

***БАКАЛАВРСКА  
РОБОТА***

**БР.ІІ – 34.00.00.000 ІЗ**

**Група ІІ-22-2**

**Грижак Максим**

**2026**

**Івано-Франківський національний технічний університет нафти і газу**

**Факультет інформаційних технологій**

**Кафедра інженерії програмного забезпечення**

**Грижак Максим Русланович**

---

(прізвище, ім'я, по батькові)

**УДК 004.942**

(індекс)

## **БАКАЛАВРСЬКА РОБОТА**

**Методи формальної верифікації програмного забезпечення**

(назва роботи)

**Інженерія програмного забезпечення**

---

(назва освітньої програми)

**121 - Інженерія програмного забезпечення**

---

(шифр і назва спеціальності)

**Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:**

Здобувач освітнього ступеня М.Р. Грижак  
(підпис, ініціали та прізвище здобувача)

Науковий керівник Гобир Лідія Мирославівна, асистент  
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

**Допущено до захисту**

**Завідувач кафедри**

доц. Бандура В.В.  
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ:**

Зав. кафедрою

ІІЗ

доцент.

В.В. Бандура

“        ”                      2025 р.

## **ЗАВДАННЯ** **НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ**

Грижак Максиму Руслановичу

(прізвище, ім'я, по-батькові)

**1. Тема проекту (роботи) "Методи формальної верифікації програмного забезпечення"**

керівник проекту (роботи) Гобир Лідія Мирославівна, асистент                     

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

**2. Строк подання студентом проекту (роботи)** 10 червня 2026 р.

**3. Вихідні дані до проекту (роботи)** Результати і матеріали отримані під час проходження переддипломної практики

**4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)**

1. Аналіз вимог до програмного забезпечення

2. Аналіз вимог і постановка задачі

3. Проектування системи

4. Реалізація проекту

5. Тестування та впровадження

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

1 Інтенсивність науково-дослідної діяльності в SE порівняно з FV (рис. 1.1, ст. 17)

2 Типи додатків (рис. 2.1, ст. 27)

3 Обсяг проекту в рядках коду (рис. 2.3, ст. 28)

4 Необхідність формальної верифікації (рис. 3.1, ст. 56)

5 Мови специфікації (рис. 3.3, ст. 62)

## 6. Консультанти розділів проекту (роботи)

| Розділ | Консультант | Підпис, дата   |                  |
|--------|-------------|----------------|------------------|
|        |             | Завдання видав | Завдання прийняв |
|        |             |                |                  |
|        |             |                |                  |

2. Дата видачі завдання 2026 р.

Керівник

\_\_\_\_\_ (підпис)

Завдання прийняв до виконання

\_\_\_\_\_ (підпис)

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проекту (роботи)                                    | Строк виконання етапів проекту | Примітка |
|-------|-----------------------------------------------------------------------------|--------------------------------|----------|
| 1     | Визначення та обґрунтування теми роботи                                     | 17.01.2026                     | виконано |
| 2     | Огляд існуючих концепцій, рішень та сервісів в даній області                | 21.02.2026                     | виконано |
| 3     | Побудова моделі або алгоритму власного рішення                              | 01.03.2026                     | виконано |
| 4     | Документування реалізації власного оригінального рішення вибраними засобами | 21.04.2026                     | виконано |
| 5     | Оформлення пояснювальної записки бакалаврської роботи                       | 02.05.2026                     | виконано |

Студент

\_\_\_\_\_ Грижак М.Р.  
(підпис)

Керівник роботи

\_\_\_\_\_ Гобир Л.М.  
(підпис)

## АНОТАЦІЯ

Бакалаврська робота містить 81 сторінок, 17 рисунків, 7 таблиць, список використаних джерел із 40 найменувань.

**Метою роботи** є дослідження методів формальної верифікації програмного забезпечення та аналіз підходів до забезпечення надійності, коректності та безпеки програмних систем.

**Об'єкт дослідження:** процеси розробки та перевірки програмного забезпечення.

**Предмет дослідження:** методи, моделі та інструменти формальної верифікації програмного забезпечення, зокрема математичні методи доведення коректності, формальні специфікації та статичний аналіз.

**Результати дослідження:** проведено аналіз сучасних підходів до формальної верифікації, досліджено класифікацію формальних методів, розглянуто інструментальні засоби їх застосування та оцінено ефективність.

**У першому розділі** розглянуто теоретичні основи формальної верифікації програмного забезпечення, основні поняття, принципи та класифікацію формальних методів.

**У другому розділі** досліджено сучасні методи та інструменти формальної верифікації, включаючи статичний аналіз, формальні специфікації та практичні приклади застосування.

**У третьому розділі** розглянуто реалізацію та оцінку ефективності методів формальної верифікації, проведено аналіз результатів їх застосування та визначено їх вплив на якість програмного забезпечення.

**Висновок:** використання методів формальної верифікації дозволяє значно підвищити надійність, безпеку та коректність програмного забезпечення, зменшити кількість помилок на ранніх етапах розробки.

**КЛЮЧОВІ СЛОВА:** ФОРМАЛЬНА ВЕРИФІКАЦІЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СТАТИЧНИЙ АНАЛІЗ, ФОРМАЛЬНІ МЕТОДИ, СПЕЦИФІКАЦІЯ, ДОКАЗ КОРЕКТНОСТІ, НАДІЙНІСТЬ, БЕЗПЕКА.

## ANNOTATION

The bachelor's thesis contains of 81 pages, 17 figures, 7 tables, and a reference list with 40 sources.

**The aim of the work** is to study methods of formal verification of software and to analyze approaches to ensuring reliability, correctness, and security of software systems.

**Research object:** software development and verification processes.

**Research subject:** methods, models, and tools of formal verification of software, including mathematical proof techniques, formal specifications, and static analysis.

**Research results:** an analysis of modern approaches to formal verification was conducted, the classification of formal methods was studied, tools for their application were reviewed, and the effectiveness of formal verification methods in software development practice was evaluated.

**The first section** describes the theoretical foundations of formal verification of software, including key concepts, principles, and classification of formal methods.

**The second section** analyzes modern methods and tools of formal verification, including static analysis, formal specifications, and practical use cases.

**The third section** covers the implementation and evaluation of the effectiveness of formal verification methods, including analysis of results and their impact on software quality.

**Conclusion:** the use of formal verification methods significantly improves software reliability, security, and correctness, reduces the number of errors at early stages of development, and ensures compliance with specified requirements.

**KEY WORDS:** FORMAL VERIFICATION, SOFTWARE, STATIC ANALYSIS, FORMAL METHODS, SPECIFICATION, CORRECTNESS PROOF, RELIABILITY, SECURITY.

# ЗМІСТ

|                                                                                          |           |
|------------------------------------------------------------------------------------------|-----------|
| <b>ВСТУП .....</b>                                                                       | <b>7</b>  |
| <b>РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ФОРМАЛЬНОЇ ВЕРИФІКАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .....</b> | <b>9</b>  |
| 1.1 Основні поняття та принципи формальної верифікації .....                             | 9         |
| 1.2 Огляд сучасних підходів і практик формальних методів .....                           | 19        |
| 1.3 Класифікація та характеристика формальних методів верифікації .....                  | 22        |
| 1.4 Висновки по розділу .....                                                            | 25        |
| <b>РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ ФОРМАЛЬНОЇ ВЕРИФІКАЦІЇ .....</b>                      | <b>26</b> |
| 2.1 Сучасний стан та тенденції розвитку формальної верифікації .....                     | 26        |
| 2.2 Інструментальні засоби та практичні приклади застосування .....                      | 34        |
| 2.3 Абстрактний статичний аналіз як метод формальної перевірки .....                     | 49        |
| 2.4 Висновки по розділу .....                                                            | 55        |
| <b>РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДІВ ВЕРИФІКАЦІЇ .....</b>             | <b>56</b> |
| 3.1 Роль та необхідність формальної верифікації у сучасних системах .....                | 56        |
| 3.2 Аналіз результатів застосування формальних методів .....                             | 66        |
| 3.3 Оцінка ефективності та обговорення отриманих результатів .....                       | 73        |
| 3.4 Висновки по розділу .....                                                            | 80        |
| <b>ВИСНОВКИ .....</b>                                                                    | <b>81</b> |
| <b>СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА .....</b>                                                  | <b>83</b> |

|           |      |               |        |      |                                                                                    |                 |      |         |
|-----------|------|---------------|--------|------|------------------------------------------------------------------------------------|-----------------|------|---------|
|           |      |               |        |      | БР.ІП – 34.00.00.000 ПЗ                                                            |                 |      |         |
| Змн.      | Арк. | № докум.      | Підпис | Дата |                                                                                    |                 |      |         |
| Розроб.   |      | Гаргат К.Ю.   |        |      | Методи формальної верифікації програмного забезпечення<br><br>Пояснювальна записка | Літ.            | Арк. | Акрушів |
| Перевір.  |      | Гобир Л.М.    |        |      |                                                                                    |                 | 7    |         |
| Реценз.   |      | Яцишин М.М.   |        |      |                                                                                    | ІФНТУНГ ІП-22-2 |      |         |
| Н. Контр. |      | Піх М.М.      |        |      |                                                                                    |                 |      |         |
| Затверд.  |      | Бандура В. В. |        |      |                                                                                    |                 |      |         |

## ВСТУП

У сучасному світі програмне забезпечення стало невід’ємною частиною критично важливих систем, таких як фінансові сервіси, медичні інформаційні системи, транспортні платформи та системи керування інфраструктурою. Зростання складності програмних систем, збільшення обсягів даних та підвищення вимог до безпеки і надійності зумовлюють необхідність використання ефективних методів забезпечення їх коректності. Традиційні підходи до тестування не завжди дозволяють виявити всі можливі помилки, особливо у складних або критичних системах, що підкреслює важливість застосування формальних методів верифікації.

Актуальність теми зумовлена необхідністю підвищення якості програмного забезпечення шляхом використання математично обґрунтованих підходів до перевірки його правильності. Формальна верифікація дозволяє довести відповідність програм заданим специфікаціям, зменшити ризик виникнення критичних помилок та забезпечити високий рівень безпеки. У сучасних умовах, коли програмні системи інтегруються у різноманітні сфери діяльності, включаючи фінансові технології, штучний інтелект та кібербезпеку, застосування формальних методів стає особливо важливим.

Метою роботи є дослідження методів формальної верифікації програмного забезпечення та аналіз їх ефективності для забезпечення надійності, коректності та безпеки програмних систем.

Завданнями дослідження є аналіз теоретичних основ формальної верифікації, вивчення сучасних підходів і практик застосування формальних методів, дослідження інструментальних засобів формальної перевірки, зокрема методів статичного аналізу, оцінка ефективності їх використання у процесі розробки програмного забезпечення, а також формування висновків щодо перспектив застосування формальної верифікації.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 7    |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Об'єктом дослідження є процеси розробки та перевірки програмного забезпечення.

Предметом дослідження є методи, моделі та інструменти формальної верифікації, включаючи математичні підходи до доведення коректності, формальні специфікації та алгоритми статичного аналізу.

Методи дослідження включають аналіз наукових джерел для вивчення теоретичних основ формальної верифікації, порівняльний аналіз існуючих методів і інструментів, моделювання процесів перевірки програмного забезпечення, а також експериментальний підхід для оцінки ефективності застосування формальних методів.

У роботі розглядаються сучасні підходи до формальної верифікації, включаючи класифікацію методів, інструментальні засоби їх реалізації та практичні аспекти застосування. Особлива увага приділяється оцінці ефективності формальних методів у підвищенні якості програмного забезпечення та зменшенні кількості помилок на ранніх етапах розробки.

Бакалаврська робота містить 82 сторінок, 17 рисунків, 7 таблиць, 3 розділи, список використаних джерел із 40 найменувань.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 8    |

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ФОРМАЛЬНОЇ ВЕРИФІКАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 1.1 Основні поняття та принципи формальної верифікації

Функціональна верифікація стала вузьким місцем для проектування складних систем. Моделювання проектів є фінансово та часомістким, а виконання повного моделювання є майже неможливим. Наразі, як вирішення цих проблем, розробники почали використовувати формальні методи для виконання формальної верифікації більшості продуктів. Але все ще існує великий розрив для верифікації великих проектів, які можуть бути побудовані, але не можуть бути верифіковані повністю через складність проблем, з якими вони стикаються. Це призвело до того, що в багатьох країнах академічний світ, промисловість і уряди мусять постати перед викликом скорочення цього технологічного розриву та пропонування нових та винахідливих рішень для специфікації, проектування, структурування та застосування тестових сценаріїв шляхом використання формальної верифікації.

Формальна верифікація є вирішальним елементом у розробці сучасних складних інформаційних систем. Закон Мура все ще застосовується для визначення темпів зростання складності програмних та апаратних продуктів, але складність верифікації стає дедалі складнішою. Фактично, теоретично, вона зростає експоненціально зі складністю продукту і подвоюється так само з часом. Спільнота комп'ютерних наук визнає, що функціональна верифікація є важливою перешкодою для методології проектування, і що вона вимагає до 70% часу та ресурсів розробки. Але, незважаючи на значну кількість зусиль і ресурсів, застосованих у верифікації, функціональні дефекти залишаються причиною значної кількості помилок кінцевого продукту [1]. В екстремальних ситуаціях помилки є артефактами моделювання, оскільки вони не виявляються через невичерпний характер верифікації, яка базується на моделюванні. Реальний факт полягає в тому, що не має значення, скільки часу витрачено на моделювання або

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 9    |

наскільки вичерпним є план випробувань, будь-яка спроба підтвердити проект за допомогою моделювання сама по собі є неповною для будь-якої системи.

Формальна верифікація (FV) - це систематичний процес, який використовує математичне обґрунтування, щоб перевірити, чи специфікація проекту залишається незмінною під час реалізації. За допомогою цієї верифікації можна подолати виклики моделювання, оскільки всі можливі вхідні значення можуть бути досліджені алгоритмічно або вичерпно. Іншими словами, для досягнення високого ступеня спостереження за продуктом не потрібно перебільшувати дизайн або створювати кілька сценаріїв.

Однією з цілей FV є гарантування повного покриття простору станів у тестованому проекті; щоб досягти цього, вона використовує та застосовує такі методи, як верифікація моделі через дослідження простору станів та автоматизовані методи для доведення теорем. Наразі найбільш автоматизованою та найбільш визнаною технікою FV є Symbolic Model Verifier або SMV і, незважаючи на її успіх як важливого методу для формальної верифікації послідовних комерційних проектів, вона все ще обмежена стосовно розміру верифікованих проектів. Формальна верифікація вимагає від інженерів думати інакше [2]. Наприклад, моделювання є емпіричним, це означає, що використання методу спроб і помилок для перевірки всіх можливих комбінацій і спроб виявити помилки може зайняти значний час. З цієї причини воно не досягає мети повністю. Крім того, оскільки інженери повинні визначати та створювати велику кількість вхідних сценаріїв, вони зосереджують свої зусилля на тому, щоб зламати проект, а не на тому, що проект повинен робити. Формальна верифікація, навпаки, є математичною та вичерпною і дозволяє інженерам зосередитися виключно на з'ясуванні того, яка поведінка дизайну є правильною.

Метою даного дослідження є проведення систематичного огляду літератури щодо досліджень формальної верифікації за останнє десятиліття, щоб визначити підходи, методи, техніки та методології дослідження, що використовувалися, а також інтенсивність цієї дослідницької діяльності. Для досягнення цього була використана парадигма дослідження на основі доказів і

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 10   |

мета полягає в тому, щоб ідентифікувати питання, на яке можна відповісти, яке могло б надати інформацію та яке може привести до доказів для цієї відповіді та оцінити її. Таким чином, систематичний огляд літератури є першим етапом проведення дослідження на основі доказів. У наступному розділі описується методологія, застосована в цьому дослідженні; третій розділ показує та аналізує отримані результати; наступний розділ показує можливі загрози та обмеження для валідації, а останній розділ містить деталі щодо висновків та пропозицій щодо майбутньої роботи.

Таблиця 1.1

### Етапи систематичного огляду

| Етапи          | Процедури                                                                                                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Планування     | <ul style="list-style-type: none"> <li>• Визначення дослідницьких питань</li> <li>• Розробка протоколу огляду</li> <li>• Валідація (перевірка) протоколу огляду</li> </ul>                                            |
| Виконання      | <ul style="list-style-type: none"> <li>• Пошук відповідних досліджень</li> <li>• Відбір первинних досліджень</li> <li>• Оцінка якості досліджень</li> <li>• Витяг необхідних даних</li> <li>• Синтез даних</li> </ul> |
| Документування | <ul style="list-style-type: none"> <li>• Написання звіту про огляд</li> <li>• Валідація (перевірка) звіту</li> </ul>                                                                                                  |

Ось технічний переклад розділу «Методологічний процес» українською мовою:

### Методологічний процес

Проведення систематичного огляду літератури можна розділити на три основні етапи: (1) планування, (2) виконання та (3) документування, які, у свою чергу, поділяються на комбінацію інших простіших процедур, як показано в Таблиці 1.1.

1. Дослідницькі питання.
2. Процес пошуку.
3. Критерії включення та виключення.
4. Оцінка якості.

5. Збір даних.

6. Аналіз даних.

Дослідницькі питання

Дослідницькі питання, застосовані під час розробки цього дослідження, були такими:

- Q1: У яких сферах формальної верифікації наразі проводяться дослідження?

- Q2: Яка методологія застосування є найбільш досліджуваною?

- Q3: У якій техніці формальної верифікації дослідження проводяться найчастіше?

- Q4: Який підхід і метод дослідження є найбільш вживаними?

- Q5: Яка інтенсивність дослідницької діяльності у формальній верифікації?

Для відповіді на Q1, Q2, Q3 та Q4 кожне первинне дослідження було пов'язане з підходом або методом дослідження, з технікою та застосованою методологією, а також із охопленою галуззю. Для встановлення показників інтенсивності дослідницької діяльності щодо Q5 було визначено дослідницький корпус за кількістю публікацій на рік. Нахил лінії для FV (формальної верифікації) порівнювався з нахилом лінії, що представляє дослідницьку діяльність у функціональній верифікації.

Процес пошуку

Систематичний огляд конкретного предмета має ідентифікувати та виділити конкретні джерела про об'єкт вивчення; однак у галузі формальної верифікації такі джерела не були знайдені, оскільки відповідні дослідження можуть бути опубліковані в журналах і на конференціях, пов'язаних як з функціональною верифікацією, так і з формальними методами. Метою пошуку було виявлення первинних досліджень, які могли бути включені або виключені з остаточного набору досліджень огляду. План передбачав автоматизований пошук у цифрових бібліотеках ACM Digital Library, IEEE Digital Library, Science

Direct та Springer Link за період з січня 2000 року по квітень 2011 року.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 12   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Параметри автоматизованого пошуку та їх розташування в дослідженні були наступними:

- Formal verification (Формальна верифікація): у назві. Для всіх дослідницьких питань.
- Discrete mathematics, declarative language, formal language, formal method, formal specification, formal verification: в анотації або в змісті. Для Q1.
- Experimentation, case study, stochastic, heuristic: в анотації або в змісті. Для Q2.
- Peer, animation, simulation, agile methods, XR: в анотації або в змісті. Для Q3.
- Спостереження за результатами Q1, Q2 та Q3 дозволило класифікувати підхід і метод дослідження для Q4. Для емпіричного дослідження було проведено пошук термінів *experiment*, *survey*, *case study*, *empirical research* в анотації та в змісті.
- Formal verification AND research: у назві та в поєднанні з кожним роком часової шкали. Для Q5.

Загальна кількість статей, знайдених під час цього пошуку, склала 552. Однак більшість із них були ідентифіковані за опосередкованим зв'язком та в результаті поєднання деяких ключових слів. Виключення нерелевантних статей проводилося вручну на основі критеріїв виключення та включення.

#### Критерії включення та виключення

Роботи, відібрані як первинні дослідження, мали бути релевантними для теми дослідження, тому було застосовано процес фільтрації:

1. Ідентифікація релевантних досліджень. Розглядалися лише повні роботи, опубліковані в журналах, повні матеріали конференцій, конгресів та воркшопів; короткі (short papers), розширені тези (extended abstracts) та постери були відхилені. Було виключено 131 дослідження.
2. Виключення досліджень на основі їх назви. Застосовувався фільтр "AND" в опції розширеного пошуку кожної цифрової бібліотеки. Було виключено 28 досліджень.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 13   |

3. Виключення досліджень на основі їх анотацій. Було виключено 49 досліджень.

4. Вибір найбільш релевантних робіт серед результатів на основі повного тексту. Було вирішено включити лише роботи, глибоко пов'язані з темою формальної верифікації. Згідно з цим критерієм було виключено 145 робіт, у результаті чого було отримано фінальний набір із 199 статей, розглянутих як первинні дослідження.

#### Оцінка якості

Метою цього етапу є підтвердження того, що відібрані первинні дослідження демонструють ґрунтовність методології та результатів [3]. Враховуючи високі стандарти процесу рецензування в обраних журналах і базах даних, було зроблено висновок, що відібрані первинні дослідження мають належну якість.

#### Збір даних

Після завершення процесу включення або виключення було структуровано набір даних первинних досліджень. На цьому етапі збиралися такі атрибути:

1. Тип заходу: журнал, конференція-конгрес, воркшоп.
2. Опубліковано в: журналі, збірнику матеріалів (proceedings).
3. Видавництво: ACM, IEEE, Springer, Elsevier.
4. Рік публікації: період з 2000 по 2011 рік.
5. Країна.

6. Класифікація підходу та методу. Основними дослідницькими та науковими підходами є: описовий, пояснювальний та емпіричний; існують три методи дослідження для оцінки технік, методів та інструментів: опитування (survey), тематичне дослідження (case study) та експеримент.

7. Класифікація галузі: математичні моделі, формальні мови, автоматизовані моделі, декларативні мови, формальні методи та формальна специфікація.

8. Класифікація методології: експеримент, тематичне дослідження, стохастичні методи та евристика.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 14   |

9. Класифікація техніки: за методом обробки (рецензування, анімація, моделювання та гнучкі методи).

Для відповіді на Q5 було включено три типи статей:

- Технологічна та науково-дослідна стаття: документ, що детально представляє оригінальні результати завершених дослідницьких проектів.
- Стаття-роздум: документ, що представляє результати завершеного дослідження з аналітичної, критичної або інтерпретаційної точки зору.
- Оглядова стаття: документ, що аналізує, систематизує та інтегрує результати опублікованих або неопублікованих досліджень, з метою поширення інформації про досягнення та тенденції розвитку.

#### Аналіз даних

Первинні дослідження були табульовані та статистично проаналізовані з метою визначення:

1. Кількість робіт, опублікованих за роками: Q5.
2. Кількість робіт, опублікованих у журналах та збірниках матеріалів: Q5.
3. Кількість досліджень за країнами: Q5.
4. Основні теми, охоплені у формальній верифікації: Q5.
5. Підхід та метод дослідження: Q4.
6. Сфера формальної верифікації, в якій проводиться дослідження: Q1.
7. Методологія застосування: Q2.
8. Використана техніка: Q3.

#### Результати та аналіз

Для розуміння категорій, призначених для кожного дослідження, характеристики набору даних первинних досліджень були табульовані. Важливо зауважити різницю між дослідницькою діяльністю та дослідницькою статтею. Дослідницька діяльність охоплює набір релевантних статей, які були включені на основі назви, тобто статті про дослідження, статті-роздуми та статті про верифікацію, тоді як дослідницькі статті є кінцевим результатом застосування

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 15   |

критеріїв включення та виключення. Таблиця 1.2 показує динаміку дослідницької діяльності за роками та типом заходу.

Згідно з цими результатами, дослідження у сфері формальної верифікації подвоїлися з 2005 року, залишаючи кількість публікацій незмінною до теперішнього часу [4]. На рисунку 1.1 показано порівняння дослідницької діяльності в інженерії програмного забезпечення (SE) проти формальної верифікації (FV). Теми, охоплені дослідженнями в SE, є різноманітними, серед них ми знайшли формальну верифікацію, але для цього аналізу було взято іншу концепцію через дослідницькі інтереси. Як показано в таблиці 2.5, промислові та академічні групи є найбільшими рушійними силами досліджень у формальній верифікації, проте університети вчетверо перевищують обсяг роботи, виконаної в промисловості. Крім того, у таблиці 1.2 можна побачити, що існують спеціалізовані журнали, воркшопи та конференції, на яких обговорюються теоретичні та практичні застосування цього предмета.

При розгляді діяльності щодо формальної верифікації за країнами можна побачити, що Сполучені Штати забезпечують більше половини від загальної кількості публікацій - 55%. За Сполученими Штатами йдуть Велика Британія, Японія, Китай і Німеччина. В інших репрезентативних країнах було виявлено, що інтерес до FV почався трохи пізніше, ніж у згаданих вище. У таблиці 3 представлено порівняння робіт, які звітують про дослідницьку діяльність, та інших типів звітів, пов'язаних із темою FV.

Таблиця 1.2.

Динаміка досліджень у галузі FV

| Рік  | Конференція-Конгрес | Журнал | Семінар (Workshop) | Усього |
|------|---------------------|--------|--------------------|--------|
| 2000 | 0                   | 3      | 0                  | 3      |
| 2001 | 0                   | 3      | 0                  | 3      |
| 2002 | 1                   | 13     | 0                  | 14     |
| 2003 | 0                   | 8      | 0                  | 8      |
| 2004 | 0                   | 3      | 0                  | 3      |
| 2005 | 11                  | 21     | 0                  | 32     |
| 2006 | 8                   | 13     | 8                  | 29     |
| 2007 | 13                  | 11     | 3                  | 27     |

|      |    |    |   |    |
|------|----|----|---|----|
| 2008 | 11 | 14 | 1 | 26 |
| 2009 | 10 | 8  | 5 | 23 |
| 2010 | 11 | 9  | 7 | 27 |
| 2011 | 0  | 4  | 0 | 4  |

Оскільки метою цього огляду літератури є запит про методи, техніки та методології, які застосовують дослідження у формальній верифікації, з цього моменту стаття буде зосереджена на 105 роботах, що поширюють результати досліджень. Таблиця 1.3 показує видання, у яких були знайдені статті про дослідження, пов'язані з формальною верифікацією, та кількість робіт, опублікованих у часових межах, охоплених цим дослідженням. На рисунку 1.2 показано профіль найбільш активних країн у дослідженнях FV.

Таблиця 1.3.

#### Наукові статті порівняно з іншими статтями

| Рік  | Наукові (Research) | Інші | % Наукових |
|------|--------------------|------|------------|
| 2000 | 1                  | 2    | 33%        |
| 2001 | 0                  | 3    | 0%         |
| 2002 | 6                  | 8    | 43%        |
| 2003 | 2                  | 6    | 25%        |
| 2004 | 0                  | 3    | 0%         |
| 2005 | 13                 | 19   | 41%        |
| 2006 | 17                 | 12   | 59%        |
| 2007 | 16                 | 11   | 59%        |
| 2008 | 16                 | 10   | 62%        |
| 2009 | 16                 | 7    | 70%        |
| 2010 | 18                 | 9    | 67%        |
| 2011 | 0                  | 4    | 0%         |

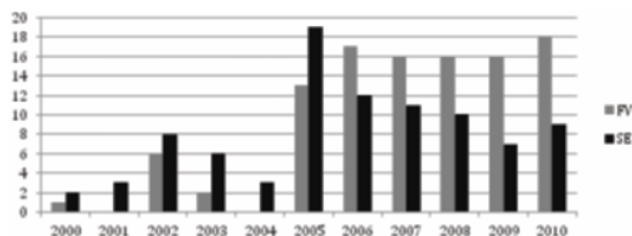


Рисунок 1.1 - Інтенсивність науково-дослідної діяльності в SE порівняно з FV

Найбільш застосовуваним методом є тематичне дослідження (case study); цей факт підкріплює результат, який вказує на те, що університети мають найбільшу частку участі в дослідженнях у цій галузі. Дослідницькі статті з формальної верифікації використовують підхід емпіричного дослідження через необхідність підтвердження методу та застосованої моделі. У Таблиці 1.3 наведено результати щодо галузей досліджень у формальній верифікації; слід враховувати, що вони є інклюзивними (взаємопов'язаними).

Галузі, у яких виконується найбільша частина роботи, - це формальна специфікація, математичні моделі та формальні методи. Останні дозволяють описувати властивості системи за допомогою суворої математики; для досягнення цього вони застосовують мову формальної специфікації, що робить можливим визначення функціональності програми. Це зумовлено тим, як вибудовується формальна верифікація: спочатку вона зосереджується на специфікації, потім будується тестова модель і, зрештою, перевіряється верифікація конкретного прикладу (case study) [5]. Це складний процес, який залучає різні інструменти; деякі [дані] показують порівняння методологій, використаних для формальної верифікації в аналізованих роботах.

Оскільки метод тематичного дослідження використовується для застосування формальної верифікації та ручної перевірки результатів, він також виступає як панівна методологія для валідації результатів емпіричного підходу. Експериментальна частина стає очевидною в участі промислових досліджень. У Таблиці 1.3 представлено техніки, використані для досліджень у FV для первинних робіт.

Сучасні техніки розробки найкраще підходять до нових парадигм, і існують комерційні інструменти, які підтримують покращення якості програмного забезпечення. Оскільки складність інформаційних систем зростає, збитки, спричинені дефектами, стають дедалі вищими [6]. Близько 98% дослідницьких статей описують техніки моделювання (симуляції), що робиться з метою контролю вхідних змінних та реакцій або очікуваних вихідних даних у тестових середовищах. Важливо звернути увагу на той факт, що лише 2%

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 18   |

використовували техніку взаємної перевірки (peer checking), яка в оглядах літератури кінця минулого століття була найбільш вживаною.

## 1.2 Огляд сучасних підходів і практик формальних методів

Перенесення технології формальних методів у промисловість було метою дослідників та практиків протягом кількох десятиліть. Потенційні переваги зменшення щільності дефектів у специфікаціях, конструкціях та коді повинні бути досягнуті за розумною ціною та в межах обмежень реальних промислових умов. На початку 1990-х років виникли питання про те, чи можуть формальні методи коли-небудь бути життєздатною частиною процесів промислової розробки. Кілька важливих досліджень того часу виявили проблеми для практики та досвіду верифікації, які наступні дослідження намагалися вирішити. Ми коротко розглянемо деякі з основних публікацій, що досліджують стан промислового застосування.

Захист Холлом формальних методів як інженерного підходу визначає сім «міфів» про формальні методи. Вінг пояснив новачкам основні концепції та принципи формальних методів. Томас представив докази економічної ефективності промислового використання формальних методів з точки зору генерального директора. Остін провів опитування щодо промислового використання формальних методів, щоб з'ясувати причини їх досить низького визнання в промисловості. Остін використав анкету для оцінки використання формальних методів як у дослідженнях, так і в застосуванні, а також для збору думок щодо перешкод для ширшого впровадження в промисловості. Більшість проаналізованих відповідей (126) повідомляли про використання модельно-орієнтованих формалізмів (таких як VDM) та зосереджувалися на специфікації, а не на верифікації.

Крейген та його колеги дослідили застосування формальних методів, прагнучи надати авторитетний запис про застосування формальних методів. Їхнє дослідження охопило 12 тематичних досліджень, кожне з яких базувалося на

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 19   |

застосуванні формальних методів у промисловому середовищі [7]. Для отримання інформації про кожне застосування використовувалося поєднання анкет, оглядів літератури та інтерв'ю. Спектр формалізмів включав модельно-орієнтовані підходи, обчислення процесів (CSP) та середовища верифікації. Жодне з досліджень не розглядало системи розміром більше 300 KLOC, а більшість були набагато меншими (здебільшого високоінтегральні програми). Опитування дійшло позитивних висновків щодо підвищення зрілості формальних методів та того факту, що вони були застосовані до значних систем. Нормативна підтримка вдосконалених практик програмної інженерії була важливою для забезпечення підвищеної мотивації до їх впровадження. Вони зазначили, що «підтримка інструментів, хоча й необхідна для повного процесу індустріалізації, на сьогоднішній день не виявилася ні необхідною, ні достатньою для успішного застосування формальних методів». Тим не менш, Блумфілд зазначив, що незрілість теорій та інструментальних баз означає, що деякі успішні застосування вимагають «героїчного» рівня зусиль, що інструменти розробляються, але не передаються між платформами чи дослідницькими групами, і що інструменти не завжди супроводжуються досягненнями в методології чи теорії [8].

Технічний звіт для NASA, пояснюючи зацікавленим сторонам з аерокосмічної галузі, що таке формальні методи та як їх можна застосовувати в розробці та сертифікації критичних систем. Усвідомлення довгострокової кризи в розробці програмного забезпечення спонукало до ширшого інтересу до потенціалу верифікації. Написана в гумористичній манері, в якій зазначено заповіді, яких слід дотримуватися під час застосування формальних методів. Вони також відредагували книгу, що містить збірку з 15 різних застосувань формальних методів з використанням різних формалізмів. Стандарти, інструменти та освіта «вирішать проблему» промислового впровадження, тоді як бачив прірву між науковцями, які «вважають формальні методи неминучими», та практиками, які «вважають формальні методи нерелевантними» [9]. В інших статтях наводяться недоліки в нотаціях, інструментах та освіті як виклики для ширшого прийняття технології формальних методів.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 20   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Незважаючи на оптимістичні висновки деяких опитувань, стаття за круглим столом у журналі *IEEE Software* у 1996 році показала розбіжність думок щодо того, чи приносять формальні методи бажані покращення на практиці. Стаття Кларка та Вінга була результатом роботи робочої групи, вона давала короткий вступ до понять формальних методів, перераховувала помітні промислові застосування та рекомендувала майбутні напрямки для спільноти формальних методів. Технічні звіти Келлі були підготовлені NASA та утворили посібник з використання формальних методів для специфікації та верифікації програмного забезпечення та комп'ютерних систем.

Широкий огляд включає оцінки дослідницьких програм, великих конференцій та галузей промислового застосування. Ключовим моментом є пропозиція, що моделі поширення технологій повинні свідомо застосовуватися до впровадження формальних методів [10]. Хоча автори спостерігали значне впровадження в критичних сферах застосування, вони визначили кілька причин загальної нездатності впроваджувати формальні методи.

В опитуваннях представлені дуже різні точки зору. Деякі, такі як Craigen, базують висновки на аналізі вибраної групи застосувань. Інші, такі як Austin, мають ширший погляд на промисловість та академічні кола. Ще інші, такі як Bloomfield, використовують огляди цілих дослідницьких програм. Незважаючи на відмінності в підходах, існує певна згода щодо значних труднощів для успішного промислового впровадження.

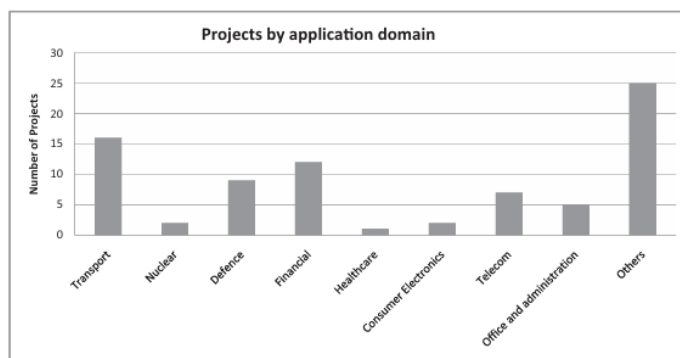


Рисунок 1.2 - Сфери застосування

### 1.3 Класифікація та характеристика формальних методів верифікації

Перші роботи з формальних методів датуються 1960-ми роками. Центральна ідея полягає в *гарантуванні поведінки* обчислювальної системи за допомогою математичних методів.

Протягом багатьох років формальні методи переважно використовувалися в академічних колах. Сьогодні формальні методи стають все більше пов'язані з прикладною інженерією, і багато галузей промисловості використовують їх у процесі розробки [11]. Вони краще застосовуються в проектуванні електронного обладнання, оскільки апаратні засоби є більш стандартизованими та стабільними. Програмні засоби та технології (такі як вимоги та методологія проектування) все ще швидко змінюються.

Ми коротко обговоримо тут три основні категорії формальних методів, які забезпечують певну гарантію якості та є високоавтоматизованими.

Абстрактна інтерпретація - це вид статичного аналізу, запроваджений у 1977 році, який автоматично обчислює інформацію про поведінку програми без її виконання. Більшість питань щодо цієї поведінки або нерозв'язні, або практично неможливо обчислити відповідь. Головна мета абстрактної інтерпретації - ефективно обчислити приблизне або абстрактне представлення програми та надати *надійні* гарантії щодо динамічних властивостей. *Надійність* означає, що наближення є надійним і що помилки не пропущені (немає *хибнонегативних результатів*). Хибний сигнал тривоги, також відомий як *хибнопозитивний результат*, - це попередження про помилку, якої не існує в програмі. Хибнопозитивні результати можуть бути згенеровані через наближення.

Цей метод був реалізований в інструментах статичного аналізу для перевірки помилок під час виконання, таких як ділення на нуль, доступ до масиву поза межами меж, переповнення та інші. Деякі інструменти також розширюють його для перевірки правил кодування MISRA [16], правил SEI CERT Secure C [20] та правил MITRE Common Weakness Enumeration CWE [15]. Astree від AbsInt, Polyspace Code Prover від MathWorks та TrustInSoft Analyzer від TrustInSoft – це

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 22   |

комерційні інструменти для надійного статичного аналізу. Інші інструменти, такі як Coverity та Klocwork, базуються на евристиці та не гарантують знаходження всіх помилок під час виконання. Вони виходять за рамки цієї статті.

Перевірка моделей – це формальний метод, запроваджений на початку 80-х років двома командами. Він визначає, чи задовольняє формальна модель системи властивість коректності зі специфікації. Він забезпечує кращу виразність властивостей, ніж абстрактна інтерпретація, але менш масштабований через деякі обмеження (наприклад, нелінійна арифметика, комбінаторний вибух) [12]. Найціннішою функцією перевірки моделей для інженерів-системних розробників є контрприклад, який створюється, коли властивість порушується. Він показує трасування виконання, що призводить до стану, в якому властивість не перевірена.

Існують різні типи перевірників моделей, але для наших випадків використання (перевірка властивостей безпеки вбудованого програмного забезпечення) найбільш підходящими є ті, що використовують індукцію та SMT (теорії задовільності по модулю), наприклад Kind2 [5], JKind [10], Prover Plugin, також відомий як Design Verifier (Prover Technologies) та GATeL (CEA). Зазвичай вони перетворюють модель, написану формальною мовою синхронного потоку даних, такою як Lustre [3], у логічні формули першого порядку, які потім перевіряються на задовільність розв'язувачем. Іноді властивості недостатньо сильні, щоб їх можна було довести за допомогою індукції. У цьому випадку корисно підсилити гіпотези індукції відомими інваріантними властивостями системи за допомогою методів генерації інваріантів [13].

Дедуктивні методи використовують математичні аргументи для встановлення кожної властивості формальної моделі. Докази зазвичай будуються за допомогою інструменту доведення теорем, автоматично або інтерактивно. Докази базуються на логіці Хоара та численні передумов Дейкстри [11]. Цей метод, як правило, є *більш виразним*, ніж абстрактна інтерпретація та перевірка моделі, але якщо властивість недійсна, зазвичай немає контрприкладу, який би допоміг інженеру. Він також потребує більше людської експертизи, ніж інші

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 23   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

методи.

Можна згадати деякі фреймворки та інструменти для дедуктивного доведення: SPARK для Ada [6], Frama-C WP для коду C [14] та його попередник SAVEAT, який успішно використовувався в Airbus [21]. Метод В [2] – це ще один дедуктивний метод, який успішно використовувався в деяких французьких залізничних компаніях.

Деякі фактори можуть сприяти впровадженню формальних методів у галузі:

- Рекомендація або вимога стандартів ;
- Наявність допоміжних *інструментів* , якими легко користуватися інженерам ;
- Загальна *вартість* їх використання менша, ніж без використання формальних методів.

Для автомобільної промисловості стандартом ISO 26262 є управління функціональною безпекою [14]. Цей стандарт рекомендує використання формальних методів для найважливіших рівнів.

У залізничному транспорті стандарт EN 50128 рекомендує використання формальних методів для менш критичних рівнів і наполегливо рекомендує їх використання для найбільш критичних рівнів, що робить їх фактично обов'язковими.

Зокрема, в авіаційній галузі існує спеціальне доповнення DO-333 до DO-178C, яке змінює, видаляє та додає цілі, характерні для формальних методів. Таким чином, використання формальних методів може замінити деякі складні завдання, такі як тестування 100% покриття MC/DC.

Стандарт безпеки програмного забезпечення «Common Criteria» вимагає використання формальних методів для його найвищого критичного рівня (EAL7).

Статичний аналіз також рекомендується або вимагається деякими стандартами. Його виконання за допомогою інструментів абстрактної інтерпретації забезпечує кращі гарантії відсутності помилок під час виконання.

Використання формальних методів вимагає відповідних інструментів.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 24   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Було розроблено багато інструментів для деяких конкретних випадків, і не існує універсального інструменту, який би охоплював усі типи формальних методів [15]. Можна також сказати, що зрілість інструментів для кожного формального методу різна. Інструменти на основі перевірки моделей для програмного забезпечення менш надійні, ніж інструменти статичного аналізу, і потребують удосконалення, щоб охопити більший фрагмент програм. Інструменти дедуктивного доведення також повинні покращити допомогу користувачам у написанні формальних специфікацій та інваріантів, а також у наданні пояснень, коли цілі неможливо довести.

Важливим моментом для галузі під час впровадження нової технології є вартість. Можна стверджувати, що використання формальних методів підвищує якість програмного забезпечення і таким чином знижує загальну вартість перевірки та верифікації (V&V), але це важко виміряти. Деякі компанії використовують формальні методи в поєднанні з тестуванням, щоб впровадити їх у діяльність, з якою знайомі інженери. Таким чином, можна виміряти зниження витрат.

#### **1.4 Висновки по розділу**

У першому розділі було розглянуто теоретичні основи формальної верифікації програмного забезпечення. Визначено основні поняття та принципи, що лежать в основі формальних методів перевірки коректності програмних систем. Проаналізовано сучасні підходи та практики застосування формальної верифікації у розробці ПЗ. Також здійснено класифікацію та характеристику основних формальних методів, що дозволило систематизувати їх особливості та сфери застосування. Отримані результати формують теоретичну базу для подальшого дослідження інструментів та практичних методів верифікації.

### **РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ ФОРМАЛЬНОЇ ВЕРИФІКАЦІЇ**

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 25   |

## 2.1 Сучасний стан та тенденції розвитку формальної верифікації

Щоб краще зрозуміти тенденції та досягнення у вирішенні проблем, визначених у вищезазначених статтях, ми провели опитування, щоб зібрати інформацію в узгодженому форматі з низки промислових проектів, у яких, як відомо, використовувалися формальні методи. Ми надіслали відкрите запрошення якомога більшій кількості громад взяти участь у нашому опитуванні. Це могло призвести до упередженості опитування в бік тих, з ким ми маємо найтісніші контакти; наприклад, у нас мало внесків щодо перевірки моделей [16]. Незважаючи на це, однаковий спосіб збору даних дозволяє порівнювати проекти та дає певне уявлення про сучасну практику та довгострокові тенденції у використанні формальних методів.

За допомогою структурованої анкети було зібрано дані з листопада 2007 року по грудень 2008 року щодо 62 промислових проектів, відомих з опублікованої літератури, списків розсилки та особистого досвіду, в яких використовувалися формальні методи. Опитувані проекти були (у порядку спадання) з Європи, Північної Америки, Південної Америки, Австралії та Азії. Якщо особа мала досвід участі в кількох проектах, то для кожного проекту заповнювалися окремі анкети. У 56 проектах дані збиралися безпосередньо від осіб, які брали участь у цих проектах, а в решті шести випадках ми використовували інформацію, доступну в літературі.

На рисунку 2.1 представлено області застосування, до яких стосувалися проекти. Найбільшою окремою областю застосування був транспорт, за ним ішов фінансовий сектор. Іншими основними секторами були оборона, телекомунікації, офісна справа та адміністрування. Інші області, з якими було надано лише одну або дві відповіді, були: ядерна енергетика, охорона здоров'я, побутова електроніка, космос, семантична мережа, планування ресурсів, автоматизоване паркування автомобілів, вбудоване програмне забезпечення, інженерія та виробництво. Близько 20% відповідей додатково вказували, що проекти стосуються самих інструментів розробки програмного забезпечення, таких як

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 26   |

операційні системи, компілятори та CASE-інструменти, а ще 10% стосувалися обчислювальних застосувань у цій області, таких як високопродуктивні обчислення, оптимізація коду виконання, реплікація файлових систем, контроль доступу, протоколи зв'язку та проектування мікрокомп'ютерів.

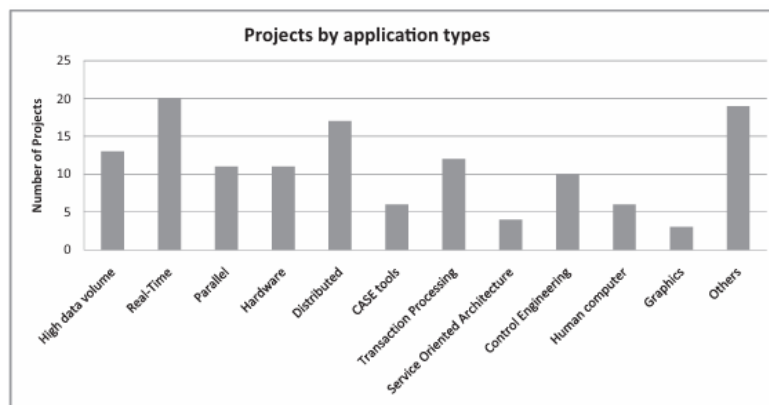


Рисунок 2.1 - Типи додатків.

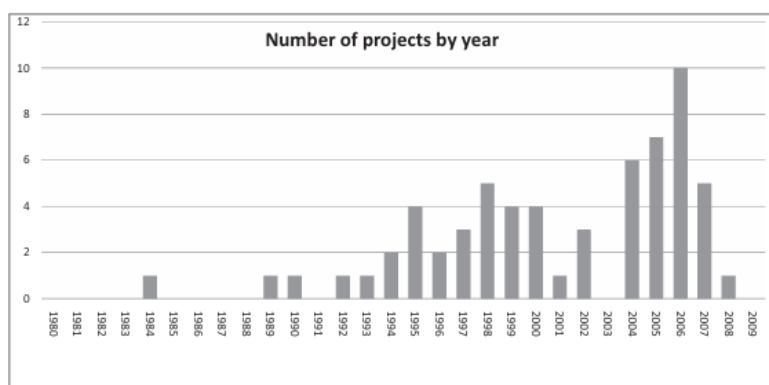


Рисунок 2.2 - Дата початку проекту

Типи застосунків представлені на рисунку 2.2. Найбільшими групами були застосунки реального часу, розподілені застосунки, обробка транзакцій та застосунки для великих обсягів даних. Інші включали паралельне програмування, апаратне забезпечення, інженерію керування, HCI, сервісно-орієнтовані обчислення та графіку. Стандарти сертифікації були вказані як такі, що застосовуються, у 30% відповідей, зокрема IEC 61508 Міжнародної електротехнічної комісії, а також Загальні критерії та рівень Е6 Великої Британії для оцінки безпеки інформаційних технологій. Інші включали CENELEC EN50128 для залізниць, DO-178B рівня А для авіоніки, оборонні стандарти 00-55

та 00-56 Міністерства оборони Великої Британії для програмного забезпечення, а також стандарт IEEE 754 для чисел з плаваючою комою.

На рисунку 2.3 представлено дати початку проектів. З урахуванням того факту, що поточні проекти навряд чи будуть представлені в опитуванні, здається, що частота проектів в останні роки вища, хоча це може бути просто відображенням способу збору даних, оскільки новіші проекти є більш чутливими до змін.

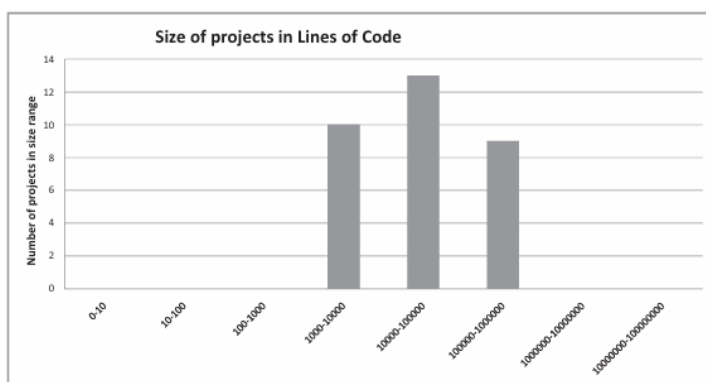


Рисунок 2.3 - Обсяг проекту в рядках коду

Як показано на рисунку 4,50% респондентів вказали розмір програмного забезпечення з точки зору рядків коду. Розподіл з них був приблизно однаковим за логарифмічною шкалою між 1-10 KLOC, 10-100 KLOC та 100-1000 KLOC. На рисунку 2.4 представлено показники використання різних методів, таких як специфікація та моделювання, виконання (інтерпретація та тестування специфікації), перевірка (специфікації та моделі) [17]. Ми шукали залежності між областю застосування та типом розробленого програмного забезпечення, але виявили лише одну значну кореляцію, яка полягала у високій представленості програмного забезпечення для обробки транзакцій у фінансовій сфері (точний тест Фішера, двосторонній,  $p < 0,01$ ). Без порівняння із загальними даними, що стосуються типу програмного забезпечення.

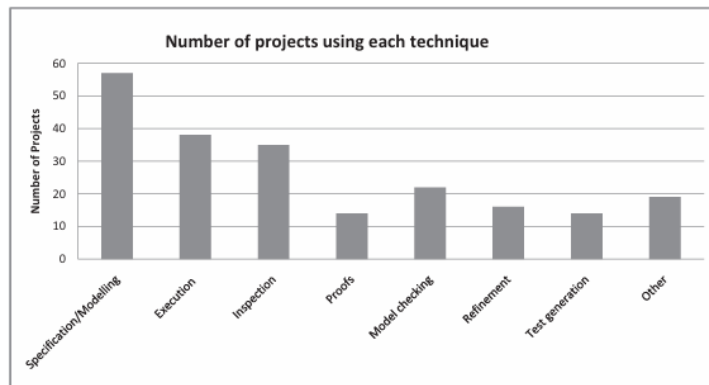


Рисунок 2.4 - Використані методи

У прикладній області це можна вважати відображенням тенденції в програмному забезпеченні в цілому, а не пов'язаним із застосуванням формальних методів. Аналогічно, не було виявлено жодної суттєвої залежності між методами, що використовуються для різних прикладних областей, і спостерігалось лише кілька, досить незначних, кореляцій між методами, що використовуються для різних типів програмного забезпечення. Ці останні кореляції демонструють вище середнього використання перевірки моделей у побутовій електроніці та інспекції в програмному забезпеченні для обробки транзакцій (точний тест Фішера, двосторонній,  $p = 0,03$  для кожного).

З іншого боку, після співвіднесення використаних методів з датою проекту, ми виявили, що використання перевірки моделей значно зросло з 13% у 1990-х роках до 51% у цьому десятилітті. Це дуже суттєва зміна (точний тест Фішера, двосторонній,  $p = 0,003$ ). Натомість, за цей період не було виявлено суттєвих змін у використанні доказів, уточнень, виконання або генерації тестових випадків.

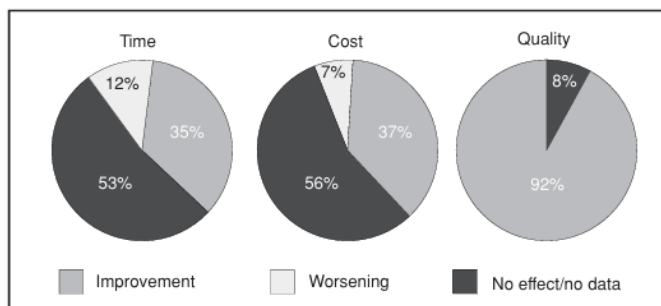


Рисунок 2.5 - Чи вплинуло застосування формальних методів на терміни,  
витрати та якість

Коли респондентів попросили вказати, які ролі входять до команди проекту з попередньо визначеного списку (управління продуктом, управління програмами, розробка, користувацький досвід, тестувальник, управління релізами, архітектор, інше), найбільше відповідей було «тестувальник» (50%) та «архітектор» (46%), а всі інші відповіді були менше 10%. Щодо попереднього досвіду у використовуваних методах, 40% повідомили про «значний попередній досвід», 45% повідомили про «деякий попередній досвід», а 22% повідомили про «відсутність попереднього досвіду». Загальна сума перевищує 100%, оскільки деякі респонденти повідомили про змішані команди з більш ніж однієї категорії. З тих, хто повідомив про «відсутність попереднього досвіду», половина була в змішаній команді з більш досвідченими колегами; інші впроваджували методи в команді, яка раніше не мала досвіду в цих методах. Щодо навчання, 33% повідомили про «не пройшли навчання», 54% повідомили про «деяке пройшло навчання» та 5% повідомили про «значне пройшло навчання».

На рисунку 2.6 показано загальний вплив використання формальних методів на час, вартість та якість. Вплив на час, витрачений на виконання роботи, був у середньому позитивним. Утричі більше респондентів повідомили про скорочення часу, а не про його збільшення. Багато відповідей вказували на те, що важко оцінити вплив на витрачений час, хоча деякі зазначили збільшення часу на етапі специфікації, яке могло бути компенсоване, а могло й ні, зменшенням часу пізніше. Наприклад: «Важко судити, але фаза ретельного уточнення, ймовірно, додала часу, хоча, можливо, й заощадила час пізніше».

«Тривала фаза специфікації, ймовірно, додалася до витраченого часу».

«Незначне збільшення часу, витраченого на ранніх етапах проектування... було багаторазово компенсовано під час системної інтеграції та тестування».

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 30   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

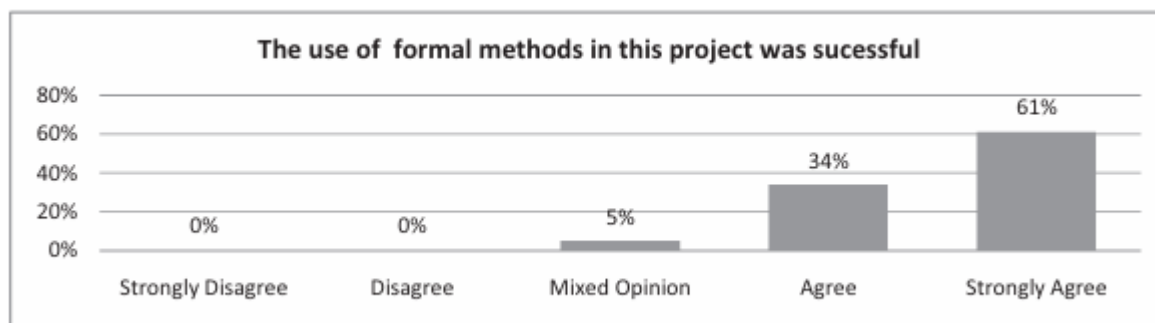


Рисунок 2.6 - Загальна задоволеність використанням формальних методів.

Серед випадків, коли висловлювалася думка, у п'ять разів більше проектів повідомили про зниження витрат, ніж тих, де повідомили про збільшення витрат [18]. Деякі помітні коментарі щодо впливу на вартість включають:

«Ми спостерігали суттєве покращення продуктивності після завантаження підсистеми генерації коду інструменту завдяки використанню генерації коду зі специфікацій...»

«Збільшення вартості було значною мірою пов'язане з відсутністю точної, повної інформації про необхідну зовнішню видиму поведінку програмного продукту. Після реалізації коду необхідна поведінка стала зрозумілою, а застосування формальної специфікації та формальної верифікації було відносно простим. Єдиною дорогою частиною процесу верифікації коду було анотування коду з попередніми та постумовами. Після того, як анотований код став доступним, показати відповідність між анотованим кодом та абстрактною специфікацією необхідної поведінки було легко [19]. Цей останній процес включав додавання додаткових анотацій та виправлення неправильних анотацій. Під час цього процесу абстрактна специфікація та необхідні властивості безпеки мало змінилися».

Натомість, респонденти вважають, що використання формальних методів покращило якість, причому 92% усіх випадків повідомляли про підвищення якості порівняно з іншими методами, і жодного випадку не повідомляло про зниження якості. Більшість із них були пов'язані з виявленням недоліків (36%). Іншими поширеними причинами покращення були: покращення дизайну (12%),

підвищення впевненості в правильності (10%), покращення розуміння (10%) та раннє виявлення недоліків або інших проблем (4%).

Як показано на рисунку 2.6, респонденти загалом позитивно оцінили успішне використання формальних методів, хоча через упередженість у виборі, описану вище, можна було б очікувати, що зацікавлені сторони, які не бачать цінності у використанні формальних методів, будуть недостатньо представлені серед відповідей. Рисунок 2.8 показує, що респонденти загалом були задоволені формальними методами, що використовуються в їхніх проектах, тоді як рисунок 2.9 показує, що у 9% проектів застосовані інструменти не повністю виправдали очікування [20]. Нарешті, рисунок 10 демонструє, що більшість респондентів бажають знову використовувати подібну технологію в нових проектах.

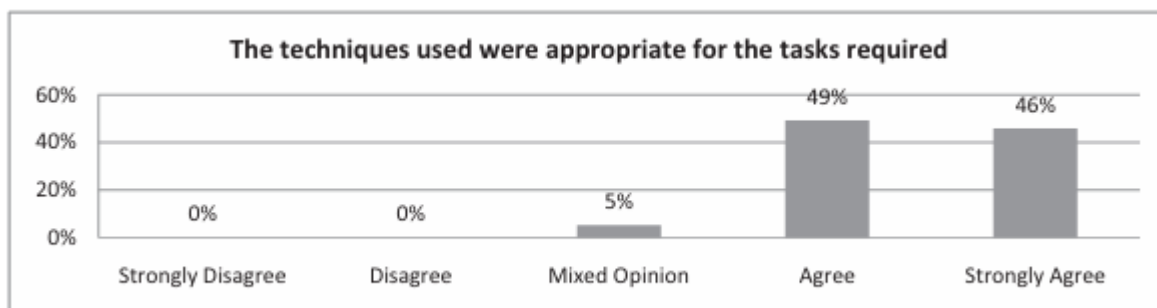


Рисунок 2.7 - Загальна задоволеність застосованими методами.

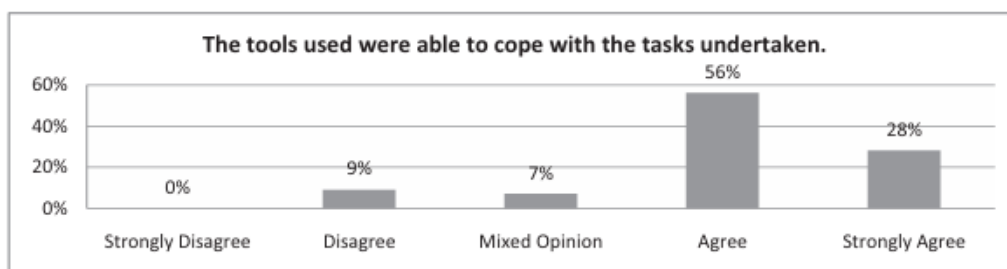


Рисунок 2.8 - Загальна задоволеність використовуваними інструментами

*Використання користувачами.* Опитування показує, що використання формальних методів розподілене по широкому спектру прикладних областей, включаючи значну кількість пов'язаних відображає той факт, що багато використовуваних інструментів є універсальними, як і мови програмування, тому дослідження універсальних методів (тобто незалежних від області застосування)

все ще актуальні та повинні продовжуватися разом з розробкою більш спеціалізованих генераторів.

*Інструменти та методи.* Деякі проекти передбачали масштабний та всебічний аналіз, значне ручне втручання у формальний аналіз, і, отже, суттєві зміни до існуючих неформальних методів. З іншого боку, деякі керівники проектів були більше зацікавлені в широкому, а не в глибокому аналізі, сподіваючись отримати певну користь від незначних або відсутніх ручних зусиль.

Зрозуміло, що закон Мура мав дуже значний вплив на доступність обчислювальної потужності протягом двох десятиліть, охоплених проектами, про які йдеться в опитуванні, причому типові обчислювальні та сховищні ресурси настільних комп'ютерів зросли, можливо, приблизно в 10 000 разів за цей період. Разом зі значним теоретичним прогресом це мало значний вплив на масштаб проблеми, яку можна вирішити за допомогою автоматизованих методів, таких як перевірка моделей, і, здається, свідчить про постійне зростання рівня автоматизації та перехід до формальних методів, які з часом зникають «під капотом». Така тенденція уособлюється наступною цитатою з проекту, який використовував значний рівень перевірки моделей:

«Щоб бути корисними під час розробки продукту, формальні методи повинні надавати відповіді за секунди або хвилини, а не за дні. Перевірка моделей може робити це дуже ефективно, якщо застосовувати її до правильних типів системних проєктів. Щоб скористатися цим, перевірка моделей має бути тісно інтегрована в комерційні інструменти проектування та верифікації, які розробники вже використовують».

Якою мірою і в які терміни математичне доведення піддається закону Мура? Один з респондентів розмірковував про доступні інструменти наступним чином:

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 33   |

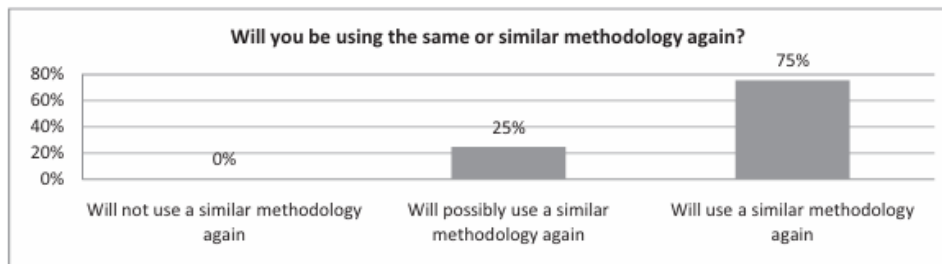


Рисунок 2.9 - Намір знову використовувати формальні методи.

*Вплив на час, вартість та якість.* З тих, хто висловив думку, значно більше проектів мали скорочення часових рамок та витрат, а також покращення якості завдяки використанню формальних методів. Але, можливо, дивно, що в більшості опитаних проектів немає даних про відносний час та витрати. З іншого боку, загальна задоволеність їх використанням серед респондентів є очевидною.

«Точних вимірювань не проводилося. За емпіричним правилом було оцінено, що зусилля на проектування перевищили звичайні зусилля приблизно на 10%. Ймовірно, витрати на обслуговування різко скоротилися. Але це судження ґрунтується лише на гіпотетично припущених рівнях помилок програмного забезпечення, розробленого традиційним способом».

## 2.2 Інструментальні засоби та практичні приклади застосування

Ми представляємо серію проектів промислових формальних методів, обраних для демонстрації зрізу робіт за останні два десятиліття. Акцент робиться на економічно ефективній розробці верифікованих систем, а застосування включають мікрокод та прошивку, залізничі, національну інфраструктуру, смарт-картки та біометричний додаток безпеки. Кожен проект включено до огляду, описаного в останньому розділі, але ми починаємо з огляду ряду інструментів верифікації, про які повідомляється в літературі, що використовуються в промисловості.

Існує компроміс між рівнем автоматизації, якого можна досягти під час перевірки, та складністю виконаного аналізу. З одного боку, є статичні

аналізатори, які виявляють потенційні дефекти програми, такі як використання змінних перед визначенням, константні умови та, можливо, вирази поза діапазоном [21]. Прикладами є евристичні інструменти, такі як 30-річна утиліта UNIX lint, яка позначає потенційні дефекти в програмах на C, та новіші інструменти, такі як Splint, яка забезпечує легку статичну перевірку за допомогою анотацій. Такі інструменти аналізу програм використовуються в промисловості; для опису інструментарію застарілого коду Microsoft з твердженнями для використання в масовому регресійному тестуванні та для короткого огляду використання інструментів у Microsoft. Сучасні аналізатори, такі як PREfix, а PREfast може аналізувати багатомільйонні рядкові програми на C та C++ на наявність потенційних розіменувань нульових вказівників, неправильного розподілу та звільнення пам'яті, використання неініціалізованих змінних, простих помилок стану ресурсів, неправильного використання бібліотек та проблемних ідіом програмування. Такі інструменти часто використовують формально некоректні аналітичні методи і тому можуть давати хибнопозитивні результати (виявляючи хибні помилки), хоча докладаються зусилля для їх мінімізації [22]. З іншого боку, інструмент SLAM може сертифікувати програму як вільну від певного типу помилок, перевіряючи, чи використання API відповідає певним правилам послідовності, заданим користувачем. Він використовувався для пошуку багатьох помилок у драйверах пристроїв Windows і розповсюджується як Static Driver Verifier, що входить до комплекту розробки Windows. Інструмент ESP схожий на SLAM, але для аналізу величезних баз коду він порівнює точність із великомасштабною оброблюваністю; він використовувався для пошуку переповнення буфера у великомасштабному системному коді та для перевірки властивостей безпеки ядра ОС.

Далі по спектру верифікації є такі інструменти, як засоби перевірки моделей та абстрактні інтерпретатори. Перевірка моделей на основі автоматів SPIN використовувалася для пошуку логічних помилок проектування в розподілених системах, таких як ті, що використовуються в космічних місіях Deep Space 1, Cassini, Mars Exploration Rovers та Deep Impact. SPIN перевіряє логічну

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 35   |

узгодженість специфікації, повідомляючи про блокування, невизначені отримання повідомлень, неповноту, умови гонки та необґрунтовані припущення щодо відносної швидкості процесів. Прикладом абстрактного інтерпретатора є статичний аналізатор вбудованого програмного забезпечення реального часу ASTREE, який намагається довести відсутність усіх помилок під час виконання в програмі на C. Він робить це повністю автоматично для основного програмного забезпечення для керування польотом Airbus A340 та кодів електричного керування польотом A380.

На найдальшому кінці спектру верифікації, доказувачі теорем можуть перевіряти довільні гіпотези в заданій логіці з різним ступенем автоматизації. Наприклад, Vampire - це автоматичний доказувач теорем для класичної логіки першого порядку. KIV, з іншого боку, - це інтерактивний доказувач теорем з логікою об'єктів, яку визначає користувач, і він використовувався в широкому спектрі застосувань, від верифікації протоколів медичних процедур до верифікації протоколів для смарт-карт.

Серія мікропроцесорних чіпів Transputer була розроблена спеціально для паралельної обробки. Описується розробка одного з Transputer: блоку обчислень з плаваючою комою T800, який поєднував 32-бітний процесор зі скороченим набором інструкцій, певну кількість пам'яті, чотири двонаправлені канали зв'язку та блок обчислень з плаваючою комою на одному чіпі. Його наступник, T9000, був досить складнішим, з багатшими можливостями підключення, моделлю пам'яті та конвеєрним процесором. Транспутер, заснований на технології T9000, ST20, досі дуже широко використовується в чіпсетах для телеприставок та GPS-додатків. Мовою програмування для Transputer є Оссам, простий, низькорівневий, виконуваний підмножина CSP.

Компанія Inmos почала розробляти T800 у 1986 році, використовуючи традиційний підхід, який вимагав місяців тестування, оскільки пристрої з плаваючою комою є, як відомо, складними пристроями та схильними до помилок у проектуванні. Коли обсяг необхідного тестування став зрозумілим, розпочалася робота над офіційною розробкою правильного за конструкцією пристрою з

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 36   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

плаваючою комою. Стандарт природної мови IEEE 754 для арифметики з плаваючою комою був формалізований у Z [23]. Специфікація описана в Barrett і виявила деякі проблеми у стандарті. Наприклад, стандарт вимагає, щоб діагностична інформація про результати недійсних операцій (таких як квадратний корінь з від'ємного числа) поширювалася через подальші операції. Але це не завжди можливо. Наступним завданням було показати, що пакет для обчислень з плаваючою комою, написаний мовою Оккама та використаний у попередніх трансп'ютерах, є коректною реалізацією IEEE 754. Спроба перевірки за допомогою логіки Хоара виявила помилки в операціях округлення та із залишками. Пізніше Барретт зауважив Гіббонсу, що «лише дуже малий клас тестових векторів міг би виявити помилки». З внесеними виправленнями пакет Оккама було перевірено на правильність і використано як проміжне представлення функціональності необхідного мікрокоду. Він був занадто абстрактним, щоб бути безпосередньо корисним для розробки апаратного забезпечення, тому для застосування законів програмування Оккама для створення еквівалентної програми мікрокоду було використано систему перетворення Оккама.

Розробка одиниці обчислень з плаваючою комою, від природної мови до кремнію, була щонайменше на три місяці швидшою, ніж альтернативна неформальна розробка, яка відбувалася на даний момент. За оцінками, щомісячна затримка виробництва коштувала 1 мільйон доларів США. В мікрокоді з плаваючою комою було знайдено рівно дві помилки. Перша була внесена програмою перекладу, яка перетворювала мікрорівневий код Оккама в код асемблера для чіпа; друга була внесена шляхом ручної оптимізації цього коду асемблера [24]. Оксфордський університет та Inmos спільно отримали Королівську премію за технологічні досягнення у 1990 році «для визнання та заохочення видатних досягнень у розвитку технології процесів або продуктів у Великій Британії».

У 1988 році компанії GEC Alsthom, MATRA Transport та RATP (паризький оператор громадського транспорту) розпочали роботу над комп'ютеризованою

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 37   |

системою сигналізації для керування приміськими поїздами регіональної експрес-мережі RER у Парижі. Метою проекту було збільшення мережевого трафіку на 25%, зберігаючи при цьому існуючий рівень безпеки. Отримана система SACEM із вбудованим апаратним та програмним забезпеченням була поставлена в 1989 році та з моменту її впровадження контролює швидкість усіх поїздів на лінії A RER у Парижі, що охоплює сім мільярдів пасажирських поїздок.

Програмне забезпечення SACEM складається з 21 000 рядків коду Modula-2, з яких 63% вважаються критично важливими для безпеки та пройшли офіційну специфікацію та верифікацію. Специфікацію було створено в В, а докази виконувалися інтерактивно з використанням автоматично згенерованих умов верифікації для коду. Зусилля з валідації всієї системи (включаючи процедури, не критичні для безпеки) становили близько 100 людино-років. Система стала безпечнішою в результаті офіційної специфікації та верифікації. Команда проекту повідомила про труднощі у спілкуванні між верифікаторами та інженерами сигналізації, які не були знайомі з В-методом. Цю проблему було подолано шляхом надання інженерам французького опису, отриманого вручну з офіційної специфікації. Система SACEM детальніше описана, де представлені різні вимоги до надійності та їх реалізація. Методи забезпечення безпеки включають виявлення помилок в режимі реального часу, перевірку програмного забезпечення та відмовостійкість бортової наземної складової системи.

Abrial (розробник В) повідомляє про два додаткові залізничні проекти, виконані з використанням В: лінія 14 Паризького метрополітену, система, що використовується з жовтня 1998 року; та безпілотний шатл до аеропорту Париж-Руассі, який використовується з 2007 року. З використанням В були розроблені лише критично важливі для безпеки частини, що становлять одну третину кожної програмної системи. У таблиці 2.1 наведено основні характеристики двох систем. Оскільки лінія 14 повністю автоматична, критично важлива для безпеки частина стосується руху та зупинки поїздів, а також відкриття та закриття дверей поїздів і платформ. Для проектів лінії 14 або шатла Roissy не проводилися жодних

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 38   |

модульних тестів; їх замінили деякими глобальними тестами, які всі були успішними. Це значно знизило загальні витрати.

Таблиця 2.1

Статистика проєктів 14-ї лінії паризького метро та шатла Roissy

| Показник                                  | 14-та лінія метро<br>(Париж) | Шатл Roissy |
|-------------------------------------------|------------------------------|-------------|
| Довжина лінії (км)                        | 8.5                          | 3.3         |
| Кількість зупинок                         | 8                            | 5           |
| Інтервал між поїздами (с)                 | 115                          | 105         |
| Швидкість (км/год)                        | 40                           | 26          |
| Кількість поїздів                         | 17                           | 14          |
| Пасажирів на день                         | 350,000                      | 40,000      |
| Кількість рядків коду на Ada              | 86,000                       | 158,000     |
| Кількість рядків коду на B                | 115,000                      | 183,000     |
| Кількість доведень (proofs)               | 27,800                       | 43,610      |
| Відсоток інтерактивних доведень           | 8.1                          | 3.3         |
| Зусилля на інт. доведення (людино-місяці) | 7.1                          | 4.6         |

На початку 1990-х років Національний Вестмінстерський банк та Platform Seven розробили систему електронних готівкових операцій на основі смарт-карт, Mondex, придатну для транзакцій з низькою вартістю, подібних до готівкових, без участі третьої сторони та без витрат на транзакцію. Було надзвичайно важливо, щоб картка була безпечною, інакше гроші могли бути підроблені електронним способом, тому Platform Seven вирішила сертифікувати Mondex за одним із найвищих стандартів, доступних на той час: ITSEC Level E6, який наближається до Common Criteria Level 7. Це вимагає суворих вимог до проектування, розробки, тестування та процедур документування програмного забезпечення. Це також вимагає використання формальних методів для визначення моделі абстрактної політики безпеки високого рівня та конкретного архітектурного дизайну нижчого рівня. Це вимагає офіційного доказу відповідності між ними, щоб показати, що конкретний дизайн відповідає абстрактним властивостям безпеки.

Цільова смарт-карта платформи мала 8-бітний мікропроцесор, низьку тактову частоту, обмежений обсяг пам'яті (256 байт динамічної оперативної пам'яті та кілька кілобайт повільнішої EEPROM) та не мала вбудованої підтримки операційної системи для таких завдань, як управління пам'яттю. Живлення могло бути відключено в будь-який момент під час обробки транзакції. Logica була укладена контракт на надання специфікації та підтвердження з використанням Z. Вони не мали особливих труднощів у формалізації конкретного архітектурного проекту з існуючих напівформальних проектних документів, але завдання створення абстрактної моделі політики безпеки, яка б одночасно фіксувала бажані властивості безпеки (зокрема, що «не створюється цінність» і що «вся цінність враховується») та довідково відповідала специфікації нижчого рівня, було набагато складнішим.

Дуже невелика зміна в проекті значно спростила б абстракцію, але вважалася занадто дорогою для реалізації, оскільки паралельна робота з впровадження вже далеко зайшла за цей етап. 200-сторінкове підтвердження було виконано вручну та виявило невеликий недолік в одному з другорядних протоколів; Це було представлено Платформі Сім у вигляді сценарію, що ставить під загрозу безпеку [25]. Оскільки це становило реальну проблему безпеки, конструкцію було змінено, щоб її виправити. Ретельні докази, які проводилися, виконувалися вручну, що було прийнято на той час для контролю витрат. Нещодавня робота (про яку повідомляється нижче) показала, що це було надмірно обережно, і що закон Мура ще більше змістив чашу терезів на користь економічно ефективної механічної перевірки.

Platform Seven - це компанія-розробник програмного забезпечення, що відокремилася від NatWest; зараз вона належить DataCard, основному акціонеру Gemplus, ще однієї компанії з виробництва смарт-карт.

У 1999 році Mondex отримав сертифікат ITSEC рівня Е6: ставши першим продуктом в історії, який отримав такий сертифікат. В рамках процесу ITSEC Е6 вся розробка Mondex додатково пройшла ретельне тестування, яке також було

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 40   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

оцінено. У жодній частині системи, що піддавалася використанню формальних методів, не було виявлено помилок.

MonDEX було відроджено у 2006 році як пілотний проект для Grand Challenge in Verified Software [26]. Головною метою було перевірити, як змінився сучасний стан механічної верифікації за десять років. Вісім груп прийняли це завдання, використовуючи такі формальні методи (з посиланнями на повне обговорення видів аналізу, що виконувалися в кожному випадку): Вартість механізації Z-доказів початкового проекту становила 10% від початкової вартості розробки, тому не домінувала у витратах, як вважалося спочатку. Цікаво, що майже всі методи, що використовувалися в пілотному проекті MonDEX, досягли однакового рівня автоматизації, створюючи подібну кількість умов перевірки та вимагаючи подібних зусиль.

Одним з їхніх найперших експериментів було визначення та вдосконалення мікро-виконавчого елемента реального часу p.RTE в нотації RAISE, RSL. Цей проект не був успішним: мова вважалася занадто складною та потребувала значного навчання, а єдиними доступними інструментами були засоби перевірки синтаксису та типів, без підтримки розробки та управління доказами.

Подальший експеримент з SRI International, спонсорований NASA, формально перевіряв мікрокод мікропроцесора AAMP5 за допомогою PVS. AAMP5 – це власний мікропроцесор, який широко використовується в продуктах Rockwell Collins [27]. Він має стекову архітектуру, великий набір інструкцій, широко використовує мікрокод, має конвеєрну архітектуру та складні обчислювальні блоки. Він містить приблизно 500 000 транзисторів з продуктивністю між Intel 386 та 486. Rockwell Collins специфікував AAMP5 як на рівні передачі регістрів, так і на рівні набору інструкцій, з відношенням пошуку між ними, щоб довести правильність інструкцій мікрокоду. Головний урок, винесений з проекту AAMP5, полягав у тому, що технічно можливо довести правильність мікрокоду, і що їхні інженери могли читати та записувати формальні специфікації.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 41   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Під час створення специфікації в мікрокоді AAMP5 було виявлено дві помилки, і це переконало їх, що є цінність у простому написанні формальної специфікації. Але вони також переконали себе, що механічні докази правильності забезпечують дуже високий рівень гарантії. Вони зробили це, додавши дві дуже тонкі помилки в мікрокод, який вони надали SRI, а потім чекаючи, чи знайдуть вони їх. SRI справді виявила помилки за допомогою систематичного процесу: єдиний спосіб не знайти помилок - це не виконати докази.

Найбільшою проблемою проекту AAMP5 була занадто висока вартість: понад 300 годин на інструкцію. Ця цифра, схоже, була завищена з різних причин, включаючи круту криву навчання при першому використанні PVS та необхідність розробки багатьох допоміжних прикладних теорій. Вони знали, що наступного разу їхні витрати різко знизяться, але не могли передбачити, наскільки, тому провели другий експеримент – перевірку мікрокоду в AAMP-FV, знову спонсорований NASA та SRI. Метою було продемонструвати різке зниження вартості шляхом повторного використання інфраструктури та досвіду AAMP5.

Важливо, що проект AAMP-FV підтвердив, що досвід, отриманий у рамках проекту AAMP5, може бути використаний для значного зниження вартості формальної верифікації. З 80 інструкцій AAMP-FV 54 були визнані правильними, а вартість їхньої верифікації знизилася майже на порядок порівняно з проектом AAMP5 [28]. Але оскільки намагалися використовувати складніші інструкції, методи доведення, розроблені вперше в рамках проекту AAMP5, виходили з ладу, і довелося розробляти нові підходи. Цей етап розвивався радше як дослідницький проект, з крутою кривою навчання та неочікуваними затримками. Одним з головних внесків проекту AAMP-FV була розробка методів обробки складного мікрокоду.

Найновіший проект стосується процесора AAMP7, у мікроархітектурі якого вбудоване ядро розділення. Роквелл виконав доведення в ACL2, що рядкова модель мікрокоду дотримується політики безпеки щодо того, як розділам дозволено взаємодіяти. Складність таких доказів полягає в складності роботи зі структурами даних, що містять вказівники [29]. Робота отримала сертифікацію

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 42   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Агентства національної безпеки як пристрій з кількома незалежними рівнями безпеки для використання в криптографічних програмах на рівні EAL-7 Загальних критеріїв.

Airbus використовував SCADe протягом останніх десяти років для розробки контролерів DO-178B рівня А для серії А340-500/600, включаючи вторинний комп'ютер керування польотом та блок керування електричним навантаженням. Короткий огляд цих та інших промислових застосувань. Esterel Technologies повідомляє про такі переваги: (i) Значне зменшення помилок кодування: для проекту Airbus А340 70% коду було згенеровано автоматично. (ii) Коротші зміни вимог: пакет інструментів SCADe керує еволюцією системної моделі в міру зміни вимог, а в проекті Airbus А340 зміни вимог здійснювалися швидше, ніж раніше, з покращеною відстежуваністю. (iii) Значне підвищення продуктивності: Airbus повідомив про значні здобутки, незважаючи на те, що кожен новий проект Airbus вимагає вдвічі більше програмного забезпечення, ніж його попередник.

Досягнувши значної економії на загальному циклі проектування, Airbus застосував SCADe для проектів А380, де більшість бортових комп'ютерів, розроблених Airbus та її постачальниками, використовують технологію SCADe. Пакет SCADe використовується для розробки більшості критично важливого бортового програмного забезпечення А380 та А400М, а також для вторинної системи керування польотом літака А340-500/600, який використовується з серпня 2002 року. Система керування та відображення кабіни пілотів А380 та А400М, а також дисплей бортової навігаційної системи аеропорту були розроблені з використанням SCADe Display, орієнтованого на специфікацію графічних інтерфейсів.

Maeslant Kering – це рухомий бар'єр, що захищає порт Роттердама від затоплення внаслідок несприятливих погодних та морських умов. Рішення про розгортання та повторне відкриття бар'єру приймається на основі метеорологічних даних за допомогою комп'ютерної системи. Відповідно до міжнародного стандарту ІЕС 61508, застосунок був віднесений до рівня цілісності

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 43   |

безпеки 4, для якого використання формальних методів «настійно рекомендується». Розробники (CMG) були навмисно обережними у визначенні цілей для розгортання формальних методів. Технологія вдосконалення не вважалася доцільною для системи такого масштабу. Також вважалося, що впровадження кількох нових методів в одному проекті є занадто ризикованим варіантом. Тому підхід полягав в інтеграції технології моделювання та верифікації в рамках звичайної траєкторії проектування. Підхід використовував формальне моделювання та верифікацію на етапах аналізу, проектування та реалізації розробки системи. Основною темою формальної роботи була підсистема прийняття рішень та її інтерфейси з навколишнім середовищем.

Дані та операції були змодельовані в Z, і це було вбудовано в модель Promela, що описує керування, а проекти були перевірені за допомогою перевірки моделі SPIN. Promela та SPIN були обрані через попередній досвід розробників з цим інструментом та вважану простоту використання, що означало, що інженери CMG могли виконувати більшу частину роботи з моделювання та аналізу без необхідності залучення зовнішньої допомоги. Зв'язок Promela/Z був *ad hoc* і сам по собі не мав формальної семантики.

У остаточному детальному проекті було визначено 29 програм з 20 000 рядків Z. Реалізація була здійснена шляхом систематичного кодування в безпечній підмножині C++. Формальної перевірки коду не було. Остаточна реалізація становила 200 KLOC для операційної системи та 250 KLOC допоміжного коду (симулятори, тестові системи тощо). Неформальне, але систематичне тестування з моделі Z призвело до покриття 80-90% коду для тестування "чорної скриньки", а решта - для тестування "білої скриньки". Проблеми, що виникли під час процесу, були зареєстровані; близько 85% з них виникли на етапах розробки та близько 15% під час тестування надійності та приймальних випробувань. Залишкові недоліки були незначними. Близько половини недоліків, виявлених під час розробки, були в коді або дизайні, решта - це недоліки в специфікаціях тестів, параметрах конфігурації або документації.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 44   |

Досвід був загалом позитивним, а звіт навмисно повторював Сім міфів Холла. Розробники вважали, що програмне забезпечення значно кращої якості, ніж було б досягнуто в іншому випадку, і що ця перевага в якості стосуватиметься також некритичних систем. Було відзначено значний зсув у зусиллях та витратах у бік фаз специфікації та проектування. Автори зазначили, що навички абстракції є важливою частиною процесу моделювання, і що легкість побудови формальних моделей не повинна спокушати інженерів від присвячення зусиль вибору відповідних абстракцій.

У системі, розробленій з використанням формальних методів, не було повідомлено про жодних суттєвих дефектів. Про оновлення в середині життєвого циклу було повідомлено у 2008 році, і розробка наступної програми продовжуватиме використовувати формальні методи.

Проект Tokeneer ID Station (TIS), що здійснювався Praxis High Integrity Systems спільно з SPRE Inc. під керівництвом NSA (Агентства національної безпеки), показав, що можливо створювати високоякісні системи з низьким рівнем дефектів, що відповідають вимогам Загальних критеріїв рівня оцінювання 5 (EAL5) [30]. Система Tokeneer була спочатку розроблена NSA для дослідження різних аспектів біометрії в контролі доступу та складається з безпечного анклаву з контрольованим фізичним входом. У безпечному анклаві розташовано низку робочих станцій, користувачі яких мають токени безпеки (наприклад, смарт-картки) для отримання доступу до робочих станцій. Користувачі пред'являють свої токени безпеки до зчитувача зовні анклаву, який використовує інформацію на токені для проведення біометричних тестів (наприклад, зчитування відбитків пальців) користувача. Якщо користувач проходить ці тести, то двері до анклаву відчиняються, і користувачеві дозволяється вхід. Під час входу система додає до токена безпеки інформацію про авторизацію, яка точно описує тип доступу, дозволений для цього відвідування анклаву, наприклад, час роботи, допуск до секретної інформації та ролі, які можна виконувати.

У 2002 році компанія Praxis завершила MULTOS, важливий проект з

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 45   |

використанням формальних методів [31]. MULTOS - це багатопрограмна операційна система, яка дозволяє розміщувати кілька програм на одній смарт-картці. Успіх цього проекту призвів до пропозиції NSA щодо демонстраційного проекту з безпечної розробки програмного забезпечення.

Таблиця 2.2

Tokeneer: Розмір та продуктивність

|              | Розмір/Рядки коду (LOC) |                             | Продуктивність (LOC/день) |          |
|--------------|-------------------------|-----------------------------|---------------------------|----------|
|              | Ada                     | Анотації та коментарі SPARK | Під час кодування         | Загальна |
| Ядро TIS     | 9,939                   | 16,564                      | 203                       | 38       |
| Допоміжне ПЗ | 3,697                   | 2,240                       | 182                       | 88       |

Praxis займався цією розробкою протягом дев'яти місяців у 2003 році, і зрештою було схвалено до публікації доповідь на конференції. Матеріали проекту Tokeneer були опубліковані відповідно до Угоди про передачу технологій NSA у липні 2008 року як внесок у Великий конкурс «Верифіковане програмне забезпечення».

У рамках проекту TIS було перероблено один компонент системи Tokeneer. Для полегшення розробки замість фактичних пристроїв TIS використовувалися симулятори пристроїв TIS, реалізовані незалежними консультантами з надійності (SPRE Inc.). Основна функціональність системи була написана на SPARK, підмножині Ada з супутнім набором інструментів, спеціально розробленим для написання програмного забезпечення для високоінтегративних програм. Допоміжне програмне забезпечення для її взаємодії з імітованими периферійними пристроями було написано на повній мові Ada. У таблицях 2.2 та 2.3 зафіксовано розмір проекту, продуктивність та зусилля. Проект вимагав 260 людино-днів роботи, включаючи трьох співробітників, які працювали неповний робочий день, протягом дев'яти місяців. Кількість дефектів, виявлених у системі під час незалежного тестування надійності системи та з моменту поставки у 2003 році, становить два. Один з них

був виявлений шляхом перевірки коду після завершення проекту.

Таблиця 2.3

Tokeneer: Розподіл за фазами проекту

| Фаза проєкту                  | Зусилля (%) | Зусилля (людино-дні) |
|-------------------------------|-------------|----------------------|
| Управління проєктом           | 11          | 28.6                 |
| Вимоги                        | 10          | 26.0                 |
| Специфікація системи          | 12          | 31.2                 |
| Проектування основних функцій | 15          | 39.0                 |
| Код та доведення ядра TIS     | 29          | 75.4                 |
| Тестування системи            | 4           | 10.4                 |
| Допоміжне ПЗ та інтеграція    | 16          | 41.6                 |
| Приймання                     | 3           | 7.8                  |
| <b>Усього</b>                 | <b>100</b>  | <b>260.0</b>         |

Другу помилку було виявлено компанією Praxis під час дослідження невиконаного зобов'язання щодо доказів. Вона стосується коду, який перевіряє цілочисельні значення, зчитані з файлу. Ці цілі числа представляють секунди та перетворюються на десяті частки секунди, що може спричинити помилку переповнення. Інструменти SPARK використовувалися для створення умов перевірки часткової коректності та помилок під час виконання, але без побічних умов, пов'язаних з Ada, Aos Overflow\_Check, через обмежену можливість скидання таких віртуальних коефіцієнтів (VC). Після вдосконалення інструментів, віртуальні коефіцієнти були повторно згенеровані та перевірені, що виявило помилку. Розробники зазначають, що дефект не був виявлений під час жодного тестування під час оригінального проекту, або під час будь-якого використання чи спроби аналізу системи з моменту початкової поставки.

Команда тестувальників SPRE Inc. фактично виявила два збої в рамках свого режиму тестування: обидва стосувалися відсутніх елементів у посібнику користувача, а не помилок у TIS Core. Запис у Таблиці для системного тестування не включає внесок SPRE Inc. у тестування [32]. Оцінюють, що більш репрезентативним показником може бути 25%. Функціональна специфікація, написана мовою Z та пояснювальною англійською мовою, складається приблизно зі 100 сторінок.

Завдання, поставлене NSA, полягало в розробці системи відповідно до вимог Загальних критеріїв для EAL5. Фактично, Praxiс перевищила вимоги EAL5 у низці областей, оскільки виявила, що використання деяких більш суворих методів було насправді економічно ефективним. Praxiс відповідала критеріям EAL5 в основній частині основної роботи з розробки, що охоплювала керування конфігурацією, управління несправностями та тестування. Вони перевищили EAL5, досягнувши рівнів EAL6 або EAL7, в областях розробки, що охоплюють специфікацію, проектування, реалізацію та відповідність між представленнями. Інші аспекти, такі як доставка та операційна підтримка, залишалися поза сферою застосування.

«FeliCa» - це технологія безконтактних IC-карт, що широко використовується в Японії, розроблена та просувається корпорацією Sony. Мобільні телефони з чіпами FeliCa можуть служити електронними гаманцями, проїзними квитками, ключами від дверей тощо. FeliCa Networks Inc. вирішила використовувати VDM++ та VDMTools для розробки прошивки для IC-чіпа нового покоління, що містить нові функції, але тим не менш працює відповідно до суворих вимог до синхронізації, передбачених попередніми поколіннями.

Проект тривав три роки і три місяці, у ньому брали участь від 50 до 60 осіб із середнім віком трохи більше 30 років. На момент запуску проекту жоден із учасників не мав знань або досвіду роботи з формальним методом. Навчання з VDM++ (японською мовою) для команди розробників було проведено компанією CSK [33]. Крім того, протягом усього проекту використовувався зовнішній консультант з VDM від CSK Systems. Нова версія прошивки згодом була випущена мільйонами мікросхем.

Велика кількість тестових випадків VDM++ була розроблена, а потім виконана за допомогою інтерпретатора VDMTools. Використовуючи засіб аналізу покриття тестами VDMTools, стало можливим відобразити інформацію про покриття тестами моделі VDM++ після виконання всього набору тестів. Тут було охоплено 82% моделі VDM++, а решта частин моделі були перевірені вручну. Основні результати включали 383-сторінковий посібник з протоколу, написаний

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 48   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

японською мовою, 677-сторінковий документ із зовнішньою специфікацією, написаний мовою VDM++ (приблизно 100 KLOC, включаючи коментарі, з яких приблизно 60 KLOC – це тестові випадки, сформульовані мовою VDM++). Реалізація становила приблизно 110 KLOC C/C++, включаючи коментарі.

FeliCa Networks вважала, що застосунок був дуже ефективним. З точки зору якості, на ранніх етапах розробки було виявлено більше помилок, ніж в інших подібних проектах FeliCa Networks. Загалом у вимогах та специфікаціях було виявлено 440 дефектів. З них 278 були виявлені безпосередньо в результаті використання VDM++. З них 162 були виявлені шляхом перевірки моделі, тоді як 116 були виявлені за допомогою інтерпретатора VDMTools з тестовими випадками на виконуваний моделі VDM++.

### 2.3 Абстрактний статичний аналіз як метод формальної перевірки

Статичний аналіз охоплює сімейство методів для автоматичного обчислення інформації про поведінку програми без її виконання. Хоча такі методи широко використовуються в оптимізації компілятора, ми зосереджуємося на їхньому використанні у верифікації програм. Більшість питань про поведінку програми є нерозв'язними або неможливими для обчислення. Таким чином, суть статичного аналізу полягає в *ефективному* обчисленні приблизних, але *надійних* гарантій; гарантій, які не вводять в оману.

Поняття апроксимації є якісним, на відміну від того, що використовується в більшості інженерних задач. Правильна апроксимація- це та, на яку можна покладатися. Наприклад, процедура виявлення помилок ділення на нуль у програмі повинна враховувати всі можливі значення змінних, що беруть участь в операціях ділення, під час виконання. Наближена процедура може обчислювати або підмножину, або надмножину цих значень і виразів. Якщо обчислюється підмножина, але помилок не знайдено, повернення «Немає помилок ділення» є неправильним, оскільки помилка все ще може існувати. Якщо обчислюється надмножина, але помилок не знайдено, повернення «Немає помилок ділення» є

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 49   |

виправданим. *Хибне попередження* - це повідомлення про помилку, якої немає в програмі. *Пропущена помилка* - це помилка, яка існує, але не повідомляється процедурою аналізу. Через нерозв'язність задач статичного аналізу розробка процедури, яка не створює хибних попереджень або пропущених помилок, неможлива.

Ми вводимо термінологію статичного аналізу в § II-A та ілюструємо абстрактний аналіз у § II-B. Ми розглядаємо абстракції числових типів даних у § II-C та купи даних у § II-D. Ми завершуємо цей розділ оглядом та обговоренням інструментів статичного аналізу. Зв'язок між статичним аналізом та перевіркою моделей коротко обговорюється в § III.

Методи статичного аналізу зазвичай поширюють набір значень через програму, доки набір *не досягне насичення* ; це значення не змінюється при подальшому поширенні. Математично такий аналіз моделюється як ітеративне застосування монотонної функції. Насичення відбувається, коли обчислюється фіксована точка функції. Ми ілюструємо цю ідею на навмисно спрощеному прикладі.

*Приклад 1:* Ми хочемо визначити значення змінної *i* в наведеній нижче програмі. Така інформація має різні застосування. Якщо *i* використовується як індекс масиву, аналіз корисний для забезпечення того, щоб не перевищувати межі масиву.

```
int i = 0;
do {
    assert(i <= 10);
    i = i+2;
} while (i < 5);
```

Граф потоку керування (ГПК) для цієї програми показано на рисунку 2.10(a). Кожен вузол анотується значеннями, які може мати *i* (які ми називаємо *набором значень* ) після досягнення цього місця в програмі. Нехай *Int* - це набір значень, які може мати змінна типу *int*. Для аналізу ми поширюємо набір значень

і вздовж ребер ГПК, додаючи нові значення в міру їх зустрічі. Анотації, обчислені в перших двох ітераціях, показані сірим і чорним кольором на рисунку 2.10(a).

Початкова множина значень  $i$ , будучи невизначеною, є  $\text{Int}$ . Після присвоєння  $i = 0$ , множина значень дорівнює  $\{0\}$ . Оператор  $\text{assert}(i \leq 10)$  не змінює  $i$ , тому  $L3$  має таку ж анотацію, як і  $L2$ . Після оператора  $i = i + 2$ , множина  $\{0\}$  змінюється на  $\{2\}$ . Оскільки  $2 < 5$ , ця множина поширюється з  $L4$  до  $L2$ . Множина  $\{0\}$  поширюється з  $L1$  до  $L2$ , а  $\{2\}$  поширюється з  $L4$  до  $L2$ , тому  $L2$  позначається об'єднанням  $\{0\}$  та  $\{2\}$ . Вузол, подібний до  $L2$ , з кількома вхідними ребрами, є *точкою з'єднання* та позначається множиною, отриманою шляхом об'єднання всіх множин, що поширюються на неї. Множина значень багаторазово поширюється вздовж CFG на рисунку 2.10(a), доки не буде досягнуто фіксованої точки.

Наведений вище аналіз є *конкретною інтерпретацією* програми. Підмножини  $\text{Int}$  представляють точні значення  $i$  під час виконання та складають *конкретний домен*. Конкретна інтерпретація є непрактичною, оскільки множини зростають неприпустимо швидко, і таке наївне поширення вздовж CFG не є масштабованим [34]. На практиці точність жертвується ефективністю шляхом використання абстракції набору значень та ігнорування структури програми під час поширення. Метод аналізу програми такий:

- *Чутливий до потоку*, якщо враховується порядок виконання операторів у програмі.
- *Чутливий до шляху*, якщо розрізняє шляхи в програмі та намагається розглянути лише можливі.
- *Контекстно-залежно*, якщо виклики методів аналізуються по-різному залежно від місця виклику.
- *Міжпроцедурний*, якщо аналізується тіло викликів методів. Якщо одна з цих умов ігнорується, метод аналізу є незалежним від потоку, шляху або контексту відповідно. Аналіз, який ігнорує тіло викликів методів, є *внутрішньоопроцедурним*.

Працюючи над компілятором Algol, зауважив, що абстрактних значень

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 51   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

може бути достатньо для аналізу програм [35]. У ранніх інструментах зв'язок між абстрактним аналізом та поведінкою програми під час виконання часто був незрозумілим. Запровадили абстрактну інтерпретацію як основу для зв'язку абстрактного аналізу з виконанням програми.

Абстрактна область – це наближене представлення наборів конкретних значень. Функція абстракції використовується для відображення конкретних значень в абстрактні. Абстрактна інтерпретація включає оцінку поведінки програми в абстрактній області для отримання наближеного рішення. Абстрактну інтерпретацію можна отримати з конкретної інтерпретації, визначивши аналоги конкретних операцій, таких як додавання або об'єднання, в абстрактній області. Якщо дотримуються певних математичних обмежень між абстрактною та конкретною областями, нерухомі точки, обчислені в абстрактній області, гарантовано є надійним наближенням конкретних нерухомих точок.

Приклад 2: Можливою абстрактною областю для аналізу в Прикладі 1 є множина інтервалів,  $\{[a, b] | a < b\}$ , де  $a$  та  $b$  є елементами  $\text{Int}$ . Конкретними операціями в Прикладі 1 є додавання та об'єднання на підмножинах  $\text{Int}$ . Додавання та об'єднання для інтервалів визначаються інтуїтивно зрозумілим чином.

Абстрактна інтерпретація програми проілюстрована на рисунку 1(b). Нехай  $\min$  та  $\max$  – мінімальне та максимальне значення цілочисельної змінної. Абстрактне значення  $i$  в  $L1$  дорівнює  $[\min, \max]$ , а в  $L2$  –  $[0, 0]$ . Цей інтервал поширюється на  $L3$ , а потім перетворюється за допомогою  $i = i + 2$  на  $[2, 2]$ . Об'єднання інтервалів  $[0, 0]$  та  $[2, 2]$  у точці з'єднання  $L2$  дорівнює  $[0, 2]$ , як показано. Інформація про те, що  $i$  не дорівнювало 1, втрачається шляхом об'єднання інтервалів. Інтервали, обчислені на другій ітерації, показані чорним кольором на рисунку 2.10(b). Поширення завершується, коли досягнуто фіксованої точки, як показано на рисунку 2.10(c).

Відповідний вибір абстрактної області та чутливості аналізу дає неточні, але обґрунтовані відповіді на проблеми аналізу програм. Однак абстрактний аналіз все ще може вимагати неможливу кількість ітерацій для досягнення фіксованої точки. Оператор розширення використовується для прискорення та

забезпечення завершення обчислень з фіксованою точкою, зазвичай шляхом подальшої втрати точності. Потім використовується додатковий *оператор звуження* для підвищення точності розв'язку.

Значення  $i$  збираються у відповідних місцях програми на рис. (а). Дані абстрагуються за допомогою інтервалів у (b), а фіксована точка досягається у (с). Мітки на ребрах позначають умову, яка має виконуватися для того, щоб потік керування пройшов по ребру; мітки на вузлах відповідають значенням змінних. Анотації точок керування, обчислені у двох послідовних ітераціях, показані відповідно сірим та чорним кольорами.

Протягом багатьох років було розроблено різні абстрактні області, зокрема для обчислення інваріантів щодо числових змінних. Клас інваріантів, які можна обчислити, а отже, і властивості, які можна довести, змінюється залежно від виразної сили області. *Реляційні області* можуть фіксувати зв'язки між змінними, а *нереляційні* – ні. Абстрактна область є точнішою за іншу, якщо втрачається менше інформації. Втрата інформації між різними областями може бути незрівнянною, як ми незабаром проілюструємо. Тепер ми розглянемо поширені абстрактні області, починаючи з простих нереляційних і переходячи до реляційних областей [36].

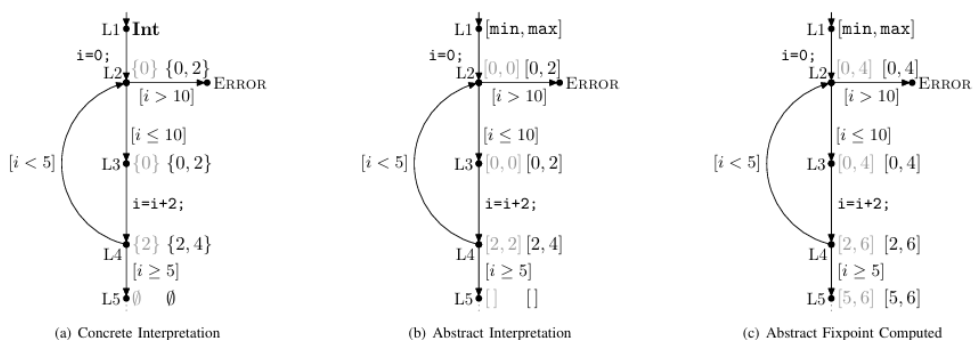


Рисунок 2.10 - Конкретна та абстрактна інтерпретації програми.

Домен *Знаків* має три значення {Положення, Від'ємне, Нуль} та має обмежене використання. Інтервали є більш виразними, оскільки значення в домені *Знаків* моделюються інтервалами [мін, 0], [0,0] та [0, макс]. Домен *Парностей*

абстрагує значення як Парні та Непарні. Хоча домен Парностей має менше елементів, ніж домени Знаків або Інтервалів, його не можна порівнювати з двома останніми. Жоден інтервал не може бути використаний для моделювання всіх парних або непарних значень. Домен Конгруенцій узагальнює Парності, представляючи значення  $v$  як  $(v \bmod k)$  для деякого фіксованого цілого числа  $k$ .

Розглянемо вираз  $1/(x - y)$ . Якщо використати абстрактну інтерпретацію, щоб показати, що  $(x \bmod k) = (y \bmod k)$ , ми можемо зробити висновок, що ділення на нуль не відбувається при обчисленні цього виразу. Конгруенції можна використовувати для доведення нерівностей, але не нерівностей між змінними. Нереляційні домени ефективно представляти та маніпулювати ними. На жаль, навіть прості властивості, такі як  $x < y$ , неможливо представити.

Тепер ми розглянемо деякі реляційні області. *Різницево-зв'язані матриці* (PBM) – це кон'юнкції рівнянь виду  $x - y < c$  та  $\pm x < c$  [19]. Спочатку їх використовували для аналізу мереж Петрі з часовими параметрами, а пізніше для перевірки моделей у системах реального часу. PBM дозволяють отримати точнішу абстракцію, ніж інтервальні, але не допускають обмежень виду  $(-x - y < c)$ . *Восьмикутники* є більш виразною областю з рівняннями виду  $(ax + by < c)$ , де  $a, b \in \{-1, 0, 1\}$ , а  $c$  – ціле число. *Октаедри* [33] узагальнюють восьмикутники для більш ніж двох змінних. Область визначення *многогранників* є однією з найдавніших і найпопулярніших реляційних областей, що використовуються для доведення числових властивостей програм та часової поведінки вбудованого програмного забезпечення. Многогранник – це кон'юнкція нерівностей виду  $a_1x_1 + \dots + a_nx_n < c$ , де  $a_i$  та  $c$  – цілі числа. Маніпулювання многогранниками включає обчислення опуклих оболонок, що є обчислювально ресурсоємним. Вимоги до часу та простору процедур маніпулювання многогранниками зазвичай експоненціальні за кількістю змінних.

Домени Знаків, Інтервалів, DBM, Восьмикутників, Октаєдрів та Багатогранників утворюють ієрархію з точки зору виразної сили. Вони корисні для обчислення обмежень нерівності щодо цілочисельних змінних та для виведення таких інваріантів для вкладених циклів програм [37]. Відповідний

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 54   |

еліпсоїдний домен кодує нелінійні залежності виду  $ax^2 + bxy + cy^2 \leq n$  та використовувався для аналізу цифрових фільтрів реального часу.

Ці реляційні області не можна використовувати для доведення нерівностей. Розглянемо задачу перевірки наявності ділення на нуль у  $1/(2 * x + 1 - y)$ . Абстрактна область для доведення цієї властивості повинна мати можливість виражати як лінійні обмеження, так і нерівності. Область *лінійних конгруенцій містить* рівняння виду  $ax + by = c \bmod k$ . Ця область розроблена шляхом поєднання ознак многогранників та конгруенцій. Загалом, нові області можуть бути розроблені з існуючих, за потреби.

Описані вище домени використовувалися для доведення властивостей авіоніки та вбудованого програмного забезпечення [25]. Числові домени також використовувалися для аналізу поведінки вказівника, маніпуляцій з рядками та завершення програми.

## 2.4 Висновки по розділу

У другому розділі було досліджено методи та інструменти формальної верифікації програмного забезпечення. Розглянуто сучасні тенденції розвитку формальних методів та їх застосування у реальних програмних системах. Проаналізовано інструментальні засоби, що використовуються для автоматизації процесів верифікації, а також наведено практичні приклади їх використання. Окрему увагу приділено методу абстрактного статичного аналізу як одному з ключових підходів формальної перевірки коректності програм. У результаті визначено ефективні інструменти та методи підвищення надійності програмного забезпечення.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 55   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

## РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДІВ ВЕРИФІКАЦІЇ

### 3.1 Роль та необхідність формальної верифікації у сучасних системах

З моменту появи автономних систем їхня інтеграція або роль у дедалі важливіших сферах застосування, таких як транспорт/охорона здоров'я, аерокосмічна галузь/ оборона , ніколи не була такою високою , тому програмне забезпечення, яке вони використовують і від якого залежать, має бути забезпечене на запланованому рівні максимально ретельного рівня [38]. Давно встановлені методи верифікації, включаючи моделювання та тестування, хоча й є цінними, не демонструють ефективності у забезпеченні правильності цих систем, оскільки вони досить вузькі та не охоплюють усі сценарії виконання. Саме тут формальна верифікація стає необхідною.



Рисунок 3.1 - Необхідність формальної верифікації

- Обмеження традиційного тестування: Класичні методи тестування передбачають використання певних вхідних даних та тестових даних у процесі, що означає, що можна протестувати лише частину поведінки системи. Це можливо лише в системах зі складністю, в системах, де поведінка може бути недетермінованою, в системах, де поведінка може бути реактивною, за умови

вимог до паралельності та часу, а також за умови невизначеності навколишнього

середовища. Через це деякі важливі помилки можуть залишитися непоміченими до моменту розгортання, коли їхні наслідки можуть бути катастрофічними.

- Гарантії через математичну ретельність: Формальна верифікація – це зовсім інший напрямок підходу до виявленої проблеми, оскільки він починається з математичних моделей та дедуктивної логіки, щоб забезпечити всі можливі поведінки системи. Перевірка моделей, доведення теорем та верифікація під час виконання – це методи, які дозволяють інженерам демонструвати такі властивості, як безпека (нічого поганого не трапляється) та життєздатність (зрештою трапляється щось хороше) та правильність відносно формальної специфікації. Граничні випадки, умови гонки та недоліки конструкції можуть стати очевидними завдяки цьому вичерпному аналізу та можуть бути пропущені під час валідації на основі моделювання.

- Критична роль у системах, критично важливих для безпеки: межа похибки дуже мала в таких галузях, як автономні транспортні засоби, безпілотні літальні апарати та роботизована хірургія. Формальна перевірка стає все більш важливою (як обов'язковий компонент стандартів безпеки, наприклад, ISO 26262 для автомобілів або DO-178C для авіоніки) або відповідність нормативним вимогам (або в деяких випадках, докази для оцінки відповідності специфікаціям). Формальні методи, що використовуються в таких контекстах, не просто зменшують ризик, але й пропонують перевірені, відстежувані докази правильності систем.

- Забезпечення довіри та впровадження: Формальна перевірка є важливим аспектом забезпечення загального сприйняття автономних систем суспільством та нормативними актами. Чим інтелектуальнішими та автономнішими стають ці системи, тим більше користувачів та інших зацікавлених сторін хочуть розуміти їхні дії та очікують, що вони діятимуть передбачувано, але безпечно [39]. Цей попит частково задовольняється

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 57   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

використанням формальних методів для забезпечення переконливих гарантій поведінки програмного забезпечення, що забезпечує основу відповідального масштабного розгортання.

#### *Формальна перевірка програмного забезпечення автономної системи*

Формальна верифікація програмного забезпечення автономної системи – це верифікація, яка використовує математично строгі засоби, щоб гарантувати, що програмне забезпечення може працювати правильно за будь-якого можливого сценарію роботи. Традиційне тестування, оскільки воно являє собою тестування лише певних місць або тестових випадків, не є вичерпним. Тим не менш, формальна верифікація є вичерпною, оскільки як система, так і специфікації, що стосуються неї, можуть бути змодельовані формальною мовою та перевірені відповідно до бажаних властивостей. Це особливо важливо для автономних систем, які повинні залишатися без нагляду в потенційно динамічних, невизначених умовах і часто викликаються в режимі реального часу для прийняття критично важливих для безпеки рішень. Вихідний код програмування їхніх систем сприйняття, планування та управління повинен бути різним; навіть найменша помилка в їхньому програмуванні може призвести до катастрофічних наслідків. Тут використовуються методи формального доведення, які включають перевірку моделей, доведення теорем та перевірку під час виконання. Перевірка моделей стосується автоматичного процесу перевірки властивостей безпеки, життєздатності, досяжності тощо шляхом визначення всіх можливих станів системи. Доведення теорем використовує дедукцію та людське конструювання доказів для доведення правильності системи та забезпечує високу впевненість у її складних компонентах.

Верифікація під час виконання доповнює ці методи пасивною перевіркою поведінки системи під час виконання для виявлення порушень заданих властивостей у режимі реального часу. Формальна верифікація автономних систем є особливою. Більшість цих компонентів (особливо тих, що стосуються машинного навчання або сенсорного сприйняття) важко формально моделювати, оскільки вони є ймовірнісними та недетермінованими. Однак операції отримання

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 58   |

доказів, алгоритми керування, комунікаційні протоколи тощо є ключовими процесами, які можна добре представити за допомогою таких механізмів, як кінцеві автомати, таймерні автомати або гібридні системи. Інструменти верифікації, такі як UPPAAL, Coq, SPIN, CBMC, ефективно використовуються для тестування автоматизованих дій, таких як запобігання зіткненням, призначення планет та виробництво контролерів [40]. Нарешті, формальна верифікація сприяє достовірності, безпеці та можливостям сертифікації програмного забезпечення будь-якої автономної системи, тому вона є надзвичайно цінним інструментом у циклі розробки майбутніх інтелектуальних машин.

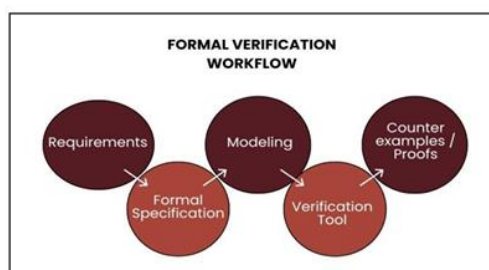


Рисунок 3.2 - Алгоритм формальної верифікації

Професіоналізація процесу верифікації має структуру робочого процесу, який забезпечує уявлення про коректність системи, вимагаючи перевіреної реалізації. [10-13] Етапи є кумулятивними та призводять до систематичних та ретельних міркувань щодо поведінки системи.

- Вимоги: Відправною точкою робочого процесу буде чітке визначення системних вимог, які визначають, що система повинна робити. Вони можуть бути як функціональними, наприклад, обмеження безпеки, часові властивості, так і нефункціональними, наприклад, продуктивність, надійність. Вони є основою для подальшої верифікації. Ці вимоги повинні бути повними, однозначними та формально вираженими під час формальної верифікації, оскільки будь-яка неточність у цій частині може призвести до неправильних або неповних результатів верифікації.

- Формальна специфікація: Після визначення вимог їх можна формалізувати та сформулювати як формальну специфікацію, яка представляє бажані властивості системи за допомогою математичної логіки або

- формальної мови. Цей крок пропонує специфікацію інваріантів, вимог безпеки, життєздатності або часових обмежень у формально коректній та машинозчитуваній формі. Офіційні специфікації не містять двозначностей та пропонують формальну основу для порівняння поведінки системи, яку вона повинна мати та не мати.

- Моделювання: На етапі моделювання система створюється за допомогою абстрактної версії. Структура та поведінка системи описуються цією моделлю з формальною семантикою з точки зору кінцевих автоматів, алгебр процесів або часових автоматів. Абстракція спрощує систему таким чином, щоб не виключати суттєві поведінкові фактори, що робить аналіз ефективним. Добре розроблена модель служить контролем між високорівневими специфікаціями та інструментом верифікації, що важливо для масштабованості та коректності.

- Інструмент верифікації: Потім, маючи модель та специфікацію, використовується адекватний інструмент верифікації для проведення автоматизованого або напівавтоматизованого аналізу. Залежно від підходу (наприклад, перевірка моделі, доведення теорем, верифікація під час виконання), використовуються такі інструменти, як UPPAAL, Coq, SPIN або CBMC. Вони передбачають дослідження простору станів системи або створення логічних доказів, які гарантують, що специфікація є істинною в моделі.

- Контрприклад / Докази: Результатом перевірки є або доказ коректності, або контрприклад, який показує порушення специфікації. Доказ гарантує той факт, що модель має всі формальні властивості незалежно від сценаріїв. Навпаки, у разі невдачі, якщо помилки не виявлено, інструмент надає контрприклад, трасу або послідовність дій, яка йде далі та призводить до порушення. Такі контрприклад також є важливими під час налагодження, оскільки розробник може отримати уявлення про фактичну причину та внести

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 60   |

корективи в модель системи або специфікацію.

Моделювання автономної поведінки означає абстрагування складної, а іноді навіть недетермінованої поведінки автономних систем за допомогою точних математичних моделей. Це необхідно для реалізації формальних схем верифікації, оскільки для поведінкового дослідження та раціоналізації систем надається шаблонна структура існування. Поведінка може бути реактивною, тобто автономні системи завжди реагують на подразники навколишнього середовища, як у випадку з безпілотними автомобілями, дронами та роями роботів. Формальна модель, така як система переходу станів, скінченний автомат або гібридна система, зазвичай описує останню. Система переходу станів описує систему як сукупність станів та переходів між цими станами, причому події або вхідні дані провокують переходи. Це дозволяє інженерам моделювати трансформацію системи з часом у відповідь на різні умови. Більш обмежений набір (відповідний для систем, які мають скінченну кількість дискретно представлених станів) - це скінченні автомати, які використовуються здебільшого у верифікації протоколів та проектуванні логіки керування. Вони мають ефективне представлення детермінованої поведінки та можуть бути легко проаналізовані за допомогою автоматичного набору інструментів, включаючи засоби перевірки моделей. Проблема полягає в тому, що більшості автономних систем доводиться працювати в умовах виникнення як дискретних, так і безперервних змін.

Прикладом є те, що дрон може бути дискретним у прийнятті рішень (наприклад, зліт, посадка, уникнення перешкод), але безперервним у своєму русі у фізичному просторі. У цих випадках застосовуються гібридні системи, що містять дискретні переходи станів разом із безперервною динамікою, зазвичай у формі диференціальних рівнянь. Гібридні автомати є скромним розширенням класичних кінцевих автоматів, оскільки вони мають змінні, що змінюються в часі. Гібридні автомати є дуже підходящими моделями вбудованих контролерів, динаміки роботів та фізичної взаємодії. Процес моделювання спирається на вибір методів, які будуть використовуватися залежно від складності, потреб та характеру аналізованої системи. Для перевірки системи дуже важливо створити

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 61   |

добре побудовану модель не лише для того, щоб систему можна було точно перевірити, але й для того, щоб знайти неоднозначності або невідповідності в конструкції системи. Таким чином, інтеграція моделювання утворює зв'язок між вимогами високого рівня та ретельним процесом верифікації.

Мови специфікацій у формальній верифікації. Саме у формальній верифікації імперативні системи визначають свої бажані властивості та очікувану поведінку будь-якої машини за допомогою формальних мов. [14-18] Це мови, синтаксис та семантика яких поєднують математичне використання для визначення системних цілей, обмежень та моделей взаємодії. Деякі з найпопулярніших різновидів мов специфікацій включають темпоральні логіки, алгебри процесів та специфікації на основі контрактів, які характеризують різні особливості поведінки системи та підходи до верифікації.

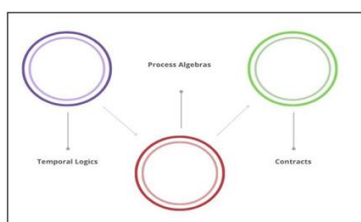


Рисунок 3.3 - Мови специфікації

- Тимпоральні логіки: це часові логіки для вираження властивостей послідовностей станів або подій під час виконання системи з плином часу. Існує дві поширені логіки: лінійна часова логіка (LTL) та логіка дерева обчислень (CTL). LTL використовується для вираження властивостей вздовж лінійних траєкторій і може бути застосована для визначення властивості на основі порядку, наприклад, подія А повинна врешті-решт слідувати за подією В або те, як умова Х завжди має бути істинною. Однак CTL допускає специфікації часу розгалуження, тому вона здатна міркувати про багато можливих майбутніх шляхів, використовуючи квантифікатори шляхів (наприклад, для всіх шляхів або існує шлях).

- Алгебри процесів: комунікація послідовних процесів (CSP) та Пі-

числення є прикладами алгебр процесів для моделювання та пояснення поведінки паралельних процесів. CSP також зосереджується на синхронній міжпроцесній комунікації та підходить для визначення протоколів комунікації та координації. Пі-числення узагальнює це та пропонує можливість динамічної структури, включаючи зміну каналів зв'язку, що важливо в розподілених або мобільних системах. Такі алгебри дозволяють композиційне мислення, в якому система визначається як конгломерат взаємодіючих частин.

- **Контракти: Специфікації.** Контрактні специфікації базуються на принципі припущень та гарантій, композиційній теорії специфікацій, в якій кожен компонент системи визначається набором припущень щодо його середовища та гарантій щодо його поведінки. Складні системи можна перевіряти по одному модулю за раз, використовуючи цей модульний підхід до міркувань, тим самим спрощуючи верифікацію. Контракти особливо корисні в системах систем або великих вбудованих конструкціях, компоненти яких розробляються та перевіряються окремо.

Існує велика різноманітність інструментів та фреймворків, що підтримують використання формальних методів, адаптованих до різних методів верифікації та стилів систем. Ці інструменти відрізняються за своїми методологіями, можливостями застосування, а також обмеженнями, і вибір інструментів зазвичай залежить від типу системи, що верифікується, тобто система реального часу, паралельна або програмна. Нижче описано деякі з найбільш поширених технологій формальної верифікації. UPPAAL – це перевірник моделей, адаптований для перевірки моделей систем реального часу, змодельованих як регульовані за часом автомати. Він добре справляється з перевіркою властивостей, обмежених у часі, правильності розкладів та досяжності у вбудованих процесах та кіберфізичних системах. UPPAAL простий у використанні та має функції моделювання та верифікації. Однак на нього впливає знайома дилема вибуху станів, коли зі збільшенням складності системи кількість станів зростає експоненціально, обмежуючи її масштабованість. Ще одним помітним перевірником моделей є SPIN, популярний у застосуваннях для

верифікації паралельних та розподілених систем.

Це робиться шляхом перевірки моделей Promela (Process Meta Language) та перевірки тупикових блокувань, умов гонки та порушень тверджень. Перевагою SPIN є те, що він може справлятися зі складними шаблонами міжпроцесної комунікації, і тому знаходить застосування в мережевих протоколах та багатопотоковому програмуванні. Основним обмеженням його використання є те, що він не повністю підтримує поведінку в реальному часі і тому може бути менш схильним до систем з часовими обмеженнями. Coq - це інтерактивний доказувач теорем, який дозволяє користувачам доводити з високою впевненістю, побудовуючи формальні докази. Він базується на основах числення індуктивних конструкцій та сприяє виразній специфікації та висновку. Coq часто використовується там, де пріоритетом є правильність доказів, наприклад, в авіоніці та криптографії. Однак він вимагає багато досвіду та ручної праці, оскільки процес доведення теорем не є повністю автоматичним, а отже, він повільний. CBMC (C Bounded Model Checker) - це інструмент для перевірки обмежених моделей програм, написаних на C та C++. Він підходить для тестування переповнення масивів, помилок вказівників та збоїв тверджень, оскільки він виконує пошук до заданої межі вздовж шляхів виконання. Хоча CBMC є більш практичним та може бути легко поєднаний з іншими кодовими базами, його основним недоліком є те, що розгортання циклів та дослідження шляху є обмеженими, а це означає, що помилки, які виходять за межі циклу, будуть пропущені.

Формальна верифікація автономних систем використовується з низкою системних компонентів, які мають різноманітні функції та вимоги до верифікації. Для забезпечення безпеки, надійності та правильності життєво важливо визначити відповідні цілі верифікації. Типовими цілями є алгоритми керування, системи сприйняття та блоки прийняття рішень, всі з яких мають різні вимоги до моделювання та специфікацій. •

Алгоритми керування: Ці антирезонансні контролери (наприклад, ПІД-контролери, вдосконалені планувальники руху) створюють безперервні

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 64   |

вихідні сигнали на виконавчі механізми, щоб підтримувати систему в рівновазі. Для перевірки таких алгоритмів закони керування повинні відповідати деяким вимогам безпеки та/або продуктивності, наприклад, не мати або бути мінімальним перерегулюванням, бути стабільними до широкого діапазону збурень або відстежувати опорну траєкторію. Гібридні автомати та моделі диференціальних рівнянь загалом є поширеними, оскільки логіка керування представляє проблеми безперервної математики та керованості в реальному часі. Перевірка алгоритмів керування має особливе значення для таких систем, як дрони, автономні транспортні засоби та роботизовані маніпулятори, в яких застосування неправильного керування може призвести до небезпечної поведінки.



Рисунок 3.4 - Мішені для верифікації

- Системи сприйняття: Системи сприйняття – це системи, які інтерпретують необроблену інформацію датчиків (наприклад, зображення, LIDAR, радар) у внутрішнє представлення середовища. Ці системи є складними за своєю природою, а також керуються даними, тому їх нелегко формально перевірити. Тим не менш, формальні методи можуть бути застосовані у випадку певних елементів конвеєра сприйняття шляхом встановлення передумов та постумов певних функцій, тобто виявлення об'єктів або локалізація об'єктів. Хоча повна формалізація сприйняття на основі глибокого навчання є відкритою проблемою, поки що обмежена верифікація може бути використана для отримання впевненості в цих елементах.

Блоки прийняття рішень: Високорівневі дії автономної системи – це навігаційні стратегії, уникнення перешкод тощо, які виконуються блоками прийняття рішень високого рівня. Вони часто застосовуються за допомогою скінченного автомата, дерева рішень або системи на основі правил. Вони є гарними об'єктами для формального моделювання та верифікації завдяки своїй дискретній та структурованій формі. Типовими цілями верифікації є забезпечення того, щоб система ніколи не досягла небезпечного, невизначеного стану або належним чином реагувала на зміни навколишнього середовища та відповідала цілям місії. Це основа автономності, і будь-яка неточність у цих блоках безпосередньо впливає на загальну поведінку системи.

### 3.2 Аналіз результатів застосування формальних методів

Незважаючи на їхні успіхи, технологія верифікації та формальні методи не отримали широкого впровадження як рутинна частина практики розробки систем, за винятком, можливо, розробки критичних систем у певних областях. Дійсно, ми очікуємо, що поширення технологій ретельної верифікації та проектування відбуватиметься поступово і не призведе до їх явного прийняття як окремої технології.

Попередні дослідження та наш нещодавній огляд свідчать про успіхи у застосуванні методів верифікації та формальних методів до проблем промислового масштабу та значення, а також у промислових умовах. Провідні розробники апаратного забезпечення продовжують застосовувати технологію перевірки моделей та доказів. У програмному забезпеченні використання формальних методів надало деякі докази потенціалу для застосувань, зосереджених на певних областях, включаючи верифікацію коду. Застосування в більш традиційних додатках з високою критичною цілісністю залишається сильним і здебільшого виконується технічними спеціалістами.

У цьому розділі ми знову розглядаємо занепокоєння, порушені в

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 66   |

попередніх опитуваннях, та визначаємо прогрес, тенденції та проблеми, що залишилися, у світлі новіших проектів. Ми сподіваємося, що ці проблеми будуть розглянуті дослідниками, які бажають удосконалити сучасний рівень розвитку формальних методів.

Вхідні витрати на формальні методи повинні здаватися досить високими для менеджера проектів розробки програмного забезпечення, який перебуває у скрутному становищі, вартість багаторазового використання може різко знизитися. Однією з головних труднощів в інженерії є економічно ефективний вибір того, що робити і де. Жоден інженер не приділяє однакової уваги всім зацепкам: ті, що нижче ватерлінії, виділяються окремо; аналогічно, формалізм не обов'язково застосовується в повній мірі до всіх компонентів цілого продукту та на всіх етапах їх розробки, і саме це ми бачимо на практиці. Різні рівні ретельності включають наступне:

— Докладаються максимальні зусилля, щоб вказати на ймовірні типові помилки. Приклади варіюються від lint до Coverity Prevent (комерційний інструмент, розроблений на основі Stanford Checker). Останній виконує два види аналізу: міжпроцедурний аналіз потоку даних, який описує характеристики функцій з наслідками для зовнішньої спостережуваної поведінки; та статистичний аналіз для виявлення важливих тенденцій у коді, що свідчать про аномалії.

— Майже гарантовано, що всі потенційні помилки певного класу були позначені. Основним прикладом тут є розширена статична перевірка ESC/Java2, система для автоматичного виявлення помилок під час компіляції, які зазвичай не виявляються до моменту виконання, таких як: помилки, пов'язані з масивами, розіменування null, умови гонки та глухі блокування в багатопотокових програмах.

—Перевірка під час виконання тверджень та іншої надлишкової інформації, наданої програмістом.

—Контрактне програмування з твердженнями на основних інтерфейсах.

—*Легкі формальні методи.* Цей термін застосовувався до різних форм

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 67   |

застосування, які не передбачають такого глибокого застосування технології. Наприклад, підхід надає перевагу точності над повною формальністю та створенню зрозумілих для людини доказів, які можна формалізувати за потреби. Пропонують цілеспрямоване застосування, проте повністю формальних методів, до часткових задач.

Усі вони заслуговують на цілеспрямоване дослідження, і наука, що лежить в основі всіх них, має багато спільного. Обов'язок інженера - визначити, де і наскільки використовувати наукові дані. Наш огляд не свідчить про існування єдиної стратегії для успішного застосування технології верифікації та формальних методів. Однак, полегшений підхід, здається, лежить в основі багатьох останніх галузевих застосувань. Використовуються повністю формальні методи, але зазвичай вони зосереджені на конкретних підсистемах та на верифікації певних властивостей. Успіх цього підходу залежить від якості інструментальної підтримки та обмежується нею.

Можна очікувати, що значний розвиток інструментів та ланцюгів інструментів вплине на форми впровадження формальних методів у майбутньому. Там, де досягнення сертифікації є важливим рушійним фактором, фахівці-практики можуть бути залучені до побудови формальних моделей, формулювання гіпотез для перевірки, а також керівництва та інтерпретації результатів напівавтоматизованого формального аналізу. Однак, зростаючі можливості автоматизованого формального аналізу дозволяють впливати на масштабніші розробки, не обов'язково критичні, які зумовлені необхідністю скоротити час виходу на ринок, рівень дефектів або витрати на тестування та обслуговування. У таких проектах ми можемо очікувати широкого використання вдосконалених інструментів статичного аналізу, що включають можливості перевірки моделей або доказів.

Тематичні дослідження, переважно стосувалися застосування формальних методів, що виконувалися технічними спеціалістами з використанням інструментів, які, за деякими винятками, не вважалися достатньо надійними для широкомасштабного застосування. Майже всі дослідження в цій галузі вказують

на важливість створення інструментів, які достатньо добре підтримуються, щоб заслуговувати на комерційне застосування. Серед виявлених проблем є: підтримка автоматизованого виведення, особливо там, де потрібна взаємодія людини з інструментом доказів; загальні формати для обміну моделями та аналізами; та відсутність адаптивної підтримки для користувачів інструментів у випадку багатьох інструментів, що пропонуються на основі «найкращих зусиль». Розробка таких надійних інструментів передбачає вирішення практичних питань, таких як забезпечення багатомовної підтримки, перенесення на різні платформи, контроль версій та допомога у спільній роботі кількох інженерів над окремими розробками. Кілька досліджень та близько чверті проектів розглянутих, визначають відсутність комерційно підтримуваних або «надійних» інструментів як перешкоду для впровадження формальних методів. Незважаючи на спостереження, що інструменти не є ні необхідними, ні достатніми для ефективного застосування формальних методів, здається майже немислимим, щоб промислове застосування зараз продовжувалося без інструментів.

Дослідницька спільнота зосередила значні зусилля на розробці нових теорій, що лежать в основі верифікації, вдосконаленні інструментів та, в обмежених, але зростаючій мірі, інтеграції інструментів. Робота над Mondex спочатку включала ручне конструювання доказів; його нещодавнє відродження свідчить про покращення рівня підтримки інструментів за останні роки, чи то в результаті покращених можливостей, чи то в результаті збільшення обчислювальної потужності, що лежить в основі закону Мура. У більш пізніх виділених проектах та в новому опитуванні є приклади надійних інструментів промислового рівня. Однак вони все ще мають лише обмежене застосування, і залишається багато цікавих проблем в теоріях, що лежать в основі верифікації, а також у взаємодії користувача з інструментами верифікації. Слід зазначити, що деякі безкоштовні коментарі, отримані в новому опитуванні, вказують на те, що інструменти досі не використовуються, за словами одного респондента, «простими смертними».

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 69   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Інструменти верифікації з ранніми радіоприймачами, з «десятками елементів керування для налаштування таких складних функцій, як регенерація та ширина смуги фільтра, а від операторів очікувалося розуміння основ радіоприйому». Сучасний радіоприймач, з іншого боку, простий в експлуатації. Перевірники моделей слідує цій тенденції, метою якої є остаточне перетворення на технологію керування натисненням кнопки для певних класів програмного забезпечення, таким чином, щоб компроміс між точністю та обчислювальною вартістю аналізу правильності можна було контролювати за допомогою кількох простих параметрів.

Скрізь, де це можливо, інструменти доказів мають зникнути, щоб вони були повністю приховані за іншими інструментами для аналізу, тестування та автоматизації проектування. Передбачають, що розробники застосовуватимуть технологію формальної верифікації «з такою ж легкістю, як і компілятори». Припускають, що інструменти мають бути невід'ємною частиною середовищ розробки, що розрив між інструментами та стандартним виробничим процесом має зникнути, і що певні аналізи можна буде використовувати в існуючих середовищах розробки. Існують вагомі аргументи на користь цього підходу як засобу перенесення технології формальної верифікації від новаторів до першопрохідців.

Дослідження рухаються в напрямку інструментів, які можна використовувати для надання аналізу з доданою вартістю «під капотом» існуючих середовищ розробки. Досвід SLAM та SDV, наприклад, свідчить про те, що цілеспрямований підхід може дати значні переваги на існуючій базі коду, коли інструменти ретельно інтегровані з існуючими середовищами розробки.

Успішне впровадження технології формальної верифікації передбачає «упаковку конкретних аналізів у простіші у використанні, але більш обмежені компоненти». Такий автоматичний аналіз повинен повертати результати, зрозумілі для користувача. Однак взаємодія з механізмами формального аналізу повинна здійснюватися на рівні мови проектування, а не формалізму.

Успішна інтеграція вимагає, щоб інструменти стали роз'єднаними

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 70   |

компонентами, які можна інтегрувати в існуючі інструменти для проектування, програмування або статичного аналізу. Інтеграція кількох підходів до верифікації була вперше застосована в середовищах розробки, таких як PROSPER, а архітектури "плагінів", такі як Eclipse, були успішно застосовані для інструментів, що підтримують Event-B та VDM. Інтеграція між графічними нотаціями проектування та математичними представленнями, необхідними для формального аналізу, стає все більш поширеною. Наприклад, зв'язок UML з B дозволяє використовувати знайому нотацію моделювання, яка буде пов'язана з формальним аналітичним фреймворком, а підтримка верифікації реалізацій систем, заданих за допомогою діаграм законів керування, була реалізована за допомогою Z, CSP та Circus..

Інтеграція з існуючими процесами розробки також є важливою. Нещодавнє опитування показує, що засоби перевірки моделей та генератори тестів мають значне застосування в промисловості. Можна припустити, що це тому, що їх можна інтегрувати в існуючі процеси розробки з меншими потрясіннями, ніж впровадження повністю нового дизайну або програмного фреймворку.

Зазначається відсутність переконливих доказів економічної ефективності використання формальних методів у промисловості як основну перешкоду для промислового впровадження, хоча вони також припускають, що це проблема, природно пов'язана зі зміною процесів. Вказують на відсутність загальноприйнятої моделі витрат як причину цієї відсутності доказів, хоча минулі опитування та висвітлені проекти надають багато історій, що підтверджують твердження, що формальні методи можна використовувати для економічно ефективного створення систем з низьким рівнем дефектів. У нашому опитуванні лише половина респондентів повідомила про наслідки використання формальних методів для витрат, в будь-якому випадку. Картина складна, навіть там, де витрати ретельно контролюються в пробних застосуваннях формальних методів. У роботі зазначається, що вартість одноразового проекту з формальних методів значно вища, ніж вартість повторних проектів у межах певної області.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 71   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Було проведено деякі дослідження, що порівнювали розробки з використанням та без використання формальних методів, наприклад, в оборонному секторі. Слабкі місця були виявлені в кількісних заявах, зроблених для попередніх проектів, що свідчить про необхідність вагомих емпіричних доказів для заохочення впровадження. Зазначають, що результати деяких наукових досліджень не масштабуються до інженерних проблем, і що мало місце переоцінка та надмірні очікування.

Дані щодо технічних та вартісних профілів комерційних застосувань можуть бути недоступними для громадськості, але, тим не менш, вони свідчать про те, що пілотні дослідження технології верифікації не завжди проводяться з метою збору інформації про наступні етапи впровадження технології. Пілотні застосування технології верифікації повинні бути доступними для спостереження, а зроблені спостереження повинні відповідати потребам тих, хто приймає критичні рішення щодо проектування. Наприклад, дослідження мереж FeliCa було зосереджено на вимірюванні запитів щодо неформальних документів з вимогами, які можна було віднести до формального моделювання та аналізу, оскільки це розглядалося як важлива особливість процесу розробки, метою якого було покращення специфікацій. Проект Deploy чітко розглядає перехід від новаторів до ранніх користувачів за допомогою досліджень, що вимірюють вплив інструментів верифікації, інтегрованих в існуючі процеси розробки.

Рішення про впровадження технології розробки приймається з урахуванням ризиків, і переконливі докази цінності формальних методів у виявленні дефектів (з якомога меншою кількістю хибнопозитивних результатів) можуть бути щонайменше такими ж вагомими, як і кількісний аргумент щодо вартості.

### 3.3 Оцінка ефективності та обговорення отриманих результатів

Це тематичне дослідження включало реалізацію, у формальному часовому

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 72   |

діапазоні, поведінки модуля запобігання зіткненням безпілотних літальних апаратів (БПЛА) та його перевірку за допомогою засобу перевірки моделі UPPAAL. Метою було переконатися, що БПЛА виявлятиме можливі зіткнення в повітрі та вживатиме безпечних та своєчасних заходів для їх запобігання. В абстрактній системі було розглянуто три надстани, а саме: Безпечний, Виявлений конфлікт та Маневр ухилення. По-перше, БПЛА перебуває в безпечному стані та постійно перевіряє повітряний простір. Він переходить у стан Виявлений конфлікт, коли виявляє можливий конфлікт, наприклад, коли інший БПЛА наближається до заданої відстані. У цьому стані система визначає серйозність конфлікту та, за потреби, виконує маневр ухилення, щоб відновити безпечну дистанцію розділення. Система повертається до безпечного стану після правильного виконання цього маневру. Така поведінка була правильною завдяки кодуванню властивостей безпеки за допомогою часової логіки, яка називається логікою дерева обчислень з часовим графіком (TCTL), що застосовується до системних міркувань реального часу. Серед найважливіших було те, що у випадку виявлення будь-якого конфлікту, маневр ухилення повинен бути виконаний до досягнення певного часового обмеження, і що система ніколи не повинна переходити в невизначений небезпечний стан, навіть за будь-яких обставин. Ці особливості відіграли життєво важливу роль у доведенні чутливості системи, а також надійності в ситуаціях обмеженого часу. Простір станів моделі був ретельно досліджений за допомогою UPPAAL за різних часових обставин та конфліктних ситуацій. Під час проведення аналізу було підтверджено, що за заданих припущень щодо навколишнього середовища та в межах часових міркувань система ніколи не потрапляла в небезпечні стани та мала правильну реакцію на конфліктні ситуації таблиця 3.1.

Таблиця 3.1:

Показники продуктивності

| Показник | Значення (%) |
|----------|--------------|
|----------|--------------|

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 73   |

|                        |     |
|------------------------|-----|
| Час верифікації        | 20% |
| Досліджені стани       | 24% |
| Використана пам'ять    | 20% |
| Порушення властивостей | 0%  |
| Кількість автоматів    | 40% |

Такий формальний підхід дозволив забезпечити високий ступінь впевненості в тому, що логіка уникнення зіткнень буде добре працювати в реальному середовищі, і це мало велику перевагу над деякими більш традиційними методами тестування за допомогою моделювання, які можуть легко пропустити дивні та нестандартні поведінки.

Аналіз процесу формальної верифікації, проведений під час оцінки його ефективності, виявив деякі основні аспекти, що виникли, включаючи час виконання, використання пам'яті та розмір досліджуваного простору станів. Усі ці показники надають інформацію про ефективність, масштабованість та повноту процесу верифікації з точки зору перевірки моделі UPPAAL (див.рис. 3.5.,

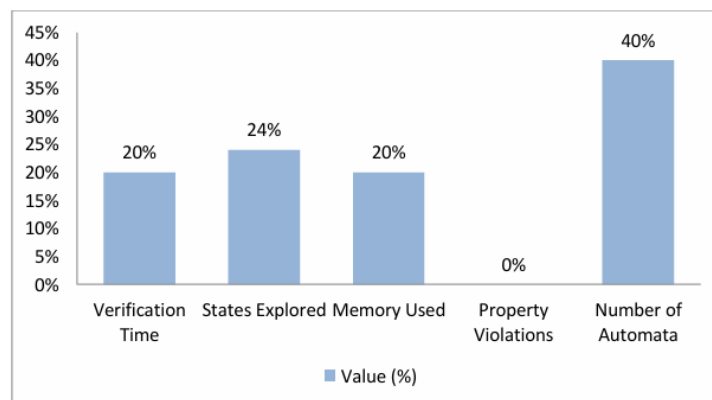


Рисунок 3.5 - Графік, що відображає показники ефективності

- Час перевірки (20%): Час перевірки становив 12 секунд, що становило 20 відсотків від максимального значення контрольного показника (прийняте за 60 секунд). Це означає, що модель легко оброблюється, і це можна пояснити помірно складними та добре розробленими переходами станів. Відносно невеликий час перевірки вказує на те, що UPPAAL має обробляти властивості безпеки системи з низьким рівнем паралельності та зручним простором станів.
- Досліджені стани (24%): UPPAAL здійснив пошук близько 1,2

мільйона станів, що становить 24 відсотки від гіпотетичної верхньої межі в 5 мільйонів. Це означає, що існує досить широкий діапазон охоплення поведінки системи за різних сценаріїв. Досліджені стани надають інформацію про те, наскільки повним є інструмент для вивчення траєкторій усіх можливих виконань тесту, включаючи граничні випадки, які стандартне тестування може не перевірити. Однак правда, що складніші системи з більшими розмірами все ще матимуть проблеми з вибухом станів.

- Використана пам'ять (20%): Використання пам'яті для перевірки становило 200 МБ (20 відсотків від передбачуваного ліміту в 1000 МБ). Це означає, що інструмент був економним на пам'яті, враховуючи розмір цієї моделі. Використання пам'яті є важливим, оскільки масштабування зростає у складності, оскільки обмеження ресурсів можуть погіршити швидкість перевірки або призвести до часткового аналізу.

- Порухення властивостей (0%): Відсутність порушень властивостей призвела до рівня порушень 0 відсотків. Той факт, що цей результат мав місце, доводить, що модель відповідає всім заданим властивостям безпеки TCTL у межах наданих припущень. Це підтверджує достовірність логіки запобігання зіткненням БПЛА та автентичність формальної моделі.

- Кількість автоматів (40%): Модель використовувала чотири компоненти часових автоматів, що становить 40 відсотків від розрахункової максимальної складності десяти автоматів. Такий рівень модульної декомпозиції забезпечив як керований, так і очевидний дизайн верифікації, а також перевірів реалістичні взаємодії систем, коли компоненти, такі як датчики, контролери та монітори навколишнього середовища, були взаємопов'язані.

Формальна верифікація системи запобігання зіткненням БПЛА змогла надати достовірну інформацію про сильні та слабкі сторони змодельованої поведінки. Серед найважливіших результатів можна відзначити, що були виявлені критичні випадки, які раніше не були виявлені за допомогою традиційного тестування на основі моделювання. Це були такі ситуації, як одночасне виявлення конфлікту двома або більше БПЛА практично одночасно та неможливість вжити

заходів ухилення через конфлікти часу або конфлікти ресурсів, і це лише деякі з них. Хоча такі випадки навряд чи трапляються в типовому застосуванні, вони надзвичайно небезпечні, коли трапляються, і надійність формальної верифікації можна розглядати як здатність всебічно охоплювати всі шляхи виконання, включаючи такі граничні випадки, які моделювання може пропустити, оскільки вони залежать від набору заздалегідь запланованих тестів або ймовірнісних вхідних даних. Хоча цей метод був досить ефективним, дослідження також виявило суттєвий недолік масштабованості/прогнозування, який існує, зокрема, у формальній верифікації та перевірці моделі. Простір станів зростає експоненціально зі збільшенням кількості БПЛА в системі та складністю їхньої взаємодії, тобто численними рівнями прийняття рішень, доданими до системи, угодами про зв'язок та динамічним моделюванням середовища. Це призвело до збільшення часу перевірки та значного збільшення використання пам'яті, і незабаром може стати непрактичним для масштабної координації в реальних системах.

Щоб впоратися з останнім, виникає необхідність методів абстракції, тобто укрупнення моделі або ізоляції компонентів, які підлягають окремому аналізу. Крім того, композиційна верифікація – це ситуація, коли система доводиться модульно, а властивості, виведені систематично, мають привабливий напрямок для управління складністю та збереження строгості. Примітно, що тематичне дослідження також довело переваги гібридного методу верифікації, включаючи поєднання формальної верифікації, традиційного моделювання та традиційного тестування. Перевірка моделі була більш застосовною для отримання хорошої оцінки властивостей безпеки та передумов системи, тоді як моделювання краще визначало динамічні характеристики та ефекти мінливості в реальності. Таке компенсуюче застосування методів дозволило використовувати сильніші аспекти кожного підходу для забезпечення ефективного підходу до верифікації – формальна верифікація наполягала на правильності, тоді як моделювання давало гарантію щодо продуктивності системи, ця стратегія призвела до поміркованого, ефективного та практичного підходу до верифікації автономних систем, критично

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 76   |

важливих для безпеки.

Формальні методи, що застосовуються до життєвого циклу розробки програмного забезпечення, виявилися дуже перспективними, особливо коли вони застосовуються на самому початку життєвого циклу програмного забезпечення, тобто на етапі розробки вимог та етапі архітектурного проектування. Формальні методи корисні на цих етапах для забезпечення кращої якості специфікацій, дозволяючи розробникам виявляти неоднозначності, невідповідності та недосліджені вимоги на ранній стадії, щоб вони не стали небажаними та важкими для подолання на етапах впровадження та тестування. Формальні нотації або специфікації на основі логіки описують вимоги і, таким чином, дають командам чітке уявлення про те, що повинна робити система, що зменшує неправильне тлумачення та покращує прозорість. Крім того, на етапі архітектурного проектування моделювання формальної семантики пропонує формалізувати відповідні аспекти поведінки на рівні системи на ранніх етапах проектування, підтримуючи сувору аргументацію щодо інтерфейсів, часу, паралельності та обмежень ресурсів, що мають велике значення в багатьох складних та критично важливих для безпеки областях, таких як автономні системи. Незважаючи на свої переваги, ширше використання формальних процесів на пізніших етапах циклу впровадження, особливо під час впровадження, інтеграції та обслуговування, все ще є відносно слабким, значною мірою тому, що інтеграція з традиційними інструментальними ланцюгами CI/CD (безперервна інтеграція/безперервне розгортання) все ще є проблемою.

Більшість формальних інструментів, включаючи UPPAAL, Coq та SPIN, створені як автономні системи без потужної підтримки автоматизації, API сценаріїв або легкого взаємозв'язку з контролем версій та DevOps, наприклад, GitHub Actions, Jenkins або GitLab CI. Отже, формальна верифікація зазвичай виконується як незалежне завдання, повністю відокремлене від автоматичних робочих процесів, що мотивують гнучкі та ітеративні процеси розробки. Слід внести кілька покращень, щоб скористатися перевагами формальних методів протягом усього життєвого циклу розробки. Це включає формулювання API

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 77   |

інструментів та стандартизованих форм, що дозволяють формальним інструментам взаємодіяти з конвеєрами DevOps, посилену підтримку поступової верифікації, оскільки код піддається подальшій розробці, навчання інженерів та розробку зручних інтерфейсів формальних інструментів, оскільки не всі інженери навчені формальним методам. Завдяки доступності та інтегрованості формальної верифікації організації можуть змістити верифікацію вліво в життєвому циклі, впроваджуючи математичну точність у щоденні інженерні процеси без шкоди для гнучкості.

Формальна верифікація є важливою для підвищення надійності та безпеки автономних систем, пропонуючи формальну математичну гарантію того, що система буде поводитися належним чином за всіх можливих обставин. На відміну від звичайного тестування, яке вибірково перевіряє лише певні шляхи виконання, формальні методи повністю гарантують, що система була протестована на всіх можливих типах поведінки, надаючи гарантії того, чи є система правильною чи неправильною, що має особливе значення для систем, пов'язаних з безпекою, таких як безпілотні літальні апарати, автономні транспортні засоби та робототехніка. Перевірка моделей та доведення теорем виявилися особливо ефективними при верифікації логіки дискретного керування, особливо кінцевих автоматів, дерев рішень та алгоритмів керування. Ці методи можуть гарантувати, що будь-які логічні невідповідності, помилки синхронізації, недосяжні та небезпечні стани будуть систематично видалені під час проектування. Тим часом, хоча ці методи роблять систему максимально безпечною, верифікація під час виконання базується на таких підходах для відстеження поведінки системи під час розгортання, щоб гарантувати, що безпека підтримується в реальних динамічних умовах. Цей багаторівневий метод підвищує загальну впевненість системи в розробці, а також у життєвому циклі операцій. Формальні методи мають ряд недоліків щодо їх практичної реалізації, хоча вони мають кілька переваг. Значною складністю є легка масштабованість: кількість станів зростає експоненціально зі збільшенням складності системи, особливо у великих системах або в паралельних чи розподілених системах, або системах, що складаються з кількох агентів.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 78   |

Другим обмеженням є моделювання систем сприйняття, наприклад, систем на основі сенсорів або глибокого навчання. Вищезазначені

частини є ймовірнісними та керованими даними за своєю природою; таким чином, складно формально описати ці частини та проаналізувати їх за допомогою звичайних інструментів аналізу на основі логіки. Більше того, більшість інструментів, доступних у формальних методах, також вимагають ґрунтового рівня знань у логіці, теорії автоматів або доведенні теорем, що знову ж таки робить криву навчання надзвичайно високою, коли вона представлена інженерам та розробникам, незнайомим з формальними методами. Відсутність інтуїтивності та зручності використання перешкоджає доступу та інтеграції таких інструментів у сучасні гнучкі функції. Щоб подолати ці недоліки, дослідження та розробки в майбутньому повинні включати посилення формальних методів, і одним із способів зробити це може бути застосування методів абстракції та уточнення, щоб складні системи можна було перевіряти шляхом їх декомпозиції на простіші системи. Можливість розширити формальні рамки верифікації на такі програми, як нейронні мережі та компоненти, що підтримують навчання, які ми поступово все більше інтегруємо в сучасні автономні системи, є ще одним вірогідним майбутнім у цьому напрямку. Такі методи, як мережева верифікація нейронних мереж, формальна абстракція машинного навчання та конверти безпеки, – це методи, які можуть допомогти подолати розрив між штучним інтелектом та формальними методами. Більше того, інтеграція формальних методів із системою проектування на основі моделей та фреймворками безпеки штучного інтелекту також дозволить ширше розширення, оскільки фази моделювання, симуляції та верифікації можна буде легко перемикати. Це дозволить розробникам проводити формальну верифікацію раніше та частіше протягом життєвого циклу розробки.

### 3.4 Висновки по розділу

У третьому розділі було розглянуто практичну реалізацію та оцінку ефективності методів формальної верифікації. Обґрунтовано роль і необхідність

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 79   |

використання формальних методів у сучасних програмних системах для підвищення їх безпеки та надійності. Проаналізовано результати застосування формальної верифікації на практиці, що дозволило оцінити її вплив на виявлення помилок і підвищення якості ПЗ. Проведено оцінку ефективності використаних методів та інтерпретацію отриманих результатів, які підтверджують доцільність застосування формальної верифікації у процесах розробки критично важливих програмних систем.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 80   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

## ВИСНОВКИ

У ході виконання дипломної роботи було досліджено методи формальної верифікації програмного забезпечення як комплексний підхід до забезпечення надійності, коректності та безпеки сучасних програмних систем. У роботі проаналізовано існуючі підходи до перевірки програм, розглянуто теоретичні основи формальних методів, їх класифікацію, а також інструментальні засоби застосування у процесі розробки програмного забезпечення. Проведене дослідження дозволило визначити ключові переваги формальної верифікації порівняно з традиційними методами тестування, зокрема можливість математичного доведення коректності програм.

На початковому етапі було здійснено аналіз теоретичних засад формальної верифікації, включаючи основні поняття, принципи та сучасні підходи. Особливу увагу приділено класифікації формальних методів, серед яких логічні методи доведення, модельна перевірка та абстрактний статичний аналіз. Це дозволило систематизувати існуючі підходи та визначити їх область застосування залежно від типу програмних систем і вимог до їх якості.

У другому розділі було досліджено сучасні методи та інструменти формальної верифікації, включаючи засоби статичного аналізу, формальні специфікації та програмні інструменти автоматизованої перевірки. Було розглянуто практичні аспекти їх застосування, що дозволило оцінити ефективність використання формальних методів у реальних умовах розробки програмного забезпечення. Особлива увага приділялася аналізу тенденцій розвитку цієї галузі, зокрема інтеграції формальних методів у процеси DevOps та автоматизованого тестування.

У третьому розділі проведено аналіз результатів застосування формальних методів та оцінено їх ефективність. Було визначено, що використання формальної верифікації дозволяє виявляти помилки на ранніх етапах життєвого циклу програмного забезпечення, зменшувати витрати на їх виправлення та

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 81   |

підвищувати загальну якість програмних продуктів. Разом із тим, було виявлено певні обмеження, зокрема складність впровадження формальних методів, необхідність високої кваліфікації розробників та значні витрати часу на формалізацію вимог.

Загалом, результати дослідження підтверджують, що формальна верифікація є ефективним інструментом підвищення якості програмного забезпечення, особливо у критично важливих системах, де помилки можуть мати серйозні наслідки. Перевагами застосування формальних методів є можливість забезпечення високого рівня надійності, точності та безпеки програмних систем.

Водночас, подальший розвиток формальної верифікації пов'язаний із спрощенням інструментів її використання, автоматизацією процесів перевірки та інтеграцією з сучасними підходами до розробки програмного забезпечення. У перспективі це дозволить зробити формальні методи більш доступними для широкого кола розробників та сприятиме їх активнішому впровадженню у практику розробки програмного забезпечення.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 82   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

## СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. MIT Press. <https://mitpress.mit.edu/9780262032704/model-checking/>
2. Baier, C., & Katoen, J. P. (2008). *Principles of Model Checking*. MIT Press. <https://mitpress.mit.edu/9780262026499/principles-of-model-checking/>
3. Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 576–580. <https://doi.org/10.1145/363235.363259>
4. Dijkstra, E. W. (1976). *A Discipline of Programming*. Prentice Hall.
5. Cousot, P., & Cousot, R. (1977). Abstract interpretation: A unified lattice model. *POPL*. <https://doi.org/10.1145/512950.512973>
6. Lamport, L. (2002). *Specifying Systems: The TLA+ Language*. Addison-Wesley.
7. Spivey, J. M. (1992). *The Z Notation: A Reference Manual*. Prentice Hall.
8. Woodcock, J., & Davies, J. (1996). *Using Z: Specification, Refinement, and Proof*. Prentice Hall.
9. Bowen, J. P., & Hinchey, M. G. (1995). Ten commandments of formal methods. *Computer*, 28(4), 56–63.
10. Abrial, J.-R. (2010). *Modeling in Event-B*. Cambridge University Press.
11. Huth, M., & Ryan, M. (2004). *Logic in Computer Science*. Cambridge University Press.
12. Kroening, D., & Strichman, O. (2016). *Decision Procedures*. Springer. <https://doi.org/10.1007/978-3-662-50497-5>
13. Barrett, C., Sebastiani, R., Seshia, S., & Tinelli, C. (2009). SMT solving. *Handbook of Satisfiability*.
14. Biere, A., Heule, M., & van Maaren, H. (2009). *Handbook of Satisfiability*. IOS Press.
15. Jackson, D. (2012). *Software Abstractions: Logic, Language, and Analysis*. MIT Press.

16. Clarke, E. M., & Wing, J. M. (1996). Formal methods: State of the art. *ACM Computing Surveys*, 28(4).
17. Meyer, B. (1992). Applying “Design by Contract”. *Computer*, 25(10).
18. Leino, K. R. M. (2010). Dafny: An automatic program verifier. *LPAR*.
19. Barnett, M., et al. (2005). Boogie: A modular reusable verifier. *FMCO*.
20. Nipkow, T., Paulson, L. C., & Wenzel, M. (2002). *Isabelle/HOL*. Springer.
21. Coq Development Team. (2025). *The Coq Proof Assistant Reference Manual*. <https://coq.inria.fr/documentation>
22. Microsoft Research. (2025). *Z3 Theorem Prover Documentation*. <https://github.com/Z3Prover/z3>
23. SPIN Model Checker. (2025). Documentation. <http://spinroot.com>
24. NuSMV. (2025). Symbolic Model Checker Documentation. <http://nusmv.fbk.eu>
25. PRISM Model Checker. (2025). Documentation. <https://www.prismmodelchecker.org>
26. Frama-C. (2025). Static analysis framework. <https://frama-c.com>
27. CBMC. (2025). C Bounded Model Checker. <https://www.cprover.org/cbmc/>
28. Infer Static Analyzer. (2025). <https://fbinfer.com>
29. SonarQube Documentation. (2025). <https://docs.sonarqube.org>
30. LLVM Static Analyzer. (2025). <https://clang-analyzer.llvm.org>
31. Microsoft Learn. (2025). Static code analysis in .NET. <https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/overview>
32. OWASP. (2024). Secure Coding Practices. <https://owasp.org>
33. ISO/IEC 25010. (2011). Systems and software quality models. <https://www.iso.org>
34. ISO/IEC 12207. (2017). Software lifecycle processes. <https://www.iso.org>
35. IEEE. (2023). Software Verification and Validation Standards. <https://www.ieee.org>

36. Rushby, J. (1993). Formal methods and the certification of critical systems. *SRI International*.
37. Hall, A. (1990). Seven myths of formal methods. *IEEE Software*, 7(5).
38. Jackson, D., & Wing, J. M. (1996). Lightweight formal methods. *IEEE Computer*.
39. Woodcock, J., et al. (2009). Formal methods: Practice and experience. *ACM Computing Surveys*.
40. Broy, M., & Stølen, K. (2001). *Specification and Development of Interactive Systems*. Springer.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП – 34.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 85   |
| ЗМН. | Арк. | № докум. | Підпис | Дата |                         |      |

## БІБЛІОГРАФІЧНА ДОВІДКА

**Тема бакалаврської роботи:** "Методи формальної верифікації програмного забезпечення"

Обсяг пояснювальної записки: 82 аркушів

Дата закінчення бакалаврської роботи 10 червня 2025р.

Підпис студента \_\_\_\_\_

