РОЗДІЛ 1
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ЯК ОБ'ЄКТ ДІАГНОСТУВАННЯ

1.1. Аналіз процесу діагностування програмного забезпечення

Надійність і достовірність роботи сучасних мікропроцесорних пристроїв, комп'ютерів та комп'ютерних систем є запорукою успішного вирішення задач, що розв'язуються ними, та довготривалої експлуатації цих пристроїв та систем. Одним з основних способів, за допомогою котрого досягається висока надійність функціювання обчислювальної техніки та вірність розв'язку прикладних задач, є діагностування програмних продуктів (у подальшому програмного забезпечення (ПЗ)) на різних етапах його життєвого циклу, зокрема, на етапах проектування, розроблення та експлуатації.

Надійність програмного забезпечення визначається як властивість програми виконувати задані функції у заданих умовах роботи і на заданій ЕОМ, аналогічно поняттю надійності апаратних засобів ЕОМ [6, 7, 45].

Якщо надійність ПЗ розглядають з точки зору імовірнісного підходу, то під надійністю ПЗ розуміють імовірність того, що при функціюванні комп'ютерного пристрою чи системи протягом певного часу не буде виявлено жодної помилки [16]. Взагалі, таке визначення неточне, тому що в ньому не врахована вага кожного типу помилок. А помилки є нерівноцінними за своїми наслідками при розв'язку задач, що вирішуються ПЗ.

Під *помилкою ПЗ* взагалі розуміють неправильність його функціювання, що призводить до хибного чи спотвореного результату обчислювального процесу [7, 13, 18].

Поняття *помилки в програмі* в літературних джерелах трактується порізному, але всі вони погоджуються з тим, що помилки не можна розглядати тільки як відхилення від еталону. Інколи програми відповідають формалізова-

ним еталонам, але порушуються правила здорового глузду, які не передбачені еталонами. Тому вважатимемо, що в програмі наявна помилка, якщо програма не виконує того, що очікує від неї користувач. Як правило, відсутність програми-еталона унеможливлює локалізацію і виправлення помилки у програмі, що розробляється, шляхом порівняння її з програмою без помилок (еталоном).

Більш детально помилки та їх типи будуть розглянути у наступному розділі.

Повернемося до розгляду надійності ПЗ та властивостей, що її характеризують. Надійність ПЗ є комплексною властивістю, яка в залежності від умов його призначення та застосування відображає, в свою чергу, ряд властивостей, зокрема безвідмовність, довговічність, відновлюваність, усталеність функціювання та ін.

Безвідмовність ПЗ оцінюють ймовірністю його роботи без відмов у певних умовах на протязі певного часу [7]. Під *відмовою* при цьому розуміють подію, яка полягає у втраті системою (ПЗ) здатності виконувати необхідну функцію [45]. З визначення надійності зрозуміло, що безвідмовність ПЗ принципово відрізняється від безвідмовності апаратних засобів комп'ютерної техніки. Програми при цьому не виходять з ладу. На відміну від апаратних засобів, їх безвідмовність залежить від кількості помилок в них, які були внесені на етапі проектування.

Порушення роботоздатності ПЗ проявляється у вигляді збоїв. Під *збоєм*, у тому числі і ПЗ, розуміють відмову, що самоусувається [45].

Щодо *відновлюваності*, то вона полягає у коректуванні і відновленні тексту програм, виправленні даних, внесенні певних змін в ПЗ і таке інше. Вона оцінюється середньою тривалістю часу і витратами праці на усунення помилки або згаданих вище операцій.

Під *усталеністю функціювання ПЗ* розуміють здатність обмежувати наслідки власних помилок і несприятливих впливів зовнішнього середовища та

протистояти їм. Джерелами несприятливих впливів можуть бути некоректність вхідних даних, помилки оператора і таке інше [46].

Розглянемо механізм виникнення відмов ПЗ, враховуючи те, що відмова ПЗ зумовлена невідповідністю програмного забезпечення поставленим задачам. Як правило, вона виникає внаслідок порушення специфікації, тобто технічних вимог до програми, або неточної чи неповної специфікації [45]. Порушення специфікації частіше зустрічається в складних програмних системах, де окремі помилки програміста важко спостережувані і тому залишаються невиявленими. Неточність чи неповність специфікації виникає за причини численних факторів при складанні специфікації. На початковому етапі розроблення програми вони невідомі і з'ясовуються поступово у процесі її експлуатації.

Відмови ПЗ може зумовлювати також *прихованість помилок*. У цьому випадку помилка проявляється тільки в окремих комбінаціях, що рідко зустрічаються. Тому вони виявляються тільки у процесі тривалої експлуатації ПЗ. Такі помилки є найбільш небезпечними.

Крім того, надійність ПЗ певною мірою залежить від кількості помилок, які внесені під час його розроблення і не усунуті. Такі помилки мають бути виявлені і усунуті у процесі експлуатації.

Якщо помилок виявляється і виправляється більше, ніж вноситься нових (або нові не вносяться), то надійність ПЗ має тенденцію зростання, і чим інтенсивніше усуваються помилки, тим швидше зростає надійність ПЗ [6, 7].

Виходячи з вищенаведених міркувань, *під надійністю* можна розуміти одиничний показник якості, який характеризує ПЗ, а саме його властивість виявляти в процесі експлуатації помилки, що залишилися в ньому.

Взагалі, для оцінки надійності ПЗ найчастіше використовують імовірнісні (статистичні) показники. Основними з них є: імовірність безвідмовної роботи, імовірність відмови та частота відмов. Їх розрахунок досить детально описаний в численних літературних джерелах, зокрема [45, 6, 7].

Як вже згадувалось, діагностування є одним з основних методів забезпечення надійності ПЗ. Загальне поняття *діагностування* викладене у [47]. Воно трактується як визначення технічного стану об'єкта з означеною точністю. При цьому мається на увазі, що об'єкт може мати один справний стан і безліч несправних.

Щодо діагностування апаратного і програмного забезпечення мікропроцесорних пристроїв та комп'ютерних систем, то при цьому використовують поняття *діагностичної програми* – програми, призначеної для виявлення, локалізації та опису пошкоджень технічного устаткування або помилок програми [48]. Таким чином, діагностична програма є інструментом для виявлення, локалізації та опису помилок програми.

Діагностування здійснюється на різних етапах життєвого циклу ПЗ, зокрема, на етапах планування, проектування та кодування. На *етапі планування* проводиться аналіз і оцінка вимог до ПЗ як до продукту, а також до його характеристик і функційних можливостей. На *етапі проектування* код програми ще не розроблений. Діагностуються вже повністю формалізовані і детально описані ідеї, на відміну від етапу планування. Основна увага приділяється тому, чи проект є таким, що на його основі буде створено компактний, ефективний, легкотестований, прийнятний для модернізації і супроводження програмний продукт, чи відповідає проект вимогам, що визначені в документації на етапі планування, чи проект описує всі взаємозв'язки і передачу даних між модулями, умови роботи кожного модуля та їх реалізацію, чи достатньо і правильно в проекті описана підсистема опрацювання помилок. На *етапі кодування* програміст складає і тестує програму.

У певних випадках виправлення однієї помилки призводить до появи інших. Тому програму, що розробляється, тестують, як правило, декілька разів. Для якісного тестування може знадобитися 8-10 циклів.

Таким чином, діагностування ПЗ складається з такої послідовності операцій [6-8]:

- тестування кожного окремого модуля - з метою встановлення різниці між його логічною схемою, інтерфейсом та зовнішніми специфікаціями програми;
- функційне тестування - з метою виявлення протиріч між функціями, які визначені специфікацією, і програмою, що розробляється;
- системне тестування з метою виявлення протиріч між програмою і цілями, визначеними користувачем;
- синтаксичний і семантичний контроль кодування модулів при їх трансляції в машинні коди.

При цьому слід мати на увазі, що *тестування* полягає у перевірці правильності результатів роботи програми з спеціальними наборами вхідних даних, які називають *тестами*.

Після здійснення вищезазначених операцій, як правило, проводяться випробування діагностованого ПЗ.

Щодо поняття “діагностування ПЗ”, то воно є більш загальним у порівнянні з поняттям “тестування ПЗ”, яке є основною складовою діагностування.

Історію розвитку діагностування ПЗ відображає еволюція розроблення самого програмного забезпечення. На протязі тривалого часу основна увага приділялась розробленню глобальних наукових програм та програм міністерств оборони, які базувались на корпоративних базах даних і проектувались на базі універсальних ЕОМ чи мінікомп’ютера. Опис тестів здійснювався на папері. Кінцевий набір тестових процедур давав можливість ефективно протестувати всю систему. Тестування проводилось, як правило, в кінці розроблення проекту і тим же персоналом, яким розроблявся цей проект. Поява персональних комп’ютерів дала змогу стандартизувати значну частину операцій проектування і тестування. Це призвело до швидкого росту комерційних розробок. Йшла жорстка конкуренція серед програмних додатків, що складали їх основу. Програмне забезпечення, що “виживало”, приймалось як стандарт. Системи пакетної обробки стали замінятися системами, що працювали у реальному часі, а во-

ни вимагають іншого, ніж раніше, підходу до діагностування ПЗ, оскільки потоки команд і даних можуть викликатися у будь-якому порядку. Це призвело до появи дуже великої кількості процедур тестування. Вони мають підтримувати нескінченну кількість сполучень і перестановок.

Наступний стан у розвитку ПЗ грунтується на перевагах архітектури “клієнт-сервер” за допомогою використання зовнішнього графічного користувацького інтерфейсу (GUI), що керує серверною базою даних. Архітектура “клієнт-сервер” описує співвідношення між двома процесами програмного забезпечення. У цьому тандемі клієнтський комп’ютер обслуговується серверним. Зазвичай клієнтські і серверні операції реалізуються на різних комп’ютерах, які об’єднані у мережу. З появою клієнт-серверних програм тестування ПЗ ускладнилося. Якщо раніше тестувальник виконував завершену діагностичну програму в одній системі, то тепер клієнт-серверна архітектура передбачає структуру системи, до складу якої входять три основні складові: сервер, робоча станція і мережа. Ймовірність виникнення помилок у такій системі зросла за причини наявності дуже малої кількості стандартів тестових програм для неї. Та й тестування стало залежати від роботоздатності трьох згаданих компонентів.

GUI-інтерфейс, здійснюючи представлення інформації у вікнах на екрані користувача, з точки зору тестування додає своїх проблем. Середовище, що керує подіями, значно відрізняється від процедурного середовища універсальної ЕОМ.

Високий ступінь випадковості при виборі об’єктів на базі клієнт-серверних додатків зумовлений різними шляхами їх виклику і виконання. Це є причиною неможливості виконання всіх функційних процедур тестовими програмами. Тому тестувальники змушені зосередити свою увагу тільки на розробленні тестових програм, які стосуються системних вимог та способів і можуть бути використані користувачем.

В міру розвитку сценаріїв навігації користувача основна увага стала приділятися можливості запису і відтворення зображення. Цю можливість стали

реалізовувати засоби автоматизації тестування [17, 49-56]. Роботи з тестування ПЗ ускладнились. Крім знання тестувальниками програми, що має тестуватися, необхідно було мати специфічні навички, пов’язані з роботою з певними платформами та мережами, а також із засобами автоматизованого діагностування. На сьогодні спостерігається посилення тенденції виконання автоматизованого тестування за допомогою програм.

Досить актуальними є проблеми, що виникають при розробленні діагностичних програм, а саме:

1) випередження розвитку архітектури комп’ютерів та їх апаратної досконалості методів та засобів побудови тестових програм;

2) недостатність вмінь для побудови діагностичних програм, котрі повністю враховували б вимоги, що ставляться до розробки сучасного ПЗ;

3) низька якість розроблення діагностичного ПЗ, що знижує ефективність експлуатації існуючого ПЗ.

1.2. Складові діагностування ПЗ. Тестування, верифікація, атестація та налагоджування

В літературних джерелах та стандартах [6, 7, 13, 16, 18, 19, 45 - 49, 57-61] найчастіше розглядаються такі складові діагностування ПЗ, як верифікація, атестація, тестування і налагоджування. Інколи частина спеціалістів їх плутає або намагається операціями однієї складової перекривати інші складові, тобто переносить частину операцій складових у процеси, які їм не притаманні. Тому варто уточнити зміст кожної з основних складових процесу діагностування як єдиного цілого і вказати конкретні технологічні операції та зв’язки між цими складовими.

Найбільш вагомими операціями, що охоплюють всі етапи життєвого циклу ПЗ, є верифікація і атестація. Суть *верифікації* полягає у доведенні того, що поведінка програми відповідає специфікації на цю програму [48]. Щодо верифі-

кації і атестації, то ними називають [19, 62] процеси перевірки та аналізу ПЗ, під час яких перевіряється програмне забезпечення на відповідність специфікації та вимогам замовників. Починається верифікація і атестація ще на етапі аналізу вимог і закінчується на етапі перевірки програмного коду готової програми шляхом тестування.

Верифікація і атестація являють собою різні технологічні операції діагностування. Результатом першої є висновок про те, чи правильно розроблене ПЗ, а другої – чи ПЗ працює правильно. Якщо під час верифікації перевіряється відповідність ПЗ системній специфікації, зокрема функційним та нефункційним вимогам, то під час *атестації* необхідно впевнитись, що ПЗ відповідає вимогам і очікуванням замовника. Атестація системних вимог дуже важлива на ранніх етапах розроблення ПЗ за причини помилок та упущень в них. Зрозуміло, що за таких обставин кінцевий продукт не буде відповідати очікуванням замовника. Але сама тільки атестація не може звичайно виявити всі помилки і недоречності у специфікації вимог. Частина помилок знаходиться і усувається тільки після завершення розроблення системи.

Верифікація і атестація реалізуються на основі двох методик перевірки і аналізу ПЗ: інспектування ПЗ та тестування ПЗ [19].

Суть *інспектування ПЗ* полягає в аналізі і перевірці представлень системи, зокрема, специфікації вимог, архітектурних схем та вихідних кодів програм. Інспектування виконується на всіх етапах процесу розроблення ПЗ. Воно може реалізовуватись паралельно з автоматичним аналізом вихідного коду програм і відповідних документів. Як інспектування, так і автоматичний аналіз є *статичними методами* верифікації і атестації внаслідок того, що для їх здійснення не потрібна виконувана система.

На сьогоднішній день, незважаючи на широке застосування інспектування ПЗ, переважаючим підходом при реалізації верифікації та атестації залишається тестування ПЗ.

Про суть тестування згадувалось у попередньому параграфі. Що ж до реалізації верифікації і атестації, то суть *тестування* полягає у перевірці роботи програм з даними, що подібні реальним і мають оброблятися у процесі експлуатації ПЗ. Під час тестування виявляються відмови та дефекти програм і невідповідність вимогам. Це здійснюється шляхом дослідження даних на виході системи і виявлення серед них таких, яких не повинно бути на виході .

Тестування здійснюється у цьому випадку як на етапі реалізації програмної системи з метою перевірки її відповідності очікуванням замовника, так і після завершення її реалізації.

Інформаційні потоки процесу верифікації і атестації представлено на рис.1.1.

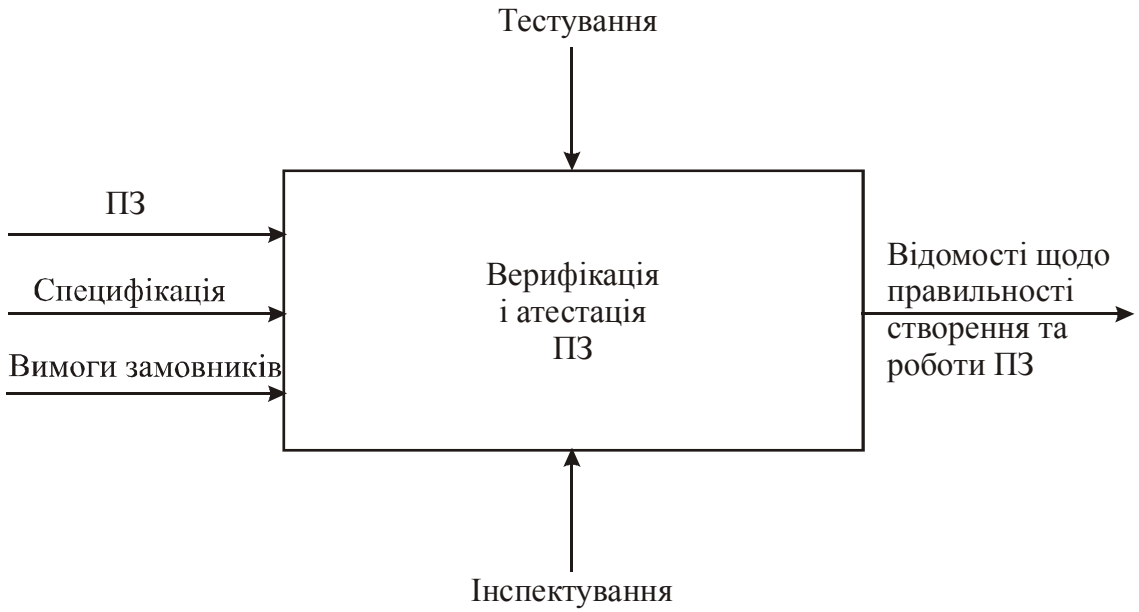

Рис.1.1. Інформаційні потоки процесу верифікації та атестації програмного забезпечення

Є багато трактувань і розумінь тестування програм та програмного забезпечення. Одним з найбільш узагальнених є таке трактування, коли під *тестуванням* розуміють процес виявлення відмов за причини наявності помилок у програмах [57].

Варте уваги трактування тестування як запуску виконуваного коду з тестовими даними і дослідження даних на виході програмної системи та робочих

характеристик програмного продукта для перевірки правильності роботи системи [19].

Найбільш поширеним трактуванням тестування є таке, коли *суть тестування* (testing) полягає у перевірці роботи програм за результатами їх виконання з використанням спеціально підібраних вихідних даних, які називають *тестами* [6, 7]. Це, по суті, метод виявлення наявності помилок у програмах шляхом опрацювання тестових даних і порівняння одержаних при цьому результатів з розрахунковими.

У багатьох випадках спеціалістам, які виконують тестування ПЗ, додають інші завдання, зокрема, виконання налагодження ПЗ. Найчастіше це робиться, коли тестувальник здійснює тестування програмних модулів та перевірку їх взаємодії, а також функціювання компонентів програмної системи. Тому інколи в процес тестування програм (рис.1.2) помилково включають операції, що відносяться до процесу налагодження, суть якого полягає у виявленні, локалізації та усуненні помилок у програмі [48].

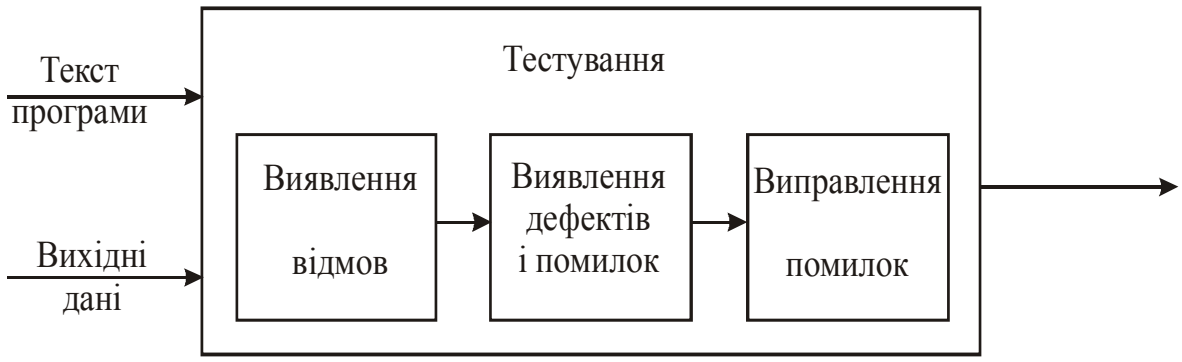

Рис.1.2. Процес тестування програмного забезпечення

Взагалі, тестування і налагодження є двома відокремленими процесами. Під час тестування виявляється тільки наявність чи ненаявність відмов за при-

чин наявності помилок. При цьому тестування не передбачає виявлення і усунення помилок. Це здійснюється тільки у процесі налагодження. Тому під *налагодженням* розуміють процес виявлення джерел відмов чи помилок і внесення в програму відповідних виправлень [14, 20, 48, 57, 63-65].

Взаємозв'язок тестування і налагодження ПЗ та інформаційні потоки цих процесів показано на рис.1.3. Результати тестування можуть бути використані для розроблення моделі надійності програм і оцінки надійності ПЗ.

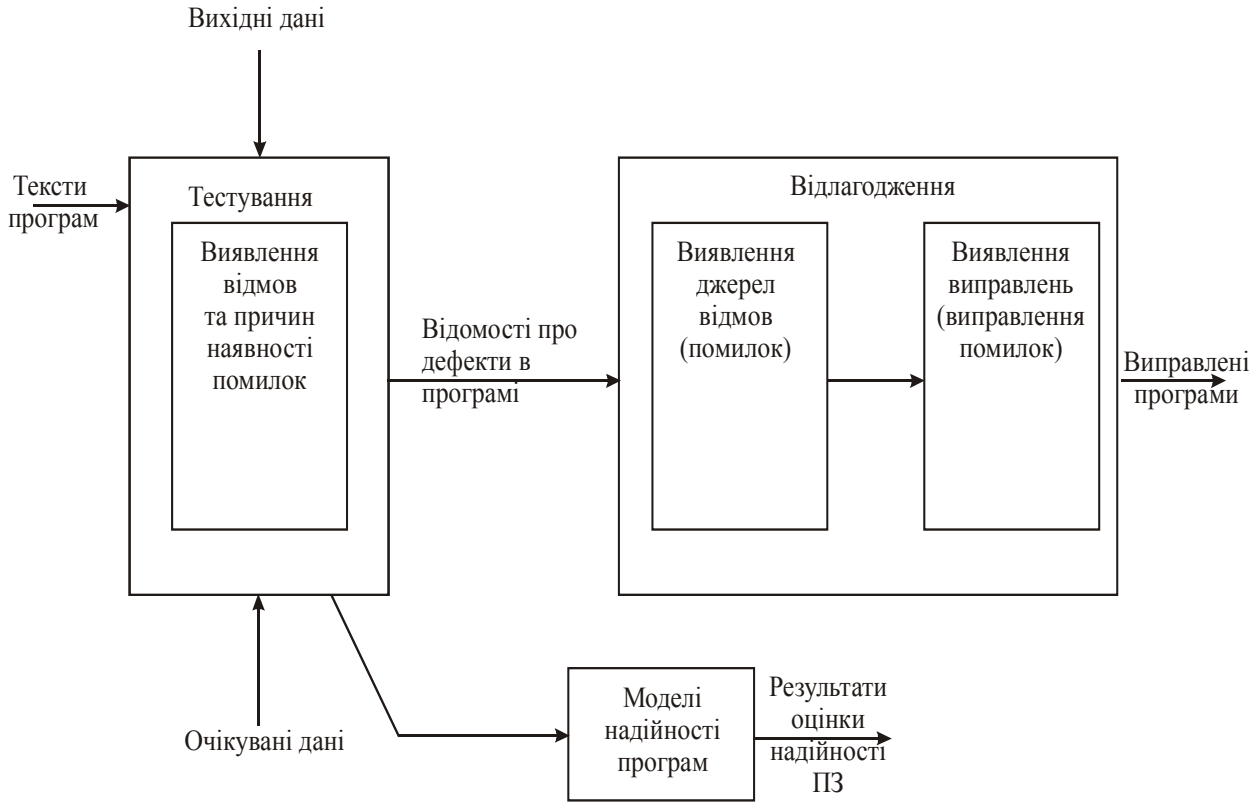

Рис.1.3. Взаємозв'язок тестування та налагодження програмного забезпечення

Якщо розглядати тестування як складову атестації чи верифікації, то на різних етапах розроблення ПЗ застосовують різні види тестування, зокрема, *тестування дефектів* і *статистичне тестування*. Суть першого полягає у виявленні невідповідностей між програмою та її специфікацією, які зумовлені де-

фектами чи помилками у програмах. При цьому слід мати на увазі, що тести розробляються для виявлення помилок у системі, а не для імітації її роботи.

Статистичне тестування здійснюється з метою оцінки продуктивності та надійності системи, а також перевірки її роботоздатності у різних режимах роботи. Тест розробляється з метою імітації реальної роботи системи з реальними вихідними даними. Надійність ПЗ оцінюється за кількістю збоїв при опрацюванні програм, а продуктивність – за тривалістю виконання операцій і відгуку системи при опрацюванні тестових даних.

Що стосується процесу налагодження, то він є найскладнішим з точки зору реалізації і складності етапів (рис.1.4).

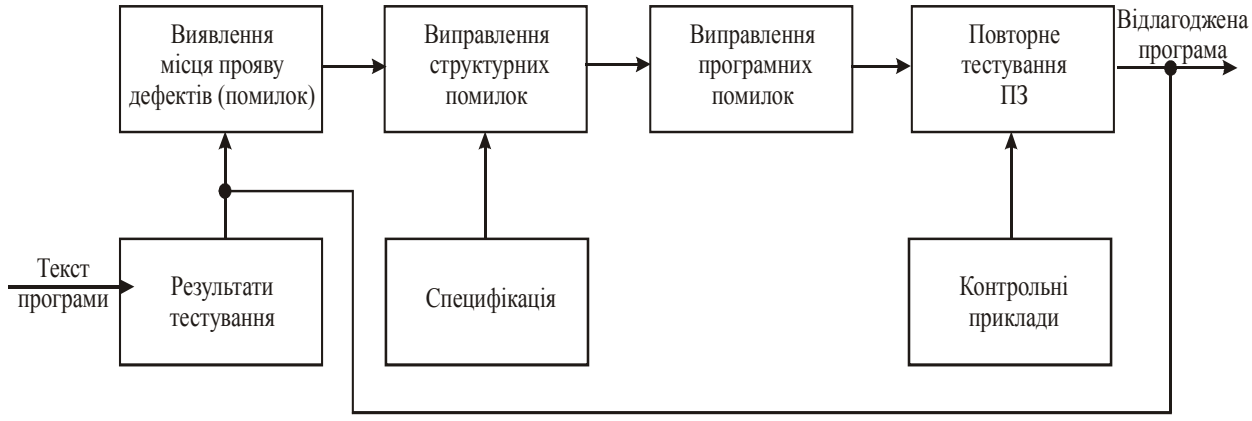

Рис.1.4. Етапи процесу налагодження

Ідеальним може бути випадок, коли на етапі тестування у програмах не було знайдено дефектів і помилок. Тоді програма вважається налагодженою. Але така ситуація на практиці є малоймовірною. Швидше за все застосовані тести не здатні виявляти серйозні помилки.

Кваліфікація налагоджувача ПЗ повинна бути досить високою, інколи вищою за кваліфікацію розробника ПЗ, тому що для знаходження і усунення помилок він повинен досконало знати суть і принципи розроблення об’єкта діагностування та налагодження.

Приймаючи до уваги вищевикладене, стає зрозумілим, що тестування ПЗ є основним з етапів як діагностування, так і налагодження. Тому розглянемо детальніше типи і методи тестування ПЗ (рис.1.5). Основну увагу приділимо тестуванню дефектів.

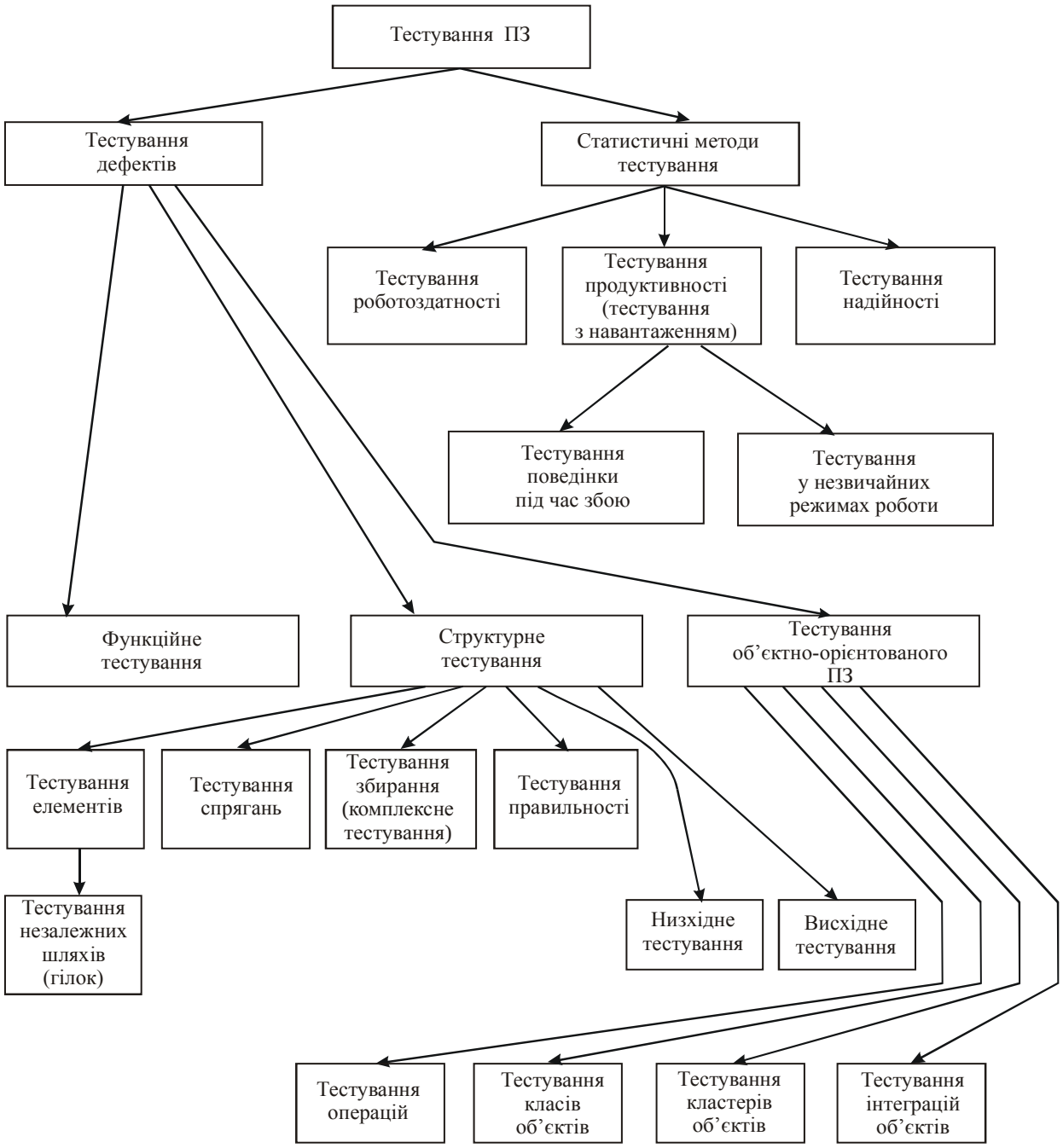

Рис.1.5. Типи і методи тестування

1.3. Особливості і порівняльний аналіз методів тестування ПЗ

Діагностування і, зокрема, тестування ПЗ здійснюють з метою впевнитися, що воно відповідає певним критеріям і вимогам кінцевого користувача, в першу чергу замовника. За допомогою тестування необхідно розпізнати дефекти ПЗ. Основна *мета тестування* полягає у підвищенні ймовірності того, що ПЗ за будь-яких умов буде функціювати належним чином і відповідатиме встановленим вимогам.

Тестування ПЗ повинно забезпечувати виявлення помилок в ньому, демонстрацію відповідності функцій ПЗ його призначенню, реалізації вимог до його характеристик, відображення якості та надійності ПЗ.

Метою *тестування дефектів* є виявлення в ПЗ прихованих дефектів до того, як воно буде передано замовнику. У процесі його розроблення і після цього не завжди перевіряється відповідність ПЗ його специфікації. Тестування дефектів передбачає запуск тесту, який викличе некоректну роботу програми, і на основі цього виявляється дефект. Тому тестування має продемонструвати наявність дефектів у ПЗ, а не їх відсутність.

Функційне тестування призначене для перевірки відповідності ПЗ 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 ПЗ шляхом аналізу вхідних і вихідних даних. В літературі цей метод називають ще *методом тестування "чорної скриньки"* [6, 7, 14, 15, 19, 20, 21, 59-61, 66-86]. Для реалізації цього методу повинні бути відомі функції програм ПЗ. Результати функційного тестування дають відповіді на такі питання:

1) як виконуються функції програм ПЗ?
2) як сприймаються вхідні дані?
3) як виробляються результати на виходах "чорної скриньки"?
4) як зберігається цілісність зовнішньої інформації?

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.

Функційне тестування, тобто тестування ПЗ 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 “скляної (білої, прозорої) скриньки”, який розглядатиметься нижче.

При *структурному тестуванні* перевіряється коректність побудови всіх елементів програм і правильність їх взаємодії між собою. Цей підхід по-іншому називають *методом тестування “прозорої скриньки”*. Інколи використовуються такі назви цього методу, як метод “білої скриньки” або “скляної скриньки” [6, 7, 14, 15, 19, 20, 59-61, 66-89]. 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 “прозорої скриньки” грунтується на керуючій структурі програм. Програми вважаються повністю перевіреними, якщо проведено вичерпне тестування всіх гілок їх графів керування [20, 59]. Для цього формуються такі тестові варіанти, які б гарантували перевірку всіх незалежних маршрутів кожної програми, перевіряли гілки True і False для всіх логічних

рішень, виконували всі цикли та аналізували правильність внутрішніх структур даних.

Структурне тестування, як правило, застосовується до невеликих програмних елементів, зокрема, до підпрограм або до ПЗ, що розроблено на об'єктно-орієнтованій мові програмування. Для розроблення тестів використовуються знання про структуру елементів і компонентів та аналізується програмний код.

Недоліками структурного тестування ("прозорої скриньки") є:

1) досить велика кількість незалежних маршрутів у графі керування [7, 59];

2) відсутність повної гарантії при вичерпному тестуванні маршрутів відповідності програм вихідним вимогам до них;

3) наявність того, що у програмах можуть бути пропущені деякі маршрути (витікає з пункту 1);

4) неможливість виявити помилки, поява яких залежить від даних, що обробляються (помилки, зумовлені виразами типу **if abs(a − b) < a / b,** **if ((a + b + c) / 3 = a,...)**.

Структурне тестування забезпечує виявлення таких видів помилок:

1. *Помилки логічних умов:*

а) помилка булевого оператора (наявність некоректних чи відсутніх, надлишкових булевих операторів);

б) помилка булевої змінної;

в) помилка булевої дужки;

г) помилка оператора відношення;

д) помилка арифметичного виразу.

2. Помилки (незалежних) маршрутів програми.
3. Помилки у гілках True, False для всіх логічних рішень.
4. Помилки в циклах у межах їх границь і діапазонів.
5. Помилки внутрішніх структур даних.

Структурне тестування (тестування модулів) ПЗ включає в себе такі складові (рис.1.5): тестування елементів; тестування спрягань між елементами, тестування збирання (комплексне тестування). Вихідним з перерахованих вище складових структурного тестування є тестування елементів.

При тестуванні модулів тестуванню підлягають:

1) внутрішні структури даних;
2) незалежні шляхи;
3) граничні умови;
4) шляхи обробки помилок;
5) інтерфейс модуля.

Помилкою внутрішньої структури даних є нецілісність збережуваних даних.

Помилками, що виявляються при тестуванні незалежних шляхів, є такі:

1. *Помилки обчислень:*

а) неправильний чи незрозумілий пріоритет арифметичних операцій;

б) змішана форма операцій;

в) некоректна ініціалізація;

г) неузгодженість при представленні точності;

д) некоректне символьне представлення виразів.

2. *Помилки порівняння і неправильних потоків керування:*

а) порівняння різних типів даних;

б) некоректні логічні операції і пріоритетність;

в) очікування еквівалентності в умовах, коли помилки точності роблять еквівалентність неможливою;

г) некоректне порівняння змінних;

д) неправильне прикорочення циклу;

е) відмова на виході при відхиленні ітерації;

є) неправильна зміна змінних циклу.

Помилки на граничних умовах:

1) помилки при опрацюванні **n**-го елемента **n**-елементного масиву;

2) помилки при виконанні **m**-ї ітерації циклу з **m** проходами;

3) помилки при задаванні максимального (мінімального) значення.

Помилками при *тестуванні шляхів оброблення помилок* є:

1) повідомлення про помилку незрозуміле;

2) текст повідомлення про помилку не відповідає виявленій помилці;

3) системні засоби реєстрації помилки прореагували на помилку ще до опрацювання її в модулі;

4) некоректне опрацювання виключної умови;

5) опис помилки не дозволяє визначити її причину.

Після тестування окремих елементів ПЗ виконується *тестування збирання* програмної системи. Метою тестування збирання є виявлення помилок спрягань.

Помилками спрягань модуля є:

1) втрата даних при проходженні через спрягання;

2) відсутність необхідного посилання в модулі;

3) несприятливий вплив одного модуля на інший;

4) відсутність утворення необхідної функції при об'єднанні підфункцій;

5) при інтеграції перехід окремих допустимих неточностей за допустиму межу;

6) проблемні ситуації при роботі з глобальними структурами даних.

Процес *тестування збирання* (інтеграції) модулів як елементів програмної структури може здійснюватись двома шляхами: низхідним тестуванням та висхідним тестуванням.

Після закінчення тестування збирання, яке дає можливість виявити інтерфейсні помилки, починається останній етап тестування програмної системи – *тестування правильності*. Його мета полягає у підтвердженні того, що функції, описані у специфікації вимог до ПЗ, відповідають очікуваним. Підтвердження правильності розробленої програмної системи здійснюється за допомо-

гою демонстрування відпрацювання тестів, розроблених за методом тестування “чорної скриньки”. При цьому важливим елементом підтвердження правильності є перевірка конфігурації системи. Вона гарантує, що всі елементи конфігурації враховані і розроблені правильно та достатньо деталізовані відповідно до вимог замовника ПЗ.

При *низхідному підході* [6, 17, 19, 20, 59] збирання і тестування системи починається з головного керуючого модуля, а підлеглі модулі додаються в структуру у результаті їх розроблення в глибину та ширину. Автономно тестують тільки головний модуль, який відповідає вершині графа. У подальшому до нього під’єднують модулі, безпосередньо зв’язані з ним, і тестують одержану структуру.

Низхідне тестування має певні недоліки. Серед них труднощі у ситуаціях, коли для повного тестування на верхніх рівнях необхідні результати обробки тестової нформації з нижчих рівнів. При низхідному тестуванні нерозроблені модулі, що мають під’єднуватися до розроблених, тимчасово замінюються, так званими, “заглушками”. Тому тестування є інколи поверхневим внаслідок незавершеності частини модулів і наявності значної кількості спрощених “заглушок”. Використовуючи низхідне тестування, часто неможливо тестувати певні лоігчні умови, зокрема, помилкові ситуації і захисні перевірки. Неможлива також перевірка виключних ситуацій у певних модулях, якщо програма працює з ними лише в обмеженому контексті. Причиною цього є недостатньо повний набір вхідних даних. Але основним недоліком низхідного тестування є необхідність “заглушок” та пов’язані з ними труднощі.

При *висхідному підході* до збирання і тестування програмної системи [6, 17, 19, 20, 59] її збирання і тестування починається з ієрархічно найнижчих модулів. Їх тестують автономно. Модулі підключаються рухом знизу-вгору. Для тестування модулів вищого ієрархічного рівня повинні бути протестовані модулі нижчого рівня, викликані ними. На відміну від низхідного тестування, підлеглі модулі завжди доступні і тому відпадає необхідність у “заглушках”.

Порядок операцій процесу висхідного тестування такий. Спочатку модулі нижчого рівня об’єднуються у кластери (групи, блоки), які виконують певну програмну функцію. Потім для координації введення-виведення тестового варіанту розробляється драйвер, що керує надходженням тестової інформації у відповідні кластери. Далі кластери тестуються, після чого драйвери видаляються, а кластери об’єднуються у загальну структуру рухом вгору. Функція драйверів – імітувати надходження інформації з попередніх модулів.

Основним недоліком висхідного підходу є те, що програмна система (ПС) не може бути окремим продуктом до того часу, поки до неї не буде включено останній модуль. Цей та інші недоліки розглянутих підходів намагаються частково усунути застосуванням комбінації раніше розглянутих підходів. Зокрема, для верхніх рівнів ієрархії застосовують низхідну стратегію, а для нижніх – висхідну [17, 20].

Свої особливості має *тестування об’єктно-орієнтованого ПЗ (ООПЗ)* [19, 57, 59, 60, 90-113]. Вони грунтуються на істотних відмінностях функційних і об’єктно-орієнтованих систем, а саме:

1) об’єкти об’єктно-орієнтованих систем, як окремі програмні компоненти, є дещо іншим і більшим, ніж окремі підпрограми чи функції;

2) об’єкти, інтегровані у підсистеми, слабо зв’язані між собою, і тому важко визначити самий високий ієрархічний рівень в системі;

3) при повторному аналізі об’єктів, що використовуються, може трапитись так, що їх вихідний код буде недоступним для тестувальників.

Тому зрозуміло, що підхід до тестування операції, яка є частиною специфікації класу і способом маніпулювання об’єктом, значно відрізняється від тестування програмного коду, який реалізує певну операцію (функцію).

Основними задачами тестування об’єктно-орієнтованого ПЗ є перевірка відповідності поведінки об’єкта, його специфікації і правильності взаємодії цього об’єкта з іншими об’єктами під час виконання програми.

Із зазначених вище відмінностей можна зробити висновок, що для тестування об’єктів варто використовувати тестування методом “прозорої скриньки”, що грунтується на аналізі коду, а для тестування збирання – інший метод, який включає такі складові: тестування операцій, тестування класів об’єктів, тестування кластерів об’єктів, тестування інтеграції об’єктів. Коротко розглянемо суть кожної із зазначених складових методу тестування збирання об’єктно-орієнтованого ПЗ.

Тестування операцій являє собою тестування функцій чи процедур. Тому вони можуть тестуватися методом тестування “чорної скриньки” чи “прозорої скриньки”. Суть цих методів була розглянута раніше.

При *тестуванні класів об’єктів* застосовується тестування методом “чорної скриньки”, але розширюється поняття класу еквівалентності. Під час тестування об’єктів виконується: окреме тестування всіх операцій, що асоційовані з об’єктом; перевірка всіх атрибутів, пов’язаних з об’єктом; перевірка всіх можливих станів об’єкта, для чого необхідно моделювання подій, що призводять до зміни стану об’єкта.

Для *тестування груп зв’язаних об’єктів (кластерів)* об’єктно-орієнтованого ПЗ низхідний та висхідний методи збирання не підходять, оскільки немає строгої ієрархії об’єктів. Тому створення кластерів здійснюється шляхом виділення методів та сервісів, що реалізуються за допомогою цих кластерів.

Тестування інтеграції об’єктів здійснюється за одним з трьох методів, а саме: а) тестування сценаріїв і варіантів використання; б) тестування потоків; в) тестування взаємодій між об’єктами.

Суть методу тестування сценаріїв і варіантів використання базується на описі сценаріїв і кластерів об’єктів, що реалізують даний варіант використання. Варіанти використання чи сценарії описують один певний режим роботи програмної системи.

Тестування потоків грунтується на перевірці системних відгуків на введення даних чи групи вхідних подій. При цьому використовується властивість подійної керованості об’єктно-орієнтованих систем. Для реалізації цього методу необхідно знати, як проходить опрацювання потоків подій у системі.

Проміжний рівень тестування збирання системи – *тестування взаємодій між об’єктами* [91] грунтується на визначенні шляхів “метод-повідомлення”, що відстежують послідовність взаємодій між об’єктами.

Найбільш ефективним з трьох зазначених методів тестування інтеграції об’єктів є тестування сценаріїв і варіантів використання.

Основним недоліком розглянутих методів тестування об’єктно-орієнтованого ПЗ є використання наслідування для класів об’єктів. Якщо клас надає методи, що успадковані від підкласів, то виникає необхідність тестувати всі підкласи з успадкованими ними методами (операціями) з класу.

На відміну від тестування дефектів, яке здійснюється на всіх етапах розроблення ПЗ, *статистичне тестування* проводять після закінчення розроблення програмної системи. Воно дає можливість оцінити роботоздатність, продуктивність та надійність програм, що входять до складу програмної системи, а також роботоздатність системи в цілому в різних режимах експлуатації. Особливість розроблення тестів для статистичного тестування полягає у тому, що вони імітують реальну роботу системи з реальними вхідними даними.

Надійність функціювання ПЗ, як вже згадувалось раніше, оцінюється за кількістю збоїв на протязі певного часу, що відбулися під час роботи системи, а продуктивність – за результатами вимірювання повного часу виконання операцій та часу відгуку системи при обробці тестових даних. Щоб упевнитись, що програмна система може працювати із заданим навантаженням, розробляються тести з постійним збільшенням навантаження. Деякі системи проектуються так, щоб вони працювали при певному навантаженні. Виконання тестів такими системами починається з максимального навантаження, яке вказується

у специфікації (технічному завданні), і продовжується до того часу, поки не виникне збій у роботі системи. Таке тестування має дві мети:

1) протестувати поведінку програмної системи під час збою – чи не призвів він до порушення цілісності даних або до втрати сервісних можливостей;

2) виявити дефекти, які не з'являються у звичайних режимах роботи системи, але можуть призвести до незвичайних комбінацій стандартних умов.

Основний недолік статистичних методів тестування ПЗ полягає у тому, що вони не дають відповідей про наявність як функційних помилок, так і помилок, які виявляють методом структурного тестування. Отже, можна сказати, що статистичне тестування взагалі не призначене для виявлення помилок ПЗ.

1.4. Висновки. Постановка задачі

1. Проведення аналізу програмного забезпечення як об'єкта діагностування дало можливість зробити наступні висновки:

1) розвиток і впровадження нових архітектур та апаратна складність комп'ютерів і систем, побудованих на їх базі, на сьогодні випереджає розроблення методів і засобів діагностування системного та прикладного програмного забезпечення цих систем;

2) існуючі діагностичні програми не завжди повністю враховують зростаючі вимоги до розробки програм, за причини постійного ускладнення ПЗ та намагання користувачів розв'язувати за його допомогою важкоформалізовані та неформалізовані задачі;

3) низька якість окремих діагностичних програм знижує ефективність використання існуючого ПЗ комп'ютерних систем.

2. Однією з основних складових діагностування ПЗ є тестування як процес виявлення дефектів у програмах. Його роль тим більше зростає в зв'язку з тим, що ПЗ сучасних комп'ютерних систем є досить складним і не може бути бездефектним. Причиною невиявлення дефектів у розроблюваному ПЗ швидше за

все слід вважати недосконаліть тестів, а не бездефектність програми, тому кваліфікація розробників тестових програм повинна бути досить високою, а у ряді випадків навіть вищою за кваліфікацію розробників самого ПЗ.

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

Для підвищення достовірності процесу тестування ПЗ необхідно вирішити такі задачі:

1). Провести аналіз методів і підходів до тестування ПЗ та виявити напрямки їх удосконалення, в тому числі за рахунок пошуку і ідентифікації прихованих помилок.

2). Розробити концептуальну модель процесу інтелектуального тестування ПЗ для розв'язку важкоформалізованої задачі ідентифікації прихованих помилок у ньому.

3). Розробити модель ідентифікації прихованих помилок ПЗ на базі штучних нейронних мереж.

4). Розробити метод на основі штучних нейронних мереж та алгоритми ідентифікації прихованих помилок ПЗ.

5). Розробити і впровадити у виробництво автоматизовану систему тестування ПЗ з можливістю ідентифікації прихованих помилок.