МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА
ФІЗИКО–МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітньо-кваліфікаційного рівня магістр

на тему «Методи тестування програмних систем»

Виконав: студент 6 курсу, групи 63
спеціальності 8.04030201
Інформатика
Плотницький Ярослав Валерійович

Керівник кандидат педагогічних
наук, доцент Кривонос О. М.

Рецензент_____
(прізвище та ініціали, науковий ступінь, вчене звання)

«Допущено до захисту»
*Завідувач кафедри*_____

_____ *(вчене звання, прізвище та ініціали)*
(підпис)
«____»_____2015 року

Житомир – 2015 рік

ЗМІСТ

Вступ

На сучасному етапі вивчення проблеми застосування методів тестування програмних систем є твердження, що одним з найбільш важливих засобів перевірки надійності програмного забезпечення є ручне тестування. Цей процес є трудомістким, часто без ефективної автоматизації, потребує багато часу та фінансових ресурсів. Щоб підняти рівень ефективності тестування програмних потрібна побудова сценаріїв тестування, що дозволить значно скоротити часові та економічні затрати. Тестування програмного продукту є основним засобом аналізу надійності програмних систем.

В останні роки складність комп'ютерних програм, що виконують важливі та відповідальні функції, швидко зростає. Актуальним завданням сучасних науковців є проблема підвищення надійності програмного забезпечення.

У розвиток та становлення програмної індустрії визначний внесок зробили українські вчені: Глушков В.М., Затонацька Т.Г., Дідковська М.В., Капітонова Ю.В., Кобринський С.Ю. Коротун Т. М., Лавріщева К. М.,Летічевський А.А., Нитребич О. О. Парасюк О.М., Перевозчикова О.Л., Плескач В.Л., Сергієнко І.В, Сердюк П.В., Тимошенко Ю.О., Федасюк Д.Ф., Ющенко К.Л., Яковина В.С. та інші. Одним із основних напрямків досліджень цих науковців є розробка нових теоретичних і практичних методів створення високоякісних програмних систем.

Основним акцентом при вивченні проблеми застосування методів тестування програмних систем є ідея про підвищення якості програмних продуктів та підвищення конкурентоспроможності українських програмних систем. Особливої ваги набувають питання про баланс вартості та тривалості процесів тестування програмних систем. Відповідь на ці питання шукають керівники та менеджери проектів, організації-

розробники програмних систем, замовники, рядові споживачі програмної продукції.

Протиріччя, що існують при вивченні проблеми застосування методів тестування програмних систем призвели до вибору теми магістерської роботи “Методи тестування програмних систем”.

Об’єктом дослідження є: процес тестування програмних систем.

Предметом дослідження є: методи тестування програмних систем.

Мета дослідження виявити і обгрунтувати ефективні методи тестування програмних систем, спрямовані на зниження ризику відмов під час експлуатації.

Завдання дослідження:

- дослідження сучасних вимог до процесу тестування програмних систем;
- аналіз існуючих моделей надійності програмних систем, розроблення алгоритмів та програм їх реалізації;
- розроблення методу оцінювання ризиків відмов модулів програмних систем;
- визначення структури базового процесу, що регламентує всі дії з підготовки, проведення та оцінювання результатів тестування;
- розроблення методики виконання процесу тестування програмних систем оброблення даних.

Методи дослідження. Метод визначення оптимального часу тестування розроблявся з використанням теорії надійності програмних засобів, теорії ймовірностей та математичної статистики, чисельних методів та методів верифікації програмних систем.

Апробація результатів дослідження. Основні результати дослідження були оприлюднені на наукових семінарах кафедри прикладної математики і інформатики Житомирського державного університету імені Івана Франка, Всеукраїнській науково-практичній Інтернет-конференції “Автоматизація та комп’ютерно-інтегровані технології у виробництві та

освіті: стан, досягнення, перспективи розвитку” (16-20 березня 2015 року, Черкаський національний університет імені Богдана Хмельницького, м. Черкаси), Всеукраїнській науково-практичній конференції ”Професійна підготовка фахівців у системі неперервної освіти”(23-24 квітня 2015 року), Міжнародній науково-практичній студентській конференції “Трансформація мовного образу сучасного фахівця” (15 квітня 2015 року, Житомирський державний університет імені Івана Франка, м. Житомир)

Структура роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел із 41 найменування. Загальний обсяг 52 сторінки.

Розділ І. ПРОБЛЕМА ЗАСТОСУВАННЯ МЕТОДІВ ТЕСТУВАННЯ ПРОГРАМНИХ СИСТЕМ В ІТ-ГАЛУЗІ

1.1. Визначення основних понять тестування програмних систем

Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок [17].

Термін *testing* (тестування) широко використовується в науковій літературі, але визначається по-різному.

Стандарт ANSI/IEEE Std. 610.12 визначає термін *testing* в самому його широкому сенсі – як будь-яку діяльність з аналізування програми (статичне та динамічне тестування) [34].

У дослідженні Гленфода Дж. Майерса цей термін тлумачиться в більш вузькому сенсі: "тестування – це процес виконання програми (або її частини) з метою виявлення помилок. Відлагодження (debugging) – визначення точної причини відомих помилок і їх виправлення" [19, с. 12].

Розділення цілей тестування та відлагодження стало першим кроком становлення інженерії тестування. І хоча динамічне тестування стосується програмного забезпечення, воно завжди виконується в контексті комп’ютерної системи. У роботі використовується термін програмна система, як складова програмного забезпечення, встановлюваного на комп’ютері.

Система тестування – це важливий етап розроблення всіх програмних систем, оскільки вона вимагає значних витрат на проведення та впливає на їх якість. Теоретично доведено, що неможливо досягти вичерпного тестування через велику потенційну вартість затрат та через

відмови у програмних системах. Тому потрібен чітко визначений та ефективний процес тестування, що базується на ухваленні ефективних рішень щодо тривалості та вартості тестування для досягнення необхідного рівня довіри до якості програмних систем. Тестування програмного забезпечення – це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126.

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів – це по суті дослідницький процес, а не тільки створення і виконання рутинної процедури.

Тестування пронизує весь життєвий цикл програмного забезпечення, починаючи від проектування і закінчуючи невизначено довгим етапом експлуатації. Ці роботи безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою тестування є якраз можливість переконатися у відповідності програм заявленим вимогам.

Тестування можемо назвати процесом також й ітераційним. Після виявлення та виправлення кожної помилки обов'язково слід повторити тести, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціального додаткового тестування. При цьому потрібно завжди пам'ятати про фундаментальний висновок, зроблений піонером інформатики, професором Едсжером В. Дейкстрі: "Тестування програм

може служити доказом наявності помилок, але ніколи не доведе їхню відсутність!" [6, с. 62].

Відповідно до ДСТУ 3918-99 для контролю якості програмних продуктів (включаючи всі робочі продукти) призначені процеси верифікації і валідації.

Основним завданням верифікації і валідації є перевірка і підтвердження того, що кінцевий програмний продукт буде відповідати своєму призначенню і задовольнить користувачів програмного забезпечення. Саме ці процеси контролюють якість програмного забезпечення шляхом виявлення помилок в продуктах процесів життєвого циклу, не досліджуючи причин появи цих помилок [13]. Std. 1012:1998. IEEE Standard for Software Verification and Validation.

Головна мета верифікації – дослідити трансформації одних (вхідних) робочих продуктів в інші (вихідні) робочі продукти і перевірити, чи правильно розробляється програмний продукт.

Основною метою валідації – є дослідження сукупності робочих продуктів, отриманих на певному етапі процесу розробки. Потрібно переконатися в тому, що вони розроблені правильно, тобто відповідають призначенню і специфікованим початковим вимогам до програмного продукту.

Процеси верифікації і валідації взаємопов'язані і зазвичай визначаються одним терміном «верифікація і валідація», якому відповідає термін «Verification and Validation» (V&V), що використовується в іноземній літературі. Процеси V&V повинні виконуватися, починаючи з самих ранніх стадій життєвого циклу програмних систем, включаючи всі робочі продукти розробки (плани тощо).

З процесом тестування тісно пов’язані такі поняття як “помилка”, “дефект”, “відмова”, “проблема”, “аномалія”, щодо визначення яких і досі існують розбіжності в науковій літературі. Ці поняття по-різному

визначаються не лише в науковій літературі з якості та надійності програмних систем, але і в стандартах.

Зокрема, стандарт ANSI/IEEE-729-83 [34] дає два визначення поняття відмови (failure):

1) відмова – це неможливість обчислювальної системи або її компоненту виконувати необхідні функції в межах специфікованих обмежень;

2) відмова – це відхилення програми від функціювання, визначеного вимогами до програми.

Поєднуючи вищенаведені визначення, термін "відмова в програмній системі" у дослідженні вживається у наступному значенні:

Відмова програмної системи (failure) – це подія, яка полягає в переході програмної системи з працездатного стану в непрацездатний або отриманні результатів за межами допустимих значень.

Відмови можуть бути обумовлені як зовнішніми чинниками (відмовами елементів середовища експлуатації, у тому числі користувача системи), так і внутрішніми чинниками – дефектами в програмній системі [12].

Дефект (fault) у програмній системі – це запис елемента програми (коду) або тексту документа (робочого продукту), використання яких може призвести до події, яка полягає в невірній інтерпретації цього елемента комп’ютером (помилці (fault) в програмі) або людиною (помилці (error) виконавця).

Дефект завжди є наслідком помилок розробника на будь-якому з процесів розробки програмного продукту. Дефекти можуть мати специфікації вимог, початкові або проектні специфікації, тексти коду, експлуатаційну документацію тощо.

Помилка також буває наслідком недоліків в одному з процесів розроблення програмної системи, що призводить до неправильної

інтерпретації початкової інформації людиною і прийняття неправильних рішень [12].

Під час тестування програмної системи та при виявленні відхилення результатів виконання від очікуваних – таке відхилення фіксують як “проблему” до з’ясування її причин.

Процес тестування проходить в динамічній перевірці поведінки програми на скінченій множині тестових даних, спеціальним чином вибраних з нескінченного вхідного простору, на відповідність встановленій очікуваній поведінці [12].

Динамічне тестування завжди призводить до виконання програми.

Скінчене тестування проводиться навіть для невеликої програми за схемою - створення теоретично такої кількості тестів, для виконання яких можливо знадобиться тривалий час. Неповнота - це одна з головних проблем тестування, адже практика показує, що повну множину тестів можна розглядати як нескінченну. Кількість тестів, які можуть бути виконані в обмежені терміни, скінчена. Отже, процес тестування завжди передбачає певний «компроміс» між обмеженими термінами і потенційно необмеженою кількістю тестів. Це призводить до відомих проблем тестування, таких як ухвалення рішень про адекватність тестування, і проблем керування, пов’язаних з оцінками витрат (вартості, часу, персоналу) на тестування [13, с. 192-193].

Сучасна наука пропонує багато підходів до вирішення завдань тестування та верифікації програмного забезпечення, але ефективне тестування нових та складних програмних продуктів – це завжди процес креативий, який не зводиться до чітких і одноманітних процедур. Він вимагає від тестувальника розгляду кожного окремого випадку в індивідуальному порядку.

Стандарт ISO 9126 визначає якість програмних засобів як сукупну характеристику досліджуваного програмного забезпечення з урахуванням

наступних складових: надійність, супроводжуваність, практичність, ефективність, мобільність, функціональність [34].

Більш повний список атрибутів і критеріїв можна знайти в стандарті ISO 9126 Міжнародної організації зі стандартизації. Склад і зміст документації, супутнього процесу тестування, визначається стандартом IEEE 829-1998 Standard for Software Test Documentation.

1.2. Інспектування програмних систем та програм

Системне тестування програмних систем та програм вимагає не тільки розробки великої кількості тестів, але і їхнього виконання і перевірки. Цей процес достатньо затратний і трудомісткий, адже майже кожен тест здатен знайти в програмі одну-дві, інколи більше помилок. Причиною такого стану є часті збої в роботі програмних систем, що відбуваються через присутні помилки в системі,що і призводять часто до руйнування введених даних.

Інспектування програм необхідно проводити на початкових стадіях розробки. Під час інспектування перевіряється остаточне представлення системи. Це може бути модель системи, специфікація або програма, написана мовою високого рівня. Виявити помилки можна шляхом використання знань розглянутої системи і семантики її вихідного представлення. Важливо, щоб кожна помилка розглядалась окремо, не зважаючи на те, яким чином вона впливає на поведінку системи в цілому.

Науковцями доведено, що інспектування програмних систем та програм являється дієвим методом виявлення помилок. Такий спосіб є значно дешевшим, аніж екстенсивне тестування. За допомогою інспектування програмних систем та програм можна знайти більше ніж 60% усіх помилок, а при більш формальному підході за допомогою математичних методів і більше 90%. Процес інспектування завжди

оцінює й інші якісні характеристики системи, такі як відповідність стандартам та зручність супроводу [14].

Інспектування програм є значно ефективнішим при виявленні помилок у системних компонентах. якщо порівнювати з тестуванням, то за один сеанс інспектування можна знайти безліч дефектів програмного коду. При застосуванні тестування, як правило, за один сеанс виявляється зазвичай лише одна помилка, оскільки помилки можуть привести до повного відмовлення системи, а ефекти помилок можуть накладатися одна на одну. Важливо зазначити, що інспектування використовує знання про предметну область і мову програмування. Фахівець, що проводить інспектування, повинний знати типи помилок, і саме це дає можливість зосередитися на зазначених видах дефектів.

Як зазначають фахівці - інспектування ніколи не замінить тестування. Існує кілька правил, яких притримуються спеціалісти: інспектування потрібно застосовувати на початкових стадіях для виявлення найбільшої кількості помилок, оскільки саме інспектуванням перевіряють відповідність програмних систем його специфікації, але таким способом майже неможливо оцінити динамічну поведінку системи в цілому. Варто зазначити, що вкрай нераціонально інспектувати закінчені системи, зібрані з декількох підсистем. Цей рівень вимагає тільки повного тестування програмних систем та програм.

Інспектування та тестування не являються антагонічними чи конкурентними методами верифікації й атестації. Кожен з методів має свої переваги та недоліки. Вважаємо за потрібне підкреслити, що і процес верифікації, і процес атестації інспектування й тестування варто обов'язково використовувати в комплексі.

Нерідко бувають випадки, коли при інспектуванні в організації виникають проблеми. Експерти, що мають великий досвід у тестуванні програм, неохоче погоджуються з тим, що інспектування є більш ефективним методом усунення дефектів системи, чим тестування.

Менеджери відносяться до цих технологій з недовірою, тому що впровадження інспектування вимагає додаткових витрат. Кінцева економія засобів при застосуванні інспектування досягається тільки завдяки досвідові провідних його фахівців.

Найчастіше інспектування програмних систем та програм пропонують для перегляду і перевірки програм з метою виявлення в них помилок. Ідея формалізованого процесу перевірки програм була сформульована корпорацією IBM у 1970-х роках. В даний час даний метод верифікації одержав широке розповсюдження. На його базі розроблена безліч інших методів, але усі вони ґрунтуються на базовій ідеї методу інспектування, відповідно до якої група фахівців виконує ретельний порядковий перегляд і аналіз вихідного коду програми. Головна відмінність інспектування від інших методів оцінювання якості програм полягає в тому, що його ціль – виявлення дефектів, а не дослідження загальних проблем проекту. Дефектами є або помилки у вихідному коді, або невідповідності програми стандартам [14].

Зазвичай, процес інспектування формалізований та мінімізований. У ньому беруть участь, як правило, три-чотири людини, кожна з яких має певні функції: автор, рецензент, інспектор, координатор.

Завданням рецензента є «озвучування» програмного коду, потім інспектор перевіряє його, а координатор забезпечує організацію усього процесу. Набуваючи досвіду, команда що інспектує може спростити процес організації інспектування і одна людина буде здатна виконувати кілька функцій, отже кількість людей в команді може змінюватись.

Розпочинаючи процес інспектування програми необхідно забезпечити наступні умови: "наявність точної специфікації коду (без повної специфікації неможливо знайти дефекти в програмному компоненті, що перевіряються,); члени інспекційної групи повинні добре знати стандарти розробки; у розпорядженні групи повинна бути

синтаксично коректна остання версія програми (нема рації розглядати код, що «майже довершений»)" [14].

Процес інспектування виглядає таким чином:

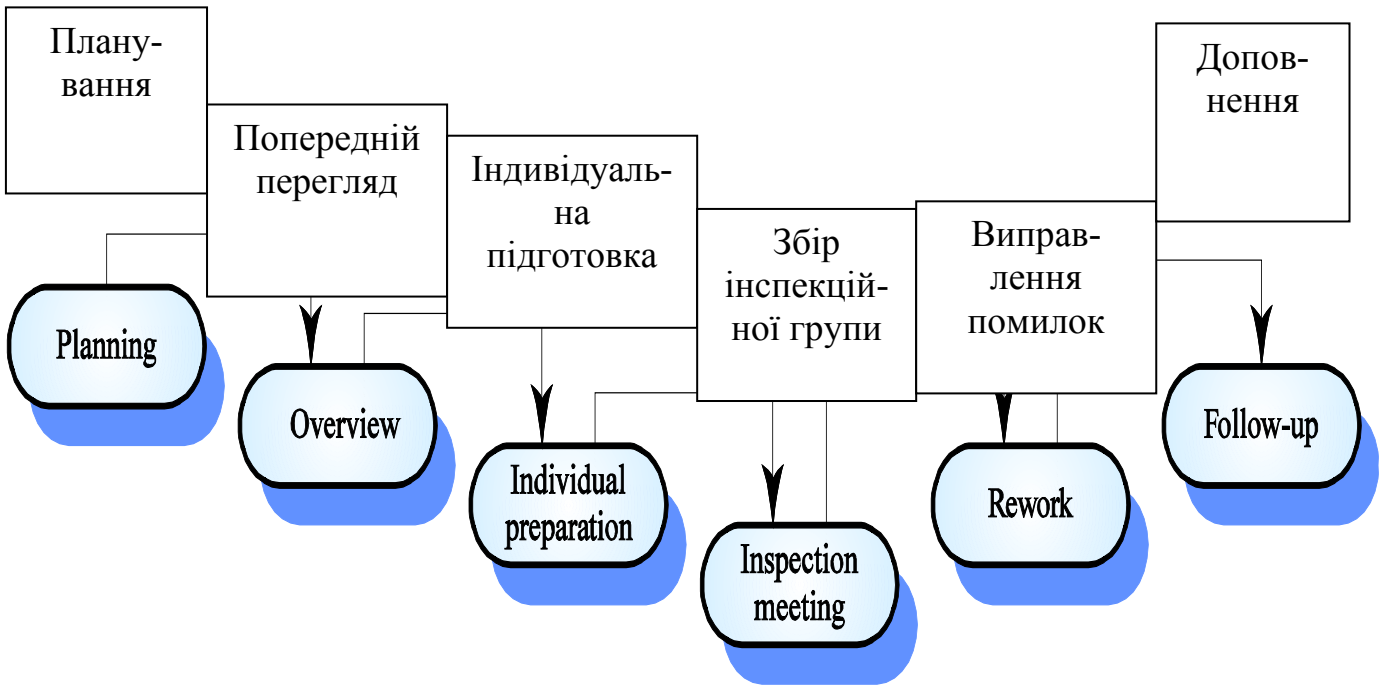

Рис. 1. Процес інспектування

Процес інспектування програмних систем та програм вимагає висококваліфікованого керівництва і правильного відношення до результатів його проведення. Інспектування – це відкритий процес виявлення помилок, оскільки помилки, допущені окремим програмістом, стають відомі усій групі програмістів. Фахівці зазначають, що "менеджери, що замовляють процес інспектування програми повинні чітко розмежовувати інспектування програмного коду й оцінку кадрів. При оцінці професійних якостей ні в якому разі не можна враховувати помилки, виявлені в процесі інспектування. Керівникам інспекційних груп необхідно пройти ретельну підготовку, щоб грамотно керувати

процесом і удосконалювати культуру відносин, що гарантувала б підтримку в процесі виявлення помилок і відсутність яких-небудь обвинувачень у зв'язку з цими помилками" [14].

1.3. Методи аналізу ризиків програмних проектів

Розробка та підтримка програмного забезпечення – це завжди високотехнологічний і складний процес, який вимагає застосування сучасних технологій, новівтнього обладнання, високопрофесійних розробників тощо. Такі складові надають конкурентоспроможності створюваному програмному забезпеченню.

Дослідження в області управління проектами зі створення програмного забезпечення показали, що в результаті неефективного управління 18 % проектів завершуються невдачею, 53 % потребують додаткових витрат фінансів і часу. В той же час в проектах реалізується тільки 69 % заявленої функціональності [33].

Наведені данні дозволяють акцентувати увагу на задачі управління ризиками в проектах. Аналіз ризиків є важливою складовою управління ризиками в проекті, від вдалого аналізу. Метою керування ризиками проектів є ідентифікація, оцінка та контроль ризиків проекту. Ризики, що ідентифікуються, потім аналізуються для визначення їх потенційного впливу та ймовірності виникнення.

Ризик програмного проекту можна визначити як "можливість зниження якості кінцевого продукту, підвищення вартості його розроблення, затримки закінчення розроблення або зриву проекту (тобто, відмови від проекту) через неефективність, недосконалість, незрілість технологічних процесів життєвого циклу програмної системи" [30, с.55].

Існує багато визначень поняття ризику як наявності невизначеності, пов'язаної з набуттям небажаної події, і збитків, понесених внаслідок настання цієї події. За визначенням SEI (Software Engineering Institute)

поняття "ризик" означає можливість зазнати втрат. Ризик проекту програмного забезпечення – це можливість отримати: зниження якості кінцевого продукту, підвищення вартості його розробки, затримки закінчення розробки або зриву проекту взагалі (відмови від проекту).

Величина ризику представляє собою формулу

R = V * P – добуток величини втрат V від небажаної події в проекті та ймовірності P настання цієї події.

Величину втрат V розглядаємо у контексті впливу небажаної події на характеристики програмного забезпечення, на складність його подальшого супроводу, а також ефективність, вартість і тривалість процесу розробки програмного забезпечення. Ймовірність P розглядаємо як ступінь визначеності, з яким можна прогнозувати прояви ризиків у проекті, тобто переростання даного ризику на проблему для проекту.

Ефективність управління ризиком передбачає прийняття компромісних рішень по кожному ризику окремо. Для цього необхідно враховувати ризики та аналіз за попередніми проектами, оцінювати трудозатратні моменти та усувати певні ризики, планувати і передбачати часові межі на боротьбу з ризиками.

Для ідентифікації та оцінки ризиків відмов програмних систем застосовують формальні і неформальні методи аналізу. Вибір того або іншого методу визначається критичністю програмних систем щодо серйозності наслідків відмов.

Найбільш відомими формальними методами аналізів ризику відмов є аналіз дерева подій (ETA - Event Tree analysis), аналіз дерева відмов (FTA - Fault Tree analysis) та аналіз режимів і наслідків відмов (FMEA - failure modes and effects analysis) [39]. Ці аналізи є достовірними методами верифікації ускладнених технічних та програмно-технічних систем на початкових стадіях аналізу вимог, проектування й реалізації, що застосовуються для аналізу можливих режимів відмов, напрацювання проектних і технічних рішень для зменшення наслідків відмов.

Аналіз дерев подій ідентифікує компоненти, події відмов яких можуть вплинути на безпеку роботи системи. Наслідком кожної події (що призводять до виникнення наступних подій) трасуються до тих пір, доки не будуть виявлені загрози (самого верхнього рівня), пов'язані з цими подіями. Під час ідентифікації ланцюга подій визначається ймовірність кожної події та комбінації подій.

Дерево подій являє собою повний спектр можливих наслідків певної події відмови компоненту. Він не відображає причин появи події відмови.

Аналізом дерева відмов називаютье процес ідентифікації ризиків, пов'язаних з компонентом системи. Дерево відмов – це ієрархічна структура, в вершині (у корені) котрої – стан відмови компоненту, прилеглі гілки – події, що призводять до переходу в даний стан. Виток кожної гілки – стан елемента компоненту, що призвів до настання відповідної події. Це дедуктивний метод пошуку умов і причин появи події відмови компонент [12].

Останнім часом з'явилися аналоги цих методів, спеціально призначені для використання в програмній інженерії. Так, аналог FTA - метод SFTA (software fault tree analysis).

Метод аналізу режимів і наслідків відмов (FMEA – failure mode and effect analysis) і його розширення для ПС (SFMEA – software failure mode and effect analysis) - табличні методи, які застосовуються як допоміжні для аналізу відмов і можуть використовуватися як разом з описаними вище методами аналізу дерев подій і відмов, так і самостійно. Ці методи застосовуються для ідентифікації типів можливих відмов і умов їх появи (режимів) в окремих компонентах системи. Використовують індуктивний аналіз з метою визначення умови виникнення відмов в компоненті (модулі) і наслідків цих відмов для всієї системи.

Головні методи та їх аналоги для програмних систем можна застосовувати спільно для аналізу відмов на рівні системи, оскільки багато відмов часто викликані комбінацією апаратних відмов, програмних

відмов або помилок користувача. У цьому випадку дерево програмних відмов підлаштовується під-дерево відмов програмних систем, а побудова структурно-логічної схеми системи виконується в два етапи:

- побудова дерев подій і відмов на рівні системи, елементами якої є функціональні компоненти системи;
- побудова дерев відмов для кожного функціонального компоненту (а також для програмних компонентів і окремих модулів) [12].

У процесі тестування розробники мають справу з ризиками, властивими проекту в цілому. Але оскільки головна мета тестування – знайти якомога більше дефектів, найбільш важлива категорія ризику для тестування – це ризики відмов. Для аналізу причин відмов та їх наслідків найбільш придатними є формальні методи верифікації – аналіз дерев подій, аналіз дерева відмов та аналіз режимів і наслідків відмов [16, с. 273].

Загальні таксономії ризиків, запропоновані SEI, корисні для ідентифікації загальних джерел ризиків проекту, але вимагають пристосування до конкретних умов.

Розділ ІІ. ОСНОВНІ ТЕХНОЛОГІЇ ТА МЕТОДИ ПОБУДОВИ ПРОЦЕСУ ТЕСТУВАННЯ ПРОГРАМНИХ СИСТЕМ

2.1. Основні принципи і типи методи тестування програмних систем

Основні принципи тестування програмних систем знаходимо у праці "Мистецтво тестування програм" видатного компютерного інженера і науковця Гленфорда Дж. Майерса [19, с. 25-31].

Необхідною частиною тестового набору має бути опис передбачуваних значень вихідних даних чи результатів. Визначення наперед передбачуваних результатів часто призводить до помилкових висновків. Щоб запобігти помилок при тестуванні потрібно включити в процес два компоненти - опис вхідних даних і опис точного та коректного результату, відповідного набору вхідних даних.

Важливою умовою вдалого тестування є обовязкова умова - програму не тестує її автор. В цьому випадку включається фактор психології людини - виявлення недоліків в своїй діяльності суперечить логіці психології. але мусимо зазначити, що найефективніше наладити програму може саме її автор. Найкраще довітири процес тестування організаціям, що спеціалізуються на тестуванні програмних засобів.

Г. Майерс зазначає, що "тестування – це процесс творчий. Цілком ймовірно, що для тестування великої програми потрібно ще більше творчий потенціал, а ніж для її проектування. ... Не можна дати гарантію побудови тесту, що виявить усі помилки. Надалі потрібно обговорювати методи побудови гарних наборів тестів, але застосовувати ці методи потрібно обов'язково творчо" [19, с. 31].

Обов'язковим правилом тестування програмних систем є доскональне вивчення результатів кожного тесту, адже це не дає можливість пропустити малопомітну на перший погляд помилку у програмі.

Необхідно ретельно підбирати тести не тільки для правильних та передбуваних вхідних даних, але і для неправильних - непередбачуваних даних. Безліч помилок можна виявити, якщо використовувати програму новим, непередбачуваним раніше способом.

При аналізі результатів кожного тесту необхідно перевіряти, чи не робить програма того, що вона не повинна робити. Є приклади побічних ефектів, які можна виявити перевіряючи саме цей параметр.

Важливим принципом тестування програмних систем є умова збереження використаних тестів тоді, коли програма вже не потрібна. Можливі варіанти доопрацювання, модифікації, внесення доповнень і виправлень після установки програми у замовника. Саме тоді повторне тестування значно скоротить процес і затрати часу на організацію.

Процес тестування не повинен плануватися виходячи з припущення, що у програмі не буде виявлено помилок. цю проблему майже повністю вирішують сучасні інструментальні засоби тестування, проте вона вимагає певної організації праці і матеріальних ресурсів.

Принцип ймовірності наявності невиявлених помилок в частині програми пропорційно відповідна до кількості помилок вже виявлених цій частині [19, с. 30]. Це наглядно продемонстровано на рис.2.

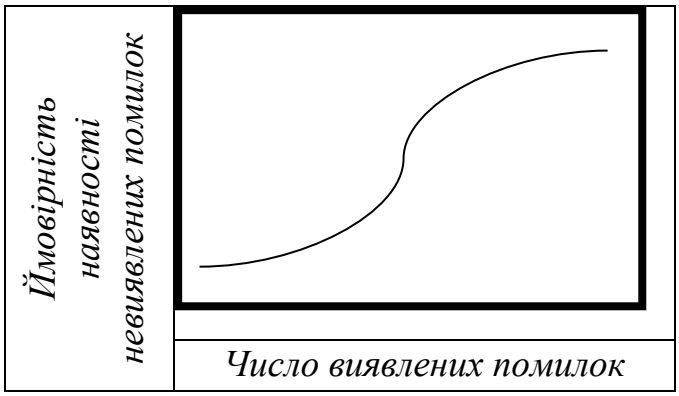

Рис 2.1. Несподіване співвідношення числа залишених і числа виявлених помилок.

Як зазначає Г Майерс, цей принцип не узгоджується із інтуїтивним уявленням про ймовірность наявності невиявлених помилок. Наведемо приклад: припустимо, програма складається з двох модулів А і В. До певного терміну в модулі А виявлено п'ять помилок, а в модулі В - тільки одна. Тоді з розглянутого принципу випливає, що ймовірность невиявлених помилок в модулі А більше, ніж в модулі В. Найчастіше, помилки розташовуються у програмі у вигляді скупчень. Цей принцип "скупчення помилок" акцентує увагу на тому, що він дозволяє ввести зворотний зв’язок у процес тестування. Якщо у будь-якій частині виявлено більше помилок аніж в інших – то саме на на цю частину потрібно направити додаткові зусилля в тестуванні [19, с. 30].

Існує кілька рівнів тестування програмних систем. серед них - тестуванн окремих елементів, завдяки якому проводиться перевірка окремих, ізольованих один від одного частин програмного забезпечення. Інтеграційне тестування дозволяє перевірити зв’язки і взаємодію інтерфейсів, що розміщені на архітектурних платформах. Тетсування системи перевіряє правильність функціонування системи в цілому, шукає та виявляє відмови і дефекти у системи, дозволяє їх усунути. При цьому контролюється виконання сформульованих не функціональних вимог (безпека, надійність і ін.) у системі, правильність подання і здійснення зовнішніх інтерфейсів системи з зовнішнім середовищем

Існує кілька ознак, за якими прийнято робити класифікацію видів тестування. Зазвичай виділяють наступні:

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 (usability testing);

- тестування інтерфейсу користувача (ui testing);
- тестування безпеки (security testing);
- тестування локалізації (localization testing);
- тестування сумісності (compatibility testing).

За знанням системи:

- тестування чорного ящика (black box);
- тестування білого ящика (white box);
- тестування сірого ящика (gray box).

За ступенем автоматизації:

- ручне тестування (manual testing);
- автоматизоване тестування (automated testing);
- напівавтоматизоване тестування (semiautomated testing).

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 (smoke testing);
- тестування нової функціональності (new feature testing);
- регресійне тестування (regression testing);
- тестування при здачі (acceptance testing);
- Бета-тестування (beta testing).

За ознакою позитивності сценаріїв:

- позитивне тестування (positive testing);
- негативне тестування (negative testing).

За ступенем підготовленості до тестування:

- тестування за документацією (formal testing);
- ед хок (інтуїтивне) тестування (ad hoc testing) [41].

Схема типів і методів тестування подано на рис.2.2.

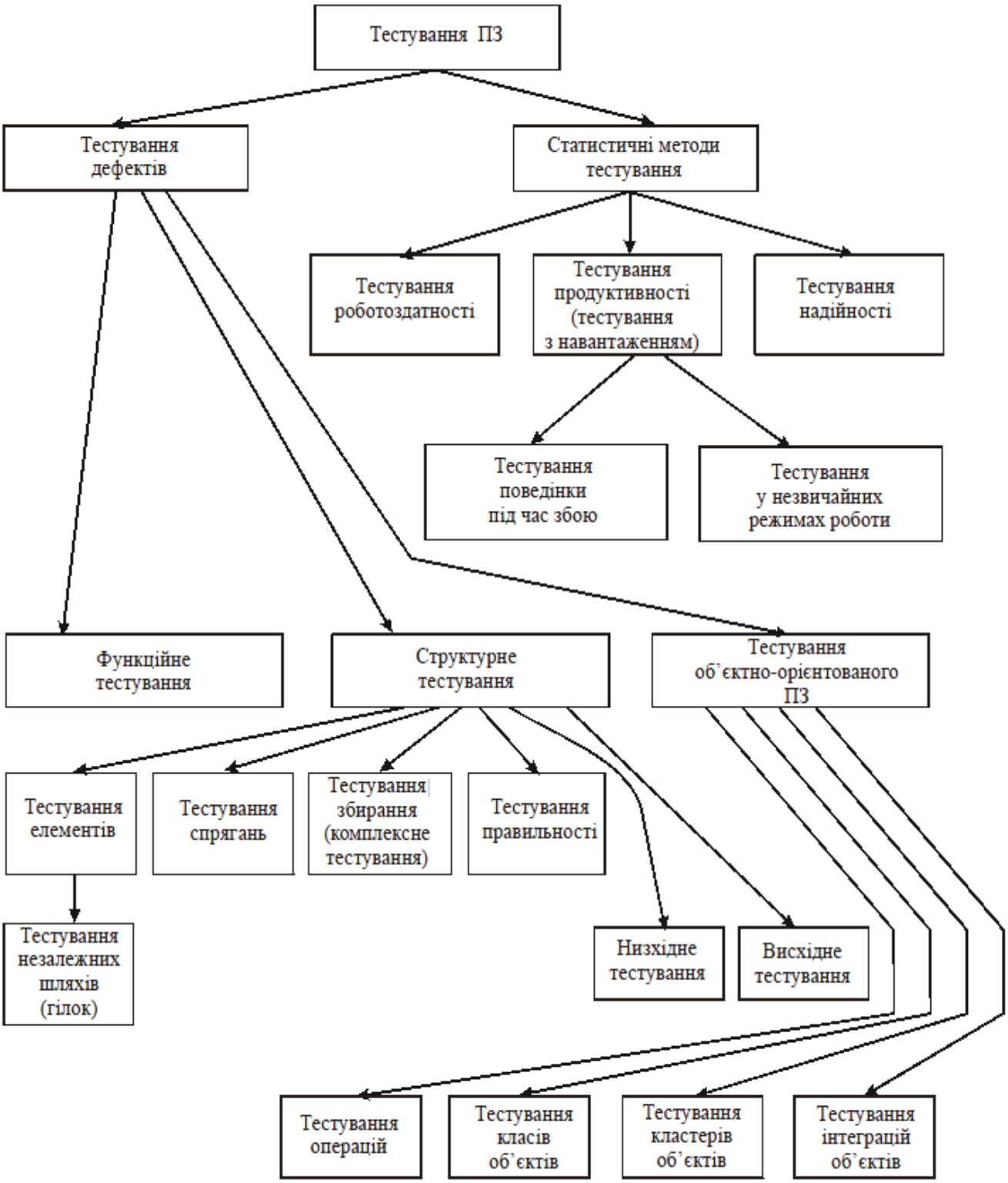

Рис.2.2. Типи і методи тестування

2.2. Класифікація методів тестування: статичні та динамічні методи тестування

Існує велика кількість сучасних техноглогій тестування програмних систем.

Тести суттєво відрізняються за параметрами використання технічних засобів, завдань, які вирішуються у тому чи іншому випадку. Різноманітні задачі, що вирішуються в процесі тестування призводять до необхідності використовувати різноманітні види та типи тестування. Сучасна наукова література поділяє види тестування за наступними категоріями:

- за обєктами (елементами) тестування. Цю категорію часто називають -розподіл за рівнями;
- за глибиною тестування. Цей критерій враховує кількість часу і об'єму компонентів програмного продукту, що тестується.

Основна класифікація розподілу тестів на види є традиційною – головний показник – високий рівень якості, що перевіряються за допомогою тестів.

Статичний метод тестування – це процес, який зазвичай асоціюється з аназізом програмного забезпечення. Статичний метод тестування напйчастішке застосовується при варифікації фактично кожного артефакту розробки: програмного коду компонент, системних вимог, системних специфікацій.

Статичні методи використовуються при проведенні інспекцій і розгляді специфікацій компонентів без їхнього виконання.

Техніка статичного аналізу полягає в методичному перегляді (або огляді) і аналізі структури програм, а також у доведенні їхньої правильності вручну за столом. Статичний аналіз направлений на аналіз документів, розроблених на всіх процесах життєвого циклу і полягає в інспекції вхідного коду і наскрізного контролю програми.

К. М. Лавріщева зазначає, що інспекція програмних систем є статичною перевіркою відповідності програми заданим специфікаціям. При цьому проводиться представлення результатів проектування шляхом аналізу різних даних - документації, вимог, специфікацій, схем або коду програм - на всіх процесах життєвого циклу. Перегляди й інспекції результатів проектування і відповідності їх вимогам замовника забезпечують більш високу якість створюваних програмних систем.

Під час інспекції програм розглядаються документи робочого проектування на процесах життєвого циклу разом з незалежними експертами й учасниками розробки програмних систем. "На початковому процесі проектування інспекція припускає перевірку повноти, цілісності, однозначності, несуперечності і сумісності документів з вимогами до програмної системи. На процесі реалізації системи під інспекцією розуміють аналіз текстів програм на дотримання вимог стандартів і прийнятих керівних документів технології програмування" [14, с. 165].

Застосування статичного методу тестування є одним із найефективніших засобів знаходження дефектів на початкових стадіях розробки програмного забезпечення та без запуску програмного коду.

Динамічні методи тестування *(dynamic mode testing)* – це виконання програм з метою встановлення причин помилок з очікуваними реакціями на ці помилки. Динамічні методи тестування використовуються в процесі тестування, який проводиться з системою чи підсистемою, яка працює. таке тестування не може використовуватись, якщо програмний код ще не запущено. Динамічне тестування складається з кількох етапів: запуск системи або підсистеми, виклику необхідних модулів чи функціональних елементів, порівнянь за допомогою графічного інтерфейсу користувача роботи системи та очікуваного результату її поведінки.

"Динамічні методи тестування використовуються в процесі виконання програм. Вони базуються на графовій структурі, що пов’язує причини помилок з очікуваними реакціями на них. У процесі тестування

накопичується інформація про помилки, що використовується при оцінці показників надійності і якості програмної системи. Динамічне тестування орієнтоване на перевірку коректності програмної системи на множині тестів, що проганяються по програмній системі, з урахуванням зібраних даних на процесах життєвого циклу, проведення виміру окремих показників (число відмов, збоїв) тестування для оцінки характеристик якості, зазначених у вимогах, шляхом виконання системи на ЕОМ" [14, с. 166].

Останні дослідження пропонують інтеграційні методи тестування, які тісно пов’язані з процесами розроблення, і таке тестування повинно розглядатись як діяльність, що виконується протягом усього процесу розроблення. Планування тестування починається на стадії аналізу вимог, а плани і процедури тестування повинні систематично і постійно уточнюватися разом з розвитком проекту розроблення системи.

Особливо важливу роль раннє планування і виконання тестування набуло з появою і розвитком об'єктно-орієнтованого і компонентного підходів, застосуванням ітераційних моделей життєвого циклу, використанням інструментів підтримки розробки.

Розподіл дій з тестування по окремих процесах прийнято зображати у вигляді V-подібної моделі [10], яка відображає процеси декомпозиції та інтеграції програмної системи та відповідні рівні і дії з динамічного тестування (рис. 2.3).

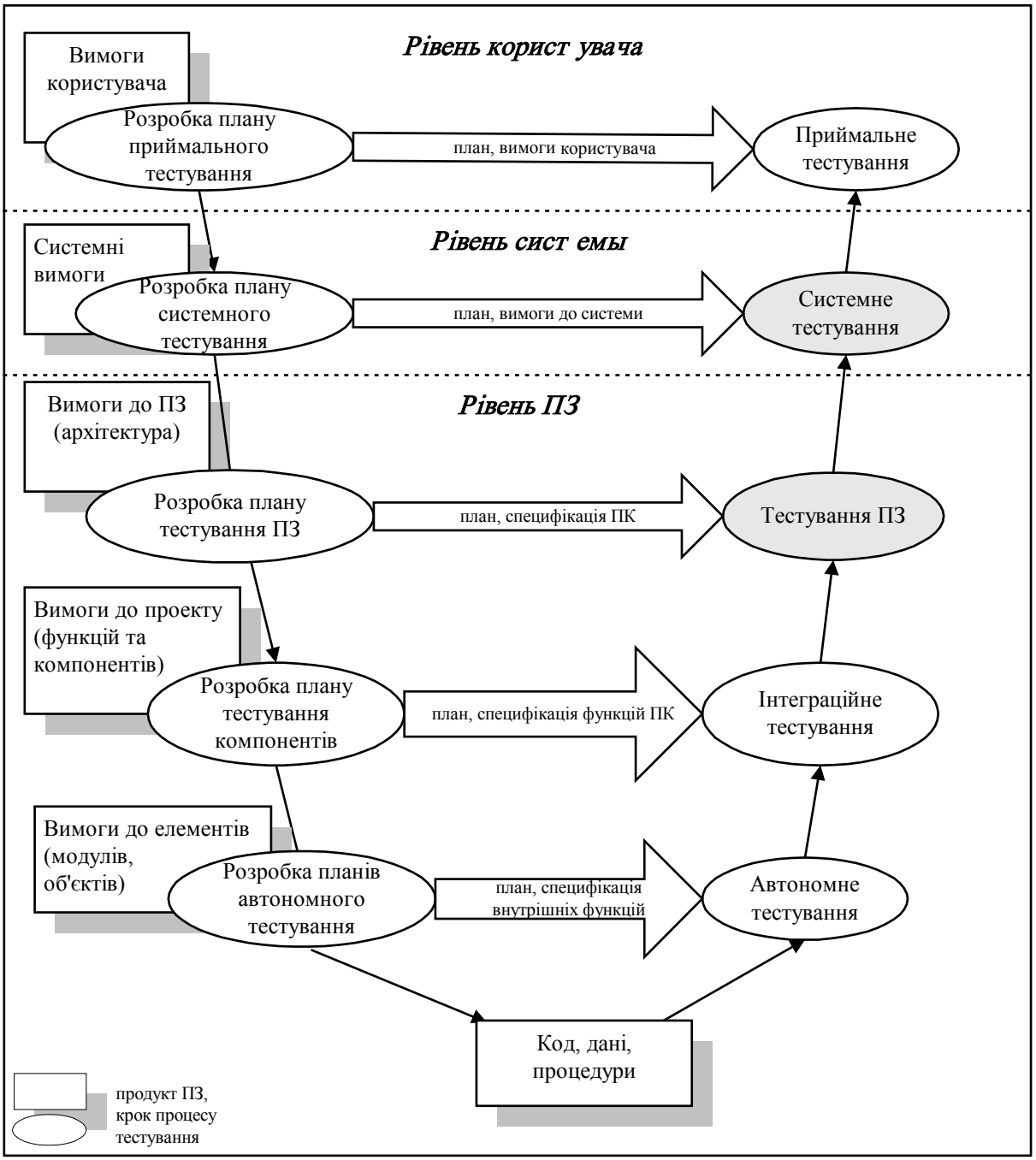

Рис. 2.3. V-подібна модель життєвого циклу із “вбудованими” задачами тестування

Статичні та динамічні методи доповнюють один одного.

Серед найбільш поширених методів тестування зазвичай виділяють метод "білої скриньки" («white-box» or «glass-box» testing) та метод "чорної скриньки"(«black-box» testing).

Стратегія "білої скриньки" передбачає детальне дослідження внутрішньої логіки і структури коду, для якого повинна бути відома повна інформація про вихідний код програмного забезпечення.

Стратегія "чорної скриньки" розглядає системні характеристики програм, ігноруючи їхню внутрішню логічну структуру.

Унаслідок врахування усіх змінних і відповідних класів еквівалентності, розроблена модель використання програмного продукту більш адекватно описує поведінку програмного забезпечення та дає можливість побудувати сценарії його тестування стратегіями як "чорної", так і "білої" скриньки.

При структурному тестуванні перевіряється коректність побудови всіх елементів програм і правильність їх взаємодії між собою. Тестування за методом "білої скриньки" характеризується ступенем, в якому тести виконують або покривають логіку (вихідний текст) програми.

Вичерпне тестування за принципом "білої скриньки" передбачає виконання кожного пункту в програмі, алу оскільки в програмі , що містять уикли, виконання кожного шляху зазвичай не реалізується то тестування усіх шляхів не розглядається.

Якщо відмовитись повністю від тестування усіх шляхів, то можна довести, що критерієм покриття виконання кожного оператора програми, принаймі, один раз. Це метод покриття операторів. такий метод є досить слабким критерієм, так як на виконання кожного оператора, хоча б один раз, є необхідною, але недостатньою умовою для прийняття тестування за принципом білої скриньки [5, с. 44].

Виконання критерію покриття шляхів забезпечує повне тестування циклічних структур, а, відповідно, і виконання як внутрішнього, так і граничного критерію тестування циклів. Зауважимо, що ці два критерії створені лише для циклів і навіть не забезпечують покриття всіх операторів, тобто вони повинні використовуватися тільки в сукупності з іншими.

Критерій покриття рішень забезпечує покриття всіх операторів. В свою чергу покриття всіх переходів (рішень) також може бути досягнуте шляхом виконання більш повних критеріїв - критерію покриття рішень/умов і критерію покриття шляхів [7].

Для реалізації цього методу повинна бути відома детальна внутрішня структура програм. Об’єктом тестування у такому випадку є внутрішня, а не зовнішня поведінка програм. Тому розглядаються внутрішні елементи програм і зв’язки між ними. Досліджується, по суті, логіка програм. Тестування методом "білої скриньки" (інколи зустрічається поняття "прозора скринька") грунтується на керуючій структурі програм. Програми вважаються повністю перевіреними, якщо проведено вичерпне тестування всіх гілок їх графів керування. Для цього формуються такі тестові варіанти, які б гарантували перевірку всіх незалежних маршрутів кожної програми, перевіряли гілки True і False для всіх логічних компонентів.

Мутаційне тестування є різновидом тестування "білої скриньки", для його здійснення необхідний доступ до вихідного коду програми [4]. Мутаційний критерій ґрунтується на штучному внесенні помилок у програму. У мутаційному критерії приймається припущення про те, що програмісти пишуть майже коректні програми, що відрізняються від правильних незначними помилками в арифметичних операціях, перестановками індексів, некоректними границями циклів, невірними константними значеннями та ін. Для виправлення дефектів подібного роду, у програму вносяться дрібні помилки (мутації). Програми, що відрізняються від вихідних програм, штучно внесеними помилками називають мутантами. Як правило, мутант відрізняється від вихідної програми невеликим числом мутацій. У вихідній програмі можуть піддаватися мутаціям ділянки коду пов'язані з перерахованими вище дефектами

(змінюються значення змінних, модифікуються індекси й границі циклів, вносяться мутації в умови). Таким чином, з первісної програми шляхом внесення n числа мутацій одержують k мутантів, n>=k (як мінімум одна мутація на один мутанта). Якщо сформована множина тестових наборів виявляє всі мутації у всіх мутантах, то воно відповідає мутаційному критерію. Якщо тестування вихідної програми та мутанту на заданій множині тестових наборів не виявило помилок, то програма оголошується еквівалентною мутанту. У випадку мутаційного тестування важливо створити таке число мутантів, яке б охоплювало всі можливі ділянки прояву помилок. Вважається, на основі дрібних помилок можна оцінити загальне число помилок, що залишилися в програмі.

2.3. Основні задачі функціонального тестування

Завданням функціонального тестування є перевірка відповідності програми своїм специфікаціям. При даному підході текст програми не доступний, і програма розглядається як "чорної скриньки". Найпоширенішими видами функціонального тестування є методи випадкового тестування, еквівалентної розбивки й аналізу граничних умов

Функціональне тестування призначене для перевірки відповідності програмного забезпечення його специфікації (завданню) або еталону, тобто перевірки, чи виконує програмна система або її модулі відповідні функції та поставлені вимоги. При цьому тестувальник перевіряє виконувані функції, а не реалізацію програмного забезпечення шляхом аналізу вхідних і вихідних даних. У науковій літературі цей метод називають ще методом тестування "чорної скриньки". Для реалізації цього методу повинні бути відомі функції програм програмного

забезпечення. Результати функційного тестування дають відповіді на запитання: як виконуються функції програм програмного забезпечення; як сприймаються вхідні дані; як виробляються результати на виходах "чорної скриньки"; як зберігається цілісність зовнішньої інформації? [21, с.12].

Під час тестування методом "чорної скиньки" розглядаються системні характеристики програм, а розгляд їх внутрішньої логічної структури ігнорується. На присутність дефектів у програмі вказує розбіжність інформації на виході з наперед відомою, еталонною інформацією. Щодо тестових даних, то вони підбираються такими, що при їх подаванні на входи “чорної скриньки” помилки програми, яка тестується, проявляються з високою ймовірністю.

Функціональне тестування, тобто тестування програмного забезпечення методом "чорної скриньки", не реагує на численні особливості програмних помилок, але воно забезпечує виявлення наступних типів помилок:

1) некоректних чи відсутніх функцій;

2) помилок інтерфейсу;

3) помилок у внутрішніх структурних даних чи в доступі до зовнішньої бази даних;

4) помилок характеристик, зокрема, необхідної ємності пам’яті та ін.

Зауважимо, що перераховані помилки не виявляються методом "білої (скляної, прозорої) скриньки", який ми розглядали вище.

Класичні методи структурного й функціонального тестування мають певні обмеження при застосуванні, розглянемо їх детальніше.

Функціональне тестування характеризується дуже великою кількістю необхідних тестів, більш того, відразу піднімається питання про забезпечення незалежності цих тестів. Якщо ж обмежувати кількість, то, все одно, виникають складності як при виділенні класів еквівалентності, так і границь, при цьому може бути пропущений ряд високоефективних

тестів, а зловмисна логіка не виявляється. До того ж, варто помітити, що методи функціонального тестування не дозволяють локалізувати несправності.

Функціональне тестування застосовується тільки коли програмне забезпечення вже створене, тобто на останніх етапах життєвого циклу програмного забезпечення. Якщо функціональне тестування виявить погану якість створення програмного забезпечення, то доводиться повертатися на попередні етапи розробки, що спричиняє як фінансові збитки так і часові витрати.

Структурне тестування перевіряє внутрішню логіку програми, що дозволяє локалізувати несправності. На жаль, із зростанням розміру вихідного коду програми повноцінне структурне тестування стає все складнішим. А спуск по представленій ієрархічній структурі взаємозв’язків критеріїв приводить до пропущення певного типу помилок і, відповідно, до втрати якості.

Можливості застосування структурного тестування для різних фаз тестування обмежені. Якщо для модульного тестування, в силу невеликих розмірів вихідного коду, представлені методи застосовні, хоча й з тими або іншими зазначеними вище складностями й обмеженнями, то для інтеграційного й системного тестування сфера застосування наведених класичних методів украй обмежена в силу різкого зростання вихідного коду або взагалі відсутності такого.

Більше того, можливість застосування структурного тестування залежить і від обраної парадигми програмування: якщо для процедурного програмування методи структурного тестування застосовні; для об‘єктно-орієнтованого застосування обмежене через зростання обсягу вихідного коду; то для компонентно-базованого програмування - дані методи часто стають недосяжними [7]. При компонентно-базованому програмуванні, компоненти в основному представлені як «чорні ящики» і доступні лише їхні автомати станів (на UML). Таким чином, класичні методи

структурного тестування, для яких рівень абстракції - це рівень операторів, стають не застосовні.

При тестуванні компонентно-базованого програмного забезпечення основним завданням є не перевірка правильності функціонування самих компонентів, тому що в більшості випадків вони вже були протестовані, а основна увага приділяється взаємодії між компонентами - їхньому інтеграційному тестуванню.

Отже, необхідні спеціалізовані критерії для інтеграційного тестування, які будуть працювати на іншому рівні абстракції, враховувати пропоновані автомати станів компонентів і концентрувати увагу саме на взаємодії між модулями, а не на їхній внутрішній роботі.

Розділ ІІІ. ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

3.1 Технологія виконання тестування програмних систем

За визначенням провідних фахівців з програмування – тестування програм та систем – це "спосіб семантичної перевірки програми, який полягає в опрацюванні програмою послідовності різноманітних контрольних наборів тестів з відомими результатами. Тести підбираються так, щоб вони охопили найрізноманітніші типи можливих ситуацій" [24].

Охарактеризуємо основні види робіт з тестування :

- верифікація результатів розроблення програмного продукту на кожному етапі життєвого циклу;
- упорядкування плану тестування і підготовки тестів для перевірки окремих елементів розробленої програми та програми в цілому;
- керування виконанням тестів та аналіз результатів тестування;
- повторне тестування.

Тестування є необхідним шляхом для оцінки якості програмного забезпечення методом експериментальної перевірки, тобто перевірки програмного забезпечення шляхом виконання тестів. Мета тестування – виявити наявність помилок або неузгодженостей. Знаходження помилок - це локалізація – задача діагностики, а досягнення відсутності помилок - відладка.

Науковці вважають, що тестування становить від 30 до 50 відсотків трудомісткості робіт зі створення коду.

Історично першим різновидом тестування було налагодження – перевірка програмного об'єкта на наявність у ньому помилок для їх усунення. При цьому можуть вноситися нові помилки.

Методи тестування й верифікації цілком залежать від методів проектування та стадій, з яких починається перевірка правильності функціонування результатів проектування.

Статичні методи використовуються під час проведення інспекцій та аналізу специфікацій компонентів без їх виконання, а динамічні застосовуються у процесі виконання програм.

Тести, фундамент яких побудовано на зовнішніх специфікаціях програмного забезпечення, застосовуються на етапі комплексного тестування для визначення повноти розв'язання функціональних задач та їх відповідності вихідним вимогам.

Перед застовуванням функціонального тестування проводиться аналіз функцій. Серед основних завдань функціонального тестування є:

- ідентифікація множини функціональних вимог;
- ідентифікація зовнішніх функцій у реалізації програмного забезпечення і побудова послідовностей функцій відповідно до їх використання;
- ідентифікація множини вхідних даних кожної функції і визначення напрямків їх зміни;
- побудова тестових наборів і сценаріїв тестування функцій;
- виявлення й подання усіх функціональних вимог за допомогою тестових наборів та проведення тестування помилок у програмі і взаємодії із середовищем.

Основна мета тестування – забезпечення повноти й узгодженості реалізованих у програмних компонентах функцій та інтерфейсів між ними.

Методи доведення правильності програм з’явились у 80-х роках XX століття. Техніка символьного виконання включає моделювання виконання коду, використовуючи символи замість змінних даних[24 плескач].

За допомогою верифікації ми перевіряємо відповідність реалізації системи специфікаціям результатів проектування й опису компоненти,

За допомогою валідації ми перевіряємо відповідність створеного програмного забезпечення потребам та вимогам замовника. Це фінансово витратний процес, але він забезпечує високий рівень якості програмного коду. Валідація завжди підтверджує той факт, що програмне забезпечення є коректною реалізацією первинних умов у системі й проводиться після завершення кожного етапу розроблення цього забезпечення.

Стан програми, при якому генеруються неправильні результати теж необхідно враховувати. Причиною помилок є недоліки в операторах програми або в технологічному процесі її розроблення, що призводить до неправильного перетворення вхідної інформації у вихідну. Дефект у програмі виникає внаслідок помилок розробника. Він може міститися у вхідних або проектних специфікаціях, текстах кодів програм, в експлуатаційній документації тощо. Відмова – це неможливість виконувати функції, визначені вимогами й обмеженнями. Вона виникає внаслідок таких причин:

- помилкової специфікації або пропущеної вимоги (специфікація точно не відображає припущення користувача);
- наявність вимоги, яку неможливо виконати на цій апаратурі і програмного забезпечення;
- помилки у проекті програми (приміром, базу даних спроектовано без захисту від несанкціонованого доступу користувача, а захист потрібен);
- помилки в алгоритмі.

Помилки у програмному забезпеченні можна класифікувати відповідно до їхнього розподілу за етапами життєвого циклу і джерел їхнього виникнення:

1) ненавмисне відхилення розробників від робочих стандартів або планів реалізації;

2) специфікації функціональних та інтерфейсних вимог без дотримання стандартів розроблення;

3) недосконала організація процесу розроблення[24].

Помилки можуть виникати під час розроблення програмного забезпечення на різних етапах життєвого циклу. Розглянемо детальніше ці етапи.

1. Етап аналізу вимог. У визначенні вхідної концепції системи та опису вхідних вимог замовника виникають помилки аналітиків, коли вони формулюють специфікації верхнього рівня і будують концептуальну модель.

Характерні помилки цього етапу:

- неадекватність опису специфікаціям вимог кінцевих користувачів;
- некоректність специфікації взаємодії програмного забезпечення із середовищем функціонування або з користувачами;
- невідповідність вимог замовника окремим і загальним властивостям програмного забезпечення;
- некоректність опису функціональних характеристик;
- незабезпеченість інструментальними засобами підтримки всіх аспектів реалізації вимог замовника тощо.

2. Етап проектування компонент. В основі проектування будь-якого програмного продукту лежить парадигма подолання складності загального завдання через декомпозицію цільових продуктів на окремі його складові - компоненти. Помилки під час проектування компонент можуть виникати при описі алгоритмів, логіки управління, структур даних, інтерфейсів, логіки моделювання потоків даних, форматів введення–виведення тощо. В основі цих помилок лежать дефекти специфікацій аналітиків та помилок проектувальників.

Найчастіше помилки виникають:

- під час визначення інтерфейсу користувача із середовищем;
- під час опису функцій (неадекватності формулювань у проекті мети та завдань окремих компонентів, що виявляються при перевірці проекту);
- у процесі опрацювання інформації або зв’язків між процесами (наслідок некоректного визначення взаємозв'язків компонентів та процесів);
- визначення даних і їх структур для окремих компонент та програмного забезпечення, що в цілому некоректно задані;
- під час опису алгоритмів модулів та їхньої логіки, що некоректно визначені в поданому проекті модуля;
- визначення умов виникнення можливих помилок у програмі;
- порушення прийнятих для проекту стандартів та технологій [24].

3. Етап кодування і налагодження. На цьому етапі виникають помилки, що є результатом дефектів проектування, помилок програмістів та менеджерів процесу розроблення і налагодження.

Характерні помилки цього етапу:

- безконтрольність допустимості значень вхідних та вихідних параметрів, ділення на 0 тощо;
- неправильна обробка нетипових ситуацій під час аналізу кодів повернення від підпрограм;
- порушення стандартів кодування (неадекватні коментарі, нераціональне виділення модулів і компонентів тощо);
- використання одного імені для позначення кількох об'єктів або кількох імен на позначення одного об'єкта;
- неузгоджене внесення змін у програму кількома розробниками.

4. Етап тестування. На цьому етапі помилки допускають тестувачі, а також програмісти, здійснюючи збирання, тестування та вибір некоректних тестових наборів і сценаріїв тестування тощо.

5. Етап супроводження. При супроводженні програмного забезпечення причиною помилок є дефекти експлуатаційної документації, слабкі показники модифікованості і зрозумілості програмного забезпечення, а також некомпетентність осіб, відповідальних за супровід та/а бо удосконалення програмного забезпечення. Залежно від сутності внесених змін на цьому етапі можуть виникати практично будь–які помилки, аналогічні раніше переліченим.

Помилки, що виникають у програмах, бувають: логічні і функціональні, обчислень, введення–виведення і маніпулювання даними, інтерфейсів тощо.

Тест – це сукупність вхідних даних і/або дій користувача із вказівкою очікуваних результатів або відповідних реакцій програми, що призначена для перевірки працездатності програми і виявлення в ній помилкових ситуацій.

Тестову перевірку можна провести також шляхом додання до програми, що перевіряється, додаткових операторів, які будуть сигналізувати про перебіг її виконання й отримання результатів.

Тестові дані, призначені для перевірки роботи системи, створюються по–різному: генератором тестових даних, проектною групою на основі документів або файлів, користувачем зі специфікації вимог тощо. Дуже часто розробляються спеціальні форми вхідних документів, у яких відображається процес виконання програми за допомогою тестових даних.

Існують такі види тестування програм з метою перевірки:

- повноти функцій;
- узгодженості інтерфейсів;

- структури програми;
- обчислення і коректності виконання функцій;
- правильності функціонування в заданих умовах;
- надійності виконання програм;
- ефективності захисту від збоїв апаратури і невиявлених помилок;
- зручності застосування та супроводження.

Багато типів тестів готує сам замовник для перевірки роботи інформаційних систем. Структура та зміст тестів залежать від виду елементу – модуля, компонентів, групи компонент, підсистем або систем. Деякі тести пов'язані з необхідністю перевірити, чи працює інформаційна система відповідно до проекту, чи задоволені вимогами самі замовника.

Для проведення тестування створюється спеціальна команда тестувачів. За функціональні тести відповідає розробник, а замовник більше впливає на складання випробувальних та інсталяційних тестів.

Як правило, команда тестувачів не залежить від штату розробників інформаційної системи. Деякі члени цієї команди є досвідченими тес–тувачами або навіть професіоналами. Це аналітики, програмісти, інженери–тестувачі, котрі присвячують увесь свій час проблемам тестування систем. Вони мають справу не лише зі специфікаціями, а й з методами та засобами тестування, організовують створення і виконання тестів на машині. Тестувачів включають до процесу розроблень з початку створення проекту для складання тестових наборів та сценаріїв, а також графіків виконання тестів.

Тести і тестові сценарії є прямим відображенням вимог та проекту в цілому. Помилки, що зустрічаються в програмі, та зміни у системі відображають у документації, вимогах, проекті. Ці помилки обов’язково вносяться до описів вхідних та вихідних даних. Зміни, що вносяться у процесі розробки, призводять до модифікації тестових сценаріїв або

заміни в планах при тестуванні. Фахівц,які проводять тестування та управляють конфігурацією повинні враховувати ці зміни і координувати упорядкування тестів.

Користувачі програми обовязково входять до складу команди що проводить тестування. Саме вони оцінюють отримані результати та зручність використання програмного забезпечення,можуть висловлювати власну думку щодо принципу роботи системи на перших етапах проекту.

Представники замовника планують роботи для тих, хто буде використовувати і супроводжувати систему. При цьому вони можуть привнести деякі зміни у проект, викликані неповнотою заданих раніше вимог, та сформулювати системні вимоги для проведення верифікації системи й ухвалення рішення про її готовність та корисність.

Типові причини, які можуть зумовити потребу змін:

- виявлення дефектів функціонування інформаційної системи під час експлуатації, не знайдених на етапі тестування (зміни, за які несе відповідальність розробник);
- з'ясування замовником під час експлуатації інформаційної системи, що вимоги до системи були висловлені недостатньо або неповно, і тому вона не відповідає окремим потребам замовника (зміни, за які несе відповідальність постановник задачі);
- зміна умов діяльності замовника, які не відповідають раніше поставленим вимогам (приміром, змінилися податкове законодавство або місцева регуляція правил бізнесу, способи комунікації замовника з бізнес–партнерами або відбувся перерозподіл їхніх ролей у бізнесі тощо).

За свідченням експертів, процес який несе зміни до процесу тестцування є досить дорогим у фінгасових затратах – оцінки його вартості сягають до 80 відсотків від загальної вартості розробки проекту.

Існують і додаткові види супроводження:

- коригувальне – внесення коректив для усунення помилок, які було знайдено після передачі системи до експлуатації;
- адаптивне – адаптація продукту до змінених обставин використання після передачі системи в експлуатацію;
- попереджувальне – діяльність із забезпечення адаптивного супроводження на старті розроблень.

3.2. Порівняльній аналіз тестування програмних систем

Науковці Харківського національного економічного університету О.В. Щербаков, Є. С. Луценко, Ю.І. Скорін пропонують таку методику кількісної оцінки роботи тестувальників, яка враховує такі фактори:

- кількість дефектів програмного продукту, знайдених тестувальником;
- значимість виявлених дефектів на підставі баг-репорту відповідно до встановленої градації помилок;
- тривалість періоду, протягом якого тестувальник виявив дефекти програмного продукту [32].

Для побудови звичайної диференціальної моделі використовуємо такі означення:

Q(t) – приведена кількість дефектів, виявлених

за певний проміжок часу;

t – час, облік якого ведеться у робочих днях;

dt / dQ – «швидкість» виявлення дефектів програмного продукту, тобто кількість помилок, знайдених в одиницю часу.

Вибір виду диференціального рівняння відносно функції Q = Q(t) ґрунтується на таких міркуваннях, що є наслідком аналізу реальних результатів роботи тестувальника.

1. Від початку тестування функція Q = Q(t) монотонно зростає.

2. «Швидкість» зростання функції Q = Q(t) уповільнюється ближче до завершення процесу тестування, тобто обернено пропорційна Q.

3. На початковому етапі тестування dt / dQ змінюється за квазілінійним законом [27].

Зазначені припущення щодо зміни Q = Q(t) добре узгоджуються з особливостями поведінки функції y = ln(x) , тому диференціальне рівняння

записуємо у вигляді:

$dQ / dt = lnQ$ (1)

з початковою умовою:

$Q(t0) = Q0$ (2)

Розв’язання цієї задачі Коші здійснюється таким чином:

$\int dQ /lnQ = \int dt + c$ (3)

або .

$t = \int dQ / lnQ - c$ (4)

де c – константа інтегрування ; а $I = \int dQ /lnQ$ - «інтегральний логарифм» який елементарно не інтегрується.

Розв’язок інтегрального логарифму можна репрезентувати у вигляді нескінченого ряду:

$I = \int dQ / lnQ = lnlnQ + lnQ + (lnQ)^2 / 2x2! + (lnQ)^2 / 3x3! + ...$ (5)

Якщо обмежимось першими числами ряду (залишаємо чотири члени), маємо наближену форму розв’язку:

$I \int lnlnQ + lnQ + (1 / 4 + 1 / 81)(lnQ)^2$

Після очевидних перетворень одержуємо:

$I = lnlnQ + lnQ + 0{,}137(lnQ)^2$, або

$I = ln(QlnQ) + 0{,}137(lnQ)^2$ (6)

Константу інтегрування C обираємо у такому вигляді $C = - lnD$, тоді на підставі (4) та (6) маємо:

$t = \ln(DQ\ln Q) + 0{,}137(\ln Q)^2$ (7)

Чисельне значення константи D знаходимо відповідно до початкової умови, що узгоджується з доступними даними результатів роботи тестувальників, а саме, Q(3) =18 :

$3 = \ln(D.18.\ln 18) + 0{,}137(\ln 18)^2$.

Звідки маємо D = 0,126.

Остаточно одержуємо формулу:

$t = \ln(0{,}126Q.\ln Q) + 0{,}137(\ln Q)^2$ (8)

Для оцінки якості роботи тестувальника пропонується (на підставі формули (8)) провести обчислення. Приведену кількість виявлених дефектів з врахуванням градації їх значимості наведено у табл. 1.

Таблиця 1

Градація значимості дефектів

Градація значимості дефектів	Кількість дефектів, знай-дених тестувальником
Блокуючий	Blt
Критичний	Crt
Значний	Mat
Незначний	Mit
Тривіальний	Trt
Разом	Nt

Коефіцієнти, що враховують значимість виявлених помилок, можна обирати відповідно до перших п’яти чисел ряду Фібоначчі (1, 1, 2, 3, 5), сума яких становить 12, а саме:

k1 = k2 = 1/12; k3 =1/ 6; k4 =1/ 4; k5 = 5/12.

Тоді кількість дефектів, виявлених тестувальником обчислюється за формулою:

$Q_m = 1/12\ (Trt + Mit) + 1/6\ Mat + 1/4\ Crt + 1/5\ Blt$

Далі за формулою (8) визначається:

$t^* = \ln(0{,}126 Q_m \ln Q_m) + 0{,}137(\ln Q_m)2.$

Якщо тривалість часу tm , фактично витраченого тестувальником на виявлення дефектів у кількості Qm , не перевищує значення параметру t* більше ніж на 5%, роботу тестувальника можна визнати ефективною. При цьому, чим більше зменшується tm у порівнянні з t* , тим ефективніше робота тестувальника [32].

Отже, тестування програмного забезпечення є досить складним і важливим етапом розробки програмного забезпечення. При цьому, оцінка ефективності роботи тестувальників програмного забезпечення є необхідним елементом комплексу заходів щодо забезпечення якості програмних продуктів. Відсутність ефективних методів такої оцінки може негативно впливати на якість програмного забезпечення. Запропонована науковцями методика дозволяє отримати кількісні оцінки ефективності тестування програмних продуктів на підставі математичної моделі. Отримані за допомогою даної методики оцінки дозволять підвищити ефективність як процесу тестування, так і процесу розробки програмного забезпечення в цілому[32]. Таким чином розроблена методика показує результати, що враховують більше факторів та відповідно мають більш високе практичне значення при застосуванні у реальному процесу тестування.

ВИСНОВКИ

На сучасному етапі розвитку ІТ-технологій тестування є одним з найбільш важливих засобів перевірки надійності програмного забезпечення. Однак, зазвичай це ручне тестування, що є трудомістким процесом без ефективної автоматизації, яке потребує багато часу та фінансових ресурсів, але не здатне забезпечити виявлення усіх відмов програмного забезпечення. Важливим завданням у сфері програмної інженерії є автоматизація процесу тестування, ефективна побудова сценаріїв тестування, що дозволить скоротити часові та економічні затратити.

Підвищення надійності програмного продукту є надзвичайно важливою та актуальною науковою задачею. Основним засобом вирішення даної проблеми є тестування програмного забезпечення, що вимагає значних затрат ресурсів, адже у більшості випадків проводиться вручну та повинно охоплювати усі сценарії використання і потоки управління та даних програмного продукту.

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 і вимогам замовника. Усунення неточностей і помилок у документації - запорука того, що створюване 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 і аналізуються дані про відмови та збої в роботі програми.

У дослідженні значну увагу приділено основним задачам функціонального тестування. Метою функціонального тестування є виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог. Функціональні тести повинні охоплювати всі реалізовані функції з урахуванням найбільш ймовірних типів помилок. Тестові сценарії, що поєднують окремі тести, орієнтовані на перевірку якості розв'язку функціональних задач.

У третьому розділі подано технологію виконання тестування програмних систем та порівняльній аналіз тестування програмних систем.

У процесі тестування розробники мають справу з ризиками, властивими проекту в цілому. Головна мета тестування – знайти якомога більше дефектів, найбільш важлива категорія ризику для тестування – це ризики відмов. Для аналізу причин відмов та їх наслідків найбільш придатними є формальні методи верифікації – аналіз дерев подій, аналіз дерева відмов та аналіз режимів і наслідків відмов.

У дослідженні ми дійшли висновку, що тестування програмних систем є досить складним і важливим етапом розробки програмного забезпечення. При цьому, оцінка ефективності роботи тестувальників програмних систем є необхідним елементом комплексу заходів щодо забезпечення якості програмного забезпечення. Відсутність ефективних методів такої оцінки може негативно впливати на якість програмного забезпечення. Запропоновані у дослідженні методи дозволяють отримати кількісні оцінки ефективності тестування програмних продуктів на підставі математичної моделі. Отримані результати дозволять підвищити ефективність як процесу тестування, так і процесу розробки програмного забезпечення в цілому.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Агапов А.С., Зенин С.В., Михайловский Н.Э., Мкртумян А.А. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504-CMM), Пер. с англ. Москва, "Книга и бизнес", 2001. – 375 с.

2. Андон Ф.И, Коваль Г.И., Коротун Т.М., Суслов В.Ю. Парадигма качества программного обеспечения //Проблемы программирования. – 1999. - №2. – С. 51-62.

3. Андон Ф. И., Коваль Г. И., Коротун Т. М., Лаврищева Е. М., Суслов В. Ю. Основы инженерии качества программных систем. – Киев: Академпериодика, 2007.– 680 с.

4. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. – СПб.: Питер, 2004. – 320 с.

5. Вакалюк Т.А. технологіх тестування програм. Навчально-методичний посібник для студентів напряму 6.040302 Інформатика.. – Житомир: Вид-во ЖДУ,2013. – 96с

6. Дейкстра Э. Дисциплина программирования. – М.: Мир, 1978.

7. Дідковська М. В. Тестування: Критерії та методи. Текст лекцій. Частина II – К., 2010. //Електронний ресурс. Режим доступу: http://mmsa.kpi.ua/studentam/study-materials-ua/testing/testuvannya-kriter456-ta-metodi-tekst-lekc456i-chastina-456456

8. Калбертсон Роберт, Браун Крис, Кобб Гэри. Быстрое тестирование. – М.: «Вильямс», 2002. – 374 с.

9. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. – Киев: Диа-Софт, 2001. – 544 с.

10. Коваль Г.И., Коротун Т.М., Остапенко А.П. Превентивное тестирование и оценка надежности программного обеспечения как форма

управления риском проекта. //Сб. Программная инженерия. – Киев., 1993. – С. 19-26.

11. Коул Д., Горэм Т., МакДональд М., Спарджеон Р.. Принципы тестирования ПО // Открытые системы. – №2. – 1998. – С. 60-63.

12. Коротун Т.М. Моделі і методи інженерії тестування програмних систем в умовах обмежених ресурсів. Дис... канд. фіз.-мат. наук: 01.05.03 – К., 2005. – 127с.

13. Лавріщева К. М. Проблеми програмування. Спецвипуск. – 2008, – № 2-3. –С.191-204.

14. Лавріщева К. М. Програмна інженерія. Підручник. – К.: Академперіодика, 2008. – 319 с.

15. Лавріщева К. М. Методы программирования. Теория, практика, инженерия. – К.: Наукова думка. 2006. – 471с.

16. Лаврищева Е.М., Коротун Т.М. Построение процесса тестирования программных систем // Проблемы программирования. –2002. – № 1-2. – С. 272-281.

17. Лаврищева Е. М., Петрухин В. А. Методы и средства инженерии программного обеспечения. – Москва: МФТИ. –2007.– 415 с.

18. Леффингуэлл, Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход : Пер.с англ. / Леффингуэлл, Д Уидриг. – Москва : Вильямс, 2002. – 448 с/

19. Майерс Г. Искусство тестирования программ /Пер. с англ. под ред. Б. А. Позина. – М.: Финансы и статистика, 1982. – 176 с. // Електронний ресурс. Режим доступу: http:// http://computersbooks.net/index

20. Макгрегор Дж, Сайкс Д. Тестирование объектно-ориентированного программного обеспечения. – К: Диасофт, 2002. – 432 с.

21. Нитребич О. О. Моделі та методи оцінювання надійності програмного забезпечення з урахуванням його архітектури : автореферат дисертації на здобуття наукового ступеня кандидата технічних наук :

01.05.03 - математичне та програмне забезпечення обчислювальних машин і систем./ Міністерство освіти і науки України, Національний університет «Львівська політехніка». - Львів, 2015. – 22 с.

22. Орлов, С. А. Технологии разработки программного обеспечения : Учебник для вузов. Доп. Мин. обр. РФ / С.А.Орлов. – Санкт–Петербург: Питер, 2002. – 464 с.

23. Основы тестирования программного обеспечения: Учебное пособие / В.П. Котляров, Т.В. Коликова. – М.: БИНОМ. ЛЗ, ИНТУИТ.РУ, 2006г. // Електронний ресурс. Режим доступу: http:// window.edu.ru/resource/713/41713/files/MSTesting.pdf

24. Плескач В.Л., Затонацька Т.Г. Інформаційні системи і технології на підприємствах: Підручник. – К: Знання, 2011. // Електронний ресурс. Режим доступу: http://pidruchniki.com/1059110247701/informatika.

25. Роберт Дж. Оберг Технология COM+. Основы и программирование = Understanding and Programming COM+: A Practical Guide to Windows 2000 First Edition. – М.:«Вильямс», 2000. – 480с.

26. Семакин И.Г., Вараксин Г.С. Структурированный конспект базового курса информатики. – М.: Лаборатория Базовых Знаний, 2000. – 167 с . Соммервилл И. Инженерия программного обеспечения: 6 издание. Москва: Вильямс, 2002. – 624 с.

27. Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения. – М.: БИНОМ, 2008. – 368 с.

28. Струбицький П.Р., Бондар О.В. Програмна інженерія – Навчальний посібник – Тернопіль: ТАНГ, 2004 – 76 с.

29. Тимошенко Ю.О., Дідковська М.В., Кобринський С.Ю. Розробка методу функціональної сегментації для тестування програмного забезпечення // Наукові вісті, №5(37), – Київ, 2004. – С. 48-56.

30. Управление риском проектов программного обеспечения / Андон Ф.И. Суслов В.Ю. Коротун Т.М., Коваль Г.И. Слабоспицкая О.А. // Проблемы программирования. – 1999. - № 1. – С. 53-62.

31. Элиенс Антон. Принципы объектно-ориентированной разработки программ/ Пер. с англ. / Антон Элиенс. – 2–е изд. – Москва : Вильямс, 2002. – 496 с.

32. Щербаков О. В. Методика кількісної оцінки ефективності роботи інженера із забезпечення якості програмного забезпечення / О. В. Щербаков, Є. С. Луценко, Ю. І. Скорін // Системи обробки інформації. – 2012. – Вип. 4(1). - С. 81-84. // Електронний ресурс. Режим доступу: http://nbuv.gov.ua/j-pdf/soi_2012_1_4_21.pdf.

33. Федасюк Д.В., Яковина В.С., СердюкП.В., Нитребич О.О. Метод побудови сценаріїв тестування програмного забезпечення на основі аналізів його змінних //Інформаційні технології та комп'ютерна інженерія. – 2014, № 2. С. 50-58.

34. ANSI/IEEE Std 729-1983. Толковый словарь терминологии, используемой при разработке ПО // IEEE, New-York, 1983, – 30 p.

35. Bourdonov I., Kossatchev A., Petrenko A., and Galter D.. KVEST: Automated Generation of Test Suites from Formal Specifications //FM'99: Formal Methods. LNCS 1708, Springer-Verlag, 1999, pp. 608–621.

36. Barnett M., Fahndrich M., P. de Halleux, Logozzo F., Tillmann N.. Exploiting the Synergy between Automated-Test-Generation and Programming-by-Contract. Proc. of ICSE 2009, Vancouver, Canada, May 2009.

37. Guide to the Software Engineering Body of Knowledge – Washington IEEE, 2004. – 324 с.

38. Leveson N. Safeware: system safety and computers. Addison-Wesley Publishing Company, 1995. - 680 p.

39. Myers G.J. The Art Of Software Testing [Text] / G.J. Myers – New York: John Wiley & Sons, Inc., 2004. – 254 p. – ISBN 0- 471-46912-2

40. Standart-ISO-9126. // Електронний ресурс. Режим доступу: http://900igr.net/kartinki/informatika/Programmnye-sredstva/010-Standart-ISO-9126-opredeljaet-kharakteristiki-kachestva-kompleksov.html

41. Електронний ресурс. Режим доступу**:** http://lib.mdpu.org.ua/e-book/vstup/zmist.htm