МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н. Каразіна

Факультет: **ННІ Каразінський банківський інститут**
Кафедра: **Інформаційних технологій та математичного моделювання**
Спеціальність: **122 Комп'ютерні науки**
Освітня програма: **Комп'ютерні науки та інформаційні технології в бізнесі**

Група: АК–41б **денна форма навчання**

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему:

«РОЗРОБКА ВДОСКОНАЛЕНОГО АЛГОРИТМУ ПІДВИЩЕННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

ЗА НАКАЗОМ № 4601–5/335 ВІД 07 ЛЮТОГО 2025 РОКУ

здобувача вищої освіти **Кулик Діани Сергіївни**

Робота допущена до захисту в ЕК
протокол кафедри ІТММ № 13 від 31.05.2025р.

Завідувач кафедри ІТММ
к.п.н., доцент
_____ Н.І. Стяглик

Науковий керівник
Ph.D з «Комп'ютерних наук»
_____ Д.М. Ковальчук

м. Харків 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Факультет навчально–науковий інститут "Каразінський банківський інститут"
Кафедра інформаційних технологій та математичного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки та інформаційні технології в бізнесі

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ **Н. І. Стяглик**
Підпис ініціали, прізвище

"08" лютого 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ)

Кулик Діани Сергіївни
(прізвище, ім'я, по батькові студента)

1. Тема роботи Розробка вдосконаленого алгоритму підвищення надійності програмного забезпечення

керівник роботи Ph.D з «Комп'ютерних наук» Ковальчук Д. М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "08" лютого 2025 року № 4601–5/335

2. Строк подання студентом роботи 15 травня 2025 року

3. Перелік питань, які потрібно розробити:
У розділі 1: розглянути теоретичні засади забезпечення надійності програмного забезпечення, визначити актуальність проблеми у контексті сучасного розвитку інформаційних технологій. Провести аналіз існуючих підходів і засобів підвищення надійності ПЗ, дослідити чинники, що впливають на надійність упродовж життєвого циклу програмного забезпечення.

У розділі 2: дослідити методи та моделі оцінювання і забезпечення надійності програмного забезпечення. Провести класифікацію типів відмов та помилок у ПЗ, проаналізувати їх вплив на функціонування інформаційних систем. Визначити метрики надійності та засоби контролю якості програмного забезпечення.

У розділі 3: вдосконалений алгоритм підвищення надійності ПЗ. Провести розрахунок надійності ПЗ при використані запропонованого рішення.

4. План роботи

№ з/п	Назви етапів роботи
1	Вибір здобувачем теми кваліфікаційної бакалаврської роботи
2	Затвердження плану і завдання кваліфікаційної бакалаврської роботи
3	Здача кваліфікаційної бакалаврської роботи керівнику
4	Підпис кваліфікаційної бакалаврської роботи керівника
5	Підпис кваліфікаційної бакалаврської роботи у нормоконтролера
6	Допуск завідувачем кафедри до захисту кваліфікаційної бакалаврської роботи
7	Захист кваліфікаційної бакалаврської роботи

5. Дата видачі завдання 08 лютого 2025 року

Студент _____ Д. С. Кулик
підпис ініціали, прізвище

Керівник роботи _____ Д. М. Ковальчук
підпис ініціали, прізвище

РЕФЕРАТ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
«РОЗРОБКА ВДОСКОНАЛЕНОГО АЛГОРИТМУ ПІДВИЩЕННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»
Кулик Діани Сергіївни

Кваліфікаційна бакалаврська робота містить 45 сторінок, 2 таблиці, 6 рисунків, список літератури з 17 найменувань.

Об'єктом дослідження є процес забезпечення надійності програмного забезпечення в інформаційних системах.

Предметом дослідження є методи, моделі та алгоритми підвищення надійності програмного забезпечення на різних етапах його життєвого циклу.

Мета кваліфікаційної бакалаврської роботи полягає у розробці вдосконаленого алгоритму підвищення надійності програмного забезпечення, який забезпечує зменшення кількості помилок та збоїв у функціонуванні системи.

Завданнями кваліфікаційної бакалаврської роботи є:

– провести аналіз сучасного стану проблеми забезпечення надійності ПЗ та обґрунтувати актуальність її дослідження;

– розглянути існуючі методи, моделі та засоби оцінювання надійності ПЗ;

– розробити вдосконалений алгоритм підвищення надійності ПЗ;

– провести розрахунок надійності ПЗ при використані запропонованого рішення та визначити сфери його практичного застосування.

Актуальність дослідження. Надійність програмного забезпечення є критичним чинником у функціонуванні сучасних інформаційних систем, особливо в умовах високої складності та інтенсивного використання ПЗ у безперервних і відповідальних процесах. Розробка ефективних методів забезпечення надійності сприяє підвищенню загальної якості ІТ-продуктів та зменшенню ризиків їх відмов.

За результатами дослідження розроблено модель підвищення надійності програмного забезпечення, яка поєднує в собі переваги багатоверсійного, дуального програмування, адаптивного вибору рішень та автоматизованого моніторингу відмов.

Практична новизна роботи полягає у створенні комплексного алгоритму підвищення надійності, що може бути застосований до широкого спектра ПЗ, зокрема для критично важливих систем.

Одержані результати можуть бути використані у практиці розробки та супроводу ПЗ, зокрема в галузях, де відмовостійкість має вирішальне значення.

КЛЮЧОВІ СЛОВА: НАДІЙНІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, БАГАТОВЕРСІЙНЕ ПРОГРАМУВАННЯ, ДУАЛЬНЕ ПРОГРАМУВАННЯ, АДАПТИВНА МОДЕЛЬ, ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ТЕСТУВАННЯ.

ABSTRACT

AT QUALIFICATION BACHELOR WORK

«DEVELOPMENT OF AN IMPROVED ALGORITHM FOR INCREASING SOFTWARE RELIABILITY»

Diana Kulyk

The bachelor's thesis contains 45 pages, 2 tables, 6 drawings, a list of references of 17 titles.

The object of the research is the process of ensuring software reliability in information systems.

The subject of the research is methods, models, and algorithms for improving software reliability at various stages of its life cycle.

The purpose of a bachelor's thesis is to develop an improved algorithm for increasing software reliability, which ensures a reduction in the number of errors and failures in the functioning of the system.

The tasks of a bachelor's degree are:

– to analyze the current state of the problem of ensuring software reliability and justify the relevance of its study;

– to consider existing methods, models, and tools for evaluating software reliability;

– to develop an improved algorithm for increasing software reliability;

– to calculate software reliability using the proposed solution and determine the areas of its practical application.

The relevance of the study. Software reliability is a critical factor in the functioning of modern information systems, especially under conditions of high complexity and intensive software usage in continuous and mission-critical processes. Developing effective methods for ensuring reliability contributes to improving the overall quality of IT products and reducing the risk of failures.

According to the results of the research: a model for improving software reliability has been developed that combines the advantages of N-version programming, dual programming, adaptive decision-making, and automated failure monitoring.

Main theoretical provisions on the topic of the practical relevance of the study: the practical novelty of the work lies in the creation of a comprehensive algorithm for improving reliability, which can be applied to a wide range of software, particularly for mission-critical systems.

The results obtained can be used in software development and maintenance practices, particularly in industries where fault tolerance is of critical importance.

KEYWORDS: SOFTWARE RELIABILITY, N-VERSION PROGRAMMING, DUAL PROGRAMMING, ADAPTIVE MODEL, SOFTWARE QUALITY, TESTING.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ 7
ВСТУП 8
РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ 10
1.1. Актуальність питання забезпечення надійності ПЗ 10
1.2. Сучасний стан питання надійності ПЗ 12
РОЗДІЛ 2. МОДЕЛІ ТА ПОКАЗНИКИ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 16
2.1. Дослідження надійності ПЗ 16
2.2. Моделі надійності ПЗ 19
2.3. Аналіз типів помилок у ПЗ 24
РОЗДІЛ 3. РОЗРОБКА ВДОСКОНАЛЕНОГО АЛГОРИТМУ ПІДВИЩЕННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 29
3.1. Підвищення надійності інформаційних систем 29
3.2. Метод багатоверсійного програмування 32
3.3. Метод дуального та комбінованого програмування 35
3.4. Модель підвищення надійності програмного забезпечення 37
ВИСНОВКИ 42
ПЕРЕЛІК ПОСИЛАНЬ 44

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ

ІС – інформаційна система

ПЗ – програмне забезпечення

MTBF – Mean Time Between Failures (середній час між відмовами)

MTTR – Mean Time To Repair (середній час на відновлення)

CI/CD – Continuous Integration / Continuous Deployment (безперервна інтеграція / безперервне розгортання)

APM – Application Performance Monitoring (моніторинг продуктивності додатків)

QA – Quality Assurance (Забезпечення якості)

ISO – International Organization for Standardization (Міжнародна організація зі стандартизації)

SQA – Software Quality Assurance (Забезпечення якості програмного забезпечення)

DevOps – Development and Operations (інтеграція розробки та експлуатації ПЗ)

ВСТУП

У сучасному інформаційному суспільстві програмне забезпечення (ПЗ) стало основою функціонування практично всіх галузей – від медицини та транспорту до оборони й фінансових систем. В умовах постійного зростання складності ПЗ та жорстких вимог до його якості ключовим чинником стає надійність. Відмова навіть незначного модуля програмної системи може призвести до суттєвих економічних, технічних або соціальних наслідків.

Актуальність дослідження. Проблема підвищення надійності ПЗ набуває особливої актуальності в умовах стрімкого розвитку складних і розподілених ІТ-систем, де збої та помилки призводять до зниження якості послуг, втрати даних або порушення бізнес-процесів. Традиційні методи забезпечення надійності не завжди відповідають сучасним вимогам гнучкості, масштабованості й автоматизації, що зумовлює потребу в розробці нових, більш ефективних підходів.

Об’єктом дослідження є процеси забезпечення надійності програмного забезпечення на всіх етапах його життєвого циклу.

Предметом дослідження є методи, моделі та алгоритми підвищення надійності ПЗ, зокрема з використанням багатоверсійного, дуального та комбінованого програмування.

Мета кваліфікаційної бакалаврської роботи полягає у розробці вдосконаленого алгоритму підвищення надійності програмного забезпечення, який забезпечує зменшення кількості помилок та збоїв у функціонуванні системи.

Завданнями кваліфікаційної бакалаврської роботи є:

– провести аналіз сучасного стану проблеми забезпечення надійності ПЗ та обґрунтувати актуальність її дослідження;

– розглянути існуючі методи, моделі та засоби оцінювання надійності ПЗ;

– класифікувати типи помилок та відмов, що впливають на надійність

систем;

– розробити вдосконалений алгоритм підвищення надійності ПЗ;

– провести розрахунок надійності ПЗ при використані запропонованого рішення та визначити сфери його практичного застосування.

За результатами дослідження розроблено модель підвищення надійності програмного забезпечення, яка поєднує в собі переваги багатоверсійного, дуального програмування, адаптивного вибору рішень та автоматизованого моніторингу відмов.

Практична новизна роботи полягає у створенні комплексного алгоритму підвищення надійності, що може бути застосований до широкого спектра ПЗ, зокрема для критично важливих систем. Алгоритм має здатність адаптуватися до умов експлуатації та підтримує механізми автоматичного виявлення, аналізу й усунення збоїв.

Одержані результати можуть бути використані у практиці розробки та супроводу ПЗ, зокрема в галузях, де відмовостійкість має вирішальне значення (фінансові системи, охорона здоров’я, транспорт, енергетика), а також при розробці систем управління якістю програмного забезпечення.

У 1 розділі розглянуто актуальність надійності як властивості ПЗ, проаналізовано наукові підходи та виклики, з якими стикаються розробники під час забезпечення надійної роботи програмних систем.

У 2 розділі проаналізовано математичні моделі надійності, класифіковано типи помилок, наведено методи кількісної оцінки й управління надійністю.

У 3 розділі запропоновано алгоритм, що інтегрує багатоверсійне, дуальне та комбіноване програмування, автоматизоване тестування, аналіз збоїв і механізми адаптивного вирішального алгоритму. Представлено модель реалізації та можливі напрями практичного використання.

У висновку підсумовуються результати дослідження, розкривається його практичне значення, а також окреслюються перспективи подальшого розвитку теми.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ

1.1. Актуальність питання забезпечення надійності ПЗ

На сучасному етапі розвитку суспільства спостерігається стрімке зростання науково-технічного прогресу та інтенсивне впровадження інноваційних цифрових технологій у різні сфери діяльності. Це вимагає від інформаційних систем, які є основою для автоматизації процесів управління, аналізу даних, моніторингу та прийняття рішень, значного підвищення ефективності та надійності. Програмне забезпечення, як базовий компонент будь-якої інформаційної системи, відіграє ключову роль у забезпеченні її функціональної спроможності та стійкості до збоїв [1-3].

Сучасне програмне забезпечення охоплює широкий спектр застосувань: від автоматизованих систем управління на підприємствах до розподілених вебсервісів, інтелектуальних систем підтримки прийняття рішень, медичних та військових систем, критичних інфраструктур. Відповідно до цього, вимоги до надійності ПЗ стрімко зростають, адже збій навіть незначного компонента в складній інформаційній системі може спричинити серйозні наслідки: втрату даних, порушення безпеки, матеріальні збитки або зупинку функціонування цілих підрозділів [1-5].

З розвитком компонентного, сервісно-орієнтованого та мікросервісного підходів до побудови ПЗ, його архітектура ускладняється, зростає кількість взаємозв’язків між елементами, збільшується ризик виникнення помилок на стиках між компонентами. У той час як апаратні ресурси постійно вдосконалюються, програмне забезпечення часто не встигає адаптуватися до нових викликів, зберігаючи слабкі місця, що знижують загальну надійність системи. Програмне забезпечення, яке ще донедавна розглядалося як допоміжний інструмент, сьогодні стало повноцінною критичною

інфраструктурою, від якої залежить функціонування транспортних, енергетичних, фінансових та інших життєво важливих систем.

За наявними статистичними даними, навіть висококваліфіковані розробники не здатні повністю уникнути помилок. Так, звичайне комерційне ПЗ, що складається приблизно з 350 тисяч рядків коду, може містити до 2000 помилок, серед яких – витоки пам’яті, некоректне використання бібліотек, логічні та синтаксичні помилки, що виникають як на етапі розробки, так і під час підтримки. Вартість їх виявлення та виправлення значно зростає на пізніх етапах життєвого циклу, особливо під час експлуатації, коли ймовірність негативного впливу на систему максимальна. У масштабних інформаційних системах навіть одинична помилка може призвести до каскадного ефекту – відмови інших компонентів, блокування доступу, порушення алгоритмів взаємодії [4].

Оцінювання та забезпечення надійності програмного забезпечення є важливою науково-технічною проблемою, вирішення якої потребує системного підходу [1-3, 5]. Сучасні методи включають:

– архітектурне моделювання надійності, що враховує структуру програмної системи, взаємодію її компонентів та інтерфейсів;

– статистичні підходи, які базуються на аналізі історичних даних про відмови, помилки та поведінку ПЗ в експлуатаційних умовах;

– динамічне тестування, що дозволяє оцінити поведінку системи в реальному середовищі або за допомогою моделювання навантаження;

– верифікацію та валідацію, що забезпечують відповідність реалізації вимогам і специфікаціям.

Попри наявність низки методів, проблема залишається недостатньо дослідженою в контексті усього життєвого циклу ПЗ. На кожному його етапі – від формулювання вимог до супроводу – виникають специфічні загрози для надійності. Наприклад, на етапі розробки часто закладаються помилки у логіці або архітектурі; під час тестування не завжди вдається виявити всі дефекти; на етапі експлуатації програма може працювати в

непередбачених умовах; під час супроводу виникає ризик внесення нових помилок при оновленнях. Більше того, межі адекватності моделей надійності часто не враховують зовнішні чинники, такі як зміни технологічного середовища, варіативність вимог користувачів, обмеження ресурсів та організаційні чинники.

Таким чином, забезпечення надійності програмного забезпечення інформаційних систем є надзвичайно актуальним завданням, вирішення якого має опиратися на глибоке розуміння внутрішньої архітектури ПЗ, принципів моделювання, механізмів тестування, типології помилок, а також на аналіз зовнішніх впливів. Створення вдосконалених алгоритмів оцінювання та підвищення надійності дозволить розробляти ефективніші, стійкіші до збоїв програмні рішення, що відповідатимуть сучасним вимогам до інформаційної безпеки, продуктивності й довготривалої стабільності функціонування складних інформаційних систем.

1.2. Сучасний стан питання надійності ПЗ

Однією з ключових проблем сучасності залишається забезпечення високої надійності технологій, зокрема – програмного забезпечення, що є центральною складовою інформаційних систем. Усі сфери діяльності – економіка, охорона здоров’я, транспорт, енергетика, оборона, освіта – дедалі більше залежать від функціонування складних програмно-апаратних комплексів, у яких програмне забезпечення виконує роль «інтелектуального ядра», відповідального за прийняття рішень, обробку даних, комунікацію, безпеку та автоматизацію процесів. З огляду на таку важливість ПЗ, питання його надійності набуває особливої ваги, адже навіть незначний збій або помилка можуть спричинити значні втрати [1-5].

Складність сучасного програмного забезпечення зростає пропорційно до ускладнення функціональних вимог і архітектурних рішень в інформаційних системах. Зі збільшенням кількості інтегрованих модулів,

зовнішніх API, взаємодії з базами даних, розподілених сервісів та кіберфізичних компонентів, розробникам усе важче забезпечити передбачувану поведінку програмної системи в умовах реальної експлуатації. Це, своєю чергою, ускладнює процес точного й правдивого оцінювання показників надійності, вимагаючи застосування відповідних формальних і емпіричних моделей.

Для кількісного оцінювання та прогнозування надійності програмного забезпечення активно використовуються моделі надійності. Їхнє завдання – передбачити кількість залишкових дефектів у програмному продукті, динаміку їх виявлення під час експлуатації або тестування, і, відповідно, визначити ймовірність безвідмовної роботи протягом заданого часу. Залежно від підходу до моделювання, всі моделі можна умовно поділити на статичні (які оцінюють характеристики ПЗ на основі структури, коду, документації) та динамічні (які ґрунтуються на аналізі поведінки програмного продукту під час тестування або експлуатації) [2, 4, 5].

У практиці розробки інформаційних систем частіше використовують динамічні моделі, які припускають, що кількість помилок у ПЗ є дискретною випадковою величиною, а їх виявлення підпорядковується певному закону розподілу. Такі моделі дозволяють оцінити:

– початкову кількість дефектів у системі,

– інтенсивність їх виявлення на етапі тестування або експлуатації,

– прогнозовану кількість помилок після завершення певного етапу.

Один з основних показників, що використовується в цих моделях, – інтенсивність відмов, яка може бути сталою, змінюватись залежно від часу або описуватись функцією відлагодження. Деякі моделі передбачають ідеальне відлагодження, тобто усунення помилок без створення нових. У таких випадках кількість залишкових дефектів є незростаючою функцією часу. Водночас більш реалістичні моделі враховують можливість неідеального відлагодження, коли під час виправлення однієї помилки можуть виникати нові або не всі дефекти усуваються повністю. Це

ускладнює процес аналізу, але краще відображає реальні умови розроблення ПЗ для складних інформаційних систем.

Варто зазначити, що на сьогодні не існує універсальної моделі, яка б була однаково придатною для всіх типів програмного забезпечення та точно описувала процеси, що відбуваються в усьому життєвому циклі інформаційної системи – від початкової розробки до супроводу та оновлень. Різні типи ПЗ (вбудоване, мережеве, хмарне, веб-орієнтоване тощо) мають свої особливості функціонування, що вимагає адаптації методів моделювання [1-5].

Крім того, надзвичайно актуальним залишається економічний аспект забезпечення надійності [1-3, 5, 7]. Вартість виправлення помилок суттєво залежить від етапу, на якому вони виявлені:

– на стадії проектування усунення дефекту може бути відносно дешевим;

– на етапі кодування – дорожчим;

– але на етапі впровадження або під час реальної експлуатації – найбільш витратним.

Це зумовлює потребу у правильному балансуванні витрат на забезпечення якості – між детальним проєктуванням, тестуванням, верифікацією та термінами розроблення. Одним з ключових чинників прийняття рішення про випуск ПЗ в експлуатацію є значення кількісних показників надійності, розрахованих на підставі відповідних моделей.

Загалом моделі надійності ПЗ можна поділити на ймовірнісні (включають стохастичні підходи до моделювання поведінки системи) та детерміністичні (ґрунтуються на фіксованих правилах і структурі системи) [6, 7]. Залежно від конкретної мети аналізу використовуються моделі:

– моделі висівання помилок;

– моделі апроксимації залежностей між виявленими та залишковими помилками;

– моделі інтенсивності відмов;

– моделі на основі неоднорідного пуассонового процесу (NHPP);

– моделі з використанням марковських ланцюгів;

– моделі зростання надійності.

Окремо варто виділити архітектурні моделі надійності, які враховують структурну організацію інформаційної системи, взаємозв’язки між її модулями та компоненти з різним рівнем критичності. Такі моделі особливо актуальні для розподілених інформаційних систем, де загальна надійність залежить не лише від кількості помилок, але й від їх розташування в архітектурі. В деяких сучасних підходах ці моделі поєднуються з уніфікованими метриками, які враховують тип джерел вхідних даних, контекст виконання, ризик виникнення нових помилок та інші фактори [6, 7].

Моделі на основі неоднорідного пуассонового процесу (NHPP) особливо поширені в практиці. Вони розглядають процес тестування або відлагодження як випадковий процес із змінною інтенсивністю, де кількість знайдених помилок залежить від функції математичного сподівання. Такі моделі дозволяють прогнозувати залишкову кількість дефектів після певного періоду, враховуючи специфіку тестування. Обчислення параметрів таких моделей зазвичай здійснюється з використанням методу Ньютона, методу максимальної правдоподібності або регресійного аналізу [1, 3].

Таким чином, сучасний стан питання надійності програмного забезпечення інформаційних систем характеризується високою науково-практичною значущістю, широким спектром моделей, але водночас – низкою нерозв’язаних проблем. Актуальними залишаються завдання адаптації існуючих моделей до особливостей складних, розподілених, динамічних інформаційних систем, розроблення комплексних інструментів, що поєднують методи формального аналізу з машинним навчанням та емпіричними підходами, а також підвищення точності оцінок надійності з урахуванням реальних умов експлуатації.

РОЗДІЛ 2
МОДЕЛІ ТА ПОКАЗНИКИ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Дослідження надійності ПЗ

Надійність програмного забезпечення визначається як здатність програмної системи безвідмовно функціонувати у встановлених режимах і умовах експлуатації протягом визначеного інтервалу часу. Відповідно до ДСТУ ISO 9000:2007, надійність (dependability) є збірним поняттям, яке охоплює готовність об'єкта до виконання функцій, а також включає характеристики безвідмовності, ремонтопридатності, забезпеченості технічного обслуговування. Таким чином, надійність програмного забезпечення є не лише технічною, а й організаційною категорією, що охоплює весь життєвий цикл – від розробки й тестування до супроводу, оновлення і виведення з експлуатації [1-5, 7, 8].

Згідно зі стандартами ДСТУ 2860-94 та ДСТУ 2861-94, які регламентують терміни і підходи до аналізу надійності технічних систем, надійність програмного забезпечення можна описати через чотири ключові властивості. Перша – безвідмовність, яка визначає здатність ПЗ виконувати задані функції без збоїв протягом певного періоду часу. Це особливо важливо для критичних систем, таких як фінансові сервіси, медичні прилади або транспортні керуючі комплекси, де навіть короткочасна відмова може призвести до суттєвих наслідків. Друга властивість – довговічність, що характеризує здатність програмного забезпечення залишатися функціональним протягом тривалого часу до настання граничного стану, наприклад, застарівання архітектури або втрати сумісності з новими технологіями. Третьою складовою є ремонтопридатність, яка відображає здатність ПЗ до ефективного оновлення, виправлення помилок та адаптації до змін середовища. Нарешті, збережуваність забезпечує здатність

програмного продукту підтримувати працездатність після періоду зберігання, перенесення або інсталяції в новому середовищі [1-5, 7, 8, 9].

Визначення і вимірювання надійності програмного забезпечення здійснюється за допомогою методів теорії ймовірностей і математичної статистики. Такий підхід дозволяє оцінити середній час безвідмовної роботи, частоту збоїв, імовірність відмови в заданий момент часу, а також середній час відновлення. Статистичні моделі будуються на основі даних про реальні помилки, отриманих у процесі експлуатації, тестування або симуляцій. У практиці використовуються також спеціальні метрики, такі як кількість збоїв на 1000 годин роботи, щільність помилок у коді, покриття тестами та інші показники, які дають змогу кількісно охарактеризувати якість ПЗ [5-9].

Імовірність безвідмовної роботи *P(t)*, або функція надійності, характеризує ймовірність того, що програмне забезпечення залишатиметься працездатним упродовж заданого інтервалу часу *t* без виникнення відмов. Це фундаментальний показник, який визначає очікувану стабільність функціонування системи у процесі експлуатації. Умовно вважається, що на початку аналізованого періоду програмний продукт є повністю працездатним. Формально цей показник обчислюється як відношення кількості працездатних екземплярів N_P до загальної кількості N, тобто [10]:

$$P(t) = \frac{N_P}{N} = \frac{N - n}{N},$$

де n - кількість відмов за час t.

Іншим ключовим показником є середній наробіток до відмови $t_{сер}$ (Mean Time To Failure, MTTF). Він визначає середню тривалість безперервного функціонування програмного продукту до моменту першої відмови. Цей показник особливо важливий для оцінки надійності неремонтопридатних програм або версій, які не підлягають оновленню чи виправленню в процесі експлуатації. Обчислення цього показника базується на часових даних відмов для кожного з примірників [10]:

$$t_{сер} = \frac{1}{N}\sum_{i=1}^{N} t_i,$$

де t_i - час до відмови для i-го примірника, а N - загальна кількість досліджуваних примірників програмного забезпечення.

Ще одним важливим параметром є інтенсивність відмов $\lambda(t)$, що відображає швидкість, з якою можуть виникати збої в системі за умови, що до цього моменту жодної відмови не сталося. Іншими словами, це умовна густина ймовірності відмов у момент часу t, яка дозволяє прогнозувати надійність програмного забезпечення у майбутньому [10]:

$$\lambda(t) = \frac{n(t)}{N_P \Delta t},$$

де $n(t)$ - кількість відмов, зафіксованих у часовому інтервалі Δt, а N_P - кількість працюючих екземплярів на початку цього інтервалу. Інтенсивність відмов є особливо інформативною для об’єктів, що не підлягають ремонту або не оновлюються в процесі роботи.

Сукупність наведених показників дозволяє здійснювати кількісну оцінку безвідмовності програмного забезпечення інформаційних систем, забезпечує обґрунтоване прийняття рішень під час розробки, тестування, впровадження й експлуатації ПЗ. У сучасних підходах до розробки велике значення надається саме статистичному моніторингу цих показників на всіх етапах життєвого циклу, що сприяє підвищенню загального рівня надійності інформаційних систем.

Надійність програмного забезпечення тісно пов’язана з іншими характеристиками якості, зокрема з безпекою, цілісністю, адаптивністю та зручністю супроводу. У складних інформаційних системах програмне забезпечення виступає не тільки засобом автоматизації процесів, а й активним учасником прийняття рішень, забезпечення відповідності нормативним вимогам і збереження критичної інформації. Тому надійність ПЗ розглядається як основа функціональної, експлуатаційної та інформаційної надійності всієї системи [10-12].

2.2. Моделі надійності ПЗ

Інформаційні системи є складними структурами, що поєднують програмне забезпечення та апаратне забезпечення для обробки даних і представлення їх у формі, зручній для користувача з метою прийняття рішень. Надійність апаратної частини таких систем можна проаналізувати за допомогою моделей, що застосовуються в теорії надійності технічних пристроїв. Проте ці моделі не повністю відображають особливості програмного забезпечення, оскільки відмови в ПЗ мають інший характер і природу виникнення.

На відміну від апаратури, яка з часом зношується та втрачає працездатність унаслідок фізичного старіння, програмні продукти не мають цієї властивості. Збої в програмному забезпеченні зазвичай пов’язані з помилками, закладеними ще на етапі розробки, а також із впливом зовнішнього середовища, у якому це ПЗ функціонує. Таким чином, надійність програмних компонентів залежить насамперед від якості програмного коду, рівня тестування та характеру експлуатаційних умов.

Практичні спостереження свідчать, що з часом інтенсивність відмов у ПЗ знижується завдяки поступовому виявленню і усуненню помилок. Як правило, лише через кілька років безперервної експлуатації – приблизно через чотири – програмне забезпечення досягає стабільного рівня функціонування. Це обумовлено поступовою адаптацією ПЗ до робочого середовища та усуненням критичних дефектів. На типовому графіку інтенсивності відмов (рис. 2.1) можна спостерігати характерні зміни рівня відмов як у програмних, так і в апаратних елементах інформаційної системи впродовж її життєвого циклу [11-13].

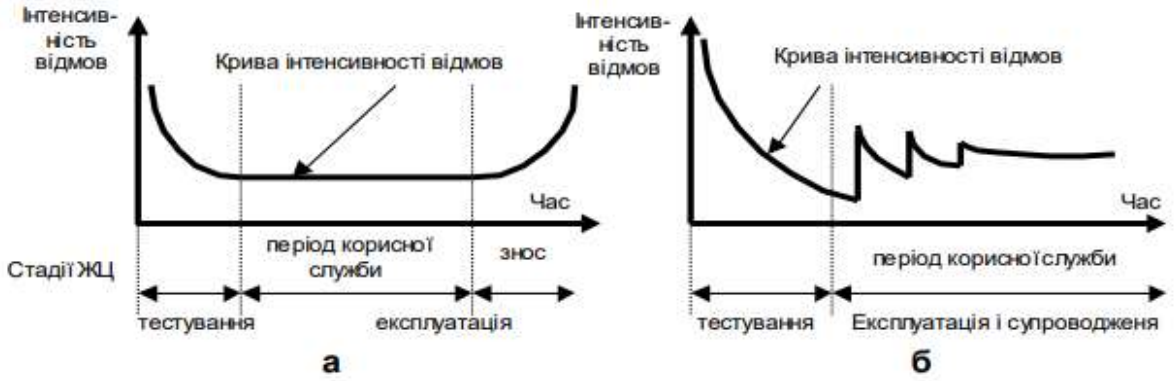

Рис. 2.1. Інтенсивність відмов програмно-апаратних модулів інформаційних систем

Недоліки, що призводять до порушення стабільної роботи програмної частини інформаційної системи, зазвичай є наслідком помилок, допущених розробниками під час реалізації проєкту на різних стадіях життєвого циклу. Джерелом таких дефектів можуть бути як помилки в технічному завданні, так і в процесі кодування, тестування чи інтеграції. На рис. 2.2 представлено графічне відображення кількості помилок, які виникають на різних етапах розробки програмного забезпечення згідно з водоспадною моделлю [11-13].

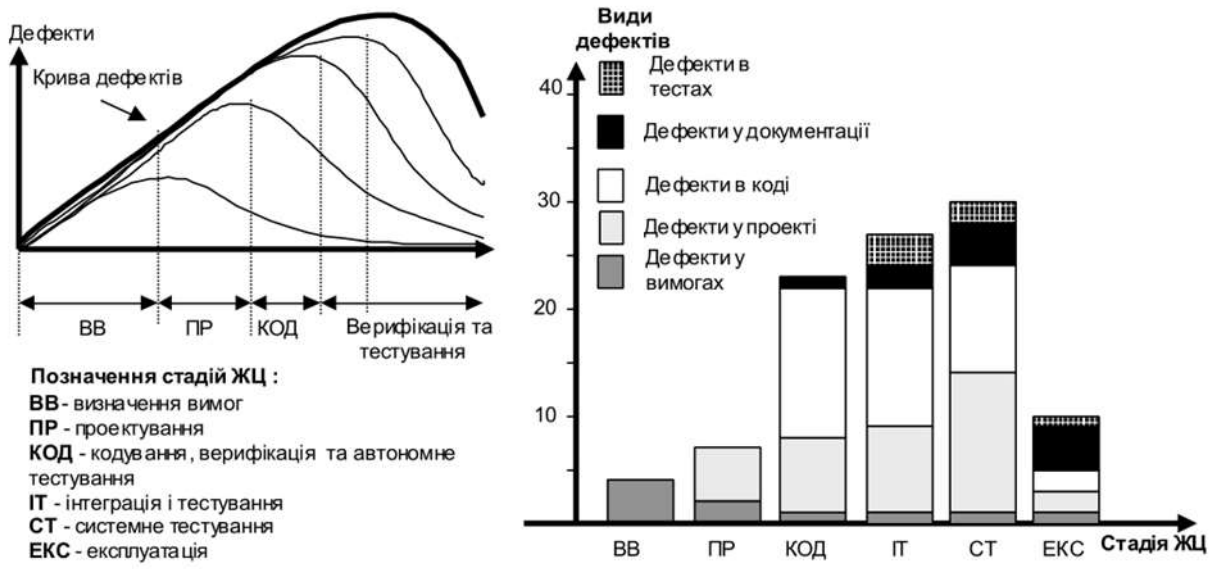

Рис. 2.2. Інтенсивність відмов програмних і апаратних складових комп’ютерної системи

Відмови, що спричиняють порушення роботи програмного забезпечення інформаційної системи, зазвичай мають різну природу та зумовлені різноманітними факторами, серед яких основними є дефекти та логічні помилки у програмних модулях. Ці фактори можуть мати як технічне, так і організаційне походження, впливаючи на надійність системи в цілому. Схематичне зображення взаємозв'язку між дефектами у програмному забезпеченні та ймовірністю виникнення збоїв наведено на рис. 2.3 [11-13].

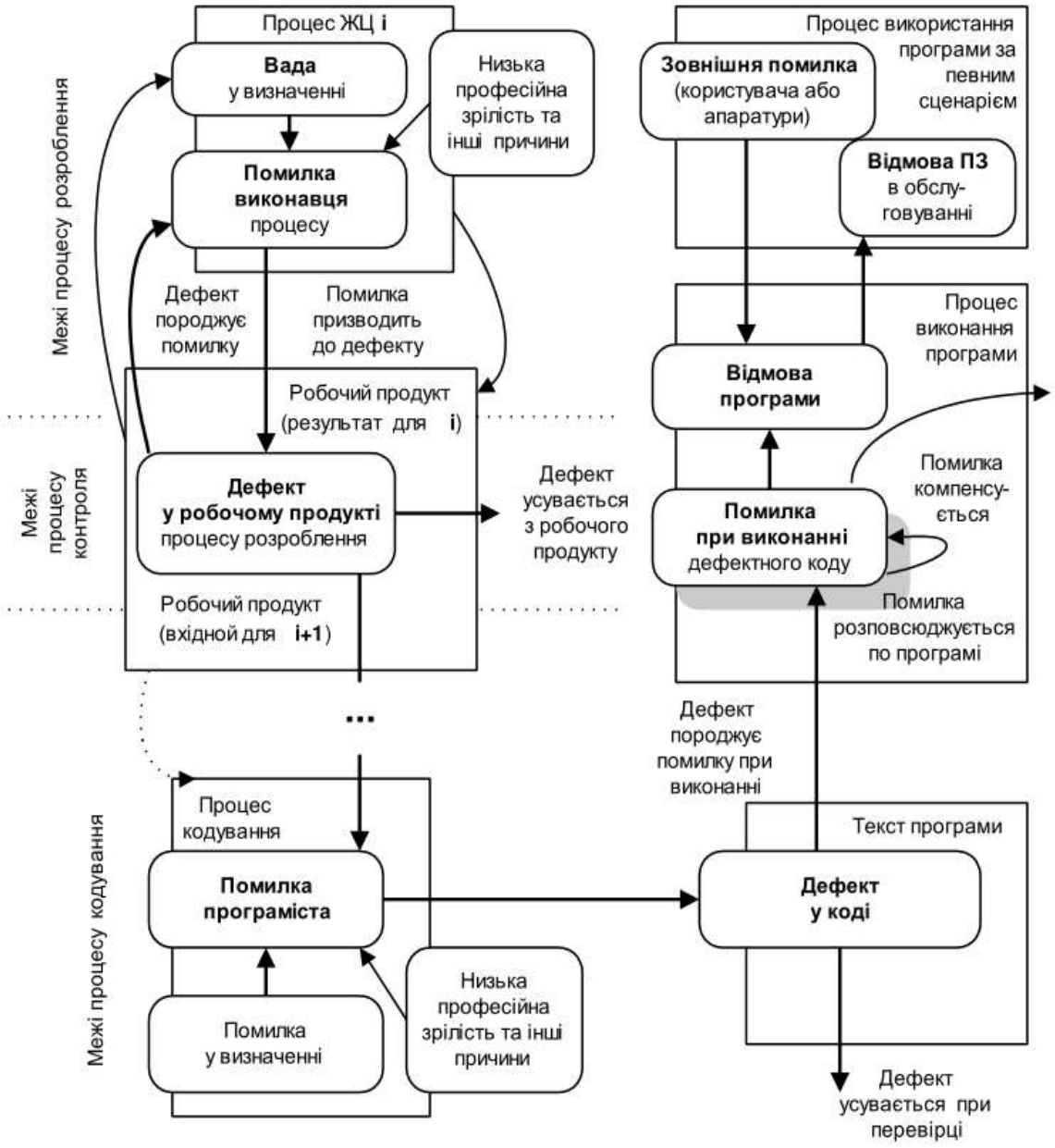

Рис. 2.3. Взаємозв'язок між виникненням відмов та наявністю помилок і дефектів у програмному забезпеченні інформаційної системи

У разі, якщо після виконання процедур перегляду коду, модульного та інтеграційного тестування в програмних компонентах залишаються невиявлені дефекти чи помилки, вони можуть проявитися вже під час реального використання інформаційної системи. Якщо такі помилки виявляються після впровадження системи в середовище кінцевого користувача, це може спричинити порушення її коректного функціонування, що, у свою чергу, негативно впливає на загальну надійність інформаційної системи та знижує її здатність протистояти відмовам.

Ураховуючи ці обставини та базуючись на аналізі даних про надійність інформаційних систем, надзвичайно важливим є завдання забезпечення заданого рівня надійності програмно-апаратного комплексу. Для цього необхідно мінімізувати кількість помилок на етапах проєктування, розробки та впровадження системи.

Однією з основних класифікацій моделей надійності є поділ їх за типом процесу, який вони підтримують: пророчі, прогнозні та вимірювальні. Пророчі моделі здатні передбачати поведінку ПЗ на основі вже існуючих даних, прогнозні – оцінюють майбутню поведінку програмного забезпечення в умовах реальної експлуатації, а вимірювальні моделі націлені на збори даних про поточний стан ПЗ, здебільшого під час тестування. Варто зазначити, що деякі моделі, зокрема ті, які базуються на даних про інтервали між відмовами, можуть одночасно відноситися як до вимірювальних, так і до оцінювальних [1-5, 7, 9-12].

Моделі надійності ПЗ поділяються також на аналітичні та емпіричні. Аналітичні моделі дозволяють здійснювати кількісні розрахунки показників надійності, використовуючи дані про поведінку системи в процесі тестування. Вони дозволяють не тільки вимірювати надійність, а й оцінювати її через розрахунок ймовірностей відмов у часі. У свою чергу, емпіричні моделі базуються на аналізі структурних характеристик програм, таких як кількість міжмодульних зв'язків або цикли в коді. Вони допомагають зрозуміти залежність показників надійності від складності структури

програми, хоча часто не дають точних числових значень, а лише вказують на потенційні проблемні зони. Емпіричні моделі використовуються здебільшого на етапі проектування ПЗ, коли відома структура системи, але вона ще не реалізована [1-5, 7, 9-12].

Аналітичні моделі, в свою чергу, поділяються на динамічні та статичні. Динамічні моделі розглядають поведінку ПЗ у часі, відслідковуючи появу відмов і зміну їх інтенсивності протягом тестування або експлуатації. Ці моделі є найбільш корисними, коли є можливість фіксувати інтервали між відмовами. Статичні моделі, на відміну від динамічних, не пов'язують відмови з часом, а аналізують залежність кількості помилок від характеристик тестових прогонів або вхідних даних. Вони дозволяють отримати прогнозовані оцінки на основі кількості виявлених дефектів під час тестування без прив'язки до часу їх виявлення [1-3].

Моделі зростання надійності використовуються на етапі тестування програмного забезпечення, де вони описують, як надійність ПЗ змінюється в залежності від кількості тестових прогонів, виправлення помилок та додавання нових компонентів. Після введення ПЗ в експлуатацію зростання надійності продовжується, і на цьому етапі використовуються еволюційні моделі, що враховують зміни в програмі через додавання нових модулів, усунення помилок та адаптацію до нових умов експлуатації.

Іншим важливим аспектом є моделі надійності з використанням функцій зусиль тестування. Такі моделі використовують метрики, що відображають зусилля, витрачені на тестування, включаючи час, витрачений розробниками та тестувальниками, кількість тестових випадків та інші ресурси. Ці функції допомагають корелювати інтенсивність тестування з кількістю виявлених дефектів у програмному забезпеченні. Зазвичай для опису таких зусиль використовуються різні статистичні функції розподілу, як-от постійний, експоненційний, логістичний, функції розподілу Релея та Вейбулла. Використання цих моделей дозволяє більш точно описати інтенсивність відмов сучасних великих і складних систем [1-3].

Більшість моделей надійності ПЗ побудовано на припущенні про ідеальне відлагодження, тобто, що при виявленні помилки під час тестування її усувають, не вводячи нових помилок. Однак на практиці цей процес є далеко не ідеальним. При виправленні однієї помилки можуть виникнути нові, і відмови часто спричинені не однією, а кількома взаємопов'язаними помилками. Це явище називається недосконалим відлагодженням, і воно значно ускладнює точність прогнозів, побудованих на цих моделях. Недосконале відлагодження не завжди враховується в існуючих моделях, що може знижувати їхню точність та реалістичність, особливо для складних сучасних програмних систем [1-5, 7].

Таким чином, моделі надійності програмного забезпечення є важливими інструментами для оцінки та прогнозування його поведінки на всіх етапах життєвого циклу. Однак складність сучасних систем, а також реалії процесу розробки, зокрема явище недосконалого відлагодження, вимагають постійного вдосконалення цих моделей. Без урахування всіх нюансів реального процесу розробки та експлуатації ПЗ отримані результати можуть бути недостатньо точними, що ставить під сумнів ефективність прийнятих на їх основі рішень.

2.3. Аналіз типів помилок у ПЗ

Забезпечення високого рівня надійності ПЗ значною мірою залежить від глибокого розуміння природи та джерел помилок, що виникають у програмних компонентах комп'ютерних систем. У міжнародній практиці застосовуються класифікації, які дозволяють систематизувати типи помилок, що можуть призвести до збоїв у роботі програмних продуктів. Згідно з усталеним підходом, виділяють три основні категорії помилок, кожна з яких по-різному впливає на функціональність програмної системи [11-14].

У загальному розумінні, програмна помилка – це ситуація, що виникає внаслідок некоректної реалізації логіки роботи програми і призводить до

отримання неправильних або недостовірних результатів. Джерелом таких помилок, як правило, є хиби, допущені під час створення алгоритмів, особливо на етапах проектування або програмування, зокрема при неправильному використанні операторів мови програмування [11-14].

Одним із поширених різновидів помилок є дефекти – недоліки, що виникають унаслідок помилок розробників під час написання програмного коду. Вони можуть бути результатом як недостатньо точного розуміння вимог до системи, так і проблем, пов’язаних з проектною або супровідною документацією. Часто дефекти фіксуються під час етапів тестування та технічного супроводу, однак їх виявлення на пізніх стадіях може негативно вплинути на стабільність роботи як окремих модулів, так і системи в цілому.

Особливу небезпеку становлять відмови – критичні помилки або їх комбінації, що призводять до повної або часткової втрати працездатності системи. Вони проявляються у вигляді поведінки програми, яка не відповідає очікуваній, визначеній у технічній документації. Причиною таких збоїв часто стають невиявлені або приховані дефекти, що закладені ще на етапах розробки або обумовлені несприятливими умовами експлуатації [13-15].

Серед основних чинників, які сприяють виникненню відмов, слід відзначити:

- порушення вимог до функціонування, визначених у специфікації;

- конфлікти між специфікацією та реальними умовами функціонування апаратного забезпечення;

- логічні помилки в коді, які можуть загрожувати безпеці або доступу до ресурсів;

- неякісну реалізацію алгоритмів, яка включає в себе неточності, неповноту або спотворення логіки обробки даних.

Таким чином, виникнення відмов не завжди пов’язане з однією конкретною помилкою – часто це наслідок комбінації факторів, у тому числі концептуальних недоліків на етапі проектування або неадекватних рішень під час програмної реалізації.

З метою підвищення ефективності процесу виявлення помилок у програмному забезпеченні, компанія IBM запропонувала ортогональний підхід до класифікації дефектів (див. таблицю 2.1). Цей підхід передбачає систематизацію виявлених помилок за незалежними ознаками, без прив'язки до типу або призначення конкретного програмного компонента. Така класифікація дозволяє формувати універсальні підходи до тестування, що однаково ефективні в різних типах програмних систем, незалежно від способу їх реалізації. Основне завдання тестувальника в цьому контексті полягає у виявленні дефектів та їх віднесенні до відповідних кластерів згідно з наперед визначеними категоріями [14].

Таблиця 2.1.

Класифікація програмних дефектів за ортогональним підходом компанії IBM

Категорія дефектів	Опис
Функціональні помилки	Виникають у разі невідповідності реалізованої функціональності вимогам або специфікації.
Інтерфейсні порушення	Пов'язані з некоректною взаємодією між модулями або компонентами, наприклад, через неправильну передачу даних.
Помилки взаємодії з середовищем	Зумовлені неправильною роботою з апаратними засобами або зовнішніми системами.
Логічні похибки	Виявляються у некоректному алгоритмічному рішенні, що дає неправильні результати навіть при правильному синтаксисі.
Помилки обробки даних	Стосуються некоректного введення, збереження, перетворення або виведення даних.
Технологічні похибки	Виникають через недотримання стандартів програмування, стилістичних або архітектурних принципів.
Документаційні неточності	Пов'язані з розбіжностями або відсутністю важливої інформації в технічній чи супровідній документації.
Помилки продуктивності	Включають проблеми з ефективністю роботи ПЗ, такі як надмірне використання ресурсів або затримки у виконанні.
Безпекові дефекти	Стосуються порушень конфіденційності, цілісності чи доступності інформації.

В ортогональній класифікаційній схемі передбачається, що кожна помилка може бути віднесена лише до одного конкретного класу. Це виключає можливість одночасної належності помилки до кількох категорій, що, своєю чергою, забезпечує чітке і недвозначне віднесення кожної помилки до певного типу. Такий підхід сприяє стандартизації та

однозначності у діях розробників під час виявлення і усунення дефектів у програмному забезпеченні [14, 15].

Як альтернативу класифікації, запропонованій компанією IBM, розглядають підхід Hewlett–Packard, що ґрунтується на класифікації, розробленій Г. Бучем. Ця модель дозволяє зіставляти різні типи помилок з відповідними етапами реалізації комп'ютерних систем. На рис. 2.4 наведено співвідношення виявлених помилок залежно від стадії життєвого циклу ПЗ.

Рис. 2.4. Співвідношення типів помилок у програмному забезпеченні

Узагальнена структура розподілу типів помилок, що зображена на рис. 2.4, характерна для більшості великих підприємств, що займаються розробкою програмного забезпечення. Водночас у сучасних умовах спостерігаються певні відхилення від цієї моделі в окремих компаніях, що спеціалізуються на створенні комп'ютерних систем [1-5, 7, 9-12].

Підхід, який дозволяє аналізувати вплив помилок на виникнення відмов, формує причинно-наслідкову модель зв'язку «помилка – відмова». Це забезпечує можливість обґрунтованого вибору інструментів тестування та засобів моніторингу для ефективного контролю стану програмних компонентів. Таким чином, досягається зниження ризику збоїв, що негативно впливають на загальну надійність комп'ютерної системи.

Відмови, що виникають у програмних компонентах інформаційних

систем, можна поділити на кілька основних категорій [1, 3]:

– апаратні збої, які спричиняють втрату працездатності системного програмного забезпечення;

– інформаційні помилки, пов’язані з некоректним введенням або передачею даних;

– збої програмних модулів, що є наслідком наявності внутрішніх дефектів;

– відмови, обумовлені людським фактором, зокрема, неправильним використанням інтерфейсу або апаратних засобів.

Аналіз факторів, які викликають помилки на різних етапах життєвого циклу програмного продукту, свідчить про відсутність єдиного уніфікованого підходу до забезпечення й моніторингу показників надійності комп’ютерних систем. У зв’язку з цим зберігається потреба у проведенні додаткових досліджень, спрямованих підвищення надійності ПЗ інформаційних систем.

РОЗДІЛ 3
РОЗРОБКА ВДОСКОНАЛЕНОГО АЛГОРИТМУ ПІДВИЩЕННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Підвищення надійності інформаційних систем

Надійність інформаційної системи визначається її здатністю зберігати працездатність протягом заданого інтервалу часу в умовах штатної чи аварійної експлуатації, та є однією з основних характеристик її якості [1-5, 7, 9-12].

Інформаційні системи піддаються впливу різних факторів ризику: технічних збоїв, людських помилок, відмов обладнання, збоїв програмного забезпечення, кібератак, впливу зовнішнього середовища тощо. У зв’язку з цим, підвищення надійності має здійснюватися на основі комплексного підходу, що включає як профілактичні заходи, так і застосування спеціалізованих технічних та програмних рішень. Одним із найефективніших підходів до підвищення надійності технічних і програмних компонентів є впровадження методів резервування.

Резервування полягає у впровадженні додаткових елементів (резервів), які можуть частково або повністю замінити основні функціональні компоненти системи у випадку їх виходу з ладу. Такий підхід дозволяє забезпечити безперервність роботи, зменшити ймовірність виникнення збоїв, а також значно підвищити середній час безвідмовної роботи системи.

У технічних системах застосовуються різні стратегії резервування, які класифікуються залежно від способу активації резерву, ступеня дублювання елементів, а також рівня реалізації. Залежно від принципу дії, ступеня участі резервного елементу у роботі системи та рівня його активації, резервування класифікується на кілька основних типів: активне, пасивне, гаряче, структурне, функціональне, а також за рівнем реалізації – апаратне, програмне й інформаційне [12-15].

Активне резервування (паралельне) характеризується одночасною роботою як основних, так і резервних елементів. При цьому навантаження розподіляється між усіма елементами, що дозволяє не лише забезпечити підвищену надійність, але й зменшити інтенсивність зношування кожного з них. У разі виходу з ладу одного з елементів, інші миттєво продовжують функціонування без переривання технологічного процесу. Цей вид резервування є типовим для обчислювальних кластерів, серверних ферм та розподілених інформаційних систем [12-14, 16].

Пасивне резервування (також зване «холодним») полягає у тому, що резервні елементи знаходяться у стані очікування і не беруть участі в роботі системи до моменту відмови основного елементу. У разі виявлення відмови відбувається перемикання на резерв, зазвичай автоматичне, хоча в деяких випадках можливе ручне втручання. Цей підхід забезпечує зниження витрат енергії та продовжує термін служби резервних компонентів, проте характеризується дещо вищим часом реакції на збої [12-14, 16].

Гаряче резервування поєднує переваги активного та пасивного підходів. Резервні компоненти постійно синхронізовані з основними, але без виконання активної функції. У разі відмови основного елементу перемикання на резерв відбувається практично миттєво, що забезпечує високу безперервність процесу. Такий підхід часто реалізується в інформаційних системах з підвищеними вимогами до доступності, зокрема у банківських структурах, телекомунікаціях та центрах обробки даних [14-16].

Окремо виділяється структурне резервування, що передбачає фізичне дублювання елементів системи. Воно може бути як повним, коли дублюються всі критичні вузли, так і частковим – застосовуваним лише до найбільш вразливих або відповідальних елементів. Такий підхід є технічно простим, але ресурсомістким, тому часто застосовується у поєднанні з іншими методами [14-16].

Особливе місце серед методів підвищення надійності інформаційних систем займає функціональне резервування. Його суть полягає не лише у

створенні фізичних копій елементів, а в організації дублювання функцій, які вони виконують. У такому підході резерв створюється на рівні функціональних блоків або підсистем, що дозволяє забезпечити цілісність процесу в разі відмови окремих елементів без необхідності їх прямого дублювання. Наприклад, у складних автоматизованих системах управління можливе використання кількох незалежних каналів досягнення однієї й тієї ж мети: якщо один підхід виходить з ладу, система переходить на альтернативний шлях виконання задачі [15-17].

Функціональне резервування широко застосовується в авіаційних, енергетичних, медичних та оборонних системах, де критичне значення має не лише безперебійність роботи окремих компонентів, а перш за все – збереження основної функціональності всієї системи. Наприклад, у літаковій авіоніці може існувати кілька незалежних каналів навігації, які використовують різні принципи (інерційну навігацію, GPS, радіолокацію), що дозволяє літаку зберегти здатність орієнтування навіть у разі виходу з ладу одного або кількох каналів. До переваг функціонального резервування належить висока адаптивність системи, можливість гнучкого перемикання між функціональними модулями, а також економія ресурсів у порівнянні з повним структурним дублюванням. Водночас впровадження такого виду резервування потребує ретельного проектування логіки функціонального перерозподілу, застосування систем моніторингу стану компонентів та алгоритмів прийняття рішень в умовах відмов.

Крім того, резервування класифікується за рівнем реалізації. Апаратне резервування полягає у фізичному дублюванні елементів, таких як блоки живлення, мережеві адаптери, контролери, дискові накопичувачі тощо. Програмне резервування реалізується за рахунок віртуалізації, реплікації даних, створення резервних копій та застосування програмних кластерів. Інформаційне резервування базується на використанні спеціальних алгоритмів надмірного кодування, які дозволяють виявити та виправити помилки, що виникають у процесі зберігання або передавання даних. До

таких алгоритмів належать, зокрема, коди Хеммінга, коди Ріда–Соломона, контрольні суми, а також різні рівні RAID-масивів.

Методи резервування знаходять широке застосування у різних галузях техніки, зокрема в автоматизованих системах управління, енергетиці, телекомунікаційних мережах, авіаційній техніці та інформаційних технологіях. Наприклад, у промислових автоматизованих системах резервування реалізується на рівні контролерів, сенсорів, каналів зв’язку та виконавчих пристроїв. В IT-інфраструктурах використовуються кластерні архітектури, багаторівневе зберігання даних із реплікацією, а також серверні комплекси з надлишковістю типу N+1 або 2N.

3.2. Метод багатоверсійного програмування

У контексті підвищення надійності інформаційних систем важливим підходом є метод багатоверсійного програмування, який передбачає створення кількох незалежних реалізацій одного й того самого програмного забезпечення. Такий підхід дозволяє значно підвищити стійкість функціонування ІС як у процесі розробки, так і під час реальної експлуатації. Наявність альтернативних версій забезпечує адаптивність до змін зовнішнього середовища та помилок, що можуть проявитися в одній з реалізацій, але не повторитися в інших [16-17].

Надмірність програмних та апаратних компонентів – один із ключових принципів побудови надійних інформаційних систем. У найбільш поширеному варіанті вона реалізується у вигляді дублювання або мультиплікації однакових модулів, що виконують однакові функції. Наприклад, багатомашинні або багатопроцесорні конфігурації із резервними копіями ПЗ забезпечують високу стійкість до апаратних відмов. Проте ця структурна надмірність не вирішує проблему логічних чи алгоритмічних помилок у коді, оскільки всі копії відтворюють одні й ті самі помилки.

Відтак, багатоверсійне програмування (multi-version programming, MVP) пропонує концептуально інший шлях – створення кількох незалежних версій ПЗ, кожна з яких реалізує ту саму функцію, але розробляється різними командами або окремими програмістами, які не мають між собою комунікації під час створення коду. Ключовим у цьому підході є незалежність реалізацій, що мінімізує ймовірність синхронної появи однакових помилок у всіх версіях.

Інформаційні системи, що працюють у режимах підвищеної відповідальності (банківські, авіаційні, енергетичні, оборонні тощо), можуть виграти від впровадження MVP завдяки підвищенню як надійності, так і безпеки. Зазвичай використовується дві або три версії програмного модуля, які запускаються паралельно на різних логічних або фізичних каналах. Отримані результати порівнюються – при розбіжностях використовується або голосування (мажоритарний вибір), або вирішальний алгоритм, що заздалегідь визначений у специфікації системи. Такий механізм дозволяє уникнути використання помилкових результатів навіть у разі одиничної відмови програмної версії.

Метод MVP реалізується як у однопроцесорних, так і у розподілених архітектурах. У деяких випадках, наприклад, у системах автоматизованого керування або промислових ІС, він може поєднуватися з апаратним резервуванням, створюючи багаторівневу систему захисту від збоїв [16-17].

Одним із базових варіантів багатоверсійного підходу є дуальне програмування, при якому дві реалізації порівнюються в контрольних точках з метою виявлення розбіжностей. Це дає змогу суттєво підвищити ефективність тестування та відладки. У разі потреби може бути використаний третій модуль як арбітр. Така структура особливо ефективна у випадках, коли необхідно гарантувати правильність виконання критичних функцій ІС.

Розробка багатоверсійних інформаційних систем вимагає наявності однозначної специфікації, що описує функціональні вимоги, формати

вхідних і вихідних даних, точки перехресної перевірки, а також формалізованого вирішального алгоритму. Він має враховувати допустимий розкид числових результатів та забезпечувати стійкість до незначних похибок, що виникають унаслідок різниці в обчислювальних середовищах або реалізаціях.

Кожна версія ПЗ проходить окреме приймальне тестування за допомогою супервізорної програми (драйвера), що фіксує помилки та передає результати до центрального механізму ухвалення рішень. Для багатоверсійного програмного блоку потрібен багатоверсійний драйвер, що координує виконання всіх версій, аналізує результати та забезпечує вибір фінального рішення. У разі виявлення помилки в одній із версій, вона може бути автоматично виключена з голосування, що мінімізує ризики впливу залишкових дефектів на роботу системи загалом.

Центральною передумовою ефективності методу є низька ймовірність однакових помилок у незалежно розроблених версіях. Саме ця властивість робить MVP одним з найперспективніших підходів для побудови надійних інформаційних систем із критичними вимогами до достовірності обробки даних. Такий підхід особливо актуальний у ситуаціях, де використання традиційних засобів верифікації виявляється недостатнім для виявлення залишкових логічних дефектів.

Узагальнюючи, можна зазначити, що багатоверсійне програмування не лише зменшує ймовірність критичних збоїв, а й відкриває можливості для динамічного управління версіями ПЗ на основі контекстних умов середовища, продуктивності окремих реалізацій або змін пріоритетів задач. У майбутньому, з урахуванням зростаючої складності інформаційних систем і необхідності підтримки безперервної роботи, цей підхід може стати ключовим елементом архітектурної стратегії проектування надійних цифрових рішень.

3.3. Метод дуального та комбінованого програмування

В межах сучасних інформаційних систем, що функціонують у відповідальних або критичних середовищах, особливо важливою є здатність програмного забезпечення не лише правильно функціонувати в умовах штатного режиму, а й демонструвати стійкість до відмов під час реальної експлуатації. У цьому контексті методи підвищення надійності, зокрема дуальне та комбіноване програмування, відіграють ключову роль у побудові безпечного програмного забезпечення.

Дуальне програмування є найбільш доступною та економічно виправданою формою багатоверсійного програмування. Його сутність полягає у створенні двох незалежних реалізацій однієї й тієї самої функціональності, які надалі запускаються одночасно. У разі невідповідності результатів неможливо застосувати мажоритарне голосування, як це відбувається при трьох або більше версіях, проте сам факт розбіжності свідчить про наявність помилки та служить потужним інструментом для тестування і виявлення дефектів на етапі верифікації. Такий підхід особливо цінний у процесі створення програмних компонентів для інформаційних систем, де відсутність помилок має вирішальне значення [14-16].

Одним із способів використання дуального підходу є розробка двох версій одного й того самого алгоритму з використанням різних мов програмування. Основна версія створюється з урахуванням вимог експлуатації у реальному середовищі, тоді як модельна – орієнтована на зручність налагодження й автоматизоване тестування. Це дозволяє значно підвищити виявлення залишкових помилок у базовій програмі за рахунок порівняння її поведінки з альтернативною реалізацією. У такій конфігурації модельна версія зазвичай не використовується в робочому середовищі, а служить виключно для забезпечення достовірності.

Цікавою варіацією є підхід, коли єдина версія ПЗ розробляється, але її незалежною перевіркою займаються два окремі фахівці. Співставлення

знайдених помилок дозволяє оцінити імовірну кількість ще не виявлених дефектів у коді. Цей підхід дозволяє мінімізувати ризик залишкових помилок навіть у традиційних одноверсійних інформаційних продуктах.

Комбіноване програмування йде далі й дозволяє організовувати динамічний, адаптивний вибір між кількома варіантами програмного вирішення однієї задачі. У цьому випадку для кожного програмного модуля готується набір альтернативних алгоритмів, реалізованих у вигляді окремих програмних компонентів. Під час роботи системи, залежно від поточних умов, інтенсивності обробки або специфіки вхідних даних, обирається найбільш релевантна реалізація. Це особливо актуально для інформаційних систем, які працюють у реальному часі, де швидкість реагування має першорядне значення.

У простих випадках така адаптація реалізується через підмножини алгоритмів з різним рівнем складності – від максимально спрощених до повнофункціональних. Наприклад, у разі підвищеного навантаження система може тимчасово використовувати більш швидкі, але менш точні алгоритми, зберігаючи здатність до обробки потоку даних без втрат. У спокійні періоди, навпаки, залучаються складніші варіанти, що забезпечують глибший аналіз і високу точність.

Ще один варіант комбінованого підходу полягає у тому, що для кожного критичного блоку програми створюється набір альтернатив, без фіксованої «основної» версії. Система у процесі роботи сама обирає оптимальний варіант на основі поточних вхідних умов. Якщо перша обрана версія не дає задовільного результату, автоматично запускається інша. Такий підхід забезпечує гнучкість, відмовостійкість та можливість автоматичного реагування на несподівані сценарії [15, 17].

Особливої уваги заслуговує концепція неідентичної надмірності, що реалізується в межах комбінованого програмування. На відміну від простого дублювання, при якому усі копії є однаковими і можуть містити одні й ті самі помилки, багатоверсійний підхід з незалежно реалізованими алгоритмами

мінімізує ймовірність того, що дефекти у різних версіях співпадатимуть. Це критично важливо для інформаційних систем, які повинні працювати безперервно, попри наявність залишкових помилок у деяких компонентах.

Ще однією перевагою такого підходу є високий ступінь автоматизації процесів тестування. Комбіновані схеми дозволяють системі не лише адаптуватися до умов функціонування, але й автоматично виявляти помилки під час перехресних перевірок, що суттєво знижує потребу у втручанні людини на етапах контролю якості.

3.4. Модель підвищення надійності програмного забезпечення

Одним із перспективних підходів для підвищення надійності програмного забезпечення є використання методу багатоверсійного програмування. Метод N-версійного ПЗ передбачає створення кількох незалежних версій програми, розроблених різними командами на основі спільної специфікації. Кожна версія реалізує один і той же алгоритм, але з використанням різних мов програмування, інструментів або підходів до проектування. Результати роботи версій порівнюються в точках перехресного контролю, і правильним вважається результат, який отримав консенсус (наприклад, більшість версій повернули однаковий результат) [17].

Запропонована модель складається з п’яти фаз, що відображені на рис. 3.1, що створені на основі аналізу життєвого циклу ПЗ.

1) Збір та специфікація вимог. Виконується формалізація функціональних і системних вимог шляхом інтерв’ю, анкетування, спостереження. Цей етап має забезпечити повноту специфікації для всіх версій.

2) Проектування. Поділяється на концептуальне, логічне та фізичне проектування інформаційної системи. Особлива увага приділяється забезпеченню різноманітності алгоритмів та структур даних у різних версіях.

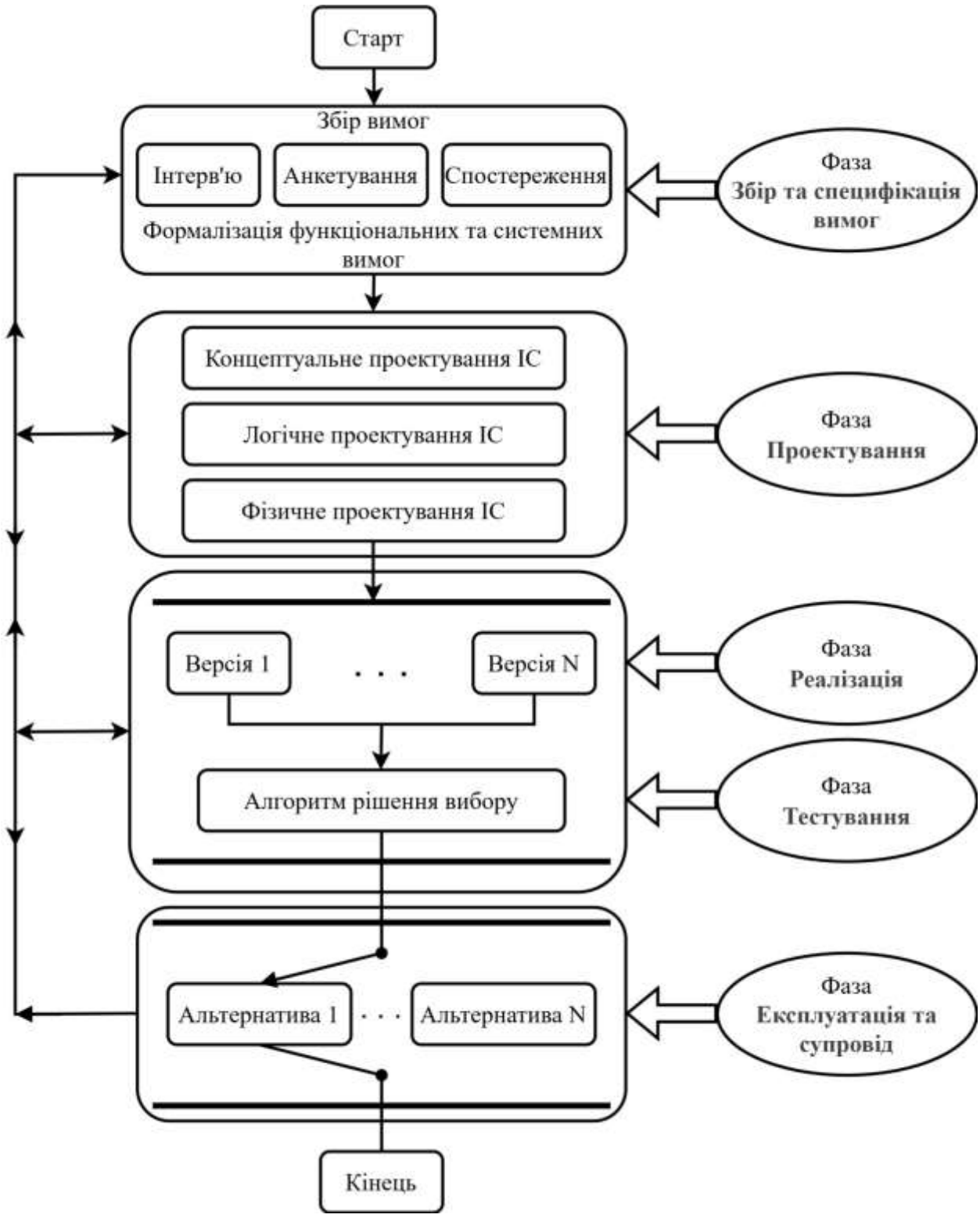

Рис. 3.1. Модель підвищення надійності програмного забезпечення

3) Реалізація. Паралельна та незалежна розробка декількох версій, які базуються на одній специфікації, але створюються різними командами, з використанням різних мов програмування або середовищ. Групи розробників ізольовані для недопущення впливу одна на одну

4) Тестування. У процесі тестування результати роботи кожної версії аналізуються супервізором, який здійснює контроль часу відповіді, повноту вихідних векторів та їх узгодженість. Вирішальний алгоритм використовує мажоритарне голосування або контекстну логіку для визначення правильного результату

5) Експлуатація та супровід. У режимі реального часу система автоматично перемикається між альтернативами (версіями або спрощеними алгоритмами) у залежності від навантаження, ресурсів або наявності відмов. Статистика роботи використовується для подальшого вдосконалення.

Запропонована модель містить низку вдосконалень у порівнянні з класичною NVS-схемою:

- адаптивний вирішальний алгоритм, що враховує не лише більшість, а й точність, швидкість виконання та історію відмов конкретної версії;

- можливість комбінації паралельного й послідовного виконання версій для економії ресурсів;

- інтеграція з механізмами контрольного програмування, такими як хешування, валідація, внутрішні перевірки;

- гнучка схема оновлення й супроводу, що дозволяє оперативно вводити нові версії без зупинки ІС.

Ці елементи дозволяють застосовувати модель в умовах обмежених обчислювальних ресурсів, а також у хмарних, мобільних або розподілених середовищах, де традиційні підходи до забезпечення відмовостійкості малоефективні.

Проведемо кількісне оцінювання надійності N-версійного ПЗ з використанням імовірнісної моделі, що базується на біноміальному розподілі, та аналізу залежності системної надійності від кількості версій.

Нехай кожна версія ПЗ має ймовірність безвідмовної роботи R, імовірність відмови – $q=1-R$. Припустимо, що:

– версії розроблені незалежно, тобто їх відмови – незалежні випадкові події;

– результат вважається правильним, якщо принаймні k з N версій дають однаковий (правильний) результат, де $k = \left\lfloor \frac{N}{2} \right\rfloor + 1$ (мажоритарний принцип).

Тоді, ймовірність безвідмовної роботи всієї системи розраховується за формулою:

$$R_{системи}(N) = \sum_{i=k}^{N} \binom{N}{i} R^i (1-R)^{N-i};$$

де: N – кількість версій;

i – кількість правильно працюючих версій;

$\binom{N}{i}$ – біноміальний коефіцієнт.

Для оцінювання впливу кількості версій на загальну надійність системи було обрано фіксоване значення надійності однієї версії R=0,95. Розрахунок здійснено для кількості версій N від 3 до 10. Обчислення проводилися програмним засобом з використанням мови Python (рис. 3.2), де було реалізовано функцію для підсумовування ймовірностей за біноміальним розподілом. Результати розрахунків наведено в таблиці 3.1.

Таблиця 3.1.

Розрахунок надійності програмного забезпечення

Кількість версій (N)	Консенсус (k)	Надійність системи
3	2	0.9928
4	3	0.9860
5	3	0.9988
6	4	0.9978
7	4	0.9998
8	5	0.9996
9	5	0.99997
10	6	0.99994

З отриманих результатів випливає, що вже при трьох версіях система демонструє надійність понад 99%. Подальше зростання кількості версій призводить до покращення, проте після N=5 приріст надійності сповільнюється і досягає асимптотичного рівня, близького до 1. Це означає, що ефективне підвищення надійності можливе вже при невеликій кількості

версій, а надмірне дублювання призводить до незначного покращення, проте потребує більше ресурсів.

```python
import math
import matplotlib.pyplot as plt

def binomial_probability(n, k, p):
    return math.comb(n, k) * (p ** k) * ((1 - p) ** (n - k))

def system_reliability(N, R_single):
    k = N // 2 + 1
    reliability = sum(binomial_probability(N, i, R_single) for i in range(k, N + 1))
    return reliability

def run_calculations(R_single=0.95, N_range=range(3, 11)):
    print(f"{'N':<5}{'k (мажоритарно)':<20}{'Надійність системи':<25}")
    print("-" * 50)
    results = []
    for N in N_range:
        k = N // 2 + 1
        R_sys = system_reliability(N, R_single)
        print(f"{N:<5}{k:<20}{R_sys:<25.5f}")
        results.append((N, R_sys))
    return results

def plot_results(results, R_single):
    N_vals, R_vals = zip(*results)
    plt.figure(figsize=(10, 6))
    plt.plot(N_vals, R_vals, marker='o', linestyle='-', color='blue')
    plt.title(f"Надійність N-версійної системи при R_однієї = {R_single}")
    plt.xlabel("Кількість версій (N)")
    plt.ylabel("Надійність системи R(N)")
    plt.grid(True)
    plt.ylim(0.9, 1.01)
    plt.xticks(N_vals)
    plt.show()

if __name__ == "__main__":
    R_single_version = 0.95
    N_range = range(3, 11)
    results = run_calculations(R_single=R_single_version, N_range=N_range)
    plot_results(results, R_single=R_single_version)
```

Рис. 3.2. Програмне забезпечення для розрахунку надійності ПЗ

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи було проведено всебічне дослідження проблеми забезпечення надійності програмного забезпечення, обґрунтовано доцільність удосконалення існуючих підходів та розроблено власну модель підвищення надійності.

У першому розділі проведено глибокий аналіз сучасного стану проблеми надійності програмного забезпечення. Доведено, що зі зростанням складності ІТ-систем і обсягів критичних застосувань (медицина, транспорт, банківські системи тощо) забезпечення високого рівня надійності стає визначальним фактором при розробці ПЗ. Показано, що традиційні підходи до надійності вже не відповідають динаміці сучасних змін і потребують адаптації та розширення.

У другому розділі розглянуто наукові підходи до моделювання, оцінювання та управління надійністю ПЗ. Проаналізовано найпоширеніші моделі надійності (експоненційні, логістичні, моделі Мусса, Джелінського-Морана тощо) та методи обробки помилок. Проведено класифікацію типів помилок (синтаксичні, логічні, архітектурні, експлуатаційні), а також вивчено фактори, що найбільше впливають на зниження надійності. Особливу увагу приділено фазам життєвого циклу, на яких найбільш ефективним є виявлення та усунення помилок.

У третьому розділі вдосконалено алгоритм підвищення надійності ПЗ, що базується на інтеграції кількох підходів:

- методу багатоверсійного програмування, що дозволяє зменшити ймовірність загального збою системи при помилці в одній з версій;
- підходу дуального програмування, який використовує результати двох незалежних реалізацій для підвищення точності та відмовостійкості;
- комбінованого вирішального алгоритму, який адаптивно обирає рішення на основі аналізу достовірності результатів.

Запропонований алгоритм реалізовано у вигляді універсальної моделі,

яка може масштабуватися залежно від типу програмного забезпечення. Модель враховує п’ятифазний життєвий цикл, на кожному етапі якого впроваджуються конкретні дії щодо забезпечення та контролю надійності. Її застосування дозволяє не лише виявляти помилки на ранніх етапах, але й реагувати на відмови в реальному часі.

Результати дослідження підтверджують, що комплексне застосування запропонованого підходу дозволяє:

- зменшити частоту критичних помилок у програмному забезпечені інформаційних систем;
- підвищити середній час безвідмовної роботи (MTBF);
- скоротити час відновлення після збоїв (MTTR);
- знизити загальні витрати на супровід ПЗ.

Практичне значення роботи полягає в тому, що розроблений алгоритм може бути використаний як основа для побудови систем управління якістю ПЗ в організаціях, які працюють у сферах, де від надійності залежить безпека, фінансова стабільність або якість обслуговування.

Перспективи подальших досліджень передбачають адаптацію запропонованого підходу до різних типів систем (розподілених, реального часу, мобільних), а також інтеграцію алгоритму з інструментами машинного навчання для прогнозування збоїв і самостійного коригування поведінки ПЗ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Добровольський Ю.Г. Інженерія надійності програмного забезпечення: навч. посіб. – Чернівці: ЧНУ ім. Ю. Федьковича, 2022. – 120 с.

2. Яковина В., Мацелюх В. Огляд і аналіз моделей надійності програмного забезпечення // Вісник Нац. ун-ту «Львівська політехніка». – 2017. – № 872. – С. 130–137.

3. Лавріщева К.М. Програмна інженерія: підручник. – Київ: Видавничий дім «Слово», 2008. – 319 с.

4. Коновалов В.С., Радоуцький К.Є. Сучасні принципи і методи проектування програмного забезпечення: конспект лекцій. – Харків: УкрДАЗТ, 2015. – 109 с.

5. Козловський С.І. Якість програмного забезпечення та тестування: базовий курс. – Тернопіль: ТНЕУ, 2020. – 180 с.

6. Куліш О.В. Тестування програмного забезпечення: навч. посіб. – Черкаси: ЧДТУ, 2017. – 150 с.

7. Lyu M.R. Software Reliability Engineering: A Roadmap // Proceedings of the Future of Software Engineering. – 2007. – С. 153–170.

8. Shatnawi O. Modelling Open-Source Software Reliability Incorporating Swarm Intelligence-Based Techniques // arXiv preprint arXiv:2401.02664. – 2024.

9. Sangeeta, Sharma K., Bala M. New Failure Rate Model for Iterative Software Development Life Cycle Process // arXiv preprint arXiv:1910.00903. – 2019.

10. AL-Saati N.A., Alabajee M.A. Selecting Best Software Reliability Growth Models: A Social Spider Algorithm based Approach // arXiv preprint arXiv:2001.09924. – 2020.

11. AL-Saati N.A., Alabajee M.A. A Comparative Study on Parameter Estimation in Software Reliability Modeling using Swarm Intelligence // arXiv preprint arXiv:2003.04770. – 2020.

12. Kapur P.K., Garg R.B., Kumar S. Software Reliability Models: Past, Present and Future // Springer. – 1995.

13. Kapur P.K., Pham H., Anand S., Yadav K. Software Reliability Assessment with OR Applications. – Springer, 2011.

14. Yamada S., Ohba M., Osaki S. S-Shaped Software Reliability Growth Models and Their Applications // IEEE Transactions on Reliability. – 1983. – Vol. R-32, No. 5. – C. 475–484.

15. Pham H. System Software Reliability. – Springer, 2006.

16. Goel A.L., Okumoto K. Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures // IEEE Transactions on Reliability. – 1979. – Vol. R-28, No. 3. – C. 206–211.

17. Xie M., Goh T.N., Ranjan P. Some Effective Control Charts for Monitoring Software Reliability // Reliability Engineering & System Safety. – 2002. – Vol. 76, No. 3. – C. 283–290.