МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
“Інженерія якості програмного забезпечення”
для студентів ОР «Магістр»
спеціальності 121 ”Інженерія програмного забезпечення”
усіх форм навчання

Частина 2.
Моделювання програмного забезпечення для превентивного обслуговування цифрових двійників

2023 р.

Методичні вказівки до виконання лабораторних робіт з дисципліни ”Інженерія якості програмного забезпечення” для студентів ОР «Магістр» спеціальності 121 ”Інженерія програмного забезпечення” усіх форм навчання. Частина 2. Моделювання програмного забезпечення для превентивного обслуговування цифрових двійників / Укл.: Г.В. Табунщик, І.Я. Зеленьова, Т.В. Голуб.- Запоріжжя: НУ«Запорізька політехніка», 2023. - 31с.

Укладачі: Г.В. Табунщик, професор, к.т.н.
І.Я. Зеленьова, доцент, к.т.н.
Т.В. Голуб, доцент, к.т.н.

Рецензент Р.К. Кудерметов, доцент, к.т.н.

Відповідальний за випуск: І.Я. Зеленьова, доцент, к.т.н.

Матеріали розроблено за підтримки міжнародного проєкту 619034-EPP-1-2020-1-UA-EPPKA2-CBHE-JP “Cross-domain competences for healthy and safe work in the 21st century” [WORK4CE], що фінансується Європейською Комісією. Підтримка Європейською комісією випуску цієї публікації не означає схвалення змісту, який відображає лише думки авторів, і Комісія не може нести відповідальність за будь-яке використання інформації, що міс- титься в ній.

Затверджено
на засіданні кафедри ПЗ
Протокол № 12 від 09.06.2023

Затверджено
на засіданні НМК КНТ
Протокол № 3 від 27.10.2023

ЗМІСТ

ПЕРЕДМОВА

Експлуатація та технічне обслуговування розумного міста стають дедалі складнішими, а громадяни очікують від міської інфраструктури все більшої функціональності. Щоб задовольнити ці очікування, міста повинні використовувати нові технології для підвищення ефективності та рентабельності своєї діяльності та технічного обслуговування. Однією з таких технологій є цифровий двійник, який забезпечує цифрове представлення фізичних систем і процесів.

Концепція цифрових двійників походить зі світу виробництва. Воно вказує на процес дзеркального відображення або "двійникування" з двонаправленими інформаційними потоками між двома об'єктами, що дозволяє виконувати певні операції, наприклад, тестування, оптимізацію та симуляцію процесів. Одне з найпопулярніших визначень належить аерокосмічній галузі, яка описує цифровий двійник як "інтегровану багатомасштабну симуляцію фізичного об'єкта, що відображає життя його відповідного двійника". Зі зростанням популярності цифрових двійників у міському контексті нещодавні дослідження спробували досягти консенсусу щодо інтерпретації цифрових двійників для міст. Деякі дослідження припускають, що цифрові двійники міст повинні уможливлювати динамічний аналіз, що виходить за межі 3D-візуалізації, наприклад, відображати просторово-часові зміни та моделювати динамічні міські сценарії. Тому, оскільки визначення різняться, для цілей цієї статті було визначено, що цифрові двійники міст повинні 1) базуватися на детальних семантичних 3D-моделях міста, 2) надавати дані в режимі, близькому до реального часу, 3) уможливлювати різноманітні операції, наприклад, аналіз, моделювання та прогнозування різних сценаріїв до того, як вони будуть реалізовані в реальності, та 4) враховувати соціальні та економічні функції у забудованому середовищі, наприклад, уможливлюючи партисипативний процес, залучаючи людей як сенсорів для вивчення місцевого контексту.

Життєвий цикл цифрових двійників класифікується як різні фази їхнього життя у виробництві, а саме: процес проектування – експлуатація – обслуговування. Віртуальна модель отримує інформацію про продукт для моделювання та перевірки сценаріїв подій, а потім надсилає зворотний зв'язок фізичному об'єкту для

оптимізації дизайну, наприклад, повідомлення про помилки або налаштування деяких деталей. Обмін інформацією між фізичними та віртуальними об'єктами утворює циклічність зв'язку.

Ще одним важливим аспектом цифрових двійників міст є їхня потреба реагувати на зміни, що відбуваються майже в режимі реального часу. Це вимагає великих баз даних, зворотного зв'язку та високочастотного інформаційного потоку протягом усього життєвого циклу. Існує кілька проблем, пов'язаних із впровадженням цифрових близнюків для функціонування та обслуговування розумного міста.

Однією з найбільших проблем є відсутність взаємодії між різними системами та джерелами даних. Для того, щоб ефективно використовувати цифрові двійники, місто має мати можливість доступу та інтеграції даних із різних джерел, таких як департаменти громадських робіт, комунальні компанії та інші зовнішні джерела. Без уніфікованої платформи для полегшення обміну даними та доступу до них міста не зможуть використовувати весь потенціал цифрових близнюків.

Нарешті, міста повинні мати необхідні ресурси для підтримки цифрових двійників. Це включає персонал з необхідними навичками та досвідом, а також необхідне обладнання та програмне забезпечення. Без належних ресурсів міста можуть бути не в змозі повністю скористатися перевагами цифрових близнюків.

Отже, цифровим двійником найчастіше називають програмне уявлення фізичного активу, системи або процесу, призначене для виявлення, запобігання, прогнозування та оптимізації за допомогою аналітики в реальному часі з метою збільшення цінності бізнесу. Оскільки цифрові двійники – це моделі реальних об'єктів, розроблені у вигляді програмного опису або програмно-апаратного комплексу, актуальним питанням є забезпечення надійності програмного забезпечення. Аналізу цього питання присвячені лабораторні роботи даного циклу.

Матеріали розроблено за підтримки міжнародного проєкту 619034-EPP-1-2020-1-UA-EPPKA2-CBHE-JP "Cross-domain competences for healthy and safe work in the 21st century" [WORK4CE], що фінансується Європейською Комісією.

1 ЛАБОРАТОРНА РОБОТА №1. ПРОГНОЗУВАННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА РАННІХ ЕТАПАХ РОЗРОБКИ

1.1 Основні показники надійності невідновлювальних систем

Надійність обчислювальних систем слід розглядати спільно з надійністю програмного забезпечення як єдину систему. Виконання завдань системи в однаковій мірі залежить від надійної роботи апаратних засобів і надійної роботи програмних засобів.

Надійність програмного забезпечення визначається як властивість програми, яка забезпечує виконання заданих функцій при заданих умовах обмежень і в заданому обчислювальному середовищі.

Відмова програмного забезпечення (ПЗ) може наступити в наступних випадках:

– несанкціоновані дії апаратних засобів вносять зміни в роботу програми;

– невірні дії оператора;

– некоректність програми.

Відмова ПЗ, зумовлена невідповідністю програмного забезпечення поставленим завданням, становить найбільший інтерес.

Невідповідність може виникнути з двох причин:

– розробник програми порушив специфікацію – технічні вимоги ПО;

– складена специфікація не відображає дійсні вимоги завдання.

Невідповідність специфікації, допущена розробником, зумовлюється складністю програмних систем, де важко виявити окремі помилки, вони не доступні для огляду і можуть залишатися тривалий час нерозкритими.

Невідповідність з другої причини виникає через різноманіття факторів, що впливають на роботу програми в ході експлуатації і визначених на момент розробки. Вони з'ясовуються поступово в ході управління процесом або натурним моделюванням.

Таким чином, програми за ступенем відповідності точності специфікації можна класифікувати на:

– програми, функції яких повністю визначаються специфікацією;

– програми, функції яких коригуються за результатами обчислень (моделювання, випробування);

– програми, що діють в обчислювальному середовищі, яке постійно змінюється (операційні системи, управління ресурсами та ін.).

Коректність програми – це відповідність її технічним умовам (ТУ).

Оскільки специфікація технічних вимог не завжди відповідає фактично необхідним вимогам до програми, можливі непередбачувані ситуації:

– відмова або вихід програми з нормального циклу роботи в область непередбачуваних дій;

– навпаки, некоректна програма за збігом обставин працює надійно.

Характерна особливість відмов програми – прихованість помилок, тому що вони проявляються лише при окремих, рідкісних комбінаціях, і тому виявляються не відразу.

Забезпечення певного рівня надійності програм можливе в результаті застосування ряду заходів, спрямованих на підвищення надійності, найбільш загальні з яких наведені в таблиці 1.1.

Таблиця 1.1 – Можливі відмови й заходи підвищення надійності по блоках

Функціональний блок ПЗ	Можливі відмови	Заходи підвищення надійності
1	2	3
Підключення бібліотек / модулів	– відсутність в системі необхідної бібліотеки; – несумісність бібліотек, що підключаються; – синтаксичні помилки.	– завантаження необхідних зовнішніх ресурсів в програмне оточення при першому зверненні; – використання компілятора для налагодження.

Продовження таблиці 1.1

1	2	3
Введення вихідних даних	– вихід з допустимого діапазону даних; – невірний формат даних; – синтаксичні помилки.	– перевірка вихідних даних на допустимість (повинні бути виділені всі групи допустимих / недопустимих даних); – перевірка формату даних, при необхідності перетворення даних до необхідного формату. – використання компілятора для налагодження.
Функціональні блоки, що містять обробку вихідних даних	– логічні помилки: – відсутня ініціалізація змінних; – невірно задані умови виходу з циклу; – невірно індексований цикл; – неправильно заданий перехід по гілках алгоритму; – не врахована можливість появи негативних значень змінних, малих і великих величин та ін.; – синтаксичні помилки.	– побудова логічних ланцюжків і тестових завдань для кожної гілки алгоритму; – акуратний і логічний стиль написання, як можна менше використання «трюків програмістів»; – аналіз коду за допомогою спеціальн. аналізаторів; – покрокове трасування та ін. ; – використання компілятора для налагодження.

Продовження таблиці 1.1

1	2	3
Вивід результату	– відсутність необхідного пристрою виведення; – невірний формат даних, що виводяться; – синтаксичні помилки.	– резервне копіювання результату в створюваний файл; – перевірка формату даних, при необхідності перетворення даних до необхідного формату.

Теорія надійності апаратури частково може бути застосована до проблеми **надійності програмного забезпечення** при врахуванні **основних відмінностей:**

– програмне забезпечення не має старіючих через зношування або втому елементів;

– більше способів контролю і шляхів їх організації;

– більше об'єктів для контролю;

– менше стандартних елементів, ніж в апаратурі;

– кількість програмної документації за обсягом більше, ніж в апаратурі;

– внести зміну в програму достатньо просто, але чи буде вона коректна - питання.

По **типах програми (таблиця 1.2)** поділяються на:

— керуючі – програми, які виконують функцію керування об’єктами поза програмним кодом.

— програми виводу – програми, які спеціалізуються тільки на безпосередньому виведенні даних, що надходять.

— програми-обчислювачі – програми, що виконують довільні розрахунки, спрямовані на отримання результату.

— програми настроювання – програми, що виконують безпосередню модернізацію і зміну зовнішніх об'єктів.

— службові – програми, що виконують внутрішнє управління прихованими ресурсами системи.

Прогнозування надійності ПЗ на ранніх етапах виконується на основі статистичних даних, отриманих при аналізі прогнозування очікуваного числа помилок в програмі.

Оцінка очікуваної кількості помилок N в програмі виражається лінійно:

$$N = \sum_{j=1}^{r} a_j z_j$$

де

z_j – j-ий параметр програми,

a_j – коефіцієнт (див. таблиці 1.2а та 1.2б), r – число істотних параметрів.

Таблиця 1.2а – Коефіцієнти впливу параметрів

Типи програм	**Коефіцієнти**				
	a_1	a_2	a_3	a_4	a_5
Керуючі	0,003	0.0405	0.171	0.43	0.036
Виводу	0,002	0	0.780	0	0
Обчислювач	0,002	0.024	0.412	0	0.416
Настроювання	0,003	0.001	0.592	0	0
Службові	2.09	0.0374	0.628	0.102	0

Таблиця 1.2б – Коефіцієнти впливу параметрів

Типи програм	**Коефіцієнти**			
	a_6	a_7	a_8	a_9
Керуючі	0	0	-0.00186	0,003
Виводу	0	0.0218	0	0,002
Обчислювач	0.016	0.082	-0.0026	0,002
Настроювання	0	0.0625	-0.0036	0,003
Службові	0	0	-0.0133	2.09

За результатами проведених статистичних досліджень, в якості **характерних переметрів z_j, які впливають на ступінь надійності програми**, обрані наступні:

– **z_1** – складність умовних операторів IF, розташованих в циклах і рекурсіях;

$$z_1 = \sum n_i w_i,$$

де **n_i – число умовних операторів і-го рівня вкладеності циклів**, **w_i – ваговий коефіцієнт**.

$$w_i = 4^{i-1} \cdot (3/4Q - 1),$$

де **Q – найвищий рівень вкладеності циклів в рамках програми (кількість циклів, вкладених один в одного)**;

– **z_2** – загальне число розгалужень, розташованих не в циклах ;

– **z_3** – загальне число зв'язків з прикладними програмами (виклик додаткових прикладних програм в коді, підключення користувальницьких файлів (наприклад, #include “my.h”));

– **z_4** – загальне число зв'язків з системними програмами (підключення бібліотечних файлів (наприклад, #include<iostream>));

– **z_5** – число операцій введення-виведення (дії з будь-яким типом введення-виведення на екран, з клавіатури, в/з файлу), одна – команда введення/виведення незалежно від складності структури цієї команди; кількість ітерацій циклів при розрахунках не враховується;

– **z_6** – число обчислювальних операторів (арифметичні, логічні, булеві операції, інкремент, декремент (що не відносяться до опису циклів));

– **z_7** – число операторів обробки даних (>, <, = - операції порівняння, присвоєння, ініціалізація і оголошення змінних);

– **z_8**- загальне число коментарів (сумарна кількість рядків коментарів з урахуванням доповнюючих код коментарів);

– **z_9** – складність операторів циклів, які не містять вкладених умовних операторів IF;

$$z_9 = \sum n_i w_i,$$

де n_i – число операторів циклу і-го рівня вкладеності, w_i – ваговий коефіцієнт (таблиця 1.4).

$$w_i = 4^{i-1} \cdot (3/4Q - 1),$$

нагадаємо, що Q – найвищий рівень вкладеності циклів в рамках програми (кількість циклів, вкладених один в одного);

Якщо число очікуваних помилок оцінено, то інтенсивність відмов програми оцінюється виразом:

$$\lambda_{пр} = \bar{\gamma} N / t_{ріш}$$

де $t_{ріш}$ – середній час одноразового проходження програми;

$\bar{\gamma}$ – середня ймовірність того, що помилка виявиться при одноразовому проходженні.

Обираємо значення $\bar{\gamma}$ із діапазону 0,001…0,01. **Приймаємо** $\bar{\gamma} = 0{,}005$ **.**

Імовірність виникнення помилок при виконанні програми, яку аналізуємо, в конкретний період часу:

$$P(t) = \exp(-\lambda_{пр} \cdot t)$$

де t – період часу, який аналізуємо.

Тоді ця ймовірність виникнення помилок за період всього виконання програми визначається як:

$$P(t) = \exp(-\lambda_{пр} \cdot t_{ріш})$$

1.2 Приклад розрахунку надійності ПЗ

Для виконання лабораторної роботи необхідно вибрати довільний код програми і проаналізувати його з розбиттям на компоненти згідно з переліком параметрів, що визначають потенційну кількість помилок.

Приклад аналізу програмного коду наведено на лістингу А.1 додатку А. Кольором позначено приналежність операції певному критерію.

1. Визначимося з вибором програми і найвищим рівнем вкладеності циклів в рамках програми (Q = 3).

2. Порахуємо кількість необхідних параметрів для розрахунку оцінки очікування числа помилок в програмі і занесемо в таблицю 1.3.

Таблиця 1.3 – Кількісна оцінка параметрів впливу

Параметри програми для оцінки z_i	**Кількість**
Умовні оператори IF, розташовані в циклах і рекурсіях **першого рівня** вкладеності (z1.1)	1
Умовні оператори IF, розташованих в циклах і рекурсіях **другого рівня** вкладеності (для z1.2)	1
Умовні оператори IF, розташованих в циклах і рекурсіях **третього рівня** вкладеності (для z1.3)	0
Умовні оператори IF, розташованих в циклах і рекурсіях **четвертого рівня** вкладеності (для z1.4)	0
Загальна кількість розгалужень, розташованих не в циклах (для z2)	2
Загальна кількість зв'язків з прикладними програмами (для z3)	0
Загальна кількість зв'язків з системними програмами (для z4)	5
Число операцій введення-виведення (для z5)	
Число обчислювальних операторів (для z6)	
Число операторів обробки даних (для z7)	53
Загальна кількість коментарів (для z8)	9
Оператори циклу **першого рівня вкладення**, які не містять IF (для z9.1)	12
Оператори циклу **другого рівня вкладення**, які не містять IF (для z9.2)	

Оператори циклу **третього рівня вкладення**, які не містять IF (для z9.3)	
Оператори циклу **четвертого рівня** вкладення, які не містять IF (для z9.4)	0

3. Проведемо розрахунки для **z1** і **z9** (зверніть увагу, що ці коефіцієнти ***мають по декілька можливих значень***, залежно від рівня вкладеності) на основі таблиць вагових коефіцієнтів (таблиця 1.4) і кількості параметрів програми. Враховуємо, що рівень складності Q = 3, тому обираємо відповідну строку в табл.1.4

Таблиця 1.4 – Таблиця вагових коефіцієнтів

Ваговий коефіцієнт	**w1**	**w2**	**w3**	**w4**
Q1	-0,25			
Q2	0,5	2		
Q3	1,25	5	20	
Q4	2	8	32	128

Для даного програмного коду Q = 3.

$$z_1 = \sum n_i w_i = 1*1,25+1*5+0*20 = 6,25$$
$$z_9 = \sum n_i w_i = 12*1,25+8*5+1*20 = 75.$$

Решта значень z_i дорівнює відповідним значенням в таблиці 1.3

4. Занесемо дані в таблицю результатів (таблиця 5.5).

Таблиця 1.5 – Таблиця кінцевих значень параметрів

Параметри програми	z1	z2	z3	z4	z5	z6	z7	z8	z9
Кількість параметрів	6,25	2	0	5	26	10	53	9	75

5. Розрахуємо оцінку очікуваного числа помилок в програмі (на основі таблиці кінцевих значень параметрів – таблиця 1.5 – і

таблиць коефіцієнтів впливу параметрів – таблиці 1.2а та 1.2б), інтенсивність відмов програми, ймовірність виникнення помилки за час виконання програми і занесемо дані в таблицю 1.6.

Для програми типу «Програма-обчислювач» очікуване число помилок знаходиться наступним чином:

$$N = \sum_{j=1}^{r} a_j z_j = 0{,}002 \cdot 6{,}25 + 0{,}024 \cdot 2 +$$
$$+ 0{,}412 \cdot 0 + 0 \cdot 5 + 0{,}416 \cdot 26 + 0{,}016 \cdot 10 +$$
$$+ 0{,}082 \cdot 53 + (-0{,}0026) \cdot 9 + 0{,}002 \cdot 75 = 13{,}2531$$

Інтенсивність відмов програми при часі виконання $t_{ріш}$ = 3,41с:

$$\lambda_{пр} = \bar{\gamma} N / \tau_{ріш} = 0.005 \times 13{,}2531 \div 3{,}41 = 0.0194$$

Ймовірність виникнення помилки за час виконання програми:

$$P(t) = \exp(-\lambda_{пр} \cdot t_{реш}) = \exp((-0{,}0194) \times 3{,}41) = 0.936$$

Таблиця 1.6 – Таблиця розрахунків

Оцінка очікуваного числа помилок в програмі (N)	13,2531
Інтенсивність відмов програми ($\lambda_{пр}$)	0,0194
Ймовірність виникнення помилки за час виконання програми P(t)	0,936

1.3 Мета роботи

Мета роботи – засвоїти методику розрахунку основних показників надійності програмного забезпечення на ранніх етапах розробки.

1.4 Порядок проведення роботи

Одержавши (або обравши самостійно) програмний код, для якого проводиться розрахунок показників надійності, та ознайомившись з ним, студент визначає загальну кількість структурних складових, що впливають на потенційну кількість виникаючих помилок, складаючи для розрахунку таблиці типу таблиць 1.3, 1.4, 1.5 та 1.6

Використовуючи методику розділу 1, виконати розрахунок показників надійності за прикладом та умовами підрозділу 1.2. Резервування та відновлення відсутні.

1.5 Обробка результатів

В звіті виконаної роботи привести основні теоретичні положення, розрахункові формули і розрахунки. Усі розрахунки звести до таблиці. Зробити висновки про якість програмного забезпечення, оформити звіт та, захистивши, здати викладачеві для заліку.

1.6 Контрольні запитання

1. Дати визначення надійності прогрмного забезпечення.
2. Умови виникнення відмов програмного забезпечення.
3. Охарактеризувати види можливих відмов програмного забезпечення та шляхи зменшення вірогідності їх виникнення.
4. Охарактеризувати типи програмних кодів за видами функцій, які ними виконуються.
5. Від яких параметрів залежить очікувана кількість помилок в програмі? Охарактеризуйте перераховані параметри.
6. Від чого залежить значення імовірності виникнення помилок?

2 ЛАБОРАТОРНА РОБОТА №6
ОЦІНКА НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА РЕЗУЛЬТАТАМИ ТЕСТУВАННЯ

2.1 Загальні відомості з надійності програмного забезпечення

Причиною відмови програмного засобу є неможливість його повної перевірки в процесі тестування і випробувань. При експлуатації програмного засобу в реальних умовах може виникнути така комбінація вхідних даних, яка викличе відмову, отже, працездатність програмного засобу залежить від вхідних даних, і чим менше ця залежність, тим вище рівень надійності.

Причиною відмови програмного засобу є помилки, які можуть бути викликані: внутрішніми властивостями програмного засобу, реакцією програмного засобу на зміну зовнішнього середовища функціонування. Це означає, що при найретельнішому тестуванні, якщо припустити, що вдалося позбутися усіх внутрішніх помилок, не можливо з повною впевненістю стверджувати, що в процесі експлуатації програмного засобу не виникне відмова.

Основним засобом визначення кількісних показників надійності є моделі надійності, під якими розуміють математичну модель, побудовану для оцінки залежності надійності від заздалегідь відомих або оцінених в ході створення програмного засобу параметрів. У зв'язку з цим **визначення показників надійності прийнято розглядати з урахуванням трьох процесів: передбачення, вимірювання, оцінювання.**

Передбачення – це визначення кількісних показників надійності виходячи з характеристик майбутнього програмного засобу.

Вимірювання – це визначення кількісних показників надійності на основі аналізу даних про інтервали між відмовами, отриманих при виконанні програм в умовах тестових випробувань.

Оцінювання – це визначення кількісних показників надійності на основі даних про інтервали між відмовами, отриманими при випробуванні програмного засобу в реальних умовах функціонування.

Всі моделі надійності можна класифікувати відносно того, який із перерахованих процесів вони підтримують (що передбачають, прогнозуючі, оцінюючі, що вимірюють). Потрібно відзначити, що

моделі надійності, які в якості вихідної інформації використовують дані про інтервали між відмовами, можна віднести і до тих, що вимірюють, і до оцінюючих в рівній мірі. Деякі моделі, засновані на інформації, отриманій в ході тестування програмного засобу, дають можливість робити прогнози поведінки програмного засобу в процесі експлуатації.

Розглянемо аналітичні та емпіричні моделі надійності.

Аналітичні моделі дають можливість розраховувати кількісні показники надійності, ґрунтуючись на даних про поведінку програми в процесі тестування (моделі, що вимірюють або оцінюють).

Емпіричні моделі базуються на аналізі структурних особливостей програм. Вони розглядають залежність показників надійності від числа міжмодульних зв'язків, кількості циклів в модулях, відношення кількості прямолінійних ділянок до кількості точок розгалуження тощо. Потрібно відзначити, що часто емпіричні моделі не дають кінцевих результатів показників надійності.

Аналітичне моделювання надійності програмного засобу включає **чотири етапи**:

– визначення пропозицій, пов'язаних з процедурою тестування програмного засобу;

– розробка або вибір аналітичної моделі, що базується на припущеннях про процедуру тестування;

– вибір параметрів моделей з використанням отриманих даних;

– застосування моделі – розрахунок кількісних показників надійності моделі.

Аналітичні моделі представлені двома групами: **динамічні і статичні моделі**. В динамічних моделях надійності програмного засобу поведінка програми (поява відмов) розглядається у часі. В статичних моделях появу відмов не пов'язують з часом, а враховують тільки залежність кількості помилок від числа тестових прогонів (по області помилок) або залежність кількості помилок від характеристики вхідних даних (по області даних). Для використання динамічних моделей необхідно мати дані про появу відмов у часі.

Статичні моделі принципово відрізняються від динамічних тим, що в них не враховується час появи помилок в процесі тестування і не використовується ніяких припущень про поведінку функції ризику $\lambda(t)$. Ці моделі будуються на твердому статистичному фундаменті.

2.2 Модель Коркорена

Застосування моделі передбачає знання наступних її показників:

– модель містить ймовірність відмов, що змінюється, для різних джерел помилок і відповідно різну ймовірність їх виправлення;

– в моделі використовуються такі параметри, як результат тільки тих випробувань, в яких спостерігається N помилок і -го типу;

– виявлення в ході N випробувань помилки і-го типу з'являються з вірогідністю a_i.

Показник рівня надійності R обчислюється за такою формулою:

$$R = \frac{N_0}{N} + \sum_{i=1}^{k} \frac{Y_i \cdot (N_i - 1)}{N} \qquad (2.1)$$

де N_0 – число безвідмовних (або безрезультатних) випробувань, виконаних в серії з N випробувань;

k – відоме число типів помилок;

Y_i – вірогідність появи помилок, при $N_i > 0$, $Y_i = a_i$, при $N_i = 0$, $Y_i = 0$.

2.3 Модель Шумана

Модель Шумана відноситься до динамічних моделей дискретного часу, дані для якої збираються в процесі тестування програмного забезпечення протягом фіксованих або випадкових інтервалів часу. ***Модель Шумана передбачає***, що ***тестування проводиться в кілька етапів***. Кожен етап являє собою виконання програми на повному комплексі розроблених тестових даних. ***Виявлені помилки реєструються, але не виправляються***. В кінці етапу розраховуються кількісні показники надійності, виправляються знайдені помилки, коригуються тестові набори і проводиться наступний етап тестування. У моделі Шумана передбачається, що число помилок в програмі постійно і в процесі коригування нові помилки не вносяться. Швидкість виявлення помилок пропорційна числу помилок, що залишилися.

Передбачається, що ***до початку тестування існує E_t помилок***. Упродовж часу тестування τ виявляється ε_c помилок в розрахунку на одну команду в машинній мові. Таким чином, питоме число помилок на одну машинну команду, що залишилися в системі після τ часу тестування, дорівнює:

$$\varepsilon_r = E_t/I_t \cdot \varepsilon_c(\tau) \qquad (2.2)$$

де **I_t – загальне число машинних команд, яке передбачається постійним в рамках етапу тестування**.

Передбачається, що значення функції **частоти відмов Z(t)** пропорційне числу помилок, що залишилися в програмі після **витраченого на тестування часу τ .**

$$Z(t) = C \cdot \varepsilon_r(\tau)$$

де C – деяке постійне значення, **t – час роботи програми без відмов**.

Тоді, якщо час роботи програми без відмови відраховується від точки t = 0, а τ залишається фіксованим, функція надійності, або ймовірність безвідмовної роботи ***R (reliability)*** на інтервалі від 0 до t, дорівнює:

$$R(t,\tau) = \exp\{C[E_t/I_t - \cdot\varepsilon_c(\tau)] \cdot t\} \qquad (2.3)$$

$$t_{cp} = \frac{1}{C[E_t/I_t - \cdot\varepsilon_c(\tau)]} \qquad (2.4)$$

Далі необхідно знайти **початкове значення помилок E_t і коефіцієнт пропорційності – C.** У процесі тестування збирається інформація про час і кількість помилок на кожному прогоні, тобто **загальний час тестування τ складається з часу кожного прогону**:

$$\tau = \tau_1 + \tau_2 + ... + \tau_n \qquad (2.5)$$

Припускаючи, що **інтенсивність появи помилок постійна і дорівнює** λ, можна обчислити її як число помилок в одиницю часу:

$$\lambda = \frac{\sum_{i=1}^{k} A_i}{\tau} \tag{2.6}$$

де A_i – кількість помилок на і-му прогоні.

Маючи дані для двох різних моментів тестування τ_a і τ_b, які обираються довільно з урахуванням вимоги, щоб $\varepsilon_c(\tau_b) > \varepsilon_c(\tau_a)$, можна зіставити рівняння (6.4) і (6.7) при τ_a і τ_b :

$$t_{cp} = \frac{\tau}{\sum_{i=1}^{k} A_i} \tag{2.7}$$

$$\frac{1}{\lambda(\tau_a)} = \frac{1}{C[E_t/I_t - \varepsilon_c(\tau_a)]} \tag{2.8}$$

$$\frac{1}{\lambda(\tau_b)} = \frac{1}{C[E_t/I_t - \varepsilon_c(\tau_b)]} \tag{2.9}$$

Тоді:

$$E_t = \frac{I_t[\lambda\lambda\tau_b)/\lambda/\tau_a) \cdot \varepsilon_c(\tau_a) - \varepsilon_c(\tau_b)]}{(\lambda\lambda\tau_b)/\lambda/\tau_a)) - 1} \tag{2.10}$$

$$C = \frac{\lambda(\tau_a)}{E_t - \varepsilon_c(\tau_a)} \tag{2.11}$$

Отримавши **значення E_t і C,** можна розрахувати **надійність безвідмовної роботи програми R** за формулою (2.3)**.**

2.4 Приклад використання моделі Шумана для розрахунку надійності програмного забезпечення за результатами тестування.

Наприклад, в програмі, яку ми обрали для тестування, маємо I_t=4381 оператор. В процесі послідовних тестових прогонів було отримано данні, наведені в таблиці 2.1.

Таблиця 2.1 – Отримані дані в процесі тестівання за десть прогонів

№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	2	1	1	1	1	1	2	1	1
Час (хв)	5	8	2	1	5	1	1	2	5	5

Оберемо дві точки, виходячи з вимоги, щоб число помилок, знайдених на інтервалі А ÷ В, було більше, ніж на інтервалі 0 ÷ А. За точку А візьмемо 2-й прогін, а за точку В – 8-й прогін. Тоді помилки, знайдені на етапах тестування на інтервалах 0 ÷ А і А ÷ В, будуть дорівнювати відповідно:

$$\varepsilon_c(\tau_A) = 3/4381 = 0{,}0007$$
$$\varepsilon_c(\tau_B) = 7/4381 = 0{,}0015.$$

Час тестування на інтервалах дорівнює: τ_A=13, τ_B=12.

Розрахуємо інтенсивності появи помилок на двох інтервалах:

$$\lambda(\tau_A) = 3/13 = 0{,}23,$$
$$\lambda(\tau_B) = 7/12 = 0{,}58.$$

Тоді число помилок, що існують до початку тестування, дорівнює:

$$E_t = \frac{4381(0{,}58/0{,}23 \cdot 0{,}0007 - 0{,}0015)}{0{,}58/0{,}23 - 1} = 0{,}763 \approx 1 \text{ помилка.}$$

$$C = \frac{0{,}23}{0{,}763 - 0{,}0007} = 0{,}301$$

Розрахуємо вірогідність безвідмовної роботи за час t при τ = 35 хв. Візьмемо t = 60 хв.

$$R(60{,}35) = \exp\{0{,}301[0{,}0002 - 0{,}0027] \cdot 60\} \approx 0{,}955$$
$$R(t{,}35) = \exp\{0{,}301[\,0{,}763/4381 - \varepsilon_c(35)] \cdot t\}.$$

Таким чином, вірогідність безвідмовної роботи досить велика і вірогідність збоїв та виникнення помилок невелика.

2.5 Мета роботи

Мета роботи – засвоїти методику розрахунку основних показників надійності програмного забезпечення на ранніх етапах розробки.

2.6 Порядок проведення роботи

Використовуючи вихідні дані згідно варіанту (таблиця 2.2) та використовуючи методику розділу, виконати розрахунок та визначити вірогідність безвідмовної роботи за прикладом та умовами підрозділів 2.3 та 2.4.

Таблиця 2.2 – Вихідні параметри згідно варіанту

Варіант 1										
I_t=4283, t = 60 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	2	2	1	2	1	1	2	1	1
Час (хв)	5	9	3	2	3	2	1	2	4	7
Варіант 2										
I_t=4391, t = 70 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	2	1	1	1	1	1	2	1	1
Час (хв)	2	1	6	3	2	5	3	2	1	4
Варіант 3										
I_t=4375, t = 80 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	2	2	1	4	2	1	2	1	1
Час (хв)	6	4	1	1	1	3	1	1	1	5

Продовження таблиці 2.2

Варіант 4										
I_t=4372, t = 50 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	2	2	1	3	1	1	2	1	1
Час (хв)	4	6	2	1	2	1	2	1	7	5
Варіант 5										
I_t=4402, t = 60 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	2	3	2	1	2	1	1	2	1	1
Час (хв)	5	15	2	1	7	4	1	2	1	4
Варіант 6										
I_t=4392, t = 70 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	3	1	2	3	1	1	1	1	2
Час (хв)	4	8	2	1	4	4	2	1	6	5
Варіант 7										
I_t=4358, t = 80 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	3	1	2	1	1	2	1	1	1
Час (хв)	4	8	2	1	1	4	1	2	4	5
Варіант 8										
I_t=4363, t = 50 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	1	2	1	1	2	1	1	1	2
Час (хв)	4	6	4	1	7	1	1	2	8	5

Продовження таблиці 2.2

Варіант 9										
I_t=4298, t = 60 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	1	2	1	2	1	1	1	1	1
Час (хв)	3	6	8	1	5	1	7	2	8	5
Варіант 10										
I_t=4392, t = 70 хв										
№ прогону	1	2	3	4	5	6	7	8	9	10
Кількість помилок	1	2	2	1	2	1	1	1	2	1
Час (хв)	5	9	2	1	5	1	1	3	5	8

2.7 Обробка результатів

В звіті виконаної роботи привести основні теоретичні положення, розрахункові формули і розрахунки. Зробити висновки про якість програмного забезпечення, оформити звіт та, захистивши, здати викладачеві для заліку.

2.8 Контрольні запитання

1. Які процеси враховуються при визначенні показників надійності? Чим вони характеризуються?
2. Дайте визначення аналітичних та емпіричних моделей надійності.
3. В чому полягають особливості аналітичних моделей надійності?
4. Які аналітичні моделі надійності Вам відомі?
5. В чому полягає особливість моделі Коркорена?
6. В чому полягає особливість моделі Шумана?

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Основи надійності цифрових систем : підручник / В.С. Харченко, В.Я. Жихарев, В.М. Ілюшко, В.А. Краснобаєв, П.М. Куліков, І.В. Лисенко, М.В. Нечипорук, Г.М. Тимонькін. – Харків: Нац. аерокосм. ун-т "Харк. авіац. інтт", 2017. – 573 с.

2. Дегтярьова Л.М., Гроза П.М., Сомов С.В., Навчальний посібник з дисципліни «Технології розробки програмного забезпечення» – Полтава: ПолтНТУ, 2017. – 218 с.

3. Табунщик Г. В. Інженерія якості программного забезпечення : навчальний посібник. 2-ге видання / Г. В. Табунщик, Р. К. Кудерметов, Т. І. Каплієнко. – Запоріжжя : Дике Поле, 2016. – 176 с.

ДОДАТОК А
АНАЛІЗ ПРОГРАМНОГО КОДУ

Лістинг А.1 -Приклад аналізу програмного коду розрахункової програми

```
# include <iostream >
# include <iomanip >
# include <ctime >
# include <cstdlib >
#include <cmath>
using namespace std;
class Matrix {
public:
      Matrix(int, int);
      Matrix(const Matrix & matrix);
      ~Matrix();
      Matrix operator = (const Matrix & matrix);
      Matrix operator + (const Matrix & matrix);
      Matrix operator !();
      void printMatrix();
      void fillMatrix();
      void zeroMatrix();       // Нулевая матрица
      void diagonalMatrix(); // Заполняет диагональ единицами
      Matrix operator*(const Matrix &) const; //
Умножение двух матриц
      Matrix operator~();// Изменения знаков на противоположные
private:
      int n; int m; double **a;
      void errMem(void *);
      static int count;};
int Matrix::count = 0;
Matrix Matrix::operator~() {// Изменения знаков на противоположные
      Matrix temp(n, m);
      temp = *this;
      for (int i = 0; i < n; i++){
            for (int j = 0; j < m; j++){
                  if (temp.a[i][j] > 0){
                        a[i][j] =  (temp.a[i][j]);
                  }
                  else{
                        a[i][j] = abs(temp.a[i][j]);
                  }}}
```

```
      return *this;
}
// Умножение двух матриц
Matrix Matrix::operator*(const Matrix &right) const {
      Matrix temp(n, m);
      if (n != right.m){
                     << "Error! n != m" << std::endl;
      }
      else{
          for (int i = 0; i < n; i++)
              (int j = 0; j < m; j++){
                 temp.a[i][j] = 0;
                     (int k = 0; k < n; k++)
                       temp.a[i][j]  = a[i][k]
right.a[k][j];
            }}}}
      return temp;}
void Matrix::diagonalMatrix() //Заполняет диагональ единицами
{
      zeroMatrix();            // Заполняет матрицу нулями
      int j = 0;
      for (int i = 0; i < n; i++)
      {if (j < m)
            {  a[i][j] = 1;
                  j  = 2;
            }}}
void Matrix::zeroMatrix(){      // Нулевая матрица
      for (int i = 0; i < n; i++){
              (int j = 0; j < m; j++){
                a[i][j] = 0;
            }}}
0Matrix::Matrix(int nn, int mm)
: n(nn), m(mm){
           << " Construstor , create object " <<
++count << endl;
      a = new double *[n]; errMem(a);
      for (int i = 0; i < n; i++){
            a[i] = new double[m]; errMem(a[i]);
      }
}
Matrix::Matrix(const Matrix & matrix){
           << "Copy - construstor , create object " <<
```

```
++count << endl;
      n = matrix.n; m = matrix.m;
      a = new double *[n]; errMem(a);
    for (int i = 0; i<n; i++){
          a[i] = new double[m]; errMem(a[i]);
      }
    for (int i = 0; i<n; i++)
        (int j = 0; j < m; j++)
          a[i][j] = matrix.a[i][j];
}
Matrix ::~Matrix(){
          << " Destructor , delete object " << count--
<< endl;
    for (int i = 0; i<n; i++)
          delete a[i];
      delete[] a;
}
void Matrix::fillMatrix(){
    for (int i = 0; i<n; i++)
        (int j = 0; j<m; j++)
          a[i][j] = (double)rand()   RAND_MAX;
}
                              {
      cout << endl;
      cout.setf(ios::fixed);
      cout.precision(3);
    for (int i = 0; i<n; i++){
          (int j = 0; j<n; j++)
                << a[i][j] << " ";
           << endl;
      }
          << endl;
}
void Matrix::errMem(void *p){
      if( p){
                << " Memory usage error !" << endl;
          exit(1);
      }}
Matrix Matrix::operator=(const Matrix & matrix){
          << " operator = \n";
      if (this == &matrix)
          return *this;
```

```
      n = matrix.n; m = matrix.m;
      a = new double *[n]; errMem(a);
      for (int i = 0; i<n; i++){
            a[i] = new double[m]; errMem(a[i]);
      }
      for (int i = 0; i<n; i++)
          (int j = 0; j < m; j++)
            a[i][j] = matrix.a[i][j];
      return *this;
}
Matrix Matrix :: operator +(const Matrix & matrix){
           << " operator + \n";
      Matrix tmp(n, m);
      for (int i = 0; i<n; i++)
          (int j = 0; j<m; j++)
            tmp.a[i][j] = this->a[i][j]
matrix.a[i][j];
      return tmp;
}
Matrix Matrix :: operator !(){
           << " operator ! \n";
      Matrix tmp(n, m);
      for (int i = 0; i<n; i++)
          (int j = 0; j<m; j++)
            tmp.a[i][j] = a[j][i];
      return tmp;
}
int main()
{   srand(time(NULL));
      Matrix m1(3, 3);
      m1.fillMatrix();
           << "\nm1 :\n";

      Matrix m2(3, 3);
      m2.fillMatrix();
           << "\nm2 :\n";

      Matrix m3(3, 3);
           << "\nm3 :\n";
      m3.fillMatrix();
```

```
        m3 = m1   m2;
             << "\nm1 + m2: \n";

        m2 =  m1;
             << "\nm2 =!m1: \n";

        Matrix m4 = m1   m2;
             << "\nm1 * m2: \n";

        ~m4;
            << "\n-(m1* m2): \n";

        system("pause");
        return 0;
}
```