МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія програмування

**Реферат**

з курсу «Основи інформатики, програмування а алгоритмічні мови»

на тему «Методи тестування програмного забезпечення»

Корнієнко Анастасії Яківни

Кордонської Анастасії Олександрівни

студенток 1П-19

варіант:

**Викладач:**

Марченко С. В.

Черкаси 2020

Зміст

Вступ

1. Філософія тестування
2. Історія розвитку тестування ПЗ
3. Методи тестування ПЗ
4. Види тестування ПЗ
5. Рівні тестування

Висновок

Джерела

Вступ

**Тестування програмного забезпечення** — це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки.

Також, це процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Може оцінюватись:

* відповідність вимогам, якими керувалися проектувальники та розробники
* правильна відповідь для усіх можливих вхідних даних
* виконання функцій за прийнятний час
* практичність
* сумісність з програмним забезпеченням та операційними системами
* відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонентів практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Тестування може проводитись, як тільки створено виконуваний код. Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог, гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

1.Філософія тестування

Тестування програмного забезпечення охоплює цілий ряд видів діяльності, дуже аналогічний послідовності процесів розробки програмного забезпечення. Сюди входять постановка задачі для тесту, проектування, написання тестів, тестування тестів і, нарешті, виконання тестів та вивчення результатів тестування. Вирішальну роль грає проектування тесту. Можливий цілий спектр підходів до вироблення, або стратегії проектування тестів, зображений на рис.2. Щоб орієнтуватися в стратегіях проектування тестів, варто розглянути два крайніх підходу, що знаходяться на кордонах спектру. Слід відзначити також, що багато хто з тих, хто працює в цій області, часто впадають в одну чи іншу крайність.

Прихильник (або прихильниця) підходу, відповідного лівої межі спектру, проектує свої тести, досліджуючи зовнішні специфікації або специфікації сполучення програми або модулі, які він тестує. Програму він розглядає як чорний ящик. Позиція його така: "Мене не цікавить, як виглядає ця програма і чи виконав я всі команди або всі шляхи. Я буду задоволений, якщо програма буде вести себе так, як зазначено у специфікаціях ". Його ідеал - перевірити всі можливі комбінації і значення на вході.

Прихильник підходу, відповідного іншого кінця спектра, проектує свої тести, вивчаючи логіку програми. Він починає з того, що прагне підготувати достатню кількість тестів для того, щоб кожна команда була виконана принаймні один раз. Якщо він трохи більш досвідчений, то проектує тести так, щоб кожна команда умовного переходу виконувалася в кожному напрямку хоча б раз. Його ідеал - перевірити кожен шлях, кожну гілку алгоритму. При цьому його зовсім (або майже зовсім) не цікавлять специфікації.

Жодна з цих крайнощів не є хорошою стратегією. Читач, однак, вже, мабуть, зауважив, що перша з них, а саме та, відповідно до якої програма розглядається як чорний ящик, краще. На жаль, вона страждає тим недоліком, що абсолютно нездійсненна. Розглянемо спробу тестування тривіальної програми, отримує на вході три числа і обчислює їх середнє арифметичне. Тестування цієї програми для всіх значень вхідних даних неможливо. Навіть для машини з відносно низькою точністю обчислень кількість тестів обчислювалося б мільярдами. Навіть якби ми обчислювальну потужність, достатню для виконання всіх тестів в розумний час, ми витратили б на кілька порядків більше часу для того, щоб ці тести підготувати, а потім перевірити. Такі програми, як системи реального часу, операційні системи і програми управління даними, які зберігають "пам'ять" про попередні вхідних даних, ще гірше. Нам треба було б тестувати програму не тільки для кожного вхідного значення, але і для кожної послідовності, кожної комбінації вхідних даних. Тому вичерпне тестування для всіх вхідних даних будь-якої розумної програми нездійсненне.

Ці міркування приводять до другого фундаментальному принципу тестування: тестування - проблема значною мірою економічна. Оскільки вичерпне тестування неможливе, ми повинні обмежитися чимось меншим. Кожен тест повинен давати максимальну віддачу в порівнянні з нашими витратами. Ця віддача вимірюється ймовірністю тою, що тест виявить не виявлену перш помилку. Витрати вимірюються часом і вартістю підготовки, виконання та перевірки результатів тесту. Вважаючи, що витрати обмежені бюджетом та графіком, можна стверджувати, що мистецтво тестування, по суті, є мистецтвом відбору тестів з максимальною віддачею. Більше того, кожен тест повинен бути представником деякого класу вхідних значень, так щоб його правильне виконання створювало у нас деяку переконаність у тому, що для певного класу вхідних даних програма буде виконуватися правильно. Це звичайно вимагає деякого знання алгоритму і структури програми, і ми, таким чином, зміщується до правого кінця спектра.

2.Історія розвитку тестування ПЗ

Перші програмні системи розроблялися в рамках програм наукових досліджень або програм для потреб міністерств оборони. Тестування таких продуктів проводилося строго формалізовано із записом всіх тестових процедур, тестових даних, отриманих результатів. Тестування виділялося в окремий процес, який починався після завершення кодування, але при цьому, як правило, виконувалося тим же персоналом.

У 1960-х багато уваги приділялося «вичерпному» тестуванню, яке повинно проводитися з використанням усіх шляхів у коді або всіх можливих вхідних даних. Було відзначено, що в цих умовах повне тестування ПЗ неможливе, тому що, по-перше, кількість можливих вхідних даних дуже велика, по-друге, існує безліч шляхів, по-третє, складно знайти проблеми в архітектурі та специфікаціях. З цих причин «вичерпне» тестування було відхилено й визнано теоретично неможливим.

На початку 1970-х тестування ПЗ розглядалося як «процес, спрямований на демонстрацію коректності продукту» або як «діяльність з підтвердження правильності роботи ПЗ». У програмній інженерії, яка в той час зароджувалася, верифікація ПЗ визначалася як «доказ правильності». Хоча концепція була теоретично перспективною, на практиці вона вимагала багато часу й не охоплювала всі аспекти тестування. Було вирішено, що доказ правильності — неефективний метод тестування ПЗ. Однак, у деяких випадках демонстрація правильної роботи використовується і в наші дні, наприклад, приймально-здавальні випробування. У другій половині 1970-х тестування представлялося як виконання програми з наміром знайти помилки, а не довести, що вона працює. Успішний тест — це тест, який виявляє раніше невідомі проблеми. Даний підхід цілком протилежний попередньому. Зазначені два визначення являють собою «парадокс тестування», в основі якого лежать два протилежних твердження: з одного боку, тестування дозволяє переконатися, що продукт працює добре, а з іншого — виявляє помилки у ПЗ, показуючи, що продукт не працює. Друга мета тестування є більш продуктивною з точки зору поліпшення якості, оскільки не дозволяє ігнорувати недоліки ПЗ.

У 1980-х тестування розширилося таким поняттям як запобіганням дефектам. Проектування тестів — найбільш ефективний з відомих методів запобігання помилок. В цей же час почали висловлюватися думки, що необхідна методологія тестування, зокрема, що тестування повинно включати перевірки впродовж усього циклу розроблення, при цьому це має бути керований процес. В ході тестування треба перевірити не тільки зібрану програму, але й вимоги, код, архітектуру, самі тести. «Традиційне» тестування, яке існувало до початку 1980-х, відносилося тільки до скомпільованої, готової системи (зараз це зазвичай називається системне тестування), але надалі тестувальники стали залучатися в усі аспекти життєвого циклу розроблення. Це дозволяло раніше знаходити проблеми у вимогах та архітектурі й тим самим скорочувати терміни та бюджет розроблення. У середині 1980-х з'явилися перші інструменти для автоматизованого тестування. Передбачалося, що комп'ютер зможе виконати більше тестів, ніж людина, причому зробить це більш надійно. Спочатку ці інструменти були вкрай простими й не мали можливості написання сценаріїв на скриптових мовах.

На початку 1990-х у поняття «тестування» стали включати планування, проектування, створення, підтримку й виконання тестів та тестових оточень, а це означало перехід від тестування до забезпечення якості, що охоплює весь цикл розроблення ПЗ. У цей час починають з'являтися різні програмні інструменти для підтримки процесу тестування: більш просунуті середовища для автоматизації з можливістю створення скриптів і генерації звітів, системи управління тестами, ПЗ для проведення навантажувального тестування. У середині 1990-х з розвитком Інтернету й розробленням великої кількості веб-застосунків особливої популярності стало набувати «гнучке тестування» (за аналогією з гнучкими методологіями програмування).

У 2000-х з'явилося ще більш широке визначення тестування, коли в нього було додано поняття «оптимізація бізнес-технологій». BTO направляє розвиток інформаційних технологій згідно з цілями бізнесу. Основний підхід полягає в оцінці та максимізації значущості всіх етапів життєвого циклу розроблення ПЗ для досягнення необхідного рівня якості, продуктивності, доступності.

3.Методи тестування ПЗ

Тестова діяльність, що пов'язана з аналізом результатів розробки програмного забезпечення, називається статичним тестуванням. Воно передбачає перевірку програмних кодів, контроль та перевірку програми без запуску на комп'ютері. Тестова діяльність, що передбачає експлуатацію програмного продукту, називається динамічним тестуванням. Динамічне та статичне тестування доповнюють одне одного.

На етапі **статичного тестування** перевіряється вся документація, отримана як результат життєвого циклу програми. Це і технічне завдання, і специфікація, і вихідний текст програми на мові програмування. Вся документація аналізується на предмет дотримання стандартів програмування. У результаті статичної перевірки встановлюється, наскільки програма відповідає заданим критеріям та вимогам замовника. Усунення неточностей та помилок у документації — запорука того, що створюваний програмний засіб має високу якість.

**Динамічні** методи застосовуються в процесі безпосереднього виконання програми. Коректність програмного засобу перевіряється на безлічі тестів або наборів підготовлених вхідних даних. При прогоні кожного тесту збираються та аналізуються дані про відмови та збої в роботі програми.

**Тестування «білої скриньки**

*Відома:* внутрішня структура програми.  
*Досліджуються*: внутрішні елементи програми і зв'язки між ними.

Об'єктом тестування тут є не зовнішня, а внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним. Зазвичай аналізуються керуючі зв'язки елементів, рідше — інформаційні зв'язки. Тестування за принципом «білої скриньки» характеризується ступенем, в якому тести виконують або покривають логіку (вихідний текст) програми.

**Особливості тестування «білої скриньки»**

Зазвичай тестування «білої скриньки» засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

* Гарантується перевірка всіх незалежних маршрутів програми.
* Знаходяться гілки True, False для всіх логічних рішень.
* Виконуються всі цикли (у межах їхніх кордонів та діапазонів).
* Аналізується правильність внутрішніх структур даних.

***Недоліки тестування «білої скриньки»:***

* Кількість незалежних маршрутів може бути дуже велика.
* Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.
* У програмі можуть бути пропущені деякі маршрути.
* Не можна виявити помилки, поява яких залежить від даних.

**Переваги тестування «білої скриньки»** пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

* Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.
* Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.
* При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).
* Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Кожна з цих причин є аргументом для проведення тестування за принципом «білої скриньки». Тести «чорної скриньки» не зможуть реагувати на помилки таких типів.

**Тестування «чорної скриньки**

*Відомі:* функції програми.

*Досліджується:* робота кожної функції на всій області визначення.

Основне місце програми тестів «чорної скриньки» — інтерфейс ПЗ.

Ці тести демонструють:

* Як виконуються функції програми.
* Як приймаються вихідні дані.
* Як виробляються результати.
* Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе. Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 1010. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми. Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

* Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми;
* Набір, утворений такими вхідними даними, які демонструють дефекти програми.

Будь-який спосіб тестування «чорної скриньки» повинен:

* Виявити такі вхідні дані, які з високою ймовірністю належать набору IT;
* Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

* Некоректних чи відсутніх функцій;
* Помилок інтерфейсу;
* Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;
* Помилок характеристик (необхідна ємність пам'яті і т. д.);
* Помилок ініціалізації та завершення.
* Подібні категорії помилок способами «білої скриньки» не виявляються.

Види тестування ПЗ

**За ступенем автоматизації**:

* Ручне тестування (manual testing)
* Автоматизоване тестування (automated testing)
* Напівавтоматизоване тестування (semiautomated testing)

**За ступенем підготовленості до тестування**:

* Тестування по документації (formal testing)
* Тестування ad hoc або інтуїтивне тестування (ad hoc testing) — тестування без тест плану та документації, що базується на методиці передбачення помилки на власному досвіді тестувальника.

**За знанням системи**:

* Тестування чорної скриньки (black box)
* Тестування білої скриньки (white box)
* Тестування сірої скриньки (grey box)

**За ступенем ізольованості компонентів**:

* Компонентне (модульне) тестування (component/unit testing)
* Інтеграційне тестування (integration testing)
* Системне тестування (system/end-to-end testing)

**За часом проведення тестування**:

* Альфа-тестування (alpha testing)
* Тестування при прийманні або Димове тестування (smoke testing)
* Тестування нової функціональності (new feature testing)
* Регресивне тестування (regression testing)
* Тестування при здачі (acceptance testing)
* Бета-тестування (beta testing)

**За об'єктом тестування**:

* Функціональне тестування (functional testing)
* Тестування продуктивності (performance testing)
* Навантажувальне тестування (load testing)
* Стрес-тестування (stress testing)
* Тестування стабільності (stability/endurance/soak testing)
* Тестування зручності використання або Юзабіліті-тестування (usability testing)
* Тестування інтерфейсу користувача (UI testing)
* Тестування безпеки (security testing)
* Тестування локалізації (localization testing)
* Тестування сумісності (compatibility testing)

**За ознакою позитивності сценаріїв**:

* Позитивне тестування (positive testing)
* Негативне тестування (negative testing)

4.Рівні тестування

**Модульне тестування**

Відноситься до тестів, які перевіряють функціональність певного розділу коду, зазвичай на функціональному рівні. В об’єктно-орієнтованому середовищі, це, як правило, тестування на рівні класу, а мінімальні модульні тести містять у собі конструктори та деструктори.

Такі типи тестів зазвичай пишуться розробниками під час роботи над кодом (стиль «білої скриньки»), щоб впевнитись, що дана функція працює так, як очікувалося. Одна функція може мати кілька тестів, щоб переглянути всі випадки використання коду. Модульне тестування саме по собі не може перевірити функціонування частини ПЗ, а використовується щоб гарантувати, що основні блоки ПЗ працюють незалежно один від одного.

Модульне тестування — це процес розробки ПЗ, що включає в себе синхронізовані застосування широкого спектра для запобігання дефектів та для виявлення стратегій з метою зниження ризиків розробки ПЗ, часу та витрат. Воно виконується розробником ПЗ або інженером, під час будівельної фази життєвого циклу розробки ПЗ. Модульне тестування спрямоване на усунення помилок проектування. Ця стратегія спрямована на підвищення якості одержуваного ПЗ, до такого рівня, як вимагає процес контролю якості.

Залежно від очікуваної організації розробки ПЗ, модульне тестування може включати статичний аналіз коду, аналіз потоку даних аналізу метрик, експертні оцінки коду, аналізу покриття коду та інші методи перевірки ПЗ.

**Інтеграційне тестування**

Інтеграційне тестування є типом тестування ПЗ, яке прагне перевірити інтерфейси між компонентами від програмного дизайну. Програмні компоненти можуть бути інтегровані як в рамках ітеративного підходу, так і всі разом.

Інтеграційне тестування працює над виявленням дефектів у інтерфейсах та взаємодії інтегрованих компонентів (модулів). Воно проводиться до тих пір, поки великі групи тестованих компонентів ПЗ, які відповідають потрібній архітектурі, починають працювати як система.

**Рівні інтеграційного тестування:**

* компонентний інтеграційний рівень (Component Integration testing). Перевіряється взаємодія між компонентами системи після проведення компонентного тестування;
* системний інтеграційний рівень (System Integration Testing). Перевіряється взаємодія між різними системами після проведення системного тестування.

**Підходи до інтеграційного тестування:**

* знизу вгору (Bottom Up Integration). Усі изько рівневі модулі, процедури або функції збираються воєдино і потім тестуються. Після чого збирається наступний рівень модулів для проведення інтеграційного тестування. Даний підхід вважається корисним, якщо всі або практично всі модулі розроблюваного рівня готові. Також даний підхід допомагає визначити за результатами тестування рівень готовності додатків;
* зверху вниз (Top Down Integration). У першу чергу тестуються компоненти верхнього рівня ієрархії об’єктів з використанням заглушок замість компонентів більш низького рівня;
* великий вибух («Big Bang» Integration). Усі або практично усі розроблені модулі збираються разом у вигляді закінченої системи або її основної частини й потім проводиться інтеграційне тестування. Такий підхід дуже хороший для збереження часу. Проте, якщо тест кейси та їхні результати записані неправильно, то сам процес інтеграції дуже ускладниться, що стане перепоною для команди тестування при досягненні основної мети інтеграційного тестування.

**Системне тестування**

Тестує інтегровану систему для перевірки відповідності всім вимогам. Крім того, системне тестування ПЗ повинно гарантувати, що програма працює так, як очікувалося, а також, що її не можна знищити або пошкодити її робоче середовище, яке викличе процеси в цьому середовищі, що переведуть систему в неробочий стан. Системне інтеграційне тестування перевіряє, чи система інтегрується в будь-яку зовнішню систему (або системи) відповідно до системних вимог.

* Альфа-тестування — імітація реальної роботи з системою штатними розробниками або реальна робота з системою потенційними користувачами/замовником. Найчастіше альфа-тестування проводиться на ранній стадії розробки продукту, але у деяких випадках може застосовуватися для закінченого продукту як внутрішнього приймального тестування. Іноді альфа-тестування виконується під відлагоджувачем або з використанням середовища, яке допомагає швидко виявляти знайдені помилки. Виявлені помилки можуть бути передані тестувальникам для додаткового дослідження у середовищі, подібному тому, в якому буде використовуватися програма.
* Бета-тестування — у деяких випадках виконується поширення версії з обмеженнями (за функціональністю або часом роботи) для певної групи осіб, з тим щоб переконатися, що продукт містить достатньо мало помилок. Іноді бета-тестування виконується для того, щоб отримати зворотній зв’язок про продукт від його майбутніх користувачів.

Часто для вільного/відкритого ПЗ стадія альфа-тестування характеризує функціональне наповнення коду, а бета-тестування — стадію виправлення помилок. При цьому, як правило, на кожному етапі розробки проміжні результати роботи доступні кінцевим користувачам.

**Тестові скрипти**

Тестувальники використовують тестові скрипти на різних рівнях: як у модульному, так і в інтеграційному та системному тестуванні. Тестові скрипти, як правило, пишуться для перевірки компонентів, у яких найбільш висока ймовірність появи відмов або вчасно не знайдена помилка може бути дорогою.

**Покриття коду**

Покриття коду, за своєю суттю, є тестуванням методом білого ящика. Тестоване ПЗ збирається зі спеціальними налаштуваннями або бібліотеками й/або запускається в особливому середовищі, в результаті чого для кожної використовуваної (виконуваної) функції програми визначається місцезнаходження цієї функції у вихідному коді. Цей процес дозволяє розробникам та фахівцям із забезпечення якості визначити частини системи, які, при нормальній роботі, використовуються дуже рідко або ніколи не використовуються (такі як код обробки помилок тощо). Це дозволяє зорієнтувати тестувальників на тестування найбільш важливих режимів.

Як правило, інструменти та бібліотеки, які використовуються для отримання покриття коду, вимагають значних витрат продуктивності та/або пам’яті, неприпустимих при нормальному функціонуванні ПЗ. Тому вони можуть використовуватися тільки в лабораторних умовах.

**Приймальне тестування**

Формальний процес тестування, який перевіряє відповідність системи вимогам і проводиться з метою: визначення чи задовольняє система приймальним критеріям; винесення рішення замовником або іншою уповноваженою особою приймається додаток чи ні.

Приймальне тестування виконується на основі набору типових тестових випадків та сценаріїв, розроблених на основі вимог до даного додатку. Рішення про проведення приймального тестування приймається тоді, коли: продукт досяг необхідного рівня якості; замовник ознайомлений з Планом приймальних Робіт (Product Acceptance Plan) або іншим документом, де описаний набір дій, пов’язаних з проведенням приймального тестування, дата проведення, відповідальні тощо.

Фаза приймального тестування триває до тих пір, доки замовник не виносить рішення про відправлення програми на доопрацювання або видачі додатка.

Висновок

Розглянувши методи діагностування і тестування ПЗ стає зрозумілим, що жоден з них не є універсальним і має певні недоліки. На сьогодні у використанні процесу тестування переважають три основні підходи: функційне тестування, структурне тестування, тестування об’єктно-орієнтованого ПЗ. Всі вони мають свої особливості, але використання окремого із згаданих методів не дає можливості розв’язувати важкоформалізовані задачі тестування ПЗ. Такі задачі можливо розв’язати, комбінуючи необхідні елементи усіх цих методів у процесі тестування ПЗ та використовуючи на певних його етапах складові компоненти штучного інтелекту.

Джерела

* «Тестування програмного забезпечення» С. Канер, Дж. Фолк
* «Мистецтво тестування програм» G. J. Myers
* «Тестування DOT COM або Посібник по жорстокому поводженню з багами в інтернет-стартапах» Роман Савін
* https://uk.m.wikipedia.org/wiki/Тестування\_програмного\_забезпечення
* https://www.quality-assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/