і є передбачення помилки.

8. Вичерпне тестування (ExhaustiveTesting – ET.)

Вичерпне тестування (ExhaustiveTesting - ET) - це крайній випадок. В межах цієї техніки необхідно перевірити всі можливі комбінації вхідних значень, і в принципі, це повинно знайти всі проблеми. На практиці застосування цього методу не представляється можливим, через величезну кількість вхідних значень.

9. Способи скорочення кількості тестових випадків.

9.1 Метод Парного Тестування (Pairwisetesting).

Метод парного тестування заснований на наступній ідеї: переважна більшість багів виявляється тестом, яким перевіряють один параметр або поєднання двох. Помилки, причиною яких стали комбінації трьох і більше параметрів, як правило, значно менш критичні.

Припустимо, що ми маємо систему, яка залежить від декількох вхідних параметрів. Так, ми можемо перевірити всі можливі варіанти поєднання цих параметрів. Але навіть для випадку, коли кожен з 10 параметрів має всього два значення (On/Off), ми отримуємо 210 = 1024 комбінацій! Використовуючи метод парного тестування, ми не тестуємо всі можливі поєднання вхідних параметрів, а складаємо тестові набори так, щоб кожне значення параметра хоча б один раз поєднувалося з кожним значенням інших тестових параметрі. Таким чином, метод істотно скорочує кількість тестів, а значить, і час тестування.

Але родзинка методу не в тому, щоб перебрати всі можливі пари параметрів, а в тім, щоб підібрати пари, що забезпечують максимально ефективну перевірку при мінімальній кількості виконуваних тестів. З цим завданням допомагають впоратися

математичні методи, звані ортогональними таблицями. Також існує ряд інструментів, які допомагають автоматизувати цей процес (наприклад, AllPairs).

9.2 Доменний аналіз (DomainAnalysisTesting).

Це техніка заснована на розбитті діапазону можливих значень змінної (або змінних) на під діапазони (або домени), з подальшим вибором одного або декількох значень з кожного домену для тестування. Багато в чому доменне тестування перетинається з відомими техніками розбиття на класи еквівалентності і аналізу граничних значень. Але доменне тестування не обмежується перерахованими техніками. Воно включає в себе як аналіз залежностей між змінними, так і пошук тих значень змінних, які несуть в собі великий ризик (не тільки на кордонах).

Стратегія доменного тестування включає відповіді на питання:

1. Який домен ми тестуємо?

Будь-який домен, який ми тестуємо, має деяку функціональність введення і функціональність виведення. Будуть введені деякі вхідні змінні, і відповідний вихід повинен бути перевірений (рисунок 1).

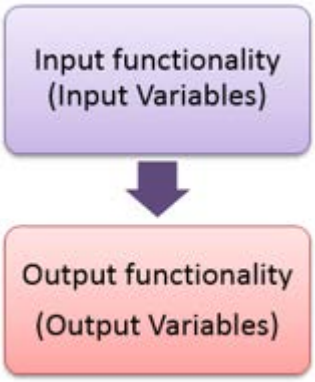

Рисунок1 – Вхідні-вихідні дані домену

2. Як згрупувати значення в класи?

Поділ деяких значень означає розбиття їх на непересічні підмножини.

3. Які значення класів повинні бути перевірені?

Як мінімум перевіряються значення на кордонах класів. Способи значень всередині класів будуть розглянуті додатково.

4. Як визначити результат?

Досягнення функціональності залежить не тільки від результатів сформованих сценаріїв. Задані дані і очікуваний результат дадуть нам результати, і це вимагає знання предметної області.

Інструкція доменного тестування може бути описана наступним чином:

1. Визначте потенційно цікаві змінні.
2. Визначте змінні, які ви можете проаналізувати зараз, і впорядкуйте їх (від найменшого до найбільшого і навпаки).
3. Створіть і визначте граничні значення і значення класу еквівалентності.
4. Визначте вторинні вимірювання і проаналізуйте кожне класичним способом.
5. Визначте і перевірте змінні, які містять результати (вихідні змінні).
6. Оцініть, як програма використовує значення цієї змінної.
7. Визначте додаткові потенційно пов'язані змінні для комбінованого тестування.
8. Уявіть собі ризики, які не обов'язково відповідають очевидному виміру.
9. Визначте і перерахуйте непроаналізовані змінні. Зберіть інформацію для подальшого аналізу.
10. Підведіть підсумок вашого аналізу з таблицею ризику/еквівалентності.

9.3. Input-OutputAnalysis

Метод Input-OutputAnalysis передбачає аналіз вмісту чорного ящика. Вхідні дані поділяються на класи відповідно до впливу на виходи. Припустимо P(A, B, C) - вхідними даними є A, B, C, але кожне з них впливає на різні вихідні дані (рисунок 2).

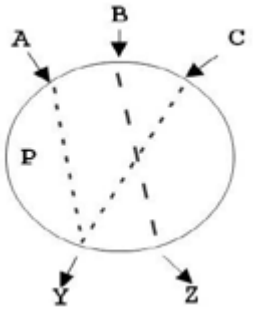

Рисунок 2 – Результати аналізу взаємовпливу вхідних та вихідних даних

Якщо дані представлені наступними значеннями A={a1, a2, a3}, B={b1, b2, b3, b4}, C={c1, c2} то Input-OutputAnalysis дозволить сформувати набори тестів, наведені на рисунку 3.

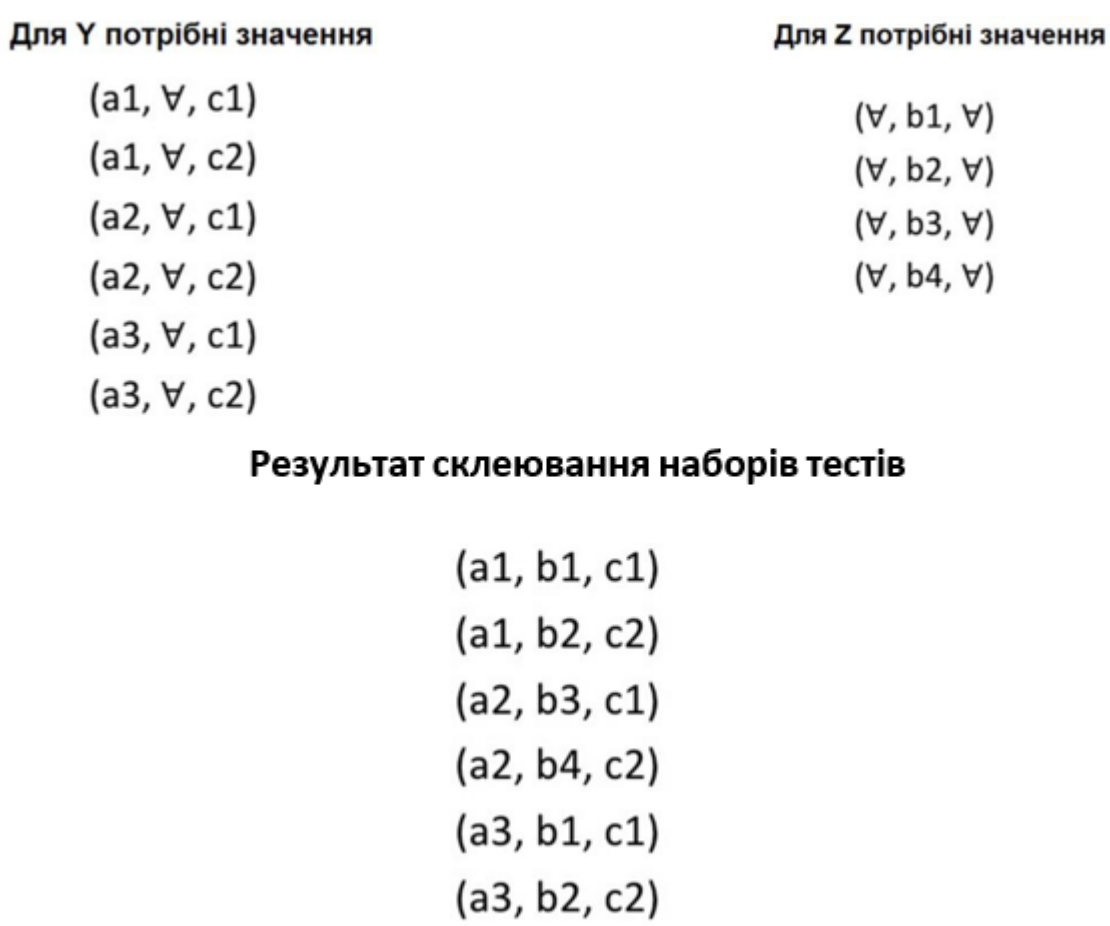

Рисунок 3 – Приклад застосування методу Input-Output Analysis