Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Оцінювання надійності програмного продукту на етапі проєктування архітектури

Виконав(ла): студент(ка) 6 курсу, групи СНм-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Нога О.В. (прізвище та ініціали)

Керівник (підпис) Марценюк В.П. (прізвище та ініціали)

Нормоконтроль (підпис) Дуда О.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

студенту Нога Олександр Васильович
(прізвище, ім'я, по батькові)

1. Тема роботи **Оцінювання надійності програмного продукту на етапі проєктування архітектури**

Керівник роботи **д.т.н., проф. Марценюк В.П.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*24*» *листопада* 2023 року № *4/7-1099*

2. Термін подання студентом завершеної роботи 25 грудня 2023 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП 1 АНАЛІЗ МОДЕЛЕЙ ПРОГРАМНТХ ПРОДУКТІВ ДЛЯ ОЦІНКИ НАДІЙНОСТІ 1.1 Моделі надійності на основі стану системи 1.2 Моделі на основі шляху виконання програми 1.3 Адитивні моделі надійності 2 МОДЕЛЬ НАДІЙНОСТІ ПРОГРАМНОГО ПРОДУКТУ НА ОСНОВІ ОБ'ЄДНАННЯ МОДЕЛЕЙ НАДІЙНОСТІ 2.1 Об'єднання моделей на основі стану 2.2 Узагальнення моделей на основі шляху виконання 2.3 Узагальнення адитивних моделей 3 ОБМЕЖЕННЯ МОДЕЛЕЙ ОЦІНКИ НАДІЙНОСТ НА ОСНОВІ ПРОГРАМНОЇ АРХІТЕКТУРИ 3.1 Рівень декомпозиції системи на компоненти 3.2 Оцінка надійності окремих компонентів 3.3 Оцінка надійності інтерфейсів 3.4 Оцінка ймовірностей переходу 3.5 Залежності відмов між компонентами та інтерфейсами 3.6 Аналіз чутливості моделі до надійності компонента 3.7 Врахування різних архітектурних стилів 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ; СПИСОК ДЖЕРЕЛ; ДОДАТКИ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Семчишин В. С., к.т.н., доц.		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викл.		

7. Дата видачі завдання " " 202 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	21.09.23-27.09.23	*Виконано*
2.	Підбір наукових джерел по темі роботи	28.09.23-04.10.23	*Виконано*
3.	Переклад та опрацювання наукових джерел	05.10.23-11.10.23	*Виконано*
	по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо теми	12.10.23-18.10.23	*Виконано*
	Кваліфікаційної роботи		
5.	Оформлення першого розділу	19.10.23-25.10.23	*Виконано*
6.	Оформлення другого розділу	26.10.23-01.11.23	*Виконано*
7.	Оформлення третього розділу	02.11.23-08.11.23	*Виконано*
8.	Виконання завдання до підрозділу «Охорона праці»	09.11.23-15.11.23	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	16.11.23-23.11.23	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	23.11.23-29.11.23	*Виконано*
11.	Нормоконтроль	30.11.23-09.12.23	*Виконано*
12.	Перевірка на плагіат	10.12.23	*Виконано*
13.	Попередній захист кваліфікаційної роботи	14.12.23	*Виконано*
14.	Захист кваліфікаційної роботи	26.12.2023	

Студент _____ Нога О.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марценюк В.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Оцінювання надійності програмного продукту на етапі проєктування архітектури" // Кваліфікаційна робота освітнього рівня «Магістр» // Нога Олександр Васильович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНм-61 // Тернопіль, 2023 // с. – 49, джерел – 30.

Ключові слова: надійність програмного забезпечення, надійність програмної архітектури, модель програми.

З орієнтацією на використання компонентів у розробці програмного забезпечення, виникає необхідність у методах моделювання, які дозволяють оцінювати надійність програмного забезпечення через взаємодію компонентів, їх використання та інтерфейси. В даній роботі виконується огляд літературних джерел для виявлення підходів до оцінки надійності програмного забезпечення на основі компонентів, який базується на архітектурі, та його можливе використання на кожному етапі розробки – від проектування до фінального впровадження. Також проводиться класифікація моделей на основі архітектури та досліджується їх взаємозв'язок, критичний аналіз припущень, обмежень і застосовності для визначення напрямків майбутніх досліджень.

ANNOTATION

“Evaluation of software reliability at the architecture design stage” // Master’s degree qualification paper // Noha Oleksandr Vasyliovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group, group СНм-61 // Ternopil, 2023 // p. – 49, references – 30.

Key words: software reliability, software architecture reliability, program model.

With an increasing emphasis on reusability in software development, there's a shift towards software design based on components. Consequently, there arises a need for modeling approaches capable of considering software architecture and assessing reliability by factoring in interactions between components, their usage, reliability, and interfaces with others. This work conducts a review of literature sources to identify approaches for assessing the reliability of software based on components, which rely on architecture. It explores their potential utilization across every stage of development, from the design phase to the final implementation.

General requirements for architecture-based models are defined, proposing a classification. Subsequently, key models within each class are detailed, discussing their interconnections. A critical analysis of primary assumptions, limitations, and applicability of these models is provided to aid in identifying directions for future research.

ЗМІСТ

ВСТУП

Актуальність задачі.

Було запропоновано ряд аналітичних моделей для вирішення проблеми кількісного визначення надійності програмного забезпечення, одного з найважливіших показників якості програмного забезпечення. Різноманітність існуючих моделей надійності програмного забезпечення можна класифікувати відповідно до кількох різних систем класифікації [3]. Класифікація, запропонована в [4], базується головним чином на фазі життєвого циклу програмного забезпечення, протягом якої моделі застосовуються: фаза відладки, фаза перевірки або робоча фаза.

Багато дослідницьких зусиль у минулому було зосереджено на моделюванні зростання надійності на етапі відладки. Ці так звані моделі чорної скриньки розглядають програмне забезпечення як монолітне ціле, враховуючи лише його взаємодію із зовнішнім середовищем, без спроб моделювання внутрішньої структури. Зазвичай у цих моделях не використовується жодна інша інформація, окрім даних про несправності. Загальною рисою моделей чорного ящика є стохастичне моделювання процесу відмови, припускаючи деяку параметричну модель сукупного числа відмов протягом кінцевого інтервалу часу або часу між відмовами. Дані про помилки, отримані під час тестування програми, потім використовуються для оцінки параметрів моделі або калібрування моделі. Для детального огляду моделей зростання надійності «чорної скриньки» читача відсилають до таких книг і статей, як [1], [2].

Із збільшенням уваги до повторного використання все більше організацій розробляють і використовують програмне забезпечення не лише як комплексні програми, як у минулому, але й як складові частини більших програм. Існуючі моделі чорного ящика явно не підходять для моделювання такої великої компонентної програмної системи. Натомість існує потреба в техніці моделювання, яка може бути використана для аналізу програмних компонентів і того, як вони поєднуються між собою. Метою підходу білої скриньки є оцінка

надійності системи з урахуванням інформації про архітектуру програмного забезпечення, що складається з компонентів.

Враховуючи, що програмне забезпечення, написане з нуля, є винятком, і що сучасна практика розробки програмного забезпечення спрямована скоріше на еволюцію існуючого програмного забезпечення, логічним наслідком є посилення прогнозів, виконаних для певного продукту, за допомогою польових даних порівняно з попередніми подібними програмними продуктами. Коли доступна лише архітектура, а не код, можна побудувати імітаційну модель і провести дослідження, подібні до тих, які призначені для вивчення впливу компонентів на надійність і продуктивність системи для даної архітектури. Наприклад, менеджери можуть використовувати цю інформацію для перегляду архітектури системи та планування розподілу ресурсів. Усі такі прогнози та рішення будуть постійно оновлюватись у міру появи нових даних тестування.

Враховуючи архітектуру програмного забезпечення, інформацію про компоненти та їхню взаємодію, можна виконувати обчислення в аналізі «що – коли». Оскільки підхід, заснований на архітектурі, дозволяє зрозуміти чутливість усієї системи до кожного компонента, його можна використовувати для розподілу зусиль на ті компоненти, які є критичними з точки зору надійності чи продуктивності.

Мета роботи.

Метою цієї роботи є виконання літературного огляду підходів до оцінки надійності програмних систем, заснованого на архітектурі.

Для досягнення мети потрібно розв'язати задачі:

– Виявити методи аналізу надійності та продуктивності додатків, створених із компонентів програмного забезпечення багаторазового використання в тому числі COTS.

– Встановити, як надійність/продуктивність системи залежить від надійності/продуктивності її компонентів та їх взаємодії.

– Виявити механізми аналізу чутливості надійності програми до надійності компонентів та інтерфейсів.

– Визначити можливість управління процесом виявлення критичних компонентів та інтерфейсів.

– Встановити перелік методів кількісного аналізу, які можна застосовувати протягом усього життєвого циклу програмного забезпечення.

Об’єкт дослідження: Процеси оцінювання надійності програмного забезпечення на основі його архітектури.

Предмет дослідження: моделі програмного продукту для дослідження надійності.

Наукова новизна отриманих результатів. Наукова новизна полягає у пропозиції фреймворку для оцінки надійності програмного продукту на основі його архітектури.

Практичне значення отриманих результатів. Підхід, заснований на архітектурі, можна використовувати для розрахунку надійності системи з інформації про компоненти або для визначення надійності системи та розрахунку розподіленої надійності компонентів. Він може показати масштаб і обсяг зусиль, необхідних для демонстрації необхідних рівнів надійності компонентів. Нарешті, це може пролити світло на доцільність побудови системи з бажаним рівнем надійності на основі повторного використання певної колекції компонентів.

Апробація результатів та особистий внесок здобувача. Основні положення роботи доповідались, розглядались та обговорювались на науковій конференції Тернопільського національного технічного університету імені Івана Пулюя та опубліковані у тезах студентської наукової конференції "Інформаційні моделі системи та технології – 2023", та " Актуальні задачі сучасних технологій – 2023", які проводились у ТНТУ.

1 АНАЛІЗ МОДЕЛЕЙ ПРОГРАМНТХ ПРОДУКТІВ ДЛЯ ОЦІНКИ НАДІЙНОСТІ

Основна мета поточного розділу полягає в тому, щоб проаналізувати на структурі, в рамках якої були розроблені існуючі моделі надійності програмного забезпечення на основі архітектури. Таким чином, різні підходи до оцінки надійності програмного забезпечення на основі архітектури базуються на наступних загальних кроках.

Крок 1. Ідентифікація модулів програмної архітектури. Основною сутністю підходу, заснованого на архітектурі, є стандартна концепція розробки програмного забезпечення модуля. Хоча загальноприйнятого визначення не існує, модуль розуміється як логічно незалежний компонент системи, який виконує чітко визначену функцію. Це означає, що модуль можна спроектувати, реалізувати та протестувати незалежно. Визначення модуля – це завдання на рівні користувача, яке залежить від таких факторів, як система, що аналізується, можливість отримання необхідних даних тощо. У цьому документі модуль і компонент використовуватимуться як взаємозамінні.

Крок 2. Архітектура на основі модулів. Поведінка програмного забезпечення щодо того, як взаємодіють різні модулі програмного забезпечення, визначається через архітектуру програмного забезпечення. Взаємодія відбувається тільки шляхом передачі контролю виконання. У випадку послідовного програмного забезпечення в кожен момент управління здійснюється лише в одному з модулів. Архітектура програмного забезпечення також може містити інформацію про час виконання кожного модуля.

Крок 3. Поведінка при виникненні збою. На наступному кроці поведінка при відмові визначається та пов’язується з архітектурою програмного забезпечення. Збій може статися під час виконання будь-якого модуля або під час передачі керування між двома модулями. Поведінка при відмові модулів та інтерфейсів між модулями може бути задана в термінах їхньої надійності або частоти відмов (постійної або залежної від часу).

Крок 4. Поєднання програмної архітектури зі збоями. Залежно від методу, який використовується для поєднання архітектури програмного забезпечення з поведінкою при відмові, література містить три принципово різні підходи: підхід на основі стану, підхід на основі шляху та адитивний підхід. У наступних розділах детально описано ключові моделі в кожному з наведених вище класів.

1.1 Моделі надійності на основі стану системи

Цей клас моделей використовує граф потоку керування для представлення архітектури системи. Передбачається, що передача управління між модулями має Марківську властивість, яка означає, що, враховуючи знання модуля, який отримав керування в будь-який момент часу, майбутня поведінка системи умовно не залежить від минулої поведінки.

Архітектура програмного забезпечення була змодельована як ланцюг Маркова з дискретним часом (DTMC), ланцюг Маркова з безперервним часом (CTMC) або напівмарківський процес (SMP). Деякі з них представляють додатки, які працюють на вимогу, для яких запуски програмного забезпечення, які відповідають завершенню виконання, можуть бути чітко ідентифіковані. Інші добре підходять для безперервно працюючих програмних додатків, наприклад, у системах керування в режимі реального часу, де або важко визначити, що являє собою цикл, або може бути дуже велика кількість таких циклів, якщо припускається, що кожен цикл також складається з циклу.

Як запропоновано в [2], [5], моделі на основі стану можна класифікувати як складені або ієрархічні. Композитний метод поєднує архітектуру програмного забезпечення з поведінкою відмов у складену модель, яка потім розв’язується для прогнозування надійності програми. Інша можливість полягає в застосуванні ієрархічного підходу, тобто спочатку вирішити архітектурну модель, а потім накласти поведінку відмов на рішення архітектурної моделі, щоб передбачити надійність.

1.1.1 Модель надійності Літтлвуда

Це одна з найстаріших, але досить загальних моделей надійності програмного забезпечення на основі архітектури.

Передбачається, що програмна архітектура безперервно працюючої програми може бути описана незвідним SMP. Програма складається з кінцевої кількості модулів, а передача управління між модулями описується ймовірністю p_{ij} = Pr {перехід з модуля i до модуля j} . Час, проведений у кожному модулі, має загальний розподіл $F_{ij}(t)$, який залежить від i та j з скінченним середнім m_{ij} .

Коли модуль i виконується, виникають збої відповідно до процесу Пуассона з параметром λ_i. Передача управління між модулями (інтерфейсами) сама піддається збою; коли модуль i викликає модуль j, існує ймовірність ν_{ij} виникнення збою.

Інтерес композитної моделі зосереджений на загальній кількості відмов інтегрованої програмної системи в інтервалі часу $(0, t]$, позначеному $N(t)$, яка є сумою відмов у різних модулях за час їх перебування, разом із відмовами інтерфейсу. Можливо отримати повний опис цього процесу точки відмов, але оскільки точний результат є дуже складним, він навряд чи матиме практичне застосування. Отримано асимптотичне наближення процесу Пуассона для $N(t)$. при припущенні, що збої дуже рідкісні. Таким чином, час між збоями буде, як правило, набагато більшим, ніж час між обмінами керуванням, тобто між послідовними збоями програм відбуватиметься багато обмінів керуванням. Частота відмов для такого Пуассонівського процесу задається рівнянням

$$\lambda_S = \sum_i a_i \lambda_i + \sum_{i,j} b_{ij} \nu_{ij}, \tag{1.1}$$

де $a_i = \dfrac{\pi_i \sum_j p_{ij} m_{ij}}{\sum_i \pi_i \sum_j p_{ij} m_{ij}}$ являє собою частку часу, витраченого на модуль i , та

$b_{ij} = \dfrac{\pi_i p_{ij}}{\sum_i \pi_i \sum_j p_{ij} m_{ij}}$ частота передачі управління між i і j . Ці терміни залежать

лише від параметрів, які характеризують архітектуру програмного забезпечення: ймовірності переходу p_{ij}, середній час виконання m_{ij} та ймовірності стаціонарного стану вбудованого ланцюга Маркова π_i.

1.1.2 Модель Чонга

Ця модель розглядає надійність програмного забезпечення стосовно використання модулів та їх надійності.

Припускається, що граф програмного потоку завершеної програми має один вхідний і один вихідний вузол, і що передача управління між модулями може бути описана поглинаючим DTMC з матрицею ймовірності переходу $P = [p_{ij}]$.

Модулі виходять з ладу незалежно, і надійність модуля i – це ймовірність R_i того, що модуль виконує свою функцію правильно, тобто модуль видає правильний вихід і правильно передає керування наступному модулю.

Додаються два поглинаючі стани C і F, що представляють правильний вихід і відмову, відповідно, і матриця ймовірності переход*у* P модифікується до $\hat{P}$. Вихідна ймовірність переходу p_{ij} між модулями i і j модифікується на $R_i p_{ij}$, що представляє ймовірність того, що модуль i дає правильний результат і керування передається модулю j. Зі стану виходу n створюється орієнтоване ребро до стану C з імовірністю переходу R_n для представлення правильного виконання. Відмова модуля i розглядається шляхом створення орієнтованого ребра до стану відмови F з ймовірністю переходу $(1 - R_i)$. Таким чином, DTMC, визначений матрицею ймовірностей переходу $\hat{P}$, є складеною моделлю програмної системи. Надійність програми – це ймовірність досягнення поглинаючого стану C DTMC. Нехай Q – матриця, отримана з $\hat{P}$ видаленням рядків і стовпців, що відповідають поглинаючим станам C і F. $Q^k(1, n)$ представляє ймовірність досягнення стану n від 1 до k переходів. Від початкового стану 1 до кінцевого стану n кількість переходів k може змінюватися від 0 до нескінченності. Це не важко показати

$$S = I + Q + Q^2 + Q^3 + \cdots = \sum_{k=0}^{\infty} Q^k = (I - Q)^{-1}, \tag{1.2}$$

тому з цього випливає, що загальну надійність системи можна обчислити як

$$R = S(1, n)R_n. \tag{1.3}$$

Модель Чонга використовується для оцінки надійності різних гетерогенних архітектур програмного забезпечення, таких як пакетно-послідовний/конвеєрний, виклик і повернення, паралельні/конвеєрні фільтри та відмовостійкість.

1.1.3 Модель Лапрі

Ця модель є окремим випадком моделі Літтлвуда [6].

Програмна система складається з *n* компонентів, а передача керування між компонентами описується СТМС. Параметрами є середній час виконання компонента *i* задано $1/\mu_i$ і ймовірність p_{ij}, що компонент *j* виконується після компонента *i* враховуючи, що під час виконання компонента *i* не відбулося збою. Кожен компонент виходить з ладу з постійною інтенсивністю відмов λ_i.

Модель системи являє собою n+1 стан СТМС, де система знаходиться в стані i, $0 \le i \le n$ (компонент *i* виконується без збоїв у стані i), а $(n + 1)$-й стан (стан поглинання) є станом вимкнення, досягнутим після виникнення збою. Асоційована генераторна матриця між станами до $B = [b_{ij}]$ задана як

$$b_{ii} = -(\mu_i + \lambda_i), \quad b_{ij} = p_{ij}\mu_i \text{ для } i \neq j. \tag{1.4}$$

Матриця *B* розглядається як сума двох породжуючих матриць, таких що

– процес виконання регулюється *B* діагональні елементи якого дорівнюють $-\mu_i$ і його недіагональні елементи рівні $p_{ij}\mu_i$;

– процес відмови регулюється *B*, діагональні елементи якого дорівнюють $-\lambda_i$ а недіагональні записи дорівнюють нулю.

Передбачається, що частота відмов набагато менша, ніж частота виконання, і, отже, очікується, що багато обмінів керуванням відбудеться до

того, як станеться відмова, тобто процес виконання збіжиться до стаціонарного стану до того, як, швидше за все, відбудеться відмова. Таким чином, наслідком припущення $\lambda_i \ll \mu_i$ є асимптотична поведінка щодо процесу виконання, що дозволяє прийняти ієрархічний метод розв'язання. Як наслідок, частота відмов системи зводиться до

$$\lambda_S = \sum_{i=1}^{n} \pi_i \lambda_i , \tag{1.5}$$

де вектор імовірності стаціонарного стану $\pi = [\pi_i]$ є розв'язком $\pi B' = 0$. Цей результат має просту фізичну інтерпретацію, враховуючи, що π_i це частка часу, проведеного в стані i коли збій не відбувається. Термін $\pi_i \lambda_i$ тому можна розглядати як еквівалентну частоту відмов компонента i.

1.1.4 Модель Кубата

Ця модель розглядає випадок завершення програми, що складається з n модулів, призначених для K різних завдань. Кожне завдання може вимагати кількох модулів, і той самий модуль можна використовувати для різних завдань.

Переходи між модулями слідують за DTMC таким чином, що з імовірністю $q_i(k)$ завдання k спочатку викличе модуль i та з імовірністю $p_{ij}(k)$ завдання k викличе модуль j після виконання в модулі i. Час перебування під час візиту в модулі i за завданням k має функцію розподілу імовірності $g_i(k, t)$. Таким чином, модель архітектури для кожного завдання стає напівмарківським процесом – SMP.

Інтенсивність відмови модуля i позначаємо, як λ_i.

Імовірність того, що під час виконання завдання k не відбудеться збій, поки в модулі i є $R_i(k) = \int_0^\infty e^{-\lambda_i t} g_i(k, t)\, dt$.

Очікувана кількість відвідувань у модулі i завданням k, позначеним $V_i(k)$, можна отримати, розв'язуючи

$$V_i(k) = q_i(k) + \sum_{j=1}^{..} V_j(k)\, p_{ji}(k). \tag{1.6}$$

Імовірність того, що не буде відмов під час виконання завдання k, можна приблизно визначити $R(k) \approx \prod_{i=1}^{n} [R_i(k)]^{V_i(k)}$, і частота відмов системи обчислюється за допомогою $\lambda_S = \sum_{k=1}^{K} r_k [1 - R(k)]$, де r_k швидкість надходження завдання k.

1.1.5 Модель Гокгейла

Новизна цієї роботи полягає в спробі експериментально визначити архітектуру програмного забезпечення та надійність його компонентів шляхом тестування програми за допомогою набору регресійних тестів [7].

Завершення програми описується поглинаючим DTMC. Дані трасування, створені інструментом аналізу покриття під назвою ATAC [8] під час тестування, використовуються для визначення ймовірностей переходу p_{ij} між модулями. Очікуваний час, витрачений на модуль i за відвідування t_i обчислюється як добуток очікуваного часу виконання кожного блоку та кількості блоків у модулі.

Поведінка відмов кожного компонента описується розширеною моделлю неоднорідного процесу Пуассона (NHPP) з використанням залежної від часу інтенсивності відмов $\lambda_i(t)$, визначеної вимірюванням покриття блоків під час тестування програми разом із підходом щільності відмов.

Очікувана кількість відвідувань модуля i, позначена V_i, обчислюється шляхом розв'язання наступної системи лінійних рівнянь:

$$V_i = q_i + \sum_{j=1}^{n} V_j p_{ji}, \tag{1.7}$$

де q позначає вектор ймовірності початкового стану.

Надійність модуля i, враховуючи залежну від часу інтенсивність відмови $\lambda_i(t)$ і загальний очікуваний час $V_i t_i$, витрачений на модуль на виконання програми, визначається як $R_i = e^{-\int_0^{V_i t_i} \lambda_i(t)\,dt}$.

Таким чином, надійність загальної програми стає $R = \prod_{i=1}^{n} R_i$.

1.2 Моделі на основі шляху виконання програми

Цей клас моделей базується на тих самих загальних кроках, що й моделі на основі стану, за винятком того, що підхід, використаний для поєднання архітектури програмного забезпечення з поведінкою відмов, можна описати як заснований на шляхові виконання, оскільки надійність системи обчислюється з урахуванням можливого виконання шляху програми або експериментально шляхом тестування, або алгоритмічно.

1.2.1 Модель Шумана

Це одна з найперших моделей, яка враховує надійність модульних програм, запроваджуючи підхід на основі шляхів із використанням частот, з якими виконуються різні шляхи [9].

Модель припускає знання різних шляхів і частот f_i , з якими шлях i виконується.

Імовірність відмови шляху i на кожному циклі, позначений q_i, характеризує поведінку відмови.

Загальна кількість відмов n_f у N тестових прогонів визначається як

$$n_f = \sum_{i=1}^{m} N f_i q_i, \tag{1.8}$$

де Nf_i загальна кількість проходжень шляху i. Імовірність відмови системи під час будь-якого тестового запуску визначається як $q_0 = \lim_{N\to\infty} \frac{n_f}{N} = \sum_{i=1}^{m} f_i q_i$.

1.2.2 Модель Крішнамурті-Матура

Цей метод поєднання архітектури та поведінки при відмові спочатку передбачає обчислення оцінок надійності шляху на основі послідовності компонентів, що виконуються для кожного тестового прогону, а потім усереднення їх для всіх тестових прогонів для отримання оцінки надійності системи [10].

Послідовність компонентів уздовж різних шляхів спостерігається за допомогою трасування компонентів, зібраних під час тестування або моделювання поведінки програми.

Кожен компонент характеризується своєю надійністю Rm.

Послідовність компонентів уздовж шляху, пройденого для тест-кейсу tc, розглядається як послідовна система. Припускаючи, що окремі компоненти виходять з ладу незалежно один від одного, випливає, що надійність шляху є продуктом надійності компонентів. Компонентна траса програми P для заданого тест-кейсу tc, позначена $M(P, \text{tc})$, є послідовністю компонентів m, які виконуються, коли P виконується для tc. Таким чином, надійність шляху в P виконується на тест-кейсі $\text{tc} \in \text{TS}$ визначається як

$$R_{\text{tc}} = \prod_{\forall m \in M(P,\text{tc})} R_m \ . \tag{1.9}$$

Оцінка надійності програми щодо тестового набору TS буде $R = \frac{\sum_{\forall \text{tc} \in \text{TS}} R_{\text{tc}}}{|\text{TS}|}$.

Цікавий випадок, коли більшість виконуваних шляхів містять компоненти всередині циклів, і ці цикли проходять достатньо велику кількість разів. Тоді надійність окремих шляхів може стати низькою, що призведе до низьких оцінок надійності системи. Виконуючи компонент довільно велику кількість разів, можна легко знизити оцінку надійності системи набагато нижче її справжньої надійності, якщо ігнорувати залежність між компонентами.

У цій роботі внутрішньокомпонентна залежність моделюється шляхом «згортання» кількох входжень компонента на шляху виконання до k входжень, де $k > 0$. k називається ступенем незалежності (Degree of Independence – DOI), так що повна залежність моделюється за допомогою встановлення DOI на 1 і повна незалежність шляхом встановлення DOI на ∞ . На даний момент незрозуміло, як визначити відповідне значення DOI.

1.2.3 Метод Якуба

Цей метод аналізу надійності є специфічним для програмного забезпечення на основі компонентів, аналіз якого суворо базується на сценаріях виконання [11]. Сценарій – це набір взаємодій компонентів, викликаних конкретним вхідним стимулом, і він пов'язаний з концепцією операцій і типів виконання, що використовуються в операційних профілях.

Використовуючи сценарії, будується імовірнісна модель під назвою компонентний графік залежності (CDG). Вузол n_i CDG моделює виконання компонента із середнім часом виконання t_i. Ймовірність переходу p_{ij} асоціюється з кожним орієнтованим ребром e, яке моделює перехід від вузла n_i до n_j. CDG має два додаткові вузли, s: початковий вузол і t: кінцевий вузол.

Процес відмови враховує надійність компонента R_i та надійність переходу $(1 - \nu_{ij})$, пов'язану з вузлом n_i і з переходом від вузла n_i до n_j відповідно.

На основі CDG представлений алгоритм обходу дерева для оцінки надійності програми як функції надійності її компонентів та інтерфейсів. Алгоритм розгортає всі гілки CDG, починаючи з початкового вузла. Розширення дерева в ширину представляє логічні шляхи «АБО» і, отже, перекладається як підсумовування надійностей, зважених за ймовірністю переходу вздовж кожного шляху. Глибина кожного шляху представляє послідовне виконання компонентів, логічне «І», і, отже, перекладається на збільшення надійності. Розширення шляху в глибину припиняється, коли наступний вузол є кінцевим вузлом (природне завершення виконання програми) або коли сума часу виконання цього потоку дорівнює середньому часу виконання сценарію. Остання умова гарантує, що цикли між двома чи більше компонентами не призведуть до тупикової ситуації.

1.3 Адитивні моделі надійності

Цей клас моделей не враховує чітко архітектуру програмного забезпечення. Швидше, вони зосереджені на оцінці загальної надійності програми за допомогою даних про відмови компонента. Слід зазначити, що ці

моделі враховують зростання надійності програмного забезпечення. Вони називаються адитивними моделями, оскільки за припущення, що надійність компонента може бути змодельована за допомогою розширеної моделі неоднорідного процесу Пуассона (NHPP), інтенсивність відмов системи може бути виражена як сума інтенсивностей відмов компонентів.

1.3.1 Модель Ксі-Войліна

Ця модель розглядає програмну систему, що складається з *n* компонентів, які можна розробляти паралельно та тестувати незалежно [12]. Якщо припустити, що відмова компонента є відмовою системи, з точки зору надійності систему можна розглядати як послідовну систему. Якщо надійність компонентів моделюється за допомогою розширеної моделі неоднорідного процесу Пуассона NHPP з інтенсивністю відмов $\lambda_i(t)$, то інтенсивність відмов системи становить $\lambda_S(t) = \lambda_1(t) + \lambda_2(t) + \cdots + \lambda_n(t)$, тобто очікувана сукупна кількість відмов системи до часу t , відома як функція середнього значення, визначається як

$$\mu_S(t) = \sum_{i=1}^{n} \mu_i(t) = \int_0^t \sum_{i=1}^{n} \lambda_i(\tau)\, \mathrm{d}\tau. \tag{1.10}$$

Коли використовується ця адитивна модель, найбільш нагальною проблемою є те, що час запуску може бути не однаковим для всіх компонентів, тобто деякі компоненти можуть бути введені в систему пізніше в процесі тестування. У цьому випадку час відповідної функції середнього значення та інтенсивність відмови повинні бути належним чином скориговані з урахуванням різних вихідних точок для різних компонентів.

1.3.2 Модель Еверетта

Цей підхід розглядає програмне забезпечення, створене з компонентів, і вирішує проблему оцінки надійності окремих компонентів. Надійність кожного компонента аналізується за допомогою моделі розширеного часу виконання (EET), параметри якої можна визначити безпосередньо з властивостей

програмного забезпечення та інформації про те, як тест-кейси та оперативне використання навантажують кожен компонент. Властивості, які впливають на надійність компонентів, поділяються на статичні та динамічні. Крім того, необхідно охарактеризувати, як використання навантажує кожен компонент як під час періодів тестування, так і протягом періодів, коли програмне забезпечення знаходиться в робочому стані. Для програмного забезпечення стресор компонента – це кількість часу обробки, витраченого на компонент. Тому цей підхід вимагає відстеження сукупного часу обробки для кожного компонента.

Комбінована модель для загальної системи накладає надійність компонентів. Коли основними моделями EET для компонентів є моделі NHPP, кумулятивна кількість відмов і функції інтенсивності відмов для суперпозиції таких моделей є лише сумою відповідних функцій для кожного компонента.

2 МОДЕЛЬ НАДІЙНОСТІ ПРОГРАМНОГО ПРОДУКТУ НА ОСНОВІ ОБ'ЄДНАННЯ МОДЕЛЕЙ НАДІЙНОСТІ

В зазначених вище роботах про підходи до надійності програмного забезпечення, заснованих на архітектурі, було запропоновано велику кількість варіантів, переважно спеціальними методами. Вони часто мали тенденцію "затіняти" об'єднуючі структурні властивості, загальні для багатьох таких варіантів. Математична обробка цих моделей стає очевидною, коли демонструється їх спільна структура. У цьому розділі обговорюється співвідношення та уніфікація існуючих моделей надійності програмного забезпечення на основі архітектури.

2.1 Об'єднання моделей на основі стану

Моделі надійності програмного забезпечення на основі архітектури припускають, що компоненти виходять з ладу незалежно один від одного і що збій компонента в кінцевому підсумку призведе до збою системи. У надійності апаратної системи зазвичай вважається, що всі компоненти безперервно активні, що відповідає звичайному рівнянню для надійності послідовної системи

$$R = \prod_{i=1}^{n} R_i. \tag{2.1}$$

Ключове питання щодо надійності програмної системи полягає в тому, як врахувати використання компонента, тобто навантаження, яке накладається на кожен компонент під час виконання.

По-перше, розглянемо завершену програму, архітектура якої описана за допомогою DTMC. Поглинаючий DTMC може надати середню кількість разів виконання кожного компонента, позначену V_i, як міру використання компонента. Тоді $R_i^{V_i}$ можна розглядати як еквівалентну надійність компонента *i* що враховує використання компонентів. Таким чином, надійність системи можна виразити як

$$R = \prod_{i=1}^{n} R_i^{V_i}, \tag{2.2}$$

що є ієрархічним підходом до оцінки надійності.

Надійність компонентів оцінюється як ймовірність того, що протягом часу виконання компонента *i* не відбудеться жодної відмови

$$R_i = \int_0^{\infty} \mathrm{e}^{-\lambda_i t} g_i(t)\,\mathrm{d}t, \tag{2.3}$$

і вбудований DTMC SMP, який описує архітектуру програмного забезпечення, забезпечує середню кількість разів V_i виконання кожного компонента. Таким чином, ієрархічна обробка цієї моделі зводиться до ієрархічної обробки моделі Чонга [14].

Ієрархічний підхід оцінює надійність компонентів, враховуючи залежні від часу частоти відмов $\lambda_\mathrm{i}(t)$ і використання компонентів через кумулятивний очікуваний час, витрачений у компоненті на виконання $V_\mathrm{i}\, t_\mathrm{i}$:

$$R_i = \mathrm{e}^{-\int_0^{V_i t_i} \lambda_i(t)\,\mathrm{d}t}. \tag{2.4}$$

Тоді надійність системи $R = \prod_{i=1}^{n} R_i$.

Як зазначалося раніше, спільною рисою наведених вище моделей є те, що запуск програмного забезпечення, який відповідає завершенню виконання програмного додатка, може бути чітко ідентифікований. Таким чином, релевантною мірою є надійність *R* одиничного виконання програмного додатку, тобто так звана ймовірність відмови на вимогу $1 - \mathrm{R}$.

Тепер розглянемо безперервно запущені програми. У [15] генеруюча функція моменту кількості відмов $N(t)$ виведена зі складеної моделі та зроблено висновок, що аналітично результати не можуть бути отримані. Однак асимптотичний аналіз моделі привів до процесу Пуассона з параметром

$$\lambda_S = \sum_i \pi_i \left[\lambda_i + \sum_{j \neq i} q_{ij} \nu_{ij} \right], \tag{2.5}$$

де $\pi = [\ \pi_i\ ]$ – вектор рівноваги незвідного СТМС із матрицею швидкостей переходів $Q = [q_{ij}]$, тобто $\pi Q = 0$. Термін у дужках можна інтерпретувати як частоту відмов компонента *i* що включає інтенсивність відмов внутрішнього компонента λ_i і частота відмов через збої, які виникають протягом взаємодії з іншими компоненти. Таким чином, $q_{ij}\ \nu_{ij}$ дає частоту відмови інтерфейсу , що характеризує відмови, які виникають під час взаємодії компонентів *i* та *j* . Отже, підсумовування по *j* дає загальну частоту відмов інтерфейсу зі стану *i*. Оскільки ймовірність усталеного стану π_i являє собою середню частку часу, витраченого в стані *i* при відсутності збою, тому термін $\pi_i \Lambda_i$ може розглядається як еквівалентна частота відмов компонента *i* що враховує використання компоненту.

Модель, представлена в [16], є окремим випадком [15], який розглядає лише відмови компонентів. У цьому випадку еквівалентну частоту відмов

$$\lambda_S = \sum_i \pi_i \lambda_i \tag{2.6}$$

отримано за допомогою ієрархічного методу на основі імовірнісних аргументів.

Ці результати заслуговують на кілька коментарів. По-перше, асимптотичний процес Пуассона можна розглядати як суперпозицію процесів Пуассона з параметрами $\pi_i \Lambda_i$, що тісно пов’язані з підходом, використаним в адитивних моделях.

Крім того, розглянемо ймовірність того, що не буде відмови до часу *t*, тобто надійність системи

$$R(t) = \mathrm{e}^{-\lambda_S t} = \mathrm{e}^{-\sum_{i=1}^{n} \pi_i \Lambda_i t} = \prod_{i=1}^{n} \mathrm{e}^{-\Lambda_i \pi_i t}, \tag{2.7}$$

π_i являє собою частку часу, проведеного в стані i за відсутності відмови; таким чином, $\pi_i t$ представляє кумулятивний час виконання, витрачений на компонент i до часу t. Для апаратних систем вважається, що всі компоненти безперервно активні, що відповідає тому, що всі π_i дорівнюють 1. З точки зору надійності, це призводить до звичайного рівняння для послідовної системи з кількома підсистемами з експоненціально розподіленим часом до відмови.

По суті, наведені вище моделі є окремими випадками універсального Марківського процесу, який еквівалентний пакетному Марківському процесу (Batch Markovian Arrival Processes – BMAP) [17]. Оскільки це багатий клас точкових процесів, для яких можна отримати багато відповідних розподілів ймовірностей, моментних і кореляційних формул у формах, які піддаються обчисленню, вони широко використовувалися для моделювання процесів надходження в теорії масового обслуговування.

Побудова універсального Марківського процесу полягає в наступному. Розглянемо CTMC з $n + 1$ станом з n перехідними станами та одним станом поглинання. Нескінченно малий генератор Q^*, отриманий після видалення стану поглинання, є незвідним і описує CTMC, отриманий скиданням вихідного CTMC миттєво, використовуючи ті самі початкові ймовірності щоразу, коли відбувається поглинання в стан $n + 1$. Часи поглинання (і відновлення) формують процес оновлення з базовим розподілом фазового типу.

Точковий процес будується, починаючи з цього фазового процесу оновлення як субстрату. Розглянуто три типи епох надходження, які пов'язані з еволюцією фазового процесу відновлення. Існують Пуассонівські надходження з довільним розподілом розміру партії під час перебування в CTMC, які регулюють процес оновлення. Швидкість надходження процесу Пуассона та розподіл розміру партії можуть залежати від стану CTMC. Базовий CTMC може змінювати стани з відповідним оновленням або без нього. Щоразу, коли процес змінює стани, надходить пакет (розмір пакета може бути 0), де розмір пакета може залежати від станів до та після зміни, а також від того, чи відбулося оновлення чи ні.

Нехай $N(t)$ і $X(t)$ позначають, відповідно, кількість надходжень у $(0, t]$ і стан незвідного СТМС, визначеного Q^* в момент часу t. Процес $\{N(t), X(t)), t \geq 0\}$ є СТМС із простором станів $\{k \geq 0\} \times \{1, \ldots, n\}$. Тісний зв'язок BMAP із кінцевим СТМС призводить до матрично-аналітичного підходу, який суттєво зменшує обчислювальну складність алгоритмічного рішення.

Таким чином, у [15] розглядаються лише надходження з розміром партії 1, що призводить до особливого випадку BMAP, так званого Марківського процесу надходження (MAP) [17]. Модель, представлена в [16], яка розглядає лише Пуассонівські надходження під час перебування в станах Марківського процесу, є подвійно стохастичним процесом Пуассона, відомим як модульований процес Пуассона за Марковим (MMPP) [17].

Модель Леду [18], яка узагальнює [15], є складовою моделлю. Метод розв'язання, який використовується в цій роботі, базується на матрично-аналітичному підході. Таким чином, багато відповідних величин, пов'язаних із процесом точки відмови, виводяться у формах, які піддаються обчисленню. До них відноситься функція розподілу кількості відмов $N(t)$ в $(0, t]$, час очікування до першої відмови, тобто надійність $R(t) = \Pr\{T > t\} = \Pr\{N(t) = 0\}$, очікувана кількість відмов $E[N(t)]$ у $(0, t]$ та функція інтенсивності відмов $h(t)$.

Добре відомо, що час між надходженнями BMAP є фазовим, а розподіл фазового типу з незвідною матрицею є асимптотично експоненціальним із власним значенням максимальної дійсної частини λ_S матриці як параметра. Скаляр λ_S, відомий як фундаментальна швидкість надходження для точкового процесу [17], можна інтерпретувати як асимптотичну частоту відмов системи, коли $t \to \infty$. Іншими словами, коли $t \to \infty$, функція інтенсивності відмов $h(t)$ стає постійним значенням $h(\infty) = \lambda_S$. Використовуючи цей підхід у [18], наближення Пуассона порівнюється з перехідним рішенням для точкового процесу складеної моделі. Якщо розглядати лише вторинні відмови, асимптотична частота відмов зменшується до отриманої в [15], яка є параметром розподілу Пуассона, що апроксимує розподіл для кількості відмов $N(t)$ у $(0, t]$.

Тепер розглянемо модель [19], яка передбачає, що архітектуру програмного забезпечення можна описати за допомогою SMP. Точний неасимптотичний аналіз складеної моделі стає дуже складним. Однак, за припущення, що між послідовними збоями програми відбуватиметься багато обмінів керуванням, процес точки збою асимптотично є процесом Пуассона, як зазначено в [19]. Еквівалентну частоту відмов асимптотичного процесу Пуассона можна отримати за допомогою ієрархічного методу на основі ймовірнісних аргументів таким чином.

Рішення моделі архітектури можна отримати за допомогою відомої граничної поведінки SMP [20]. Таким чином, модель архітектури забезпечує граничну частку часу, витраченого на кожен модуль i і гранична кількість передачі керування за одиницю часу від i до j, позначені як a_i та b_{ij} відповідно, як показники використання компонентів та інтерфейсів. Потім поведінка відмов накладається на рішення з моделі архітектури, щоб отримати еквівалентну частоту відмов

$$\lambda_S = \sum_i a_i \lambda_i + \sum_{i,j} b_{ij} v_{ij}. \tag{2.8}$$

2.2 Узагальнення моделей на основі шляху виконання

Подібно до моделей на основі стану, моделі на основі шляхів явно розглядають архітектуру програмного забезпечення та припускають, що компоненти виходять з ладу незалежно. Однак метод поєднання архітектури програмного забезпечення з поведінкою відмов компонентів та інтерфейсів не є аналітичним. По-перше, послідовність компонентів, що виконуються вздовж кожного шляху, отримується або експериментально шляхом тестування [10], або алгоритмічно [11], а надійність шляху отримуємо множенням надійності компонентів уздовж цього шляху. Потім надійність системи оцінюється шляхом усереднення надійності шляху по всіх шляхах. Таким чином, ці моделі

враховують використання кожного компонента на кожному шляху, а також між різними шляхами.

Різниця між підходом на основі стану та підходом на основі шляху стає очевидною, коли граф потоку керування програми містить цикли. Моделі на основі стану аналітично враховують нескінченну кількість шляхів через петлі, які можуть існувати; у випадку моделей на основі шляху кількість шляхів або обмежується тими, що спостерігаються експериментально під час тестування [10], або глибинний обхід кожного шляху завершується за допомогою середнього часу виконання програми [11].

2.3 Узагальнення адитивних моделей

Адитивні моделі [12], [13] розглядають фазу тестування програмного забезпечення та припускають, що надійність кожного компонента може бути змодельована за допомогою NHPP. Їх можна розглядати як суперпозицію NHPP. Добре відомо, що якщо процеси відмови компонентів моделюються за допомогою NHPP, то процес відмови системи також є NHPP із сукупною кількістю відмов і функцією інтенсивності відмов, які є сумами відповідних функцій для кожного компонента. Однак моделі відрізняються наступним.

У [12] час відповідних функцій налаштовано відповідним чином, щоб враховувати час, коли різні компоненти включені в систему. Ця модель не враховує архітектуру програмного забезпечення, тобто різне використання компонентів.

Модель, використана для фіксації зростання надійності компонента в [13], включає інформацію про відносне навантаження на кожен компонент як параметр моделі. Крім того, кумулятивний час виконання, витрачений на кожен компонент, отриманий під час тестування, використовується для обчислення сукупної кількості відмов і функції інтенсивності відмов для окремих компонентів. Таким чином, ця модель розглядає використання компонентів програмного забезпечення, тобто неявно архітектуру програмного забезпечення.

Іншими словами, цей підхід тісно пов’язаний з підходом для оцінки надійності компонентів, представленим у [7], який явно враховує архітектуру програмного забезпечення.

3 ОБМЕЖЕННЯ МОДЕЛЕЙ ОЦІНКИ НАДІЙНОСТ НА ОСНОВІ ПРОГРАМНОЇ АРХІТЕКТУРИ

Перевага підходу, заснованого на архітектурі, очевидна в контексті програмної системи, яка розроблена як гетерогенна суміш нещодавно розроблених, повторно використаних і COTS компонентів. Однак цей підхід, здається, ускладнює моделі та збір даних. Багато питань, пов'язаних із підходом до кількісної оцінки програмних систем на основі архітектури, досі залишаються без відповіді, і необхідні додаткові дослідження в цій галузі, зокрема, коли йдеться про проблеми, зазначені нижче.

3.1 Рівень декомпозиції системи на компоненти

Існує компроміс у визначенні компонентів. Занадто багато малих компонентів може призвести до великого простору станів, що може створити труднощі у вимірюваннях, параметризації та моделі. З іншого боку, занадто мала кількість компонентів може спричинити втрату різниці в тому, як різні компоненти сприяють системним збоям. Рівень декомпозиції явно залежить від компромісу між кількістю компонентів, їхньою складністю та наявною інформацією про кожен компонент.

Численні статті стосуються моделювання надійності на основі архітектури, але поки опубліковано лише кілька експериментальних досліджень. У цих експериментах вибір рівня розкладання дуже різноманітний. Наприклад, у [21] розглянуто дві декомпозиції програмної системи: декомпозиція на чотири групи відповідно до основних функцій системи та декомпозиція на чотири класи відповідно до наслідків відмови. Експеримент, описаний у [10], використовує декомпозицію на основі стандартних компонентів утиліти UNIX grep. У [7] декомпозиція інструменту, що використовується для вирішення стохастичних моделей надійності та продуктивності, базується на рівні файлу; кожен із файлів розглядається як окремий компонент. У прикладі, представленому в [12],

використовується програма, яка моделює поведінку черг очікування, що складається з шести повторно використовуваних компонентів..

3.2 Оцінка надійності окремих компонентів

Більшість документів про оцінку надійності на основі архітектури припускають, що надійності компонентів доступні, тобто ігнорують питання про те, як їх можна визначити. Оцінка надійності окремих компонентів явно залежить від таких факторів, як наявність чи відсутність коду компонента, наскільки добре компонент було протестовано та чи є він повторно використаним чи новим компонентом.

Звичайно можна припустити, що модель надійність може бути застосовується до кожного програмного компоненту, який використовує дані про збій компонента. Наприклад, в [21] застосували модель, засновану на неоднорідному Пуассонівському процесі, щоб оцінити стаціонарну інтенсивність відмов компонентів. Це дослідження було проведено на основі інформації про збої в системі, які спостерігалися протягом 3 років, включаючи етапи перевірки та експлуатації. В [7] використовували розширену модель NHPP, запропонувавши метод визначення залежної від часу інтенсивності відмови компонента на основі вимірювання покриття блоку під час тестування. В [13] визначили рекомендації для оцінки надійності нещодавно розроблених компонентів на основі моделі, параметри якої пов’язані зі статичними та динамічними властивостями компонента та використанням кожного компонента.

Однак, це не завжди можливо використовувати моделі надійності програмних для оцінювання надійності окремих компонентів. Труднощі можуть виникнути через брак даних про несправності. Прогнози на основі даних про збої становлять інтерес тільки якщо їх є, щоби були статистично репрезентативними. Подальші труднощі пов’язані з тим фактом, що базові припущення моделей надійності програмного забезпечення, такі як випадкове тестування, що

виконується відповідно до робочого профілю, і незалежність послідовних прогонів тестування можуть бути серйозно порушені під час фази модульного тестування [5].

Інший клас моделей оцінює надійність шляхом явного розгляду виконання без збоїв [22]. Такі моделі можна використовувати для оцінки надійності програмного забезпечення на основі результатів тестування, проведеного на етапі валідації. На цьому етапі програмне забезпечення не змінюється, і ці моделі можна назвати моделями стабільної надійності. У цьому контексті тестування – це не розробка для виявлення помилок, а незалежна оцінка виконання програмного забезпечення в типовому робочому середовищі.

Проблема, яка виникає з цими моделями, полягає в кількості виконань, необхідних для оцінки рівнів надійності, які відповідають розумним очікуванням. Імовірнісна оцінка надійності програмного забезпечення до рівнів, необхідних, наприклад, у критично важливих для безпеки програмах, становить в термінах частоти збоїв 10^{-9} год$^{-1}$ або 10^{-5} год$^{-1}$, а ймовірність відмови наразі визначити неможливо, оскільки випадкове тестування може вимагати десятиліть тестування, перш ніж воно зможе встановити прийнятну статистичну впевненість в оцінці надійності. Загальноприйнято, що практична поточна межа в оцінці інтенсивності відмов перед експлуатаційним використанням лежить у діапазоні від 10^{-2} до 10^{-4} год$^{-1}$ [23].

Було запропоновано кілька інших методів оцінки надійності компонента. В [24] досліджували метод визначення якості компонентів COTS за допомогою тестування компонентів методом чорної скриньки та впровадження помилок на системному рівні. Однак, складність методів, заснованих на несправностях, полягає в тому, що вони настільки потужні, наскільки потужні класи несправностей, які вони моделюють.

3.3 Оцінка надійності інтерфейсів

Виявляється, що доступної інформації про збої інтерфейсів мало, окрім загальної згоди, що вони існують окремо від збоїв компонентів, які виявляються під час модульного тестування. Інтерфейсом між двома компонентами може бути інший компонент, набір глобальних змінних, набір параметрів, набір файлів або будь-яка їх комбінація.

На практиці необхідно оцінити надійність кожного інтерфейсу. Коли інтерфейс є програмою, будь-який із згаданих вище методів можна використовувати для оцінки їх надійності. Однак, коли інтерфейс складається з таких елементів, як глобальні змінні, параметри та файли, незрозуміло, як оцінити надійність. Але, наприклад, метод інтеграційного тестування, запропонований [25] виглядає перспективним для оцінки надійності інтерфейсу.

3.4 Оцінка ймовірностей переходу

У підході, заснованому на архітектурі, необхідно моделювати взаємодію всіх компонентів. У добре розробленій системі, взаємодія між компонентами є обмеженою. в більшість випадків різноманітна взаємодія між компонентами буде неможливою, тобто матриця ймовірностей переходу буде розрідженою.

На ранніх етапах життєвого циклу програмного забезпечення кожен компонент можна перевірити, щоб визначити, з якими компонентами він не може взаємодіяти. Інші ненульові ймовірності переходу можна отримати, аналізуючи структуру програми та використовуючи відомі оперативні профілі. На етапі проектування, перед фактичним етапом розробки та інтеграції, це можна зробити за допомогою моделювання. Під час фази інтеграції, коли нові дані стають доступними, оцінки потрібно оновлювати, тим самим покращуючи прогнози. На щастя, хороші оцінки можна отримати дуже рано завдяки високій швидкості обміну елементами керування. Оцінка ймовірностей переходу має бути максимально точною, оскільки помилка в цьому процесі безумовно вплине на надійність системи та чутливість компонентів.

Наступні дві роботи демонструють два різні підходи до оцінки ймовірностей переходу. У [11] спочатку отримано оцінки ймовірностей виконання різних сценаріїв на основі робочого профілю. Потім за допомогою аналізу сценаріїв розраховуються ймовірності переходу. У [7] ймовірності переходу обчислюються на основі підрахунків виконання, отриманих із файлів трасування, отриманих під час тестування.

3.5 Залежності відмов між компонентами та інтерфейсами

Існуючі моделі без винятку припускають, що процеси відмови, пов’язані між різними компонентами, є взаємно незалежними. При розгляді помилки інтерфейсу також вважаються взаємно незалежними та незалежними від процесів відмови компонента.

Проте, якщо на поведінку компонента при відмові будь-яким чином впливає попередній компонент, що виконується, або інтерфейс між ними, ці припущення більше не є прийнятними. Цю залежність можна назвати міжкомпонентною залежністю [26]. Одним із способів зіткнутися з міжкомпонентною залежністю у випадку моделей простору станів було б розглянути ланцюг Маркова вищого порядку, як обговорювалося вище.

Внутрішньокомпонентна залежність може виникнути, наприклад, коли компонент викликається більше одного разу в циклі іншим компонентом. Спроба розглянути внутрішньокомпонентну залежність була зроблена в [10].

3.6 Аналіз чутливості моделі до надійності компонента

Оскільки підхід, заснований на архітектурі, дозволяє зрозуміти внесок кожного компонента в ненадійність усієї системи, його можна використовувати для виконання аналізу чутливості. Хоча ця перевага визнається багатьма дослідниками [7], [10], метод обчислення чутливості надійності системи щодо надійності заданого компонента розроблено лише для моделі, представленої в [14]. Цей підхід до аналізу чутливості використовується для розподілу надійності

кожного компонента, який базується на цільовій надійності для всієї системи та чутливості системи до компонента.

Аналіз чутливості може допомогти визначити критичні компоненти, які мають найбільший вплив на надійність і продуктивність системи. Цю інформацію можна використовувати для планування та сертифікації на різних етапах життєвого циклу програмного забезпечення.

3.7 Врахування різних архітектурних стилів

Сучасні програмні додатки набагато складніші, часто працюють на різних апаратних платформах, під різними операційними системами та на територіально розподілених машинах. Як наслідок, спостерігається поширення дослідницької діяльності, зосередженої на архітектурі програмного забезпечення.

Підґрунтя для вивчення архітектури програмного забезпечення було закладено в [26]. Автори визначили корисний набір архітектурних стилів програмного забезпечення. Архітектурний стиль визначається набором типів компонентів, топологічним розташуванням цих компонентів, що вказує на їхні взаємозв’язки, і набором механізмів взаємодії, які визначають, як вони координуються. Наприклад, деякі системи мають стиль каналів і фільтрів (наприклад, UNIX), тоді як інші є об'єктно-орієнтованими, орієнтованими на дані (наприклад, системи транзакційних баз даних) або системи подій (наприклад, об'єкти транслюють події через брокер запитів).

Ці стилі відрізняються як видами компонентів, які вони використовують, так і способом взаємодії цих компонентів один з одним. Компоненти визначають обчислення на рівні програми та сховища даних системи. Приклади включають клієнти, сервери, фільтри, бази даних і об’єкти. Взаємодія між цими компонентами може бути такою ж простою, як виклики процедур, канали та трансляція подій, або набагато складнішою, включаючи протоколи клієнт-сервер, протоколи доступу до бази даних тощо.

Оскільки архітектурне рішення зазвичай приймаються на ранніх етапах життєвого циклу, то їх найважче змінити а, отже, є найбільш критичними. Таким чином, архітектура програмного забезпечення відіграє головну роль у визначенні якості системи та її придатності до супроводу та підтримки. Необхідно розробити стандартизовані архітектурні стилі разом із методами їх кількісної оцінки. Це допоможе дизайнерам вибрати архітектурний стиль, який найбільше підходить для даної системи.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Синдром професійного вигорання в ІТ

У сучасних дослідженнях наголошується, що вигорання виникає у фахівця в контексті роботи і має негативні наслідки як для нього самого, так і для організації в цілому, психічного благополуччя всіх тих, з ким він взаємодіє в процесі професійної діяльності (клієнтів, учнів, пацієнтів, колег) [27]. Подальше вивчення даного феномену важливе для дослідження закономірностей і механізмів професійного становлення механізмів професійної адаптації; виявлення зв’язку між професійним стресом і соматичними розладами, та встановлення характеру цього зв’язку тощо.

Дослідниками з часу появи перших даних про вигорання як професійного феномену неухильно наголошується його поширеність. Якщо раніше він досліджувався у представників професій «людина – людина» – вчителів, викладачів коледжів, лікарів і медичних сестер, менеджерів, торгівельних агентів, соціальних працівників, представників інших професій, пов’язаних з безперервними комунікаціями з великим потоком людей, то тепер коло професіоналів, схильних до професійного вигорання, розширилося. Дослідниками отримані дані про наявність вигорання у представників управлінських структур і у представників професії «людина – знак», «людина – машина» – бухгалтерів, програмістів, пілотів, водіїв, операторів та ін. У зв’язку з цим, дослідження феномену професійного вигорання виступає одному з пріоритетних завдань сучасної науки [28].

Якісний і кількісний склад вигорання залежить від змісту професійної діяльності: у професіях «суб’єкт-суб’єктної» сфери вигорання має трикомпонентну структуру, а в «суб’єкт-об’єктній» сфері – близьку до двофакторної структури. Психічне вигорання набуває статусу загально професійного феномену і розглядається як компонент категоріального апарату психологічної науки. Даний феномен виявляється в різних сферах особистості –

емоційній, когнітивній, мотиваційній, сфері відношення людини до роботи – і найменше розроблена проблема впливу вигорання на поза професійні сфери життя людини [30].

Аналіз сучасних досліджень показав, що «психічне вигорання» ширше поняття, ніж «професійне вигорання», оскільки воно може бути викликане різними причинами – особовими, родинними, професійними. Психічне вигорання може виявлятися не лише в професійній сфері, а, наприклад,у сім'ї, в учбовій діяльності. Професійне вигорання становить один з варіантів психічного вигорання: перше частіше використовується в контексті трудової діяльності. Поняття «емоційне вигорання» вживається в тому разі, коли акцент робиться на емоційній складовій вигорання і йдеться про те, що вигоряє лише емоційна сфера психіки. Термін психічне вигорання використовується в разі, коли акцентується увага на тому, що вигорання зачіпає емоційну, інтелектуальну та мотиваційно-споживацьку сфери, а також вольовий механізм. На сучасному етапі вигорання вивчається одночасно як соціальний, професійний та особистісний феномени.

Отже, в контексті теоретичного вивчення проблем професійного вигорання виділяють основні етапи. Перший етап присвячений виявленню й опису окремих випадків вигорання. На другому етапі відбувається розуміння та узагальнення феномену вигорання. Третій період відзначається методологічною фазою дослідження вигорання. Зміст четвертого етапу полягає в аналітичному вивченні узагальненого концепту вигорання. Сучасний етап вивчення питання професійного вигорання характеризується появою нових напрямів досліджень, зокрема він характеризується різними сенсами, які зв'язуються з феноменом вигорання.

4.2 Створення і функціонування системи моніторингу довкілля з метою інтеграції екологічних інформаційних систем, що охоплюють певні території

Законом України "Про охорону навколишнього природного середовища" (ст.20, 22) передбачено створення державної системи моніторингу довкілля (ДСМД) та проведення спостережень за станом навколишнього природного середовища, рівнем його забруднення. Виконання цих функцій покладено на Мінприроди та інші центральні органи виконавчої влади, які є суб'єктами державної системи моніторингу довкілля, а також підприємства, установи та організації, діяльність яких призводить або може призвести до погіршення стану довкілля.

Основні принципи функціонування ДСМД визначені у постанови Кабінету Міністрів України від 30.03.1998 № 391 „Про затвердження Положення про державну систему моніторингу довкілля".

На даний час, у державній системі моніторингу довкілля (далі – ДСМД) функції і задачі спостережень та інформаційного забезпечення виконують 8 суб`єктів системи моніторингу: Мінприроди, МНС, МОЗ, Мінагрополітики, Мінжитлокомунгосп, Держводгосп, Держкомлісгосп, Держкомзем.

Кожний із суб`єктів ДСМД здійснює моніторинг тих об`єктів довкілля, що визначаються Положенням про державну систему моніторингу довкілля та порядками і положеннями про державний моніторинг окремих складових довкілля. Основні нормативні акти, що регламентують моніторинг об'єктів довкілля:

– постанова Кабінету Міністрів України від 09.03.1999 № 343 «Про затвердження Порядку організації та проведення моніторингу в галузі охорони атмосферного повітря»;

– постанова Кабінету Міністрів України від 20.07.1996 № 815 «Про затвердження Порядку здійснення державного моніторингу вод»;

– постанова Кабінету Міністрів України від 20.08.1993 № 661 «Про затвердження Положення про моніторинг земель»;

– постанова Кабінету Міністрів України від 26.02.2004 № 51 «Про затвердження Положення про моніторинг ґрунтів на землях сільськогосподарського призначення».

З метою координації діяльності міністерств та відомств, визначення основних принципів державної політики з питань розвитку системи моніторингу навколишнього середовища, забезпечення її функціонування на основі єдиного нормативно-методологічного забезпечення постановою Кабінету Міністрів України від 17.11.2001 № 1551 утворено Міжвідомчу комісію з питань моніторингу довкілля. Мінприроди здійснюється організаційно-технічне забезпечення роботи комісії та її профільних секцій.

Існуюча система моніторингу довкілля базується на виконанні розподілених функцій її суб'єктами і складається з підпорядкованих їм підсистем. Кожна підсистема на рівні окремих суб'єктів системи моніторингу має свою структурно-організаційну, науково-методичну та технічну бази.

Функціонування ДСМД здійснюється на трьох рівнях, що розподіляються за територіальним принципом:

– загальнодержавний рівень, що охоплює пріоритетні напрямки та завдання моніторингу в масштабах всієї країни;

– регіональний рівень, що охоплює пріоритетні напрямки та завдання в масштабах територіального регіону;

– локальний рівень, що охоплює пріоритетні напрямки та завдання моніторингу в масштабах окремих територій з підвищеним антропогенним навантаженням.

Моніторинг якості повітря.

Державною гідрометеорологічною службою (МНС) здійснюються спостереження за забрудненням атмосферного повітря у 53 містах України на 162 стаціонарних, двох маршрутних постах спостережень та двох станціях

транскордонного переносу. Ведуться спостереження за хімічним складом атмосферних опадів та за кислотністю опадів.

Програма обов`язкового моніторингу якості атмосферного повітря включає сім забруднюючих речовин: пил, двоокис азоту (NO_2), двоокис сірки (SO_2), оксид вуглецю, формальдегід (H_2CO), свинець та бензпірен. Деякі станції здійснюють спостереження за додатковими забруднюючими речовинами. Проводиться аналіз наявності забруднюючих речовин в опадах та сніговому покриві.

Державна екологічна інспекція (Мінприроди) здійснює вибірковий відбір проб на джерелах викидів. Вимірюється понад 65 параметрів.

Санітарно-епідеміологічна служба (МОЗ) здійснює спостереження за якістю атмосферного повітря у житловій та рекреаційній зонах, зокрема поблизу основних доріг, санітарно-захисних зон та житлових будинків, на території шкіл, дошкільних установ та медичних закладів в містах та в робочий зоні. Крім того, здійснюється аналіз якості повітря у житловій зоні за скаргами мешканців.

Моніторинг стану вод суші.

Державна гідрометеорологічна служба (МНС) проводить моніторинг гідрохімічного стану вод на 151 водному об`єкті, а також здійснює гідробіологічні спостереження на 45 водних об`єктах. Отримуються дані по 46 параметрах, що дають можливість оцінити хімічний склад вод, біогенні параметри, наявність зважених часток та органічних речовин, основних забруднюючих речовин, важких металів та пестицидів. На 8 водних об`єктах проводяться спостереження за хронічною токсичністю води. Визначаються показники радіоактивного забруднення поверхневих вод.

Державна екологічна інспекція (Мінприроди) відбирає проби води та отримує дані по 60 вимірюваних параметрах.

Державний комітет по водному господарству проводить моніторинг річок, водосховищ, каналів, зрошувальних систем і водойм у межах водогосподарських систем комплексного призначення, систем водопостачання, транскордонних водотоків та водойм у зонах впливу атомних електростанцій.

Контроль якості води за фізичними та хімічними показниками здійснюється на 72 водосховищах, 164 річках, 14 зрошувальних системах, 1 лимані та 5 каналах комплексного призначення. Крім того, у рамках радіаційного моніторингу вод водогосподарськими організаціями здійснюється контроль вмісту радіонуклідів у поверхневих водах.

Санітарно-епідеміологічна служба (МОЗ) проводить спостереження за джерелами централізованого та децентралізованого постачання питної води, а також місцями відпочинку вздовж річок та водосховищ.

Підприємствами Державної геологічної служби (Мінприроди) здійснюється моніторинг стану підземних вод. У місцях моніторингу проводиться оцінка рівня залягання підземних вод (наявність), їх природного геохімічного складу. Проводяться визначення 22 параметрів, в тому числі концентрації важких металів та пестицидів.

Санітарно-епідеміологічна служба (МОЗ) здійснює хімічний аналіз підземних вод, які призначаються для питного споживання.

Моніторинг прибережних вод.

Державна гідрометеорологічна служба (МНС) управляє мережею моніторингу стану прибережних вод, яка складається з станцій моніторингу у місцях скиду стічних вод та науково-дослідних станцій, що розташовані на прибережних територіях Чорного та Азовського морів. На існуючих станціях проводяться вимірювання від 16 до 26 гідрохімічних параметрів вод та донних відкладів.

Державні інспекції охорони Чорного та Азовського морів (Мінприроди) мають власні системи спостережень. До їх повноважень відносяться щомісячні відбори проб та аналіз впливу джерел забруднення, які розташовані на узбережжі; моніторинг скидів з кораблів; забруднення від діяльності з пошуку та видобування нафти, газу і будівельних матеріалів на морському шельфі; нагляд за використанням живих ресурсів моря.

Державна санітарно-епідеміологічна служба (МОЗ) здійснює моніторинг якості морської води в зонах рекреаційного та оздоровчого водокористування.

Моніторинг стану ґрунтів.

Державна гідрометеорологічна служба (МНС) здійснює моніторинг забруднення ґрунтів сільськогосподарських земель пестицидами та важкими металами у населених пунктах. Проби відбираються раз у п`ять років, проби на важкі метали у містах Костянтинівка та Маріуполь відбираються щороку.

Державна екологічна інспекція (Мінприроди) здійснює відбір проб на промислових майданчиках в межах країни. Загальна кількість параметрів, що вимірюються – 27.

Установи МОЗ здійснюють моніторинг стану ґрунтів на територіях їх можливого негативного впливу на здоров`я населення. Найбільше охоплені території вирощення сільськогосподарської продукції, території в місцях застосування пестицидів, ґрунти в зоні житлових масивів, дитячих майданчиків та закладів. Досліджуються проби ґрунту в місцях зберігання токсичних відходів на території підприємств та поза територією підприємств у місцях їх складування або захоронення.

Мінагрополітики здійснює спостереження за ґрунтами сільськогосподарського використання. Здійснюються радіологічні, агрохімічні та токсикологічні визначення, залишкова кількість пестицидів, агрохімікатів і важких металів.

Моніторинг показників біологічного різноманіття.

Через обмежене бюджетне фінансування моніторинг здійснюється тільки за видами, які представляють промисловий інтерес (дерева, риба, дичина).

Підприємства Держкомлісгоспу проводять моніторинг лісової рослинності у 24 областях країни. Здійснюється оцінка біомаси, пошкодження її біотичними та абіотичними чинниками; мисливської фауни, біорізноманіття; радіологічні визначення.

Деякі дослідження здійснюються через надання міжнародної допомоги, або в рамках міжнародних програм.

Моніторинг радіаційного випромінювання.

Державна гідрометеорологічна служба (МНС) здійснює спостереження за радіоактивним забрудненням атмосфери шляхом щоденних замірів доз гамма-радіаційної експозиції (ГРЕ), осідання радіоактивних частинок з атмосфери та вмісту радіоактивного аерозолю в повітрі. Здійснюються заміри радіоактивного забруднення поверхневих вод на 8 водних об`єктах. Поблизу атомних електростанцій Державна гідрометеорологічна служба здійснює заміри радіоактивного забруднення поверхневих вод цезієм-137 у та забруднення ґрунтів.

Лабораторії моніторингу Мінагрополітики проводять контроль у місцях концентрації радіоактивних речовин у ґрунтах та харчових продуктах. МНС здійснює моніторинг доз ГРЕ на 10 автоматизованих пунктах поблизу атомних електростанцій. У межах 30-кілометрової зони навколо Чорнобильської АЕС (зони відчуження), МНС здійснює спостереження за концентрацією радіонуклідів; радіонуклідами в атмосферних опадах, а також концентрацією «гарячих» частинок у повітрі. Міжнародна радіоекологічна лабораторія Чорнобильського центру атомної безпеки, радіоактивних відходів та радіоекології у Славутичі, здійснює моніторинг впливу радіації на біоту у зоні відчуження.

Суб`єктами ДСМД створені, або розробляються відомчі бази даних моніторингової інформації. Існуюча система інформаційної взаємодії відомчих підсистем моніторингу довкілля передбачає обмін інформацією на загальнодержавному та регіональному рівнях. Організаційна інтеграція суб`єктів моніторингу довкілля на всіх рівнях здійснюється Мінприроди та його територіальними органами.

Для упорядкування процесу обміну інформацією за показниками та термінами надання екологічної інформації між Мінприроди та суб’єктами ДСМД укладено двохсторонні угоди про співробітництво у сфері моніторингу навколишнього природного середовища, до яких розроблені відповідні регламенти обміну екологічною інформацією.

Оперативна моніторингова інформація передається територіальними органами суб'єктів ДСМД до регіональних центрів моніторингу довкілля, або державних управлінь охорони навколишнього природного середовища в регіонах.

Узагальнена аналітична інформація надається міністерствами та відомствами-суб'єктами ДСМД Мінприроди.

Отримані дані передаються до Інформаційно – аналітичного центру Мінприроди та накопичується у банках екологічних даних.

На основі отриманої щомісячної та щоквартальної інформації Мінприроди видається інформаційно – аналітичний огляд „Стан довкілля в України ”, який розповсюджується серед заінтересованих користувачів.

Функціонування Інформаційно-аналітичного центру Мінприроди забезпечує інформаційний обмін з регіональними центрами моніторингу довкілля, суб’єктами державної системи моніторингу довкілля, створення уніфікованого банку екологічних даних, проведення комплексного аналізу стану довкілля, тощо.

Постановою Кабінету Міністрів України від 05.12.2007 №1376 затверджено Державну цільову екологічну програму проведення моніторингу навколишнього природного середовища. З метою забезпечення інтеграції інформаційних ресурсів суб`єктів системи моніторингу довкілля передбачено створення та забезпечення функціонування єдиної автоматизованої підсистеми збору, оброблення, аналізу і збереження даних та інформації, отриманих в результаті здійснення моніторингу.

В межах Державної цільової екологічної програми проведення моніторингу навколишнього природного середовища, у тому числі, передбачено розширення мережі автоматизованих постів спостережень за забрудненням атмосферного повітря в екологічно небезпечних містах.

ВИСНОВКИ

У цій роботі виконано огляд підходів до оцінки якості надійності програмної системи на основі її архітектури. Використовуючи методи подання опису архітектури програмного забезпечення та поєднуючи їх з описом відмов компонентів та інтерфейсів, ідентифіковано три класи моделей.

Моделі на основі стану описують архітектуру програмного забезпечення різними типами ланцюгів Маркова: DTMC, CTMC або SMP і аналітично оцінюють надійність програмного забезпечення, поєднуючи архітектуру з поведінкою відмов.

Моделі класу на основі шляхів виконання програми обчислюють надійність програмного забезпечення з урахуванням можливих графів виконання програми або експериментально шляхом тестування, або алгоритмічно.

Третій підхід до моделювання, який називається адитивним, не розглядає архітектуру явно. Він скоріше зосереджений на оцінці залежної від часу інтенсивності збою програми за допомогою даних про збій компонента.

Далі, в роботі виконано огляд цих моделей з метою їх уніфікації між собою та з існуючими архітектурними поданнями.

Далі обговорено обмеження моделей надійності програм, заснованих на архітектурі, щоб дати розуміння корисності таких моделей, що може допомогти у визначенні напрямків для майбутніх досліджень.

Мета в цій роботі полягала в тому, щоб представити стан досліджень і практики в підході, заснованому на архітектурі, до кількісної оцінки надійності програмного забезпечення і вказати, що знання, які вже існують у різних дослідницьких спільнотах, повинні використовуватися для покращення процесів розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Farr, William. "Software reliability modeling survey." Handbook of software reliability engineering 222 (1996): 71-117.

2. Gokhale, Swapna S., Michael R. Lyu, and Kishor S. Trivedi. "Reliability simulation of component-based software systems." Proceedings Ninth International Symposium on Software Reliability Engineering. IEEE, 1998.

3. Xie, Min. Software reliability modelling. Vol. 1. World Scientific, 1991.

4. Ramamoorthy, C. V., and Farokh B. Bastani. "Software reliability – Status and perspectives." IEEE Transactions on Software Engineering 4 (1982): 354-371.

5. Gokhale, Swapna S., and Kishor S. Trivedi. Structure-based software reliability prediction. North Carolina State University. Center for Advanced Computing and Communication, 1998.

6. Laprie, Jean-Claude. "Dependability evaluation of software systems in operation." IEEE Transactions on software engineering 6 (1984): 701-714.

7. Gokhale, Swapna S., et al. "An analytical approach to architecture-based software performance and reliability prediction." Performance Evaluation 58.4 (2004): 391-412.

8. Horgan, Joseph R., and Saul London. "A data flow coverage testing tool for C." Proceedings of the Second Symposium on Assessment of Quality Software Development Tools. IEEE Computer Society, 1992.

9. Shooman, Martin L. "Structural models for software reliability prediction." Proceedings of the 2nd international conference on Software engineering. 1976.

10. Krishnamurthy, Saileshwar, and Aditya P. Mathur. "On the estimation of reliability of a software system using reliabilities of its components." Proceedings The Eighth International Symposium on Software Reliability Engineering. IEEE, 1997.

11. Yacoub, Sherif, Bojan Cukic, and Hany H. Ammar. "A scenario-based reliability analysis approach for component-based software." IEEE transactions on reliability 53.4 (2004): 465-480.

12. Xie, Min, and Claes Wohlin. "An additive reliability model for the analysis of modular software failure data." Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95. IEEE, 1995.

13. Everett, William W. "Software component reliability analysis." Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99. IEEE, 1999.

14. Cheung, Roger C. "A user-oriented software reliability model." IEEE transactions on Software Engineering 2 (1980): 118-125.

15. Littlewood, Bev. "A reliability model for systems with Markov structure." Journal of the Royal Statistical Society Series C: Applied Statistics 24.2 (1975): 172-177.

16. Laprie, Jean-Claude. "Dependability evaluation of software systems in operation." IEEE Transactions on software engineering 6 (1984): 701-714.

17. Lucantoni, David M. "New results on the single server queue with a batch Markovian arrival process." Communications in Statistics. Stochastic Models 7.1 (1991): 1-46.

18. Ledoux, James. "Availability modeling of modular software." IEEE Transactions on Reliability 48.2 (1999): 159-168.

19. Littlewood, Bev. "Software reliability model for modular program structure." IEEE Transactions on Reliability 28.3 (1979): 241-246.

20. Kulkarni, Vidyadhar G. Modeling and analysis of stochastic systems. Crc Press, 2016.

21. Kanoun, Karama, and Thierry Sabourin. "Software dependability of a telephone switching system." 17th International Symposium on Fault Tolerant Computing, FTCS-17, pp. 236-241, 1987.

22. Littlewood, Bev, and David Wright. "Some conservative stopping rules for the operational testing of safety critical software." IEEE Transactions on software Engineering 23.11 (1997): 673-683.

23. Laprie, J-C. "Dependability of computer systems: concepts, limits, improvements." Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95. IEEE, 1995.

24. Voas, Jeffrey M. "Certifying off-the-shelf software components." Computer 31.6 (1998): 53-59.

25. Delamaro, Marcio Eduardo, Jose Carlos Maldonado, and Aditya P. Mathur. "Integration testing using interface mutation." Proceedings of ISSRE'96: 7th International Symposium on Software Reliability Engineering. IEEE, 1996.

26. Shaw, Mary, and David Garlan. Software architecture: perspectives on an emerging discipline. Prentice-Hall, Inc., 1996.

27. Вовк О. В. Особливості синдрому професійного вигорання в працівників сфери інформаційних технологій. [Електронний ресурс]. – Режим доступу: http://maup.com.ua/assets/files/psihologz/2019-1/02.pdf

28. Назарук Н. Каузально-телеологічний формат профілактики «професійного вигорання» вчителя / Н. Назарук // Психологія особистості. 2012.- No 1 (3). – С. 119–128.

29. Вдосконалення охорони праці в ІТ-індустрії. // Харківський національний дорожний університет. [Електронний ресурс]. – Режим доступу: https://www.khadi.kharkov.ua/fileadmin/P_vcheniy_secretar/%D0%9E%D0%A5%D0%9E%D0%A0%D0%9E%D0%9D%D0%90_%D0%9F%D0%A0%D0%90%D0%A6%D0%86/R_IT-INDUSTRIA.pdf

30. Батлук В.А., Гогіташвілі Г.Г. та ін. Охорона праці в галузі телекомунікацій. – Львів: Афіша, 2003. – 320 с.

ДОДАТКИ

УДК 004.41

А. Вивюрка, Л. Мариненко, О. Нога, Б. Хоміцький, Т. Ланевич
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСІВ CI/CD В ГНУЧКИХ ТЕХНОЛОГІЯХ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

A. Vyviurka, L. Marynenko, B. Khomitskyi, O. Noha, T Lanevych

RESEARCH OF THE EFFICIENCY OF CI/CD PROCESSES IN AGILE SOFTWARE DEVELOPMENT TECHNOLOGIES

Процеси безперервної інтеграції (Continuous Integration – CI) та безперервного розгортання (Continuous Deployment – CD) були популяризовані наприкінці 90-х як частина eXtreme Programming [1], що загалом належить до гнучких (Agile) технологій розробки [2]. Поточні IT-стратегії значною мірою залежать від здатності компаній впроваджувати зміни або виправлення гнучким чином та безпомилково. Потреба в автоматизації різко зростає з розвитком технологій, оскільки останні десятиліття методи водоспадної розробки програмного забезпечення були замінені гнучкими підходами до розробки програмного забезпечення.

В даній доповіді пропонується фреймворк для дослідження проєктів з розробки програмних продуктів з метою оцінювання ефективності процесів розробки з впровадженням CI/CD. Для досягнення цієї мети пропонується оцінювати процеси за такими характеристиками.

1. Можливість впровадження технологій Agile в процеси тестування.
2. Ефективність комунікації в команді та в рамках проєкту між командами.
3. Збільшення продуктивності роботи розробників.
4. Збільшення передбачуваності проєкту.

Можливість впровадження гнучких технологій тестування завдяки гнучкій інтеграції розглядалась в роботі [3]. Безперервна інтеграція вважається важливою для підтримки гнучкого тестування. Вважається, що гнучке тестування включає в себе такі практики, як визначені клієнтом приймальні (acceptance) тести, автоматизація цих тестів і їх виконання в наборі регресійних тестів щонайменше щодня, а також розробка модульних (unit) тестів для всього нового коду під час кожної ітерації (спринту) з наступним виконанням цих модульних тестів з кожною збіркою. Задача дослідження полягатиме виявленні того, чи використовуються на проєкті модульні тести з їх автоматичним виконанням.

Задача оцінки ефективності спілкування полягає в тому, щоби при дослідженні проєкту інформація про дефекти автоматично формувалась на надсилалась усім зацікавленим сторонам включно з розробником. Тобто даний пункт досліджень логічно побудований на наявності автоматичних модульних тестів та необхідності застосування процесів інтеграції. Тоді при невдалому завершенні будь-якого тесту формуватиметься відповідний звіт та надсилатиметься тому, хто створив частину програмного коду, котра спричинила збій. Тобто проєктний менеджер чи тестувальник не затрачатиме час на формування таких звітів і їх розсилку. Особливо це актуально, коли над проєктом працює декілька команд, і коли часка затрат комунікацію та узгодження процесів розробки між командами значно зростає.

Безперервна інтеграція сприяє збільшенню продуктивності розробників завдяки очевидній можливості паралельної розробки, тобто роботи декількох програмістів над одним і тим же програмним кодом. Як наслідок, зменшується час на компіляцію та тестування кожним розробником та надає йому можливість впровадити більше нових властивостей (вимог) та змін. Крім того, в роботі [4] розраховано чистий прибуток від

впровадження безперервної інтеграції шляхом вимірювання часу, зекономленого завдяки тому, що розробники не здійснювали компіляцію вручну та не проводили тестування перед кожною операцією коміту, оскільки ці процеси виконувались автоматично в рамках безперервної інтеграції. Це означає, що основна перевага постійної інтеграції полягає в економії часу для розробників.

Можливість точнішої оцінки прогресу проєкту та термінів його завершення відносно проміжних етапів та кінцевого терміну відповідно розглядалась, наприклад, в роботах [5] та [6]. Актуальна інформація про покриття продукту тестами, відсоток успішно виконаних та кількість дефектів дозволяє оперативно вживати управлінські заходи, як то зміна пріоритетів, перерозподіл завдань, залучення необхідних ресурсів, та дотримуватись графіку виконання проєкту. Зокрема, проєктний менеджер матиме змогу виявляти проблеми на ранніх етапах, запроваджувати раннє тестування нефункціональних вимог.

Таким чином, впровадивши даний фреймворк, можна досягнути позитивних результатів завдяки впровадженню процесів CI/CD у Agile-проєктах. Ми дійшли висновку, що існує не одна, а декілька переваг постійної інтеграції. Для кожного проєкту можна виявити взаємозв'язок між безперервною інтеграцією та гнучкими методами тестування автоматизованих тестів клієнта та написанням модульних тестів у поєднанні з новим початковим кодом. При цьому недослідженими залишаються наступні питання. Наприклад, чи підтримує безперервна інтеграція практику модульного тестування, чи, навпаки, вони підтримують один одного. Також важко відокремити вплив безперервної інтеграції на практику автоматизованого α- та β-тестування від контекстних факторів (таких як організаційна структура, культура та доступність клієнтів).

Питання підвищення продуктивності розглянуто в аспекті збільшення кількості завдань, виконаних розробником. Проте його продуктивність в контексті обсягу написаного програмного коду залишається незмінною. Тобто економія часу не зовсім очевидна в кожному проєкті. Звичайно, ефекти, які обговорюються в цій статті, не становлять вичерпний перелік переваг постійної інтеграції. Однак можна стверджувати, що краще розуміння потенційних відмінностей у реалізаціях CI/CD-процесів та їхніх наслідків може допомогти проєктам сформувати свою безперервну інтеграцію таким чином, щоб оптимізувати переваги, яких вони прагнуть досягнути.

Література

1. Beck, K. (2000). Extreme Programming Explained Addison-Wesley. Reading.

2. Agile alliance: manifesto for agile software development. Available at: http:// agilemanifesto.org [retrieved 10.10.2022].

3. Stolberg, S. (2009, August). Enabling agile testing through continuous integration. In 2009 agile conference (pp. 369-374). IEEE.

4. Miller, A. (2008, August). A hundred days of continuous integration. In Agile 2008 conference (pp. 289-293). IEEE.

5. Goodman, D., & Elbaz, M. (2008, August). "It's Not the Pants, it's the People in the Pants" Learnings from the Gap Agile Transformation What Worked, How We Did it, and What Still Puzzles Us. In Agile 2008 conference (pp. 112-115). IEEE.

6. Liu, H., Li, Z., Zhu, J., Tan, H., & Huang, H. (2009, July). A unified test framework for continuous integration testing of SOA solutions. In 2009 IEEE International Conference on Web Services (pp. 880-887). IEEE.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ
2023

УДК 004.41

Володимир Семенюк, Андрій Вивюркка, Олександр Нога, Богдан Хоміцький, Олександр Кучма
Тернопільський національний технічний університет імені Івана Пулюя

ПОРІВНЯЛЬНЕ ОЦІНЮВАННЯ ПРОГРАМНИХ АРХІТЕКТУР

Volodymyr Semeniuk, Andrii Vyviurka, Oleksandr Noha, Bohdan Khomitskyi, Oleksandr Kuchma

COMPARATIVE EVALUATION OF SOFTWARE ARCHITECTURES

Якість архітектурного рішення оцінюється за сукупністю критеріїв, і його вибір завжди є компромісом, оскільки для заданого набору функціональних вимог та вимог якості не існує єдиного найкращого рішення і покращення одних показників веде до погіршення інших та навпаки. Тому при виборі архітектурного рішення для ПС необхідно використовувати методи багатокритеріального оцінювання з врахуванням конфліктів між показниками якості та пошуком компромісів.

Існує раннє і пізнє оцінювання архітектур. Раннє оцінювання базується на досвіді розробників та логічному обгрунтуванні, оскільки відсутні артефакти, які дають змогу імітувати роботу ПС. Методи, які реалізують раннє оцінювання, базуються на сценаріях. До цих методів належать наступні: SAAM і ATAM [1].На основі пріоритетів зацікавлених сторін визначаються критерії якості. Для перевірки задоволення кожного атрибута якості розробляється сценарій і проводиться оцінка рівня задоволення даного атрибуту варіантом архітектури.

Метод ATAM подібний до SAAM, але в ньому на основі аналізу сценаріїв для відібраних архітектур проводиться оцінка ризиків задоволення атрибутів якості. Оцінку ризиків проводить група експертів, яка також ранжує альтернативні варіанти за рівнем ризику і визначає так звані точки чутливості у компонентах чи зв'язках архітектури, також аналізуються компроміси між критеріями якості.

Для обгрунтованого вибору рішення в методі SAAM/ATAM вибрані альтернативні архітектури аналізуються на ефективність витрат методом CBAM [2]. Цей метод забезпечує економічний аналіз ПС, яка базується на вибраних в попередніх методах варіантах архітектури та сценаріях моделювання. Експерти призначають оцінки критеріям якості в балах від 1 до 100 і ранжують архітектури за значенням, яке ці архітектурні рішення забезпечують для атрибуту якості. Оцінка кожного варіанта архітектури обчислюється за формулою:

$$B(A_i) = \sum_{j=\overline{1,K}} \left(Cont_{i,j} \cdot Q_j \right) \; i = \overline{1,n}. \qquad (1)$$

Тут $Cont_{ij}$ – вага і-ї архітектури відносно j-го атрибута;

Q_j – пріоритет j-го атрибута.

Метод забезпечує оцінку витрат на реалізацію кожної альтернативи і дає можливість обчислити показник бажаності як відношення прибутку до витрат. На основі отриманих даних проводиться вибір кращого рішення.

Спільним недоліком розглянутих методів є послідовне оцінювання архітектури по одному критерію якості що викликає необхідність кожного разу розробляти новий сценарій і проводити експертне оцінювання ризиків, що робить процес вибору трудомістким і неформалізованим. Тут також неможливо отримати порівняльні оцінки для множини альтернатив.

Подальші дослідження в цій області показали ефективність методу аналізу ієрархій при рішенні цих задач. Однією з перших публікацій по застосуванню MAI до

оцінювання архітектур ПС по множині показників якості є робота [2]. В ній приведений покроковий алгоритм вирішення задачі оцінювання множини альтернативних архітектур по сукупності показників якості та вибору найкращої архітектури ПС. Застосування методу продемонстроване на конкретному прикладі.

Для подолання недоліку MAI, пов'язаного з неузгодженістю матриці парних порівнянь при великій кількості альтернатив (n≥7) в роботі пропонується коригувати елементи матриці. А це приводить до неповного використання та перекручування експертних даних, що зменшує правдивість отриманих результатів.. В роботі [3] для коректного використання MAI у випадку великої кількості альтернатив (n>9) було використано модифікований метод аналізу ієрархій.

В методі MAI використовується порівняльне оцінювання альтернатив по їх реалізації атрибутів якості. Він дає змогу визначити відносні ваги альтернатив по кожному атрибуту якості і проранжувати їх. За призначеними, зацікавленими сторонами, пріоритетами атрибутів якості обчислюється їх усереднене значення і визначаються ваги альтернатив відносно сукупності атрибутів якості.

Отримані відносні оцінки альтернатив можуть використовуватись для аналізу конфліктів між атрибутами якості і пошуку компромісного рішення.

Пропонований підхід базується на використанні методу аналізу ієрархій з оптимізаційним алгоритмом визначення ваг альтернатив, що дає змогу розширити межі застосування методу на більшу кількість порівнюваних альтернатив (n>9). Також слід дослідити чутливість рішення до зміни пріоритетів критеріїв якості і проаналізувати конфлікти і компроміси між критеріями якості.

Для розширення меж коректного застосування MAI слід застосувати оптимізаційний метод обчислення (визначення) ваг альтернатив, який базується на моделі мінімізації неузгодженості матриці парних порівнянь [3].

Література

1. Kazman, R. ATAM: Method for Architecture Evaluation [Text] / Rick Kazman, Mark Klein, Paul Clements. – Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, August 2000. – CMU/SEI-2000-TR-004, ADA377385. – 83 p

2. Kazman, R. Quantifying the costs and benefits of architectural decision [Text]/ Kazman, R., Asundi, J., and Klein // Proceedings of the 23rd International Conference on Software Engineering (ICSE), 2001. – pp. 297–306

3. Harchenko Alexandr, Bodnarchuk Ihor, Halay Iryna. Stability of the Solutions of the Optimization Problem of Software Systems Architecture // Proceeding of VIIth International Scientific and Technical Conference CSIT 2012. pp. 47–48, Lviv, 2012