Міністерство освіти і науки України
Харківський національний університет ім. В. Н. Каразіна
Факультет комп’ютерних наук

Курсова робота

з дисципліни «Математичні методи та технології тестування та верифікації програмного забезпечення»

Тема «Випадкове тестування(Random testing)»

Оцінка _____ / _____

Члени комісії:

Нарєжній О. П._____

Мелкозьорова О. М._____

Виконала:
студентка 2 курсу, групи КС- 21
Спеціальності:
122 «Комп’ютерні науки»
Бурсак Єкатерина Геннадіївна

Харків – 2020

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ПЗ – програмне забезпечення.

ІТ-технології – інформаційно- комунікаційні технології.

DOS (англ. Disk Operating System) – родина тісно пов’язаних операційних систем, які домінували на ринку сумісних із IBM PC комп’ютерів до 1995 року.

CP/M (англ. Control Program/Monitor або Control Programs for Microcomputers) – операційна система, спершу призначена для 8-розрядних мікропроцесорів.

API (англ. Application Programming Interface) – прикладний програмний інтерфейс.

QA (англ. Quality Assurance) – забезпечення якості.

Превентування – попередження появи багів.

Баг – дефект.

ЗМІСТ

ВСТУП

В наш час тестування є невід'ємною частиною процесу створення веб-сайтів, додатків та є основою роботи усього віртуального світу. Усі програми проходять етап тестування перед тим, як вони розпочнуть працювати за своїм призначенням та одержать популярність серед користувачів.

Тестування програмного забезпечення – процес аналізу програмного засобу та супутньої йому документації з метою виявлення дефектів та підвищення якості продукту.

Протягом багатьох років розвитку розробки ПЗ до питань тестування та забезпечення якості підходили з різноманітних точок зору. Можна виділити декілька, так званих, «епох тестування».

У другій половині минулого століття даний процес був формалізований, відокремлений від процесу розробки ПЗ та був «математизованим». В ті часи тестування базувалося на налагодженні програм. Також існувала концепція, яка мала назву «вичерпного тестування», яке займалося перевіркою шляхів виконання кодів з усіма можливими вхідними даними. Але, через деякий час було визначено, що є велика кількість можливих шляхів та вхідних даних, а також дуже важко знайти помилку в документації. Всі ці фактори унеможливили використання вичерпного тестування[1].

У 70-х роках минулого століття тестування дозволяло переконатися, шо ПО відповідає вимогам, а також визначало умови, які сприяють некоректній роботі програми.

80-ті роки ознаменувалися тим, що було змінено ключове місце тестування в розробці ПЗ: тестування почало застосовуватися протягом всього циклу розробки замість його використання на фінальних стадіях розробки. В багатьох випадках це дозволило не тільки швидко виявляти та усувати проблеми, але й передбачати та запобігати їх утворенню.

В цей період часу було зафіксовано швидкий розвиток та формалізацію методологій тестування та появу перших спроб автоматизувати процес тестування.

У 90-ті роки відбувся перехід від тестування як такого до більш всеохоплюючого процесу, який має назву «забезпечення якості». Цей процес охоплює цикл розробки ПЗ та має відношення до процесу планування, проектування, створення та виконання тест-кейсів та їх підтримку.

На початку XXI століття продовжувався процес пошуку нових шляхів, методологій, технік та підходів до забезпечення якості. Поява гнучких методологій розробки та таких підходів, як «розробка під керуванням тестуванням» значно сприяла розумінню тестування. Автоматизація тестування сприймалася як невід'ємна частина більшості проектів. Все більшу популярність одержували ідеї, того, що головним є здатність програми надати кінцевому користувачу можливість ефективно вирішувати свої задачі, а не відповідність програми вимогам.

На сучасному етапі розвитку технологій ми маємо: гнучкі методології та тестування, глибока інтеграція з процесом розробки, широке використання автоматизації, величезний набір технологій та допоміжних засобів, кросфункційність команди (тестувальник та програміст можуть виконувати роботу один іншого).

За допомогою тестування користувач може зробити свій код надійніше та знайти і виправити різноманітні помилки до релізу продукту. Особливо тестування корисно при розробці великих додатків в великій команді.

Автоматизоване тестування ПЗ – частина процесу тестування на етапі контролю якості в процесі розробки ПЗ. Воно використовує програмні засоби для виконання тестів та перевірки результатів його виконання. Автоматизоване тестування дозволяє скоротити час тестування та спростити його процес[5].

Перші спроби автоматизувати процес тестування з'явились в епоху операційних систем DOS та CP/M. В ті часи автоматизація полягала у видачі

додатком команд через командний рядок і подальший аналіз результатів. З часом для того, щоб працювати з мережі було додано віддалені виклики через API. Автоматизоване тестування вперше було згадано в книзі Фредеріка Брукса «Міфічний людино-місяць», де йдеться про перспективи використання модульного тестування. По-справжньому автоматизація процесу тестування розвивається тільки в 1980-х роках[1].

Всього існує два основних підходи до автоматизації тестування: тестування на рівні коду і тестування інтерфейсу користувача. До першого типу належить модульне тестування, а до другого – імітація дій користувача за допомогою спеціальних тестових фреймворків.

Фреймворк – програмне забезпечення, яке полегшує розробку та поєднання різних компонентів одного великого програмного проекту.

Для того, щоб стати професійним тестувальником необхідно постійно перебувати у русі, а також безперервно розвиватися і не цуратися ставити незручні питання.

Метою даної курсової роботи є освоєння дисципліни «Математичні методи та технології тестування і верифікації програмного забезпечення». Тематикою даної роботи є вивчення та освоєння специфічної техніки тестування, а саме: випадкове тестування (англ. Random testing).

Предметом вивчення цієї курсової роботи є дисципліна «Математичні методи та технології тестування і верифікації програмного забезпечення».

Об’єктом вивчення курсової роботи є випадкове тестування.

Актуальність даної роботи полягає в тому, що тестування в наш час займає дуже велике місце в сфері ІТ-технологій. Це означає, що дуже важливо мати здатність орієнтуватися та розбиратися в цій науковій дисципліні та глибше у сфері тестування. Також дуже важливо розрізняти види тестування, баги та помилки і інші особливості, які виникають під час цього процесу. Також актуальність цієї роботи полягає, шо програмісту необхідно враховувати інівідуальні моменти виробничих процесів і визначати оцінку якості продукту.

1. ВИПАДКОВЕ ТЕСТУВАННЯ

Дуже доцільно було б розглянути філософію тестування програмного забезпечення, яка охоплює цілий ряд видів діяльності: постановка задачі для тесту, проектування, написання тестів, тестування тестів, виконання тестів, вивчення результатів тестування.

Задача тестувальника полягає у відтворенні великої кількості багів. Відтворення багів тестувальнику необхідно для того, щоб побачити результат роботи. Чим більше буде відтворено багів, тим більше людину будуть цінувати як тестувальника. Також до задач тестувальника входить пропускання якомога меншої кількості пріоритетних для користувача багів. Чим менше багів пропущено, чим менше незадоволення висловив клієнт – тим вище тестувальник оцінює ефективність своєї праці.

Простіше кажучи, тестування це піклування про якість у вигляді виявлення багів до моменту, коли їх знайдуть користувачі.

QA – це турбота про якість у вигляді превентування появи багів. Спільним в тестуванні та QA є те, що їх головним завданням є покращення ПЗ. Різниця полягає в тому, що тестування покращує програмне забезпечення через виявлення багів. QA, в свою чергу, покращує програмне забезпечення через вдосконалення процесу розробки ПЗ.

Випадкове тестування (англ. Random testing) – техніка тестування з використанням методу чорної скрині, в якій вхідні дані, дії або навіть самі тест-кейси обираються на основі випадкових та псевдо-випадкових значень так, щоб вона відповідала операційному профілю (рисунок 1.1)[2].

Операційний профіль – підмножина дій, які відповідають деякій ситуації або сценарію роботи з додатком.

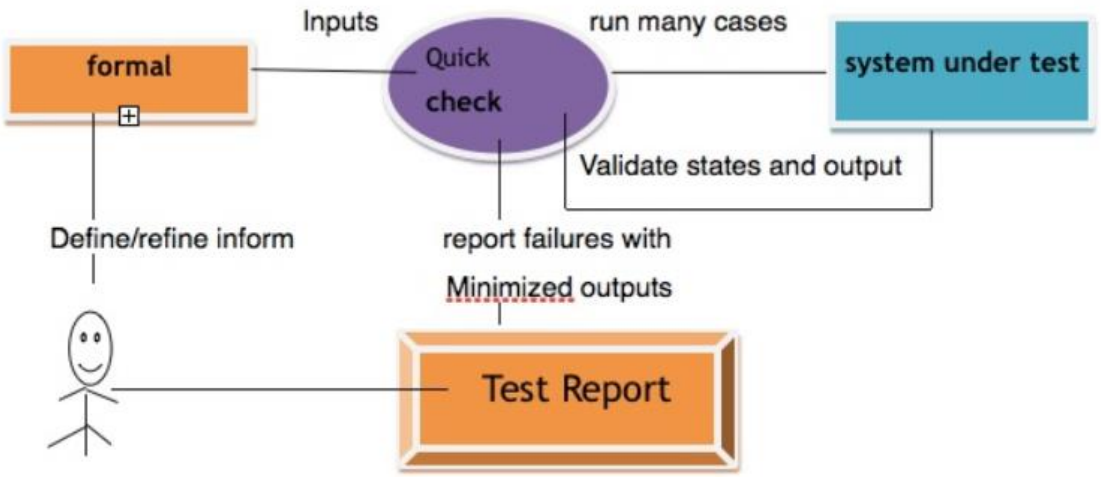

Рисунок 1.1 – Випадкове тестування [9]

Випадкове тестування також називають тестуванням мавп. Тестування мавп – техніка тестування програмного забезпечення, де користувач тестує додаток, надаючи випадкові введення та перевіряючи поведінку або намагаючись зірвати додаток (рисунок 1.2). Переважно ця методика виконується автоматично, коли користувач вводить будь-які випадкові недійсні входи та перевіряє поведінку додатку [6]. Таке тестування виконується, коли не вистачає часу для написання та виконання тестів.

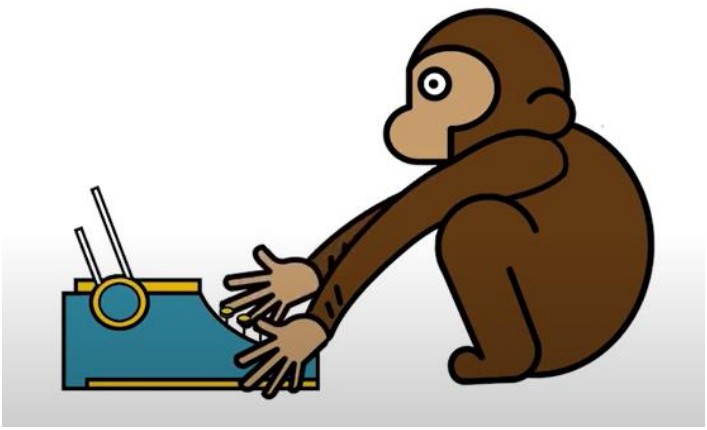

Рисунок 1.2 – Тестування мавп

В цій галузі випадкове тестування проводиться, коли дефекти не ідентифікуються через деякий проміжок часу. Спеціаліст використовує випадкове введення для того, щоб перевірити надійність та продуктивність заданої системи. Цей процес сприяє економії часу та зусиль, порівняно з реальними випробуваннями. Розглядаючи інші методи тестування, можна зробити висновок, що не кожний з вид поданих методів підходить для досягнення поставленої мети. Для оцінки за системою тестування випадкові входи ідентифікуються. В свою чергу тестові входи обираються незалежно від тестової області. На практиці дані тести виконуються за допомогою випадкових входів.

В ході роботи випадкового тестування результати виходу порівнюються зі специфікаціями програмного забезпечення для того, щоб переконатися, чи було пройдено тест, чи його неможливо пройти і виникає помилка. У тому випадку, якщо специфікації відсутні, використовуються винятки мови. Виникнення винятку під час виконання тесту означає, що програма містить помилку. Винятки мови також використовуються для того, щоб уникнути упередженого тестування.

Випадкове тестування також можна віднести до тестування на основі поведінки додатку або тестування на основі моделі.

Одним з найголовніших показників є час тестування, який необхідний для того, щоб виявити помилки. Дана, на перший погляд, дурна, стратегія тестування у цілому може виявитись набагато краще інших стратегій, адже вона є доволі швидкою. На відміну від інших стратегій, які можуть виявляти помилки, але потребують багато часу для того, щоб продумати процес тесту. Це, звісно ж, збільшує загальний час тестування.

Тестування – доволі складний процес, і іноді так стається, що після нашого об’єктивного аналізу, виявляється, що не всі ідеї виявляються корисними. Прикладом цього служить випадкове тестування. На інтуїтивному рівні може здатися, що будь-яка стратегія, яка використовує знання про

програму буде краще, ніж використання випадкового вводу. Однак, об’єктивні показники, такі як число знайдених помилок, вказують на те, що випадкове тестування доволі часто перевершує ідеї, які здавалися розумними та раціональними[10].

Ідея випадкової генерації полягає в тому, що всі дані відносяться до якихось множин або діапазонам, кожний тест – комбінація значень, кожне з котрих обирається випадково із відповідного існуючого діапазону або множини (класу еквівалентності) (рисунок 1.3).

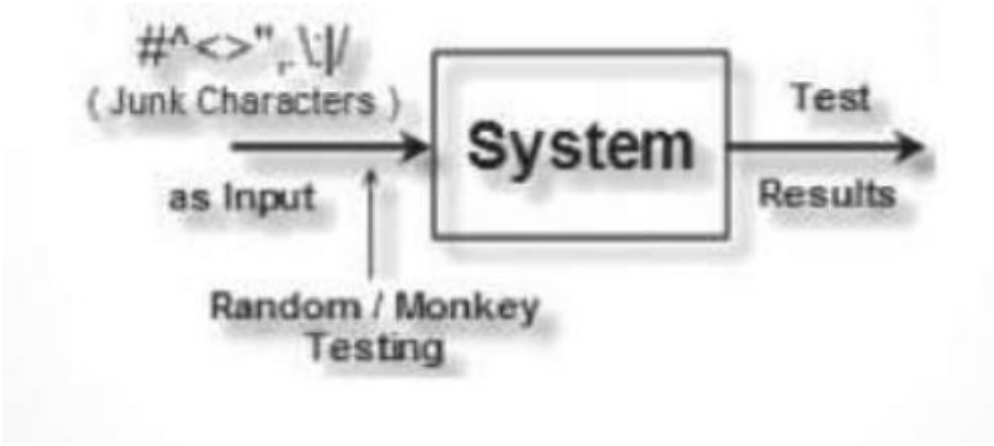

Рисунок 1.3 – Ідея випадкового тестування [11]

До властивостей можна віднести те, що випадкову генерацію доволі просто реалізувати. Існують інструменти, які дозволяють генерувати синтаксично коректні юніт-тести автоматично. Це означає, що можна дуже швидко згенерувати багато тестів. При цьому, виявлення помилок відбувається настільки д випадково, як і самі вхідні дані.

Випадкове тестування можна використати у невеликих програмах або частинах програм, де кількість елементів усередині класів еквівалентності не є дуже великим. До сфер використання також можна віднести фазинг.

Фрагмент коду наведено у додатку А.

1.1 Історія випадкового тестування

На початку 1970-х років тестування програмного забезпечення розглядалося як «процес, який був направлений на демонстрацію коректності продукту». Також можна додати, що у 1980-х роках тестування розширило свої кордони та почало спеціалізуватися на попередженні дефектів[2].

Усередині 1990-х років з розвитком Інтернету та розробкою великої кількості веб-додатків та видів тестування (на нашому прикладі – випадкове тестування). Особливу популярність стало отримувати гнучке тестування.

Вперше випадкове тестування для обладнання було вивчене Мелвіном Брюером у 1971 році, а перші зусилля для того, щоб оцінити його ефективність були зроблені Патімою та Вишвані Аграваль у 1975 році[6].

1979 року випадкове тестування вперше було згадано у книзі «The Art Of Software Testing» Гленфорда Дж. Майерса.

У 1984 році випадкове тестування було розглянуто у програмному забезпеченні Ntafos і Duran.

1.2 Сильні сторони випадкового тестування

Зазвичай випадкове тестування оцінюється за наступні сильні сторони:

1. Воно легке у використанні: не треба знати деталі про програму, яка тестується.
2. Не має жодних упереджень: не пропускає помилки, тому що не має неправильної довіри до якогось коду на відміну від ручного тестування.
3. Пошук помилок відбувається дуже швидко: звичайно тестування потребує лише декілька хвилин.
4. Якщо програмне забезпечення правильно вказане: воно знаходить реальні помилки.

1.3 Слабкі сторони випадкового тестування

Наступні недоліки зазвичай були вказані недоброзичливцями:

1. Випадкове тестування знаходить тільки основні помилки.

2. Погано порівнює з іншими методами пошуку помилок.

3. Має таку ж точність, як і специфікації, а специфікації зазвичай не дуже точні.

4. Деякі спеціалісти стверджують, що краще було б продумано охопити всі відповідні випадки ручним тестами зробленими вручну(використовуючи тестування за стратегією білої скрині), ніж покладатися на випадковість.

5. Якщо при кожному запуску тесту випадково обираються різні вхідні дані, може виникнути проблема для безперервної інтеграції, тому що одні й ті ж самі тести будуть проходити або не проходити випадково.

2. ФАЗИНГ

Фазинг (англ. fuzzing) – техніка тестування програмного забезпечення, суто автоматична або напівавтоматична, яка полягає в тому, що додатку передаються на вхід неправильні, несподівані або випадкові дані (рисунок 2.1).

Рисунок 2.1 – Місце фазингу у процесі розробки [8]

Падіння та зависання, порушення внутрішньої логіки та перевірок у коді додатку, витік пам’яті, які були викликані випадковими або неправильними даними на вході, є предметом інтересу фазингу. Фазинг, по суті, є різновидом випадкового тестування (англ. random testing), яке часто використовують для перевірки проблем безпеки в програмному забезпеченні та комп’ютерних системах. Зазвичай фазер використовують для тестування програм, які сприймають на ввід визначені структури даних (файли відомого формату). Ефективний фазер може згенерувати потік даних, який стане «досить валідним» для того, щоб пройти початкові перевірки парсеру. При подальшій обробці фазер виявить поведінку програми при перевищенні допустимих значень для певних параметрів або інші неочікувані ситуації.

Тестування коду, який порушує «рівні довіри», є найбільш цікавим з міркувань безпеки.

Фазинг дозволяє виявити велику кількість помилок в коді, які нікуди не зникнуть навіть з часом. Зазвичай фазингом завершують процес розробки, але його можна застосувати й до інших фунцій, які входять до продукту розробки.

2.1 Історія фазингово тестування

У 1950-х роках минулого століття розпочалася історія фазингового тестування. В той час для введення даних використовувалися перфокарти. Перевірка поточних програм на некоректну поведінку або баги відбувалася із застосуванням використаних перфокарт, які вже були викинуті, або пробиті випадково. Різновидом таких перевірок стало випадкове тестування або тестуванням мавп[3].

Зазначимо, що 1950-х роках вчений Джеррі Вейнберг використав набір карт з випадковими числами для того, щоб передавати їх на вхід програм. Техніки схожі з фазинговим тестуванням вже існували задовго до появи та формалізації даного терміну та процедури

1981-го року з’явилися перші наукові публікації, які було присвячено випадковому тестуванню програмного забезпечення. В той час його пропонували як дешеву альтернативу звичайному тестуванню, адже вибіркове тестування вважалося неефективним.

Випадкові дані використовувались у тестуванні додатків і раніше. Наприклад, додаток «Мавпа» (англ. The Monkey) для Mac OS, яке було створене Стівом Капсом ще в 1983 році. Цей додаток генерував випадкові події, які потім відправлялися на вхід програмам, які тестуються для пошуку багів. Програма була названа на честь теореми про те, що серед нескінченної кількості мавп, які були посаджені за клавіатури комп’ютерів, знайдеться хоча б одна, яка здатна набрати літературний текст[4].

Сам термін «fuzz» з'явився 1988-го року під час семінару Бартона Міллера в Університеті Вісконсину. На цьому семінарі було створено просту програму fuzzer. Ця програма була призначена для тестування надійності

додатків для операційної системи Unix. Вона працювала через командний рядок. У програмі генерувалися випадкові дані. Потім вони передавалися до інших програм у вигляді параметрів, які передаються доки не спровокують помилку. Це стало першим в історії тестуванням з використанням випадкових неструктурованих дані. Fuzzer став першим спеціалізованим додатком для тестування широкого спектру програм під різні операційні системи, який мав систематичний аналіз типів помилок, які виникали в процесі такого тестування. У 1995-му році було відтворено тестування, з допомогою якого додаток було допрацьовано для тестування додатків, які мають GUI, а також мережевих протоколів та системних бібліотек під Mac OS та Windows[3].

У 1991-му році тестування надійності програм під Unix и Unix-подібних операційних систем відбувалося через додаток crashme шляхом виконання випадкового набору інструкцій процесора.

У наш час фазинг-тестування є невід'ємною частиною більшості перевірок безпечності та надійності програмного забезпечення та комп'ютерних систем.

2.2 Різновиди фазингового тестування

Фазери можна класифікувати за тим, як змінюються дані в процесі тестування, за наявністю знань про структуру вихідних даних та за використанням знань про структуру вихідних даних. Дані в процесі тестування можуть змінюватись на основі мутацій або на основі поколінь. Ми можемо не мати даних про код програми і використовувати метод «чорної скриньки», знати абсолютно всю інформацію і використовувати метод тестування «білої скриньки» або знати обмежену кількість інформації та використовувати метод «сірої скриньки». Вихідні дані за стркутурою можна поділити на прості та розумні[3].

При тестуванні за мутаційним підходом фазери модифікують деякий набір відомих зразків. З іншого боку, фазери на основі поколінь генерують усі вхідні дані на початку тестування, при цьому, результати не залежать від

якості вхідного набору даних. Фазери, які не використовують дані про структуру вводу програми (наприклад, формат файлів або мережевий протокол). Якщо фазер дійсно ефективний, він має здатність вгадати правильність структуру вхідних даних і, за допомогою цього, використовувати їх у роботі. Розумні фазери мають структуру вхідних даних і, відповідно до певних правил, генерують тестові дані.

2.3 Переваги фазингового тестування

До переваг фазингового тестування перед іншими методами можна віднести[8]:

1. Автоматизоване тестування може виявити помилки, які при ручному тестуванні не вдалося виявити або було пропущено, за рахунок більшого покриття коду.
2. Фазингове тестування дозволяє зібрати загальну характеристику щодо захищеності коду, який тестується.
3. Фазер надає можливість забути про процес тестування і повернутися до нього вже за наявності готових результатів.

3. AD-HOC TESTING

Ad-hoc testing – вид тестування, який використовують без попередньої підготовки до тестів, без постановки очікуваних результатів та проектування тестових сценаріїв [12]. Ad-hoc testing також називають інтуїтивним тестуванням. Цей вид тестування можна віднести до імпровізованого та неформального. Даний вид тестування не має чітких процесів, яких слід дотримуватися при виконанні, а також не потребує ніякої документації та планування. Інтуїтивне тестування не передбачає наявність тест-кейсів. Але це може призвести до деяких утруднень при спробі відтворити дефекти наявні у системі. За відсутності потреби тестування такого типу виконується тільки один раз. Інтуїтивне тестування одразу може дати значно більше результатів, ніж будь-який вид тестування, яке планувалося заздалегідь. Так відбувається тому, шо тестувальник виконує тестування основного функціоналу проекту через деякі нестандартні перевірки.

Зауважимо, слід відрізняти інтуїтивне тестування від дослідного. Ad-hoc testing – це більш невпорядковане тестування, яке проводиться на інтуїтивному рівні. В цьому випадку тестувальник перевіряє те, що вважає потрібним, без наявності чітко мети, структури та будь-якої системи. Дослідне тестування є більш структурованим. При дослідному тестуванні тестувальник знає, що йому слід перевірити, заздалегідь має мету і якусь систему проведення тестів, тестовий сценарій.

3.1 Виповнення ad-hoc testing

Інтуїтивне тестуваннння виконується, коли відсутній час на точне та послідовне тестування. При цьому тестувальник фокусується на загальній уяві щодо додатків та здоровий глузд[12].

Відсутність інформації про продукт унеможливлює використання тестувальником ad-hoc testing. Доцільна інформація та гарне уявлення про цілі

проекту, його призначення та основні функції та можливості необхідні для того, щоб тестувальник не витрачав час для його вивчення, особливо у тому випадку, коли проект доволі складний на великий.

3.2 Види ad-hoc testing

До видів інтуїтивного тестування відносять:

1. Buddy testing або спільне тестування поєднує юніт-тестування та системне тестування між розробником та тестувальником.
2. Pair testing або парне тестування виконують тільки тестувальники з різним рівнем знать та досвідом.
3. Monkey testing або тестування мавп – випадкове тестування програми з метою її зламати. Даний вид тестування було розглянуто вище.

3.3 Переваги ad-hoc testing

Переваги ad-hoc testing полягають у тому, що:

1. відсутня необхідність витрачати час для створення та підготування документації;
2. існує можливість виявлення, так званих, «хитрих» дефектів, які було б неможливо знайти при використанні стандартних сценаріїв перевірок;
3. виявлення найважливіших дефектів на ранніх етапах;
4. використовується для форсованого навчання нових співробітників.

4. МЕТОД ЧОРНОЇ СКРИНІ

Для того, щоб розуміти які існують підходи до тестування програмного забезпечення необхідно знати які існують типи тестування взагалі (рисунок 4.1). Вважаємо доцільним зазначити, що до основних типів тестування відносяться ті, що визначають високорівневу класифікацію тестів.

Тип займає найвищим рівень в ієрархії. В свою чергу, одному типу можуть відповідати декілька видів.

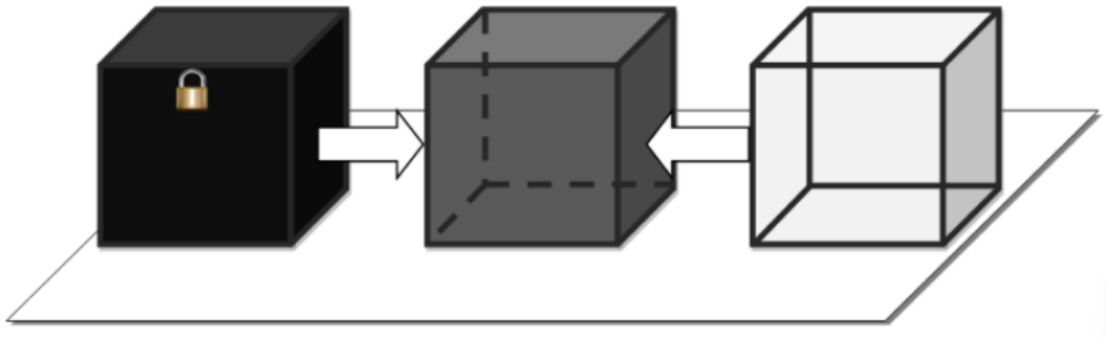

Рисунок 4.1 – White/Black/Grey Box-тестування [7]

Метод чорної скрині можна назвати техніку тестування, яка базується на роботі виключно з зовнішніми інтерфейсами системи. Функціональне і нефункціональне тестування входять до цього методу та не передбачають знання тестувальником внутрішнього устрою компонента чи системи. Для тест-дизайну, заснованому на методі чорної скрині виконується процедура написання, яка базується на аналізі функціональній або нефункціональній специфікації компоненту або системи та не вимагає наявності знань про її внутрішній устрій[7].

Цей метод має таку назву, тому що програма для тестувальника ніби чорна непрозора скриня, вміст якої невідомий і його неможливо роздивитися.

Мета цієї техніки полягає у пошуку помилок у таких категоріях:

1. неправильність реалізації функцій або їх відсутність;
2. помилки інтерфейсу;
3. помилки, які знаходяться в структурах даних або в організації доступу до зовнішніх баз даних;
4. недостатня продуктивність системи або помилки поведінки.

Тестування методом чорної скрині спонукає нас концентруватися та тому, що саме робить програма, а не яким чином. Даний метод тестування не дає уяви про структуру та внутрішній устрій системи.

Протилежністю методу тестування чорної скрині є метод тестування білої скрині. Мова про цей метод піде у наступному розділі.

4.1 Переваги методу

Переваги даного методу полягають у тому, що:

1. Тестування допомагає виявити неточності та протиріччя в специфікаціях.
2. Даний метод не потребує знань у сфері програмування та глибокого розуміння особливостей реалізації програми.
3. Допомагає уникнути упередженого ставлення, адже тестування відбувається незалежно від відділу розробки.
4. Є можливість написання тест-кейсів одразу після завершення специфікації.

4.2 Недоліки методу

Недоліки даного методу полягають у тому, що:

1. В цій області тестується дуже обмежена кількість шляхів програм, які виконуються.
2. Дуже важко розробляти тест-кейси без наявності чіткої специфікації.

3. Буває так, що іноді тести виявляються надмірними. Це виникає в тому випадку, коли вони розроблені на рівні модульного тестування.

4.3 Приклад використання

Розглянемо приклад, коли відбувається тестування веб-сайту без вказання особливостей його реалізації. При цьому використовуються тільки ті поля вводу та кнопки, які передбачив розробник. В такому випадку, джерелом очікуваного результату слугує специфікація.

Функціональне тестування передбачає перевірку працездатності системи, а нефункціональне, відповідно, загальні характеристики програми.

Спеціалісти використовують техніку чорної скрині на всіх рівнях тестування, починаючи з модульного і закінчуючи приймальним, які мають відповідну специфікацію.

Якщо розглядати техніки тест-дизайну, то можна виокремити що вони включають в себе: класи еквівалентності, аналіз кордонних позначень, таблиці рішень, діаграми зміни стану та тестування всіх пар.

5. МЕТОД БІЛОЇ СКРИНІ

Тестування білої скрині – метод програмного забезпечення, який позначає, що внутрішня струкура системи відома тестувальнику. Він обирає вхідні дані, спираючись на знання коду, мета якого їх обробка.

Для розуміння цієї техніки обов'язково потрібне знання всіх особливостей програми та її реалізації. Не відходячи від теми, зазначимо, що тестуваннння білої скрині це занурення у внутрішній устрій системи, а конкретніше, за кордони її зовнішних інтерфейсів[7].

Щодо термінології білої скрині можна сказати: це тестування, яке побудоване на аналізі внутрішньої структури компоненту або системи. Тест-дизайн, у свою чергу, також базується на техніці білої скрині. Це процедура написання або вибору тест-кейсів за наявності аналізу внутрішнього устрою системи або компоненту.

В даному випадку ця тестувальна програма має прозору скриню, яка дозволяє бачити всю внутрішню будову проекту.

5.1 Переваги методу

Переваги даного методу полягають у тому, що:

1. Для проведення тестування не має потреби очікувати створення інтерфейсу користувача. Воно здійснюється на ранніх етапах.
2. На практиці проводится більш чітке тестування з покриттям більшої кількості шляхів виконання програми.

5.2 Недоліки методу

Недоліки даного методу полягають у тому, що:

1. Для того, щоб розпочати процес виконання тестування білої скрині необхідна велика кількість знань та вмінь.

2. Підтримка тестових скриптів може здатися доволі складною, якщо програма часто змінюється і на цьому рівні використовується автоматизація тестування.

5.3 Приклад використання

Програміст, який вивчає реалізацію коду поля вводу на веб-сторінці визначає всі передбачені та непередбачувані вводи користувачів та завжди порівнює фактичний результат виконання програми з очікуваним. Очікуваний результат зазвичай визначається роботою коду програми.

Техніка білої скрині використовується саме для реалізації модульного тестування компоненту його автором.

5.4 Порівняння чорної та білої скриньок

Порівняння чорної та білої скриньок можна переглянути в таблиці 5.1. Дана таблиця відображує більш чітку різникю між двома концепіями[7].

Таблиця 5.1 – Порівняння Black Box та White Box.

Критерій	Black Box	White Box
Визначення	Тестування, як функціональне, так і нефункціональне; не передбачає знання внутрішнього устрою компоненту або системи.	Тестування, засноване на аналізі внутрішньої структури компоненту або системи
Рвіні, до яких може бути застосована техніка	Приймальне тестування. Системне тестування	Юніт-тестування. Інтеграційне тестування
Виконавець	Зазвичай, тестувальник	Зазвичай, розробник
Знання програмування	Не потрібно	Необхідно
Знання реалізації	Не потрібно	Необхідно
Основа для тест-кейсів	Специфкації, вимоги	Проектна документація

6. МЕТОД СІРОЇ СКРИНІ

Тестування методом сірої скриньки передбачає комбінацію білої та чорної скриньок. Це означає, що з внутрішнім устроєм програми ми ознайомлені частково. Передбачається доступ до внутрішньої структури та алгоритмів роботи програмного забезпечення, яке пристосоване для написання максимально ефективних тест-кейсів. Саме тестування проводится за допомогою техніки чорної скрині[6].

Також цю техніку тестування називають методом напівпрозорої скриньки, адже одна частина інформації нам доступна, а іншу частину ми не бачимо.

6.1 Приклад використання

Тестувальник вивчає код програми для того, щоб краще розуміти принципи її роботи та можливі шляхи її виконання. Таке знання допомагає розробляти тест-кейси, з допомогою яких буде проводитися перевірка певної функціональності. Техніка сірої скрині застосовується на інтеграційному рівні для перевірки взаємодії різних модулів програм.

ВИСНОВКИ

Підсумовуючи все вищезазначене, можна дійти висновку, що в процесі роботи ми переконалися у необхідності процесу тестування в сучасному світі. Тестування – дуже нелегкий процес, але знання, які ми отримали в ході вивчення дисципліни «Математичні методи та технології тестування і верифікації програмного забезпечення», допоможуть у майбутньому з легкістю розбиратися та розуміти головні аспекти роботи будь-якої програми чи системи.

Вивчаючи весь матеріал курсу ми ще раз переконалися у гнучкості та різноманітності процесу тестування. Також ми засвоїли багато навичок, які ми можемо застосувати на практиці, а саме: як працює випадкове тестування та де воно використовується; навчилися розрізняти види фазерів та розуміти де дозволяється їх експлуатувати; детально розглянули та надали можливість найлегшим способом розібратися у Ad-hoc тестуванні на прикладі можливих видів; методи чорної, білої та сірої скринь, які зазначалися вище, головним чином відносяться до теми випадкового тестування, яку ми вивчали у цій роботі, та вказують на шляхи роботи с інтерфейсами системи.

Виконання даної курсової роботи допомогло розширити знання даної дисципліни з точки зору історії та вивчення, так званих, епох тестування.

В наш час технології є дуже розвинутими. кожного дня в світі винаходять щось нове. Все це дуже пов’язане з веб-технологіями. Кожного дня створюються тисячі або навіть мільйони нових додатків, які проходять через вмілі руки тестувальників. Концепція скриньок, на різних етапах існування, буде розвиватися надалі та завжди перебувати у числі найголовніших та необхідних складових у галузі тестування. Всі технології тестування будуть оновлюватися. Деякі з них будуть зникати, а нові будуть з’являтися і займати їх місце.

Також за допомогою вивчення теми даної курсової роботи ми зрозуміли принципи та основи пошуку помилок (багів) у програмах та додатках, що і є основною метою цієї концепції. Велика кількість різновидів процесу тестування надають можливість тестувальнику обирати будь-який тип та вид, який буде підходити під розроблений тест або програму, з метою визначення конкретної проблеми або задачі. Ми розглядали тестування програмного забезпечення з погляду широкого спектру, який охоплює всі можливі аспекти конкретного випадкового тестування. В результаті розбору усіх нюансів, ми дійшли висновку, що процес тестування програмного забезпечення дуже широкий, та складається з декількох взаємопов'язаних процесів. Іншими словами сукупності процесів. Тестування, з точки зору спеціалістів, ніщо інше як те, за допомогою чого створюється щось інноваційне. Кожна людина має право вивчати продукт з різних боків, задавати різні питання та вносити свої особисті зауваження. Виходячи з переліку вимог кожен здатен крок за кроком пройти тест-кейс або що-небудь перевірити. Щоб добре та систематично зробити тестування, необхідно якісно вивчити та розібратися в усіх тонких особливостях цієї діяльності, та отримати навички задля гарного вміння керувати тестуванням з усіма його складовими. Також можна виокремити, що тестування – це структурована технічна діяльність, яка не виявляється легкою, навіть через наявність багатьох різновидів цієї сфери. Після початку вивчення випадкового тестування потрібно навчатися щось автоматизувати, а це досить непроста задача. На етапі вивчення, дуже важливо розуміти рамки автоматизування, визначення самих моментів коли час розпочинати автоматизування, окреме місце, у свою чергу, займає знання коду, знання інструментів та багатьох концепцій.

Отже, ця робота передбачала засвоєння випадкового тестування яка походить з дисципліни «Математичні методи та технології тестування і верифікації програмного забезпечення», та надала можливості набратися досвіду для полегшеної працездатності в майбутньому з різновидами та індивідуалізацією видів тестування програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Куликов С.С.. Тестирование программного обеспечения. Базовый курс. 2-е издание. М.: ОДО «Четыре четверти», 2017. 321 с. Дата звернення: 25.04.2020.
2. Випадкове тестування – Wikipedia. // URL: https://en.wikipedia.org/wiki/Random_testing. Дата звернення: 01.05.2020.
3. Фаззинг – Википедия. // URL: https://ru.wikipedia.org/wiki/Фаззинг. Дата звернення: 02.05.2020.
4. Фазинг – Вікіпедія. // URL: https://uk.wikipedia.org/wiki/Фазинг. Дата звернення: 02.05.2020.
5. Автоматизированное тестирование – GitHib. // URL: https://gist.github.com/codedokode/a455bde7d0748c0a351a. Дата звернення: 04.05.2020
6. Случайное тестирование – Qwe.wiki. // URL: https://ru.qwe.wiki/wiki/Random_testing#Overview. Дата звернення: 12.05.2020
7. Тестирование методом чёрного, белого и серого ящиков – QALight. // URL: https://qalight.com.ua/baza-znaniy/white-black-grey-box-testirovanie/. Дата звернення: 14.05.2020
8. Фаззинг – важный этап безопасной разработки - Habr. // URL: https://habr.com/ru/company/dsec/blog/450734/. Дата звернення: 15.05.2020
9. Random Testing – H2KInfosys. // URL: https://www.h2kinfosys.com/blog/random-testing/. Дата звернення: 17.05.2020

10. Автоматическая генерация тестов: подходы и иструменты – DOU // URL: https://dou.ua/lenta/articles/automatic-test-generation/. Дата звернення: 25.05.2020

11. Random Testing – SlideShare. // URL: https://www.slideshare.net/cankaya07/random-testing-15393318. Дата звернення: 27.05.2020

12. Ad-hoc testing – QAevolution. // URL: https://qaevolution.ru/testirovanie-po/vidy-testirovaniya-po/ad-hoc-testing/. Дата звернення: 30.05.2020

ДОДАТОК А
ФРАГМЕНТИ ВИХІДНОГО КОДУ

Випадкова генерація юніт-тестів на мові Java з допомогою Randoop.

Створюємо файл myTestingclasses.txt, в який записуємо назви класів, які ми б хотіли протестувати.

Викликаємо Randoop:

У ході компіляції та запуску згенерованих тестів, отримаємо два набори: RegressionTest та ErrorTest. Перший набір тестів пройде успішно (RegressionTes). Тести ж з другого набору не пройдуть тестування (ErrorTest).

Лістинг 1 – Приклад вихідного коду

```
@Test
       public void test001() throws Throwable {

             if (debug) {
System.out.format("%n%s%n","RegressionTest0.test001");
}

             MerArbiter.TestMyMerArbiterSym testMyMerArbiterSym0
= new MerArbiter.TestMyMerArbiterSym();
             testMyMerArbiterSym0.run2((int)'a', true, (int)'a',
false, (int)'4', (int)(byte)100, true, (int)(short)100, false,
(int)' ', 10, false, 100, false, (-1), (int)' ', false,
(int)(short)(-1), true, 0, true, true, (int)'a', (int)(short)1);
             testMyMerArbiterSym0.run2((int)(byte)100, false, 10,
true, (int)'4', (int)' ', false, 1, false, 100, 100, false,
(int)(byte)(-1), false, (int)'#', (int)(byte)1, false, 0, true,
(int)(byte)100, true, true, (int)(short)10, (int)(short)10);
             testMyMerArbiterSym0.run2((int)(short)100, false, 0,
false, (int)(byte)1, (int)(byte)1, true, (int)(byte)0, true,
(int)(byte)100, (int)'a', true, (int)' ', false, (int)(byte)1,
```

```
(int)(byte)100, false, (int)'4', false, (int)' ', true, false,
(int)(short)100, (int)'4');

      }
```