Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Назва теми

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.240170.01.10.ПЗ

Виконав студент 2 курсу, група ІПЗм-24-1 _____ Ілля ЧЕРВОНЧУК
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ Оксана ОНИШКО
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. пед. наук, доцент _____ Яшина О.М.
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ініціали, прізвище

15 грудня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський) Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення» Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри _____
Л. П. Бедратюк
01 09 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Червончуку Іллі Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи)
Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Керівник проєкту (роботи) канд. пед. наук, доцент Онишко О.Г.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 25.08.2025 №65.
2. Строк подання студентом проекту (роботи) на кафедру 01.12.2025 р.
3. Вихідні дані до проекту (роботи) Матеріали науково-дослідної практики 4.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1 Аналіз предметної області та рішень з програмного забезпечення.
2 Удосконалення методу автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання 3 Архітектура програмної реалізації системи автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання 4 Програмна реалізація системи на основі диференціального аналізу логів виконання
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

Антиплагіат	Форкун Ю.В., доцент	1.12.2025	15.12.2025
Нормоконтроль	Яшина О.М., доцент	30.11.2025	13.12.2024

7. Дата видачі завдання «01» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1. Вивчення предметної області; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	20.10 - 26.09.2025	
2. Робота над розділом 1 кваліфікаційної роботи - аналіз предметної області та постановка завдання	20.10 - 26.09.2025	
3 Робота над розділом 2 кваліфікаційної роботи - визначення теоретичних засад автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання	27.10 - 02.11.2025	
4. Робота над науковими публікаціями, статтями	03.11 - 09.11.2025	
5. Робота над розділом 3. Проектування архітектури системи для вирішення задачі, розробка вимог.	10.11 - 16.11.2025	
6 Робота над розділом 4 кваліфікаційної роботи - формалізація, оцінювання та порівняльний аналіз запропонованого підходу	17.11 - 23.11.2025	
7 Попередній захист кваліфікаційної роботи	24.11 - 30.11.2025	
8 Узгодження постановки задачі, отриманих результатів та висновків; оформлення пояснювальної записки та графічних матеріалів згідно вимог чинних стандартів	01.12-07.12.2025	
9 Перевірка роботи на наявність плагіату: нормоконтроль; брошурування пояснювальної записки; підготовка супровідних документів	08.12 - 14.12.2025	
10 Підготовка до захисту кваліфікаційної роботи	З 15.12.2025	

Студент _____ Ілля ЧЕРВОНЧУК
Підпис Ініціали, прізвище

Керівник проєкту (роботи) _____ Оксана ОНИЩКО
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання».

Автор роботи: Червончук Ілля Сергійович.

Керівник роботи: Онишко Оксана Григорівна.

Пояснювальна записка: 88 с., 7 рис., 2 дод., 24 джерела.

ЛОГИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ДИФЕРЕНЦІАЛЬНИЙ АНАЛІЗ, МАШИННЕ НАВЧАННЯ, АВТОМАТИЗАЦІЯ, ВИЯВЛЕННЯ ПОМИЛОК ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

Об'єктом дослідження є процес автоматизованого виявлення помилок у програмному забезпечення із використанням автоматизації на основі диференціального аналізу логів виконання.

Мета дослідження - удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Предметом дослідження є сучасні методи логування, що використовуються для виявлення помилок.

Враховуючи мету, предмет, а також об'єкт дослідження можна виділити наступні задачі:

1. Провести аналітичне дослідження предметної області досліджуваної проблеми.
2. Здійснити аналітичний огляд методів логування.
3. Здійснити удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Наукова новизна:

1. Удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

2. Розроблено алгоритм роботи методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Практичне значення дослідження. Метод автоматизованого виявлення помилок на основі диференційного аналізу логів виконання є актуальним та корисним, особливо в сучасних складних, розподілених та мікросервісних архітектурах, оскільки він забезпечує проактивне виявлення дефектів та підвищення операційної ефективності.

Ключова практична цінність полягає у скороченні середнього часу на виявлення та локалізацію помилки. Система не чекає на збій або скаргу клієнта; вона автоматично генерує сповіщення в той момент, коли ступінь аномальності перевищує встановлений поріг. Це дозволяє команді реагувати на деградацію до того, як вона перетвориться на повний збій. Завдяки векторизації та трасуванню, аномалія миттєво прив'язується до конкретного компонента та шаблону логу, що значно зменшує час, який інженери витрачають на ручний перегляд мільйонів рядків логів.

05.12.2025

ABSTRACT

Master's thesis: «Method for automated detection of errors in software based on differential analysis of execution logs».

Author: Ilya Chervonchuk.

Head of work: Oksana Onyushko.

Master's thesis consists of: 88 pages of the general text, 7 graphics, 2 supplements, 24 literature sources.

SOFTWARE LOGS, DIFFERENTIAL ANALYSIS, MACHINE LEARNING, AUTOMATION, SOFTWARE ERROR DETECTION.

The object of the study is the process of automated error detection in software using automation based on differential analysis of execution logs.

The purpose of the study is to improve the method of automating the detection of errors in software using differential analysis of execution logs.

The subject of the study is modern logging methods used to detect errors.

Considering the purpose, subject, and object of the study, the following tasks can be identified:

1. Conduct an analytical study of the subject area of the problem under investigation.

2. Conduct an analytical review of logging methods.

3. Improve the method of automating the detection of errors in software using differential analysis of execution logs.

Scientific novelty:

1. Improve the method of automating the detection of errors in software using differential analysis of execution logs.

2. Develop an algorithm for the method of automating the detection of errors in software using differential analysis of execution logs.

The method of automated error detection based on differential analysis of execution logs is relevant and useful, especially in modern complex, distributed, and microservice

architectures, as it provides proactive defect detection and improved operational efficiency.

The key practical value lies in reducing the average time to detect and localise an error. The system does not wait for a failure or customer complaint; it automatically generates an alert when the degree of anomaly exceeds a set threshold. This allows the team to respond to degradation before it turns into a complete failure. Thanks to vectorisation and tracing, the anomaly is instantly linked to a specific component and log pattern, significantly reducing the time engineers spend manually reviewing millions of log lines.

05.12.2025

ЗМІСТ

ВСТУП

Актуальність дослідження. Величезні обсяги даних журналів логів щодня генеруються програмним забезпеченням. Ці дані містять цінну інформацію про поведінку та стан системи, яка рідко використовується через свій обсяг та неструктурованість. Ручне переглядання файлів журналів є трудомісткою та тривалою процедурою для розробників.

Проте інформація журналів може допомогти у виявленні проблемного виконання програмного забезпечення, навіть якщо кінцевий результат здається нормальним. Сьогодні автоматичний аналіз файлів журналів має вирішальне значення для виявлення проблем, але головним чином для розуміння того, як поводиться програмне забезпечення, що було б корисним для запобігання збоям та вдосконалення самого програмного забезпечення. В цьому напрямку дана кваліфікаційна робота спрямована на виявлення неочікуваних виконань програмного забезпечення та визначення корінної причини їх виникнення. Більш детально, очікувану поведінку програмного забезпечення можна апроксимувати за допомогою методів модельного висновку, а нові спостережувані дані можна проаналізувати, щоб перевірити, чи відповідають вони очікуваній поведінці.

Логування є фундаментальним компонентом життєвого циклу будь-якого програмного продукту, виступаючи сполучною ланкою між так званою чорною скринькою виконуваного коду та реальним світом. Це процес запису подій, помилок, статусів та транзакцій, що відбуваються під час роботи програми. Актуальність цієї практики розкривається по-різному залежно від того, хто саме взаємодіє із системою: той, хто її створює, чи той, хто нею користується.

Для інженера-програміста логування є чи не єдиним способом зазирнути всередину працюючої системи без необхідності зупиняти її роботу. У процесі розробки та особливо на етапі підтримки, логи виступають у ролі деталізованого бортового журналу.

Для кінцевого користувача процес логування зазвичай залишається невидимим, проте його результати мають прямий вплив на якість користувацького досвіду.

Також варто зазначити роль логування у комунікації зі службою підтримки. Коли користувач звертається по допомогу, наявність унікального ідентифікатора помилки, який базується на записі в логах, дозволяє оператору підтримки миттєво зрозуміти контекст проблеми, не змушуючи клієнта довго пояснювати технічні деталі. Це значно пришвидшує вирішення індивідуальних проблем і знижує рівень стресу при взаємодії з сервісом.

Отже, логування створює симбіоз між розробником та користувачем. Для першого це технічний інструментарій для забезпечення якості, безпеки та керованості коду. Для другого - це невидимий фундамент, який гарантує, що його дані будуть цілісними, помилки виправлятимуться оперативно, а сервіс працюватиме передбачувано. Відсутність належного логування перетворює програмне забезпечення на некерований хаос, де розробник сліпий до проблем, а користувач беззахисний перед збоями.

Об’єктом дослідження є процес автоматизованого виявлення помилок у програмному забезпечення із використанням автоматизації на основі диференціального аналізу логів виконання.

Мета дослідження - удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Предметом дослідження є сучасні методи логування, що використовуються для виявлення помилок.

Враховуючи мету, предмет, а також об’єкт дослідження можна виділити наступні задачі:

1. Провести аналітичне дослідження предметної області досліджуваної проблеми.
2. Здійснити аналітичний огляд методів логування.

3. Здійснити удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Наукова новизна:

1. Удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.
2. Розроблено алгоритм роботи методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

1 ТЕОРЕТИЧНИЙ ВИКЛАД ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області

Коли виникає критична помилка або неочікувана поведінка, розробник звертається до журналів подій, щоб відтворити хронологію дій, яка призвела до збою. Без якісних логів процес виправлення помилок перетворюється на здогадки, що критично збільшує час простою системи та витрати на розробку.

Більше того, логування відіграє ключову роль у моніторингу продуктивності та оптимізації. Завдяки часовим міткам (timestamps) у журналах, розробники можуть виявити так звані вузькі місця, тобто фрагменти коду або запити до бази даних, які виконуються занадто повільно і гальмують роботу всього застосунку. Це дозволяє приймати обґрунтовані рішення щодо рефакторингу коду або масштабування інфраструктури ще до того, як користувачі почнуть скаржитися на повільну роботу сервісу.

Окремим важливим аспектом для розробника є безпека та аудит. У сучасних розподілених системах та мікросервісній архітектурі вкрай важливо відстежувати шлях запиту між різними компонентами системи. Логування дозволяє фіксувати спроби несанкціонованого доступу, підозрілу активність або зміни критичних даних. Це дає можливість проводити форензік-аналіз, тобто розслідування інцидентів у випадку кібератак, розуміючи, який саме вразливий компонент було використано зловмисниками.

Актуальність логування для користувача насамперед полягає у надійності сервісу. Коли програма перестає працювати або видає помилку, саме наявність автоматично відправлених логів дозволяє розробникам швидко випустити оновлення з виправленням. Користувач отримує стабільніший продукт, навіть не усвідомлюючи, що саме його попередній невдалий досвід, зафіксований у логах, став основою для покращення.

Крім того, логування забезпечує прозорість та довіру у випадках спірних ситуацій, особливо у фінансових або бізнес-застосунках. Якщо користувач

стверджує, що здійснив транзакцію, а гроші не надійшли, або виконав певну дію, яка не збереглася, саме журнали аудиту стають доказовою базою. У багатьох корпоративних системах користувачам навіть надають доступ до спрощених версій логів, що дозволяє їм самостійно контролювати безпеку свого облікового запису та перевіряти, хто і коли входив у систему.

Також величезне зростання складності сучасних програмних систем (ПС) та мікросервісних архітектур призводить до того, що вартість та час, необхідні для виявлення регресивних дефектів, стають критичними обмежувальними факторами у CI/CD циклах. Традиційні методи тестування, наприклад, асерції ефективні для верифікації очікуваних результатів, але часто нездатні виявити непередбачувані побічні ефекти або аномалії у поведінці, які ще не призвели до явного збою.

Журнали (логи) виконання, за умови їх структурованості, є високоінформативним, хоча й зашумленим, відбитком реальної поведінки системи. Можна припустити, що коректно функціонуюча програмна система за детермінованих вхідних умов (тестів) генерує статистично стабільний та передбачуваний потік подій у журналі. Таким чином, будь-яке значуще відхилення у цьому потоці між двома версіями програмної системи є потужним індикатором регресивного дефекту або непередбачуваної зміни поведінки.

Величезні обсяги даних журналів щодня генеруються програмним забезпеченням. Ці дані містять їх короткий зміст, що є цінною інформацією про поведінку та стан системи, яка рідко використовується через її обсяг та неструктурованість. Ручний перегляд файлів журналів є часовитратною та трудомісткою процедурою для розробників.

Тим не менш, інформація журналів може виявити проблемне виконання програмного забезпечення, навіть якщо кінцевий результат здається нормальним.

Логування передбачає перехід від традиційних локальних лог-файлів до централізованих, структурованих та інтелектуальних систем обробки логів. Сучасне логування є не просто записом подій, а повноцінною частиною архітектури DevOps та SRE (Site Reliability Engineering).

Методологія SRE відноситься до інженерного підходу щодо управління великими виробничими системами. По суті, SRE застосовує принципи розробки програмного забезпечення до завдань, які традиційно виконували команди з експлуатації створюючи міст між розробниками, які хочуть швидко впроваджувати зміни та системними адміністраторами, які прагнуть стабільності.

Цей підхід був вперше розроблений та застосований компанією Google і став стандартом для високомасштабованих, надійних хмарних сервісів. SRE фокусується на автоматизації, вимірюванні та управлінні ризиками для забезпечення заданого рівня надійності. SRE часто розглядається як конкретна реалізація філософії DevOps.

Логування - це процес запису та зберігання інформації про події, що відбуваються під час роботи комп'ютерної системи, програмного забезпечення або мережі. По суті, логування створює хронологічний запис (журнал, лог-файл) усіх дій, операцій, помилок і станів системи. Цей запис використовується для діагностики, моніторингу, аудиту та забезпечення безпеки.

Логування є життєво необхідним інструментом для:

– діагностики та налагодження, що полягає у виявленні та виправленні помилок, збоїв або несподіваної поведінки програмного забезпечення;

– моніторингу продуктивності, а саме відстеження часу виконання функцій, використання ресурсів (пам'ять, процесор) та вузьких місць у системі;

– аудиту безпеки, що забезпечує відстеження доступу користувачів, спроб авторизації, критичних змін у конфігурації чи системі, наприклад, у журналі безпеки Windows;

– аналізу поведінки користувачів, наприклад, у вебзастосунках відстежуються шляхи користувачів, популярні функції та конверсії.

Кожен окремий запис тобто подія у лозі зазвичай містить стандартизований набір даних, а саме:

– мітку часу, тобто точний час, коли подія відбулася;

– рівень, тобто важливість події, наприклад, TRACE, DEBUG, INFO, WARNING, ERROR, CRITICAL;

– джерело, а саме компонент, модуль чи функція, яка згенерувала запис;

– повідомлення, тобто детальний опис події;

– контекст, тобто додаткова інформація, наприклад, ідентифікатор користувача, ідентифікатор запиту, стек викликів.

Логи розподіляються на рівні і використання рівнів дозволяє ефективно фільтрувати записи та відокремлювати критичні проблеми від рутинної інформації (Таблиця 1).

Таблиця 1 – Рівні логування

Рівень	**Призначення**	**Опис**
TRACE	Найнижчий	Дуже детальна інформація, використовується лише для глибокого налагодження.
DEBUG	Розробка	Інформація, корисна для розробників при покроковому відстеженні логіки.
INFO	Рутина	Стандартні повідомлення про нормальну роботу, наприклад, *Служба запущена, Користувач успішно увійшов.*
WARNING	Увага	Потенційна проблема, яка не зупиняє роботу, але може призвести до помилки в майбутньому, наприклад, *Застарілий метод.*
ERROR	Критична	Помилка, яка перешкоджає нормальній роботі частини функціоналу.
FATAL (CRITICAL)	Катастрофа	Критична помилка, яка призводить до зупинки всієї системи або застосунку.

Логування також має типи і залежно від місця збору, логування поділяється на:

– логування застосунків (Application Logging), куди входять записи, згенеровані самим програмним забезпеченням;

– системне логування (System Logging) - це записи, згенеровані операційною системою, наприклад, Windows Event Log, /var/log у Linux;

– мережеве логування (Network Logging), тобто записи від мережевих пристроїв (маршрутизатори, брандмауери).

У великих розподілених системах використовується централізоване логування, наприклад, через системи ELK Stack або Splunk, де лог-файли збираються з усіх компонентів в єдине сховище для подальшого аналізу.

Аналіз системи логування Windows є критично важливим для забезпечення безпеки, діагностики несправностей, моніторингу продуктивності та аудиту на будь-якому підприємстві. Основний механізм логування в операційній системі Windows реалізований через Журнал подій Windows, а саме Windows Event Log.

Журнал подій представляє собою ієрархічну систему, яка зберігає інформацію про події, що відбуваються в системі. Замість простих текстових файлів, Windows використовує бінарні файли формату .evtx, що забезпечує структуроване зберігання даних.

Windows розділяє події на кілька основних журналів, щоб спростити їхній перегляд та управління:

– журнал застосунків (Application Log), який записує події, що стосуються програм і застосунків, встановлених користувачем або системою, наприклад, збої програм, попередження від антивірусу;

– журнал безпеки (Security Log), який є найважливішим журналом для аудиту та містить події, пов'язані з безпекою, такі як спроби входу/виходу з системи, використання прав користувачів, доступ до об'єктів (файлів, ключів реєстру) та зміни в політиці безпеки;

– журнал системи (System Log), що містить події, згенеровані самою операційною системою та її компонентами, наприклад, запуск чи зупинка служб, помилки драйверів, проблеми з апаратним забезпеченням;

– журнал Setup (Setup Log), що фіксує події, пов'язані з інсталяцією Windows та її оновлень.

Починаючи з Windows Vista, було додано розширені журнали, які надають більш деталізовану інформацію від конкретних компонентів або застосунків, наприклад, Microsoft-Windows-TerminalServices, Microsoft-Windows-Hyper-V.

Кожна подія в журналі має чітку структуру, що є ключем до її аналізу:

– ID події або Event ID є унікальним числовим ідентифікатором, який визначає тип події, наприклад, Event ID 4624 у Security Log завжди означає успішний вхід у систему;

– джерело або Source є назвою компонента або програми, яка згенерувала подію, наприклад, Service Control Manager, Security;

– дата та час (Date and Time) показує дату та час, коли подія була зафіксована;

– користувач (User) та комп'ютер (Computer) вказує хто і де ініціював подію;

Також є рівень (Level), який визначає важливість події:

– Error (помилка) вказує на критична проблему, наприклад, збій служби;

– Warning (попередження) вказує на можливу проблему, яка не є критичною, але потребує уваги;

– Information (інформація) вказує успішна робота чи рутинна операція;

– Success Audit (успішний аудит) повідомляє чи успішна дія, пов'язана з безпекою;

Failure Audit (невдалий аудит) показує чи невдала спроба дії, пов'язана з безпекою (наприклад, невдала спроба входу).

Для забезпечення безпеки, ключова увага приділяється Журналу безпеки та правильному налаштуванню політики аудиту Windows (Windows Audit Policy):

– аудит входу чи виходу (Logon/Logoff) є життєво важливим для виявлення несанкціонованого доступу. Події 4624 - успішний вхід та 4625 - невдалий вхід є основними індикаторами;

– аудит доступу до об'єктів заключається у відстеженні доступу до конфіденційних файлів, папок, реєстру та допомагає виявити спроби ексфільтрації даних;

– аудит змін у системі полягає у відстеження змін у групах безпеки, користувацьких облікових записах, політиках та системних службах, наприклад, створення нового адміністратора;

– очищення журналу – це подія, що фіксує очищення журналів (Event ID 1102), є критичним індикатором того, що зловмисник намагається приховати свої сліди.

Разом з тим існують такі проблемні питання, що доцільно розглядати в якості дослідницьких напрямків. До таких питань відносяться:

– об'єм даних;

– контекст;

– налаштування політики.

На великих підприємствах генеруються терабайти логів, що вимагає централізованої обробки SIEM-системами. Логи часто потребують кореляції з іншими джерелами, мережеві логи, логи застосунків, для повного розуміння події. Неправильне налаштування політики аудиту, занадто широке або занадто вузьке може або перевантажити систему непотрібними даними, або пропустити критичні події.

1.2 Аналіз існуючих рішень

Нові способи логування включають перехід від текстів до структури, а саме централізоване логування та структуроване логування.

Концепція централізованого логування полягає у тому, що замість того, щоб зберігати лог-файли локально на кожному сервері чи в контейнері, усі логи збираються та агрегуються в єдиному місці. Перевагами є спрощення пошуку, а такж кореляція події, що відбуваються в різних мікросервісах, і значно покращує моніторинг розподілених систем.

Концепція структурованого логування полягає у тому, що логи записуються у форматі, який легко читається машиною, найчастіше JSON або XML. Замість неструктурованого рядка, кожен запис є об'єктом із чітко визначеними полями, наприклад, {*level: Error, transaction_id: ABC123, message: Database timeout}*. Перевагами є забезпечення високої швидкості обробки та аналізу, а також можливість створювати складні фільтри та запити, а також легко інтегрується із системами моніторингу та аналітики. Це є стандартом для сучасних хмарних і мікросервісних архітектур.

Трасування відноситься до розширеного методу логування, що сфокусований на відстеженні повного шляху виконання запиту через усі мікросервіси у розподіленій системі та відоме як Distributed Tracing.

Перевагами є можливість виявити вузькі місця та затримки у складному ланцюжку викликів, що є неможливим при звичайному логуванні. Стандарти, такі як OpenTelemetry, стають ключовими для реалізації трасування.

Найпопулярнішим набором інструментів для централізованого, структурованого логування є стек ELK або Elastic Stack, що показано у таблиці 5.

Таблиця 1 - Нові засоби та технології - The ELK Stack та його аналоги

Технологія	**Роль у системі логування**
Elasticsearch (E)	Сховище та індексація – це високопродуктивна пошукова та аналітична база даних, оптимізована для зберігання логів та швидких повнотекстових запитів.
Logstash (L)	Парсинг та перетворення є інструментом для збору, обробки, фільтрації та перетворення неструктурованих логів у структурований формат (JSON).
Kibana (K)	Візуалізація та інтерфейс здійснюється через графічний інтерфейс для пошуку, аналізу, створення дашбордів та візуалізації логів у реальному часі.

Beats (додатково)	Агенти збору - це легкі агенти, які встановлюються на кінцевих точках (серверах, ПК) для ефективної передачі логів до Logstash або безпосередньо в Elasticsearch.

Вбудовані та сторонні інструменти, що значно спрощують аналіз великих обсягів логів показано у таблиці 2.

Таблиця 2 – Типи інструментів для аналізу логів

Інструмент	Тип	Призначення
Event Viewer	Вбудований	Основний графічний інтерфейс для перегляду, фільтрації та експорту локальних журналів.
wevtutil	Командний рядок	Дозволяє керувати, експортувати та очищати журнали подій через командний рядок , що є ідеальним для автоматизаціїі.
PowerShell	Скриптинг	Командлети *Get-WinEvent* та *Get-EventLog* дозволяють гнучко фільтрувати, обробляти та експортувати дані для автоматизованого аналізу.
SIEM-системи	Сторонній	Системи управління інформацією та подіями безпеки, наприклад, Splunk, Elastic Stack, Microsoft Sentinel. Вони збирають логи з багатьох Windows-машин централізовано, корелюють події та автоматично виявляють аномалії та загрози.

Аналоги та хмарні рішення:

– Splunk - це комерційний лідер у сфері SIEM та логування, відомий потужними можливостями аналізу та кореляції даних;

– Loki (Grafana Labs) - альтернатива, орієнтована на ефективне зберігання та індексацію логів, де індексація відбувається лише за метаданими, а не за повним текстом, що робить її економічною;

– хмарні сервіси, наприклад, AWS CloudWatch, Azure Monitor, Google Cloud Logging, що надають повністю керовані сервіси для збору, аналізу та зберігання логів.

Сучасні технології виходять за рамки простого пошуку та візуалізації. В сучасних реаліях широко використовуються інтелектуальний аналіз логів, що включає:

– аномальне виявлення, що полягає у використанні машинного навчання для аналізу шаблонів логування та автоматичного виявлення нетипової поведінки, яка може свідчити про збої, атаки або несподівані зміни;

– автоматичне усунення неполадок, що має на увазі інтеграцію систем логування з інструментами автоматизації, наприклад, системами оповіщення та автоматичним перезапуском сервісів для мінімізації часу простою;

– Log-as-Code полягає в управлінні конфігурацією логування, фільтрами та дашбордами за допомогою коду, наприклад, Terraform, що забезпечує узгодженість та можливість відстеження змін.

Впровадження цих засобів і способів дозволяє IT-фірмам не лише діагностувати проблеми, а й проактивно їх запобігати, перетворюючи логи з архіву даних на оперативний інтелект.

1.3 Огляд методів логування програмного забезпечення

Логування програмного забезпечення є критично важливим аспектом розробки та експлуатації, слугуючи основним механізмом для збору даних про поведінку системи та її оточення. Методи логування можна узагальнити у кілька основних підходів, кожен з яких має свої переваги та недоліки.

Традиційне структуроване логування відноситься до класичного методу та передбачає запис подій у звичайні текстові файли з додаванням метаданих, таких як мітка часу та рівень важливості. Його головна перевага - простота реалізації та

легкість читання для людини. Розробник може швидко додати *print* або *log.info()* для діагностики. Проте, з аналітичної точки зору, цей метод є малоефективним у великомасштабних системах. Аналіз таких логів вимагає складного парсингу (розбору) за допомогою регулярних виразів, що є повільним, схильним до помилок і ускладнює автоматизовану обробку та агрегацію даних. Зі зростанням обсягу даних, вилучення смислової інформації стає дуже ресурсоємним завданням, обмежуючи можливості для глибокого аналізу тенденцій.

Структуроване логування є більш сучасним та аналітично ефективним підходом. Воно передбачає запис логів у форматах, які легко читаються машиною, найчастіше це JSON або XML. Замість вільного тексту, кожна подія логу є об'єктом із чітко визначеними полями. Це кардинально покращує аналітичну цінність логів, оскільки дані можуть бути безпосередньо завантажені в системи управління логами, наприклад, ELK Stack, Splunk без потреби в складному парсингу. Аналітик може швидко фільтрувати, групувати та агрегувати дані за будь-яким полем (наприклад, *user_id*, *transaction_status* або *latency_ms*). Недоліком є трохи вища складність впровадження порівняно з простим текстовим логуванням та потенційно більший обсяг файлів через включення метаданих у кожен запис.

Наступним методом є розподілене трасування, що виходить за рамки простих подій і зосереджується на поведінці запитів у складних, розподілених мікросервісних архітектурах. Розподілене трасування, наприклад, із використанням OpenTelemetry або Jaeger, створює унікальний ідентифікатор трасування для кожного запиту, що проходить через кілька сервісів. Логи від різних сервісів пов'язуються цим ідентифікатором, формуючи повний шлях виконання запиту від початку до кінця. З аналітичної точки зору, це надає безцінну інформацію для аналізу продуктивності, дозволяючи точно виміряти затримку між сервісами, виявити вузькі місця та зрозуміти каскадні збої. Це критично для Root Cause Analysis (RCA) у складних системах. Недоліки включають значну складність впровадження та потребу в додаткових інструментах для візуалізації трасування.

Ще одним методом є семантичне логування та рівні логування, що підкреслює змістовність та контекст подій, а не просто їхнє текстове

представлення. Воно тісно пов'язане зі структурованим логуванням, але акцентує увагу на стандартизації повідомлень і параметрів. Крім того, використання рівнів логування (DEBUG, INFO, WARN, ERROR, FATAL) є базовим аналітичним інструментом. Аналітик використовує ці рівні для керування обсягом даних, виводячи лише критичні події на етапі продакшну та вмикаючи деталізований рівень (DEBUG) лише для діагностики. Ефективне застосування семантичних рівнів дозволяє оптимізувати витрати на зберігання та значно прискорити пошук критичних проблем.

Сьогодні автоматичний аналіз файлів журналів є критично важливим для виявлення проблем, але головним чином для розуміння того, як поводиться програмне забезпечення, що було б корисним для запобігання збоям та вдосконалення самого програмного забезпечення. У цьому напрямку актуальним є спрямування на виявлення неочікуваних виконань програмного забезпечення та визначення їх першопричини. Більш детально, очікувану поведінку програмного забезпечення можна апроксимувати за допомогою методів модельного висновку, а нові спостережувані дані можна проаналізувати, щоб перевірити, чи відповідають вони очікуваній поведінці.

Метод перевірки відповідності, який називається повторенням полягає у тому, що вхідні траси будуть відтворені на графіку, і в момент їх неперевірки діятиме алгоритм вирівнювання.

Вирівнювання послідовності виконується трьома різними способами. Два методи шукають найкраще вирівнювання на певному радіусі навколо проблемного вузла. Крім того, актуальним є метод глобального вирівнювання, який базується на відомому алгоритмі Нідлмана та Вунша для послідовностей ДНК.

Сьогодні кожна система створює величезну кількість журналів, які, ймовірно, ніколи не будуть досліджені людським оком. Однак вони містять інформацію, яка може бути надзвичайно корисною для розуміння результатів виконуваних процесів та загальної поведінки системи. Використовуючи дані журналів, можна зробити різні висновки про запущене програмне забезпечення, яке їх створило. Наприклад, дані журналів можуть бути корисними для виявлення

потенційних аномалій, які можуть бути спричинені неочікуваним збоєм системи або зловмисною дією. Файли журналів містять неструктуровану текстову інформацію, що робить їх обробку досить дорогою процедурою. Тим не менш, це варте зусиль, оскільки міститься в них інформація має багато застосувань.

Також цікавим є дослідження, що намагаються вирішити проблему автоматичного розрізнення файлів журналів. Більш детально, це порівняння файлів журналів з метою виявлення потоків, які не повинні були відбуватися ні в одному, ні в іншому файлі. Такого роду інструменти можуть мати багато застосувань, таких як виявлення аномалій у нещодавно випущеному програмному забезпеченні або покращення тестового покриття шляхом виявлення непротестованих випадків у реальному середовищі.

Однак, оскільки щодня генерується кілька файлів журналів, наївний підхід порівняння файлів не може бути застосований у наших сценаріях. Вищезгадану проблему доцільно розглядати шляхом моделювання поведінки програмного забезпечення з використанням даних журналу та порівняння моделі (графу специфікації) з іншими записами журналу, що відбуваються в режимі реального часу, які можуть бути створені іншим програмним забезпеченням або іншою версією програмного забезпечення.

Моделювання поведінки програмного забезпечення – це проблема, яка неодноразово досліджується в різних роботах, тому доцільно застосовувати один конкретний підхід з літератури. Головним завданням, при такому підході є використання методу, який дозволить виконувати таке порівняння, з кінцевою метою виявлення потоків, що відхиляються від очікуваної поведінки. Крім того, вкрай важливо також знайти першопричину цього відхилення.

Загалом, щодня створюються трильйони рядків журналів даних. Одна система може створювати понад 10 рядків для однієї операції, наприклад, для однієї транзакції з карткою. Ця операція може складатися з кількох кроків, які реєструються для того, щоб транзакція була авторизована та оброблена. Ці журнали містять важливу інформацію щодо необхідних базових операцій відповідного програмного забезпечення. Розробники переглядають ці файли лише тоді, коли

трапляється щось критичне, що перешкоджає загальній роботі програмного забезпечення. Тим не менш, багато речей могли бути виконані неправильно, що не впливає на кінцевий стан процесу. Наприклад, такий випадок може спостерігатись в транзакції з кредитною карткою, де з якоїсь причини не було запрошено PIN-код, бо пропущено частину автентифікації. Транзакція буде виконана без проблем, але якщо хтось перегляне її журнал, буде виявлена аномалія. Ручне дослідження журналів займає величезну кількість часу та може бути занадто складним для людського мозку. З цією метою виявляються способи, за допомогою яких цю інформацію можна витягти для виявлення відмінностей у поведінці програмного забезпечення. В епоху великих даних критично важливо знайти способи вилучення інформації з такого величезного пулу даних. Однак, пов'язані роботи [4, 5] довели важливість інструментів, які використовують файли журналів, щоб отримати уявлення про процеси виконання програмного забезпечення з іншої точки зору. Також дуже важливо перевірити, чи працює впроваджене програмне забезпечення відповідно до своїх специфікацій. Цього можна досягти, порівнявши очікувану поведінку, засновану на специфікаціях, з фактичною поведінкою програмного забезпечення, заснованою на виконанні.

Розробка програмного забезпечення спирається на використання моделей, які можуть визначити, як система повинна поводитися або як вона насправді працює. Програмне забезпечення може бути складним, тому розробникам зазвичай важко виявити проблеми або відмінності між програмним забезпеченням. Файли журналів містять інформацію, яку можна було б детально розглянути для виявлення відмінностей або проблем, проте це непросте завдання.

Існує багато досліджень, зосереджених на аналізі журналів. Наприклад, дослідник Пітер Еверс розробив інструмент під назвою Logness з вебінтерфейсом, який використовує алгоритм Longest Common Subsequence для кластеризації журналів з рівнем серйозності WARN та ERROR. Logness надсилає сповіщення розробникам, якщо створюється новий кластер журналів, який також можна використовувати для моніторингу релізу. Над цим самим інструментом працював Квінсі Баккер, який намагався вдосконалити його, використовуючи різні методи,

такі як аналіз вимог, інтерв'ю, візуалізація даних та аналіз користувачів. Рік Віман [6] оцінює різні методи пасивного навчання та використовує їх, щоб надати розробникам інтуїтивне уявлення про поведінку програмного забезпечення, що працює на POS-терміналах. Йооп Ауе досліджує використання WEB API в Adyen та рекомендує способи уникнення помилок API. Даан Шиппер [5] працював над оцінкою алгоритмів відстеження даних журналу до їх походження за допомогою статичного аналізу.

У [4] Фу та ін. розробили метод автоматичного виявлення аномалій у файлах журналів, що генеруються розподіленими системами (тип аномалій, на якому вони зосереджуються, - це низька продуктивність). Їхній метод складається з двох процесів: процесу навчання та процесу виявлення. Процес навчання стосується навчання моделі з використанням нормальних даних. Для досягнення цього вони спочатку перетворюють повідомлення журналу з файлів журналів навчання на ключі журналів. Використовуючи ці ключі, використовується FSA для моделювання шляху виконання системи. Також необхідно обчислити час виконання для кожного переходу станів. У процесі виявлення нові вхідні журнали можна перевірити на наявність аномалій за допомогою вищезгаданої вивченої моделі. Тип аномалії, на якому вони зосереджуються відноситься до низької продуктивності. Більш детально вони поділяють низьку продуктивність на два типи: один - моделювання часу виконання переходів станів, а другий - моделювання кількості циклів.

Подібне дослідження, але з іншим підходом, було проведено також Маріані та Пасторе [5]. Вони також представляють методику автоматичного аналізу файлів журналів та отримання необхідної інформації для виявлення збоїв. Це дослідження зосереджено на іншому виді аномалії, а саме на збоях. Їхній підхід складається з трьох фаз. Фаза 1 – моніторинг, фаза 2 – генерація моделі та фаза 3 – аналіз збоїв. На етапі моніторингу відбувається збір файлів журналів, які можуть бути згенеровані або під час тестування, або в результаті фактичного використання системи. Тим не менш, вони мають бути успішними виконаннями, оскільки вони будуть використані для процесу навчання. Фаза генерації моделі складається з

трьох основних завдань: виявлення подій, перетворення даних та виведення моделі. Завдання виявлення подій відповідає за перезапис подій початкового файлу журналу, щоб зробити його структурованим. Завдання перетворення даних видаляє конкретні значення атрибутів, що містяться в журналі, більш змістовною інформацією про потік даних. Останнім завданням цієї фази є виведення моделі, тобто тут задіються механізм виведення. Під час останньої фази, аналізу збоїв, аномальні закономірності виявляються за допомогою моделі, вивченої на попередньому етапі. Недоліком цього методу є те, що розробник повинен анотувати потенційні аномальні закономірності як аномальні або нормальні.

Цікавий підхід до виявлення аномалій з використанням формального аналізу запропоновано в [6]. Метою їхньої роботи є перевірка достовірності реалізації на основі специфікацій. Для цього вони пропонують формальний підхід до виявлення вразливостей з використанням символьної алгебри.

Таким чином, вони моделюють специфікацію як набір поліномів, так що кожен поліном відповідає за опис усіх допустимих станів та відповідних їм переходів, за допомогою яких можна досягти певного стану. Наступним кроком є також моделювання за допомогою поліномів. Оскільки і специфікація, і реалізація є моделями, наступним кроком є перевірка еквівалентності цих двох наборів. Для порівняння двох наборів була використана теорія базисів Грьобнера, шляхом зменшення набору реалізації та використання решти можна було знайти вразливості. Більш детально, нульовий залишок означає, що реалізація відповідала специфікаціям і їй можна довіряти. З іншого боку, ненульовий залишок означає, що в реалізації, а також в умовах є приховані несправності, які їх активують, визначено.

Їхнє дослідження стосується наступних трьох проблем. По-перше, проблема полягає в тому, що формулювання специфікацій загальної схеми не може бути змодельоване як один простий та вичерпний поліном, саме тому використовуються набори поліномів, один поліном для кожного стану. По-друге, існує проблема циклів у послідовних схемах, де ці цикли можуть зробити процедуру редукції нескінченною. І останнє, але не менш важливе, полягає в тому, що аномалія може

з'явитися після тривалого повторення нормальних станів, тобто після великої кількості циклів.

Аналіз журналів може використовуватися для багатьох різних цілей, таких як виявлення вузьких місць у продуктивності або помилок. У цій галузі кінцеві автомати широко використовуються завдяки своїй здатності точно моделювати поведінку системи.

Досить цікавим є метод із використанням кінцевих автоматів для моделювання поведінки системи з метою виявлення проблем продуктивності. Для цієї мети було використано три різні алгоритми: kTails [3], Synoptic [4] та Perfume [5].

Найкращі результати були отримані при використанні Perfume, який є розширенням Synoptic принциповим чином для врахування інформації про продуктивність, часто доступної в системних журналах. Мета Perfume полягає у створенні репрезентативної моделі системи з журналу виконання, що містить приклади поведінки системи. Для досягнення цієї мети вдосконалюються найсучасніші методи виведення моделей, щоб використовувати інформацію про продуктивність у журналі для керування процесом виведення та точніше прогнозувати неспостережувані виконання.

Процес, який Perfume використовує для виведення моделей, складається з етапу парсингу, аналізу властивостей продуктивності, початкової побудови моделі, уточнення CEGAR (уточнення абстракції на основі контрприкладів) та укрупнення kTails. Після цих кроків будується остаточна модель. Автори стверджують, що остаточна модель буде легшою для розуміння розробниками. Таким чином, ці моделі можна використовувати для тестування продуктивності.

1.4 Постановка задачі

Перед початком роботи над кваліфікаційною роботою магістра було визначено мету дослідження, що полягає у вдосконаленні методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Об’єктом дослідження є процес автоматизованого виявлення помилок у програмному забезпечення із використанням автоматизації на основі диференціального аналізу логів виконання

Предметом дослідження є сучасні методи логування, що використовуються для виявлення помилок.

Відповідно до поставленої мети, предмету та об’єкту дослідження було встановлено наступні задачі:

1. Провести аналітичне дослідження предметної області досліджуваної проблеми.
2. Здійснити аналітичний огляд методів логування.
3. Здійснити удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Наукова новизна:

1. Удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.
2. Розроблено алгоритм роботи методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Таким чином, в даній кваліфікаційній роботі має бути проведено удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Результати цієї роботи можуть в подальшому бути використаними програмістами для автоматизації роботи із журналами логів, їх аналізу та перегляду.

1.5 Висновки до 1-го розділу

У першому розділі здійснено введення поняття логування та його значення під час реалізації та користування програмним забезпеченням.

Виявлено, що логування - це процес запису та зберігання інформації про події, що відбуваються під час роботи комп'ютерної системи, програмного забезпечення або мережі. По суті, логування створює хронологічний запис (журнал, лог-файл) усіх дій, операцій, помилок і станів системи. Цей запис використовується для діагностики, моніторингу, аудиту та забезпечення безпеки.

Логування є життєво необхідним інструментом для:

– діагностики та налагодження, що полягає у виявленні та виправленні помилок, збоїв або несподіваної поведінки програмного забезпечення;

– моніторингу продуктивності, а саме відстеження часу виконання функцій, використання ресурсів, наприклад, пам'ять, процесор та вузьких місць у системі;

– аудиту безпеки, що забезпечує відстеження доступу користувачів, спроб авторизації, критичних змін у конфігурації чи системі;

– аналізу поведінки користувачів, наприклад, у вебзастосунках відстежуються шляхи користувачів, популярні функції та конверсії.

2 МЕТОД АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ПОМИЛОК У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ НА ОСНОВІ ДИФЕРЕНЦІАЛЬНОГО АНАЛІЗУ ЛОГІВ ВИКОНАННЯ

2.1 Концептуальна модель для диференціального аналізу логів виконання

Концептуальна модель методу автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання зосереджена на ідентифікації аномалій та відхилень у поведінці системи шляхом порівняння її поточного стану, представленого у логах із відомим, коректним або нормальним станом. Цей підхід є ключовим для систем проактивного моніторингу та діагностики та включає декілька етапів:

- етап збору та нормалізації логів;
- етап формування базової лінії;
- етап диференціального аналізу.

На початковому етапі відбувається збір сирих даних та їх підготовка для аналізу, що включає збір логів, парсинг та нормалізацію, генерацію шаблонів логів.

Збір логів відбувається, коли система збирає структуровані або неструктуровані логи виконання з різних компонентів програмного забезпечення, наприклад, сервіси, бази даних, зовнішні інтерфейси користувача тощо. Важливо збирати як позитивні логи від нормальних, успішних транзакцій, так і негативні логи, наприклад, від транзакцій, що завершилися збоєм, у разі їхньої наявності.

Парсинг та нормалізація (Parsing and Normalization) відбувається, коли сирі текстові логи трансформуються у структурований формат, наприклад, JSON, де виділяються ключові поля, а саме ідентифікатор транзакції, мітка часу, рівень логування, ідентифікатор компонента та текст повідомлення. У свою чергу це створює масив подій.

Генерація шаблонів логів (Log Template Generation) використовується для подальшого порівняння використовується метод логаналізу для перетворення варіативних текстових повідомлень на стандартизовані шаблони.

На етапі формування базової лінії (Baseline Creation) визначається нормальна поведінка системи, що складається із визначення референсної множини (Reference Set); формування профілю базової лінії, до якої в свою чергу входять розподілу шаблонів, послідовності подій, параметри часу, вектор логування.

На етапі визначення референсної множини вибирається набір логів, які відповідають відомому коректному періоду роботи системи. Це може бути період роботи після успішного розгортання або період, коли всі метрики продуктивності були в нормі.

Формування профілю базової лінії відбувається на основі референсної множини, тобто створюється статистичний профіль нормальної роботи. Цей профіль включає:

– розподіл шаблонів, що полягає у частоті появи кожного шаблону логу в нормальному стані;

– послідовність подій містить типові послідовності викликів функцій або взаємодії компонентів;

– параметри часу, тобто включає нормальний час виконання певних блоків коду або транзакцій;

– вектор логування, коли кожному шаблону логу та його частоті присвоюється векторне представлення.

Етап диференціального аналізу (Differential Analysis) відноситься до ядра моделі, де відбувається порівняння та включає формування поточного профілю, обчислення диференціала, виявлення аномалій

Формування поточного профілю, коли логи, отримані в поточному часовому вікні тестовані або розгорнуті, проходять етапи парсингу та шаблонізації для створення поточного профілю за тією ж векторною моделлю.

Обчислення диференціала відбувається шляхом обчислення різниці (диференціалу) між поточним профілем та профілем базової лінії. Це може бути реалізовано за допомогою метрик косинусної відстані або відносної зміни частоти шаблонів.

Виявлення аномалій (Anomaly Detection) відбувається, коли значна зміна у векторі логування, а саме високе значення сигналізує про аномалію. Аномалія може бути виражена через нові шаблони, коли відбувається поява шаблонів логів, яких не було в базовій лінії.

Наступним кроком може бути зникнення шаблонів, тобто несподіване припинення логування нормальних подій. Зміна частоти відбувається через різке зростання частоти шаблонів помилок або попереджень.

Четвертим етапом в концептуальній моделі є ідентифікація та локалізація помилок, коли виявлені аномалії пов'язуються з конкретними помилками. Даний етап складається із таких складових:

- кластеризації аномалій;
- ранжування помилок;
- локалізація;
- генерації звітів.

Кластеризація аномалій полягає у тому, що схожі аномальні події групуються в кластери для визначення кореня проблеми.

Ранжування помилок відбувається, коли аномалії, пов'язані з шаблонами з високим рівнем критичності, наприклад, FATAL, ERROR, або ті, що мають найбільше відхилення від базової лінії, ранжуються як найбільш ймовірні помилки.

Локалізація полягає у тому, що ідентифіковані аномальні події трасуються за ідентифікатором транзакції та ідентифікатором компонента, щоб вказати точне місце у коді чи компоненті системи, де відбулося відхилення.

Генерація звітів відбувається через автоматичне створення звітів для інженерів, що містять ідентифікатор аномалії, диференціал, задіяні шаблони логів та посилання на код компонента.

Отже, концептуальна модель складається з декількох етапів, які в свою чергу включають певну послідовність кроків, після здійснення яких відбувається реалізація запланованого процесу логування.

На рисунку 2.1 продемонстровано концептуальну модель для диференціального аналізу логів виконання.

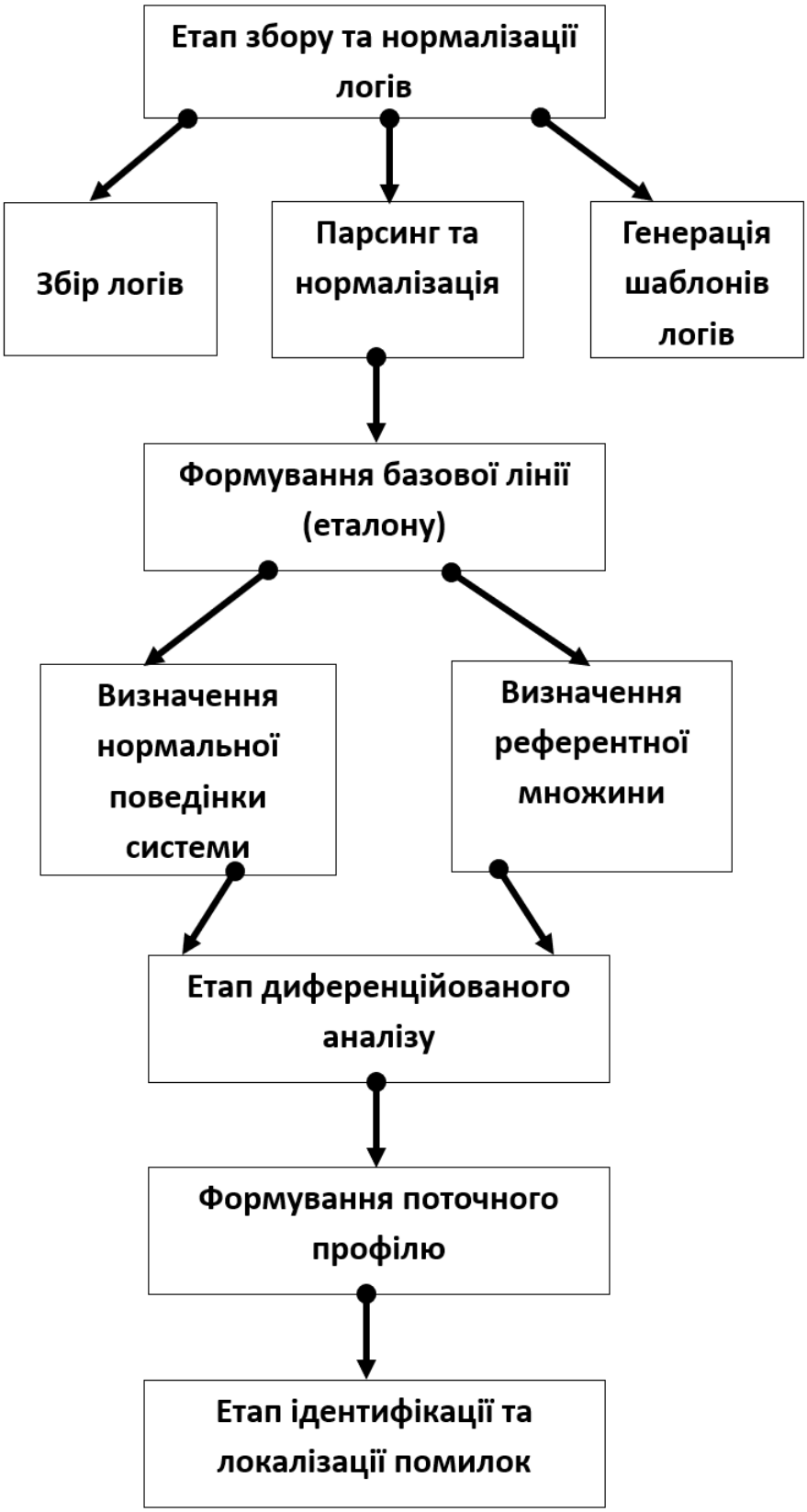


Рисунок 2.1 – Концептуальна модель

2.2 Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Метод перевірки відповідності, який буде використано, називається повторенням, коли відбувається відтворення на графіку вхідних трас, і в момент їх неперевірки діятиме алгоритм вирівнювання. Вирівнювання послідовності виконується трьома різними способами. Це відбувається, коли два методи шукають найкраще вирівнювання на певному радіусі навколо проблемного вузла. Крім того, реалізовано метод глобального вирівнювання, який базується на відомому алгоритмі Нідлмана-Вунша для послідовностей.

Розробка методу, який дозволить ефективно виявляти аномалії в поведінці програмного забезпечення, використовуючи дані журналу, на льоту. Як відбуватиметься моделювання поведінки, вже відомо з попередніх розділів, у даному випадку з використанням кінцевих автоматів, а точніше інструменту.

Початкова ідея полягає в тому, щоб знайти та реалізувати алгоритм порівняння графів, який отримуватиме на вхід два кінцеві автомати, що представляють поведінку програмного забезпечення, з метою візуалізації відмінностей у компактному та зрозумілому вигляді. Однак, є складність, яка полягає в тому, що недостатньо знайти першу різницю в трасі, а потім позначити її як розбіжний шлях до листка, як це робиться зазвичай. Дуже важливо визначити, що змінилося, що призвело до цього розбіжного шляху.

Почавши впроваджувати різні алгоритми порівняння графів, стало досить зрозуміло, що складність порівняння всіх можливих ребер між ними була нестерпною для великих обсягів даних, особливо якщо вони часто змінюються. Потреби даної роботи вимагають розробки інструменту, який зможе обробляти швидкість надходження даних журналу транзакцій, іноді сотні тисяч записів за хвилину.

Потреба в алгоритмі без експоненціальної складності з точки зору існуючих шляхів була основним фактором процедури прийняття рішень. Спроба порівняння дискретних послідовностей, тобто потоків транзакцій, які насправді є

послідовностями станів, базується на тій самій ідеї, що й порівняння між послідовностями.

Метою в обох задачах є визначення того, що призводить до їх розбіжності в кожній точці, та модифікація їх так, щоб вони стали максимально схожими (вирівнювання). Таким чином, цю ідею було використано для розробки інструменту порівняння разом із концепцією відтворення перевірки відповідності, де траси відтворюються в моделі, щоб перевірити, чи вони валідовані. Крім того, для пошуку оптимального маршруту до запитуваного вузла доцільно використати більше одного алгоритму.

Попередня обробка використовується, коли для запропонованого методу необхідні дані, що відображають поведінку системи, а також інший набір даних, який слід перевірити на відповідність моделі системи.

Першим кроком конвеєра є попередня обробка файлів журналів; це виконується за допомогою інструменту, створеного Ріком Віманом. Потім ці попередньо оброблені дані подаються до алгоритму машинного навчання станів, тобто виводиться (кінцевий автомат), яка буде вважатися графом специфікації.

Перевірка відповідності проблеми та ідеї вказують на використання конвеєра. Для цього конвеєра прийнято припущення, що є деякі дані специфікації та деякі вхідні дані, які необхідно перевірити.

Можна припустити, що дані специфікації здатні відображати очікувану або бажану поведінку програмного забезпечення та вважаються основою, за допомогою якої будуть виявлені аномалії у вхідних трасах.

Тому досить доцільно називати дані, що використовуються для навчання моделей, даними специфікації, а траси, що перевірені на наявність аномалій, - спостережуваними даними.

На рисунку 2.2 показано зображення структури методу високого рівня, що розглядає дані журналу і відповідно їх попередню обробку, далі ідентифікацію відмінностей із подальшим їх сортуванням. І кінцевим етапом роботи такого методу є відповідно візуалізація.

Рисунок 2.2 - Структура методу високого рівня

Керуючись бажанням створити швидку та ефективну систему для великих обсягів даних, застосовується підхід, який повністю відрізняється від того, що досі використовується для порівняння журналів.

Відомо, що моделювання поведінки займає багато часу, оскільки процедура навчання є трудомісткою, тому перебудову кожного разу, коли надходять нові дані, не є реалістичним варіантом. Для цього запропонований метод використовує модель очікуваної або бажаної поведінки програмного забезпечення та спостережувані дані, які необхідно перевірити з журналів, лише попередньо оброблені в текстовому форматі.

Тому доцільно знаходити аномалії в даних, перевіривши, чи вони підтверджені моделлю специфікації. У цьому випадку, виконується перевірка відповідності, перевіряючи, чи відповідають спостережувані траєкторії графу специфікації. Граф специфікації повинен моделювати очікувану поведінку системи. Чим повніша ця модель, тим кращі результати отримуються. Наведений підхід поділено на дві основні частини. Перша – це конвеєр для виявлення відмінностей, а друга, редукція або групування цих результатів

Після розбору кожної траси з файлу журналу, який містить спостережувані дані, першим кроком є проходження алгоритму перевірки відповідності. Перевірка відповідності відповідає за ідентифікацію трас, які неперевірені, шляхом їх відтворення на графі. Під перевіреним мається на увазі, що траса від першого стану до останнього повинна з'являтися в моделі та повинна закінчуватися в одному й тому ж кінцевому стані. Кожне ребро та стан траси повинні існувати в точно такому ж порядку на графі.

Для досягнення цього відбувається проходження всього дерева, та здійснюється пошук наступного переходу вхідної траси на кожному кроці. Якщо в якийсь момент наступний перехід послідовності не може бути знайдений в очікуваному порядку на дереві, вирівнювання послідовності має розпочатися, щоб перевірити, що пішло не так. Підсумовуючи частину перевірки відповідності, якщо трасування перевірено, воно позначається як існуюче, і алгоритм продовжує роботу з наступним, у випадку, якщо воно не перевірено, необхідно ідентифікувати відмінності, що буде виконано алгоритмом вирівнювання послідовностей.

Мета вирівнювання послідовності полягає у визначенні однакових та неоднакових частин траси. Траса модифікується шляхом додавання та видалення необхідних дій, щоб вирівняти її з найбільш схожою частиною з кінцевого автомата. Алгоритм може виявити кілька відмінностей (рисунок 2.3).

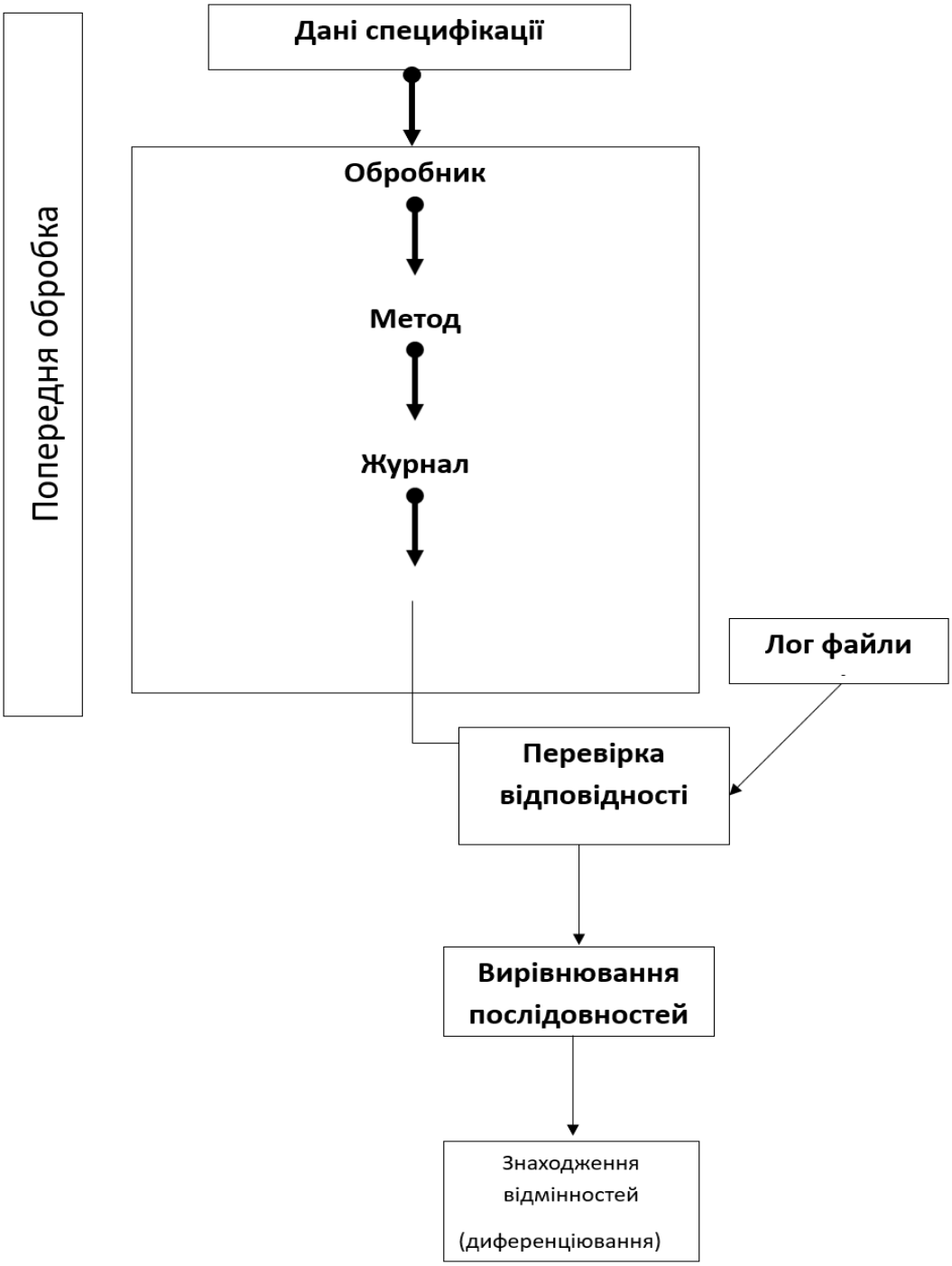


Рисунок 2.3 – Перевірка відповідностей

Випадок, коли з'являється багато послідовних пропущених або доданих станів, слід дослідити далі, оскільки протягом низки послідовних пропущених станів шляхи абсолютно різні, і немає сенсу знаходити частини, які є однаковими. Представлений алгоритм повинен повертати всі модифікації лише в одній з послідовностей, оскільки інша послідовність, у нашому випадку, є цілим деревом.

Результатом реалізованого алгоритму для буде: T/+, A, C/-, Ґ, T, C/-, Ґ/+, A/-, T, C, A Знак + (плюс) означає, що цю дію додано, отже, вона існує в послідовності , але не в дереві. Знак - (мінус) означає, що цю дію видалено, отже, дії немає в послідовності, хоча вона очікувалася згідно з деревом. Перші два представлені алгоритми (жадібний та метод пошуку найкращого) просто шукають нерівномірний стан на кожному кроці та способи його виправлення. Ці два алгоритми названі на честь алгоритму пошуку, який використовувався для визначення пропущених або доданих елементів.

Щоб використовувати деякі основні кроки для всіх алгоритмів, достатньо модифікувати три варіанти вирівнювання послідовностей, щоб вони відповідали потребам поточної задачі, і в результаті отримали такі випадки.

Якщо дві дії однакові, крок виконується в обох послідовностях. У першій послідовності можна зробити один крок, і якщо тоді послідовності вирівнювання це випадок доданого стану. У другому вирівнюванні (у даному випадку дерево) робиться один крок, якщо тоді послідовності вирівнюються, це випадок пропущеного стану. Окрім вищезгаданих випадків, є ще один, необхідний через природу даних. Якщо жоден із вищезгаданих випадків не виконується, ймовірно, має місце випадок заміненого стану. Це означає, що якщо алгоритм переміщується на один крок як у дереві, так і в послідовності, наступні стани вирівнюються.

Вибір одного з вищезгаданих випадків залежить від використаного алгоритму. Для цього використовуються три різні алгоритми, кожен з яких має різні переваги та недоліки.

Вирівнювання між послідовністю та деревом було виконано трьома різними способами. Спочатку - наївним методом, де на кожному кроці відбувається пошук відсутніх станів. Однак, перше рішення повертається незалежно від балів. Потім - покращеною версією, де всі чотири варіанти, розглядаються на кожному кроці, і вибирається той, що має найкращі результати щодо наступних станів. Нарешті, реалізовано підхід динамічного програмування, єдиний глобальний алгоритм вирівнювання послідовностей, що означає, що необхідно враховувати всі стани, а не лише сусідні, як це було зроблено попередніми двома методами.

Дані для побудови моделі специфікації повинні мати певні характеристики, вони повинні бути максимально повними, що означає, що функціональність програмного забезпечення має бути охоплена, а не лише найчастіші потоки. З цієї причини дані від роботів (тестових транзакцій) здаються найбільш доцільними, оскільки вони охоплюють широкий спектр можливих потоків, а не лише ті, що трапляються добре.

Тим не менш, це також залежить від порівняння, тобто у деяких випадках дані повинні включати лише конкретні потоки з більшою деталізацією, наприклад, під час порівняння двох послідовних тижнів даних для конкретного продавця, тоді для графіка специфікації було б корисно мати модель, яка охоплювала б усі звичайні потоки, але якщо це неможливо, необхідні лише найчастіші.

Загрози валідності зберігаються, оскільки дані є одним із найважливіших факторів, що впливають на результати. Важливо порівнювати дані, які мають певну спільну поведінку, і виявлення відхилення потоків матиме певний сенс.

Коректність моделей відноситься до загальних проблем, разом з тим, до якої нелегко підійти, адже її можна оцінити з точки зору простоти, придатності, узагальнення та точності.

Доцільно розглядати коректність використовуваних моделей і зробити припущення, що коли алгоритм порівняння повертає різницю, це фактична різниця, а не помилка процесу навчання. Продуктивність інструменту чутлива до специфікації моделі. Чим складніша модель, тим гіршими будуть результати вирівнювання послідовностей. Це також означає, що для більших моделей продуктивність гірша.

Важливим питанням є оптимальний розмір моделі - це тема, яка дуже хвилює дослідницьку спільноту, оскільки для малих моделей (більше злиттів) інформація втрачається, а для великих моделей (багато станів) вона стає дуже складною без належного узагальнення.

Отже, необхідно знайти баланс між цими двома поняттями. Однак, з більшою кількістю даних неминуче будуть створені більші моделі, тому питання, на яке потрібно відповісти, полягає в тому, скільки даних достатньо. Динамічне

програмування значною мірою залежить від визначених штрафів. бали відіграють дуже важливу роль у підході динамічного програмування. Для деяких балів точність може бути меншою.

Методом спроб і помилок випробовуються різні значення штрафів і обираються ті, що мають найкращі результати. Можна спробувати покращити результати, впровадивши динамічне оцінювання. Тим не менш, конкретна проблема потребує подальшого дослідження, і цілком можливо, що найкраще можливе значення не буде знайдено. Висновок полягає в тому, що вплив балів на продуктивність алгоритму можна вважати обмеженням, оскільки відсутня інформація, як ефективно налаштувати параметри.

Оскільки немає маркованих даних щодо відмінностей між порівнюваними журналами, можна самостійно змінити набір даних. Можна наблизити їх до фактичних відмінностей, проте завжди було б точніше оцінити їх за допомогою реальних даних, щоб ми знали, які відмінності вони представляють.

Характер даних змушує інструмент створювати занадто багато відмінностей, а це означає, що ручна перевірка займає забагато часу. Однак доведено, що деякі відмінності не є значущими та повинні бути вилучені з результатів або принаймні фільтрується, якщо користувач цього забажає. Наприклад, відмінності в конфігураціях між тестовою та реальною версіями становлять значну частину повернутих результатів і перешкоджають виявленню важливих відхилень, які можуть бути помилками системи або відсутніми тестовими випадками. Відповідно, можна проводити подальші дослідження щодо того, як можна відфільтрувати ці надлишкові відмінності. Крім того, можна створити базу даних з неаномальними відмінностями, щоб уникнути вищезгаданої проблеми. Інструмент розрізнення журналів між реальними та тестовими даними можна використовувати для перевірки покриття тестами та автоматичного створення тестових сценаріїв. За допомогою конвеєра, реальні транзакції, які відхиляються від очікуваної поведінки (тієї, що вказується тестовими даними), можуть автоматично створювати тестовий випадок, який запускатиметься на тестовій платформі та перевірятиме правильність або неправильність виконання відповідно до специфікацій.

На початкових етапах потрібно буде вручну анотувати потоки, які необхідно протестувати, а потім автоматично створюватиметься тестовий випадок для досягнення цієї мети. У певний момент покриття тестами буде набагато кращим, тому кожну різницю, виявлену інструментом розрізнення журналів, можна буде протестувати та анотувати як аномалію або ні для подальшого покращення ідентифікації.

2.3 Алгоритм автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Зазначений метод є високотехнічною процедурою, що базується на методах обробки природної мови (NLP), статистичного аналізу та машинного навчання. Нижче подано алгоритм виконання диференціального аналізу логів для автоматизованого виявлення помилок.

Алгоритм диференціального аналізу логів виконання включає декілька етапів:

- формування еталонної базової лінії;
- збір цільового логу та попередня обробка;
- вилучення диференціальних властивостей;
- диференціальний аналіз та локалізація;
- генерація звіту та адаптація.

На етапі формування еталонної базової лінії (Baseline Construction) відбувається визначення множини успіху, а саме визначається і збирається представницька множина лог-файлів з успішних або безпомилкових запусків системи. Ця множина слугує еталоном нормальної поведінки.

Створення еталонної моделі засновано на розподілі очікуваних частот кожного типу події або модель послідовності подій, наприклад, Марковська модель або граф виконання.

Статистична фіксація відбувається для кожної події логу, тобто фіксується її очікувана частота та дисперсія, що визначає статистичну норму.

Збір цільового логу та попередня обробка включає збір логу помилки, коли збирається лог-файл із цільового виконання, що призвело до зареєстрованої помилки або аномальної поведінки. Далі здійснюється парсинг логів (Log Parsing), лог-файли проходять через спеціалізований парсер, наприклад, Drain, Spell. Парсер перетворює неструктурований текстовий лог у структуровані шаблони подій (Event Templates) та присвоює кожному унікальний ідентифікатор.

Нормалізація полягає у тому, що з логів видаляються змінні дані (timestamp, ідентифікатори потоків, змінні даних), які не є суттю події, для забезпечення чистого порівняння шаблонів.

Вилучення диференціальних властивостей полягає у векторизації, коли обидва лог-файли перетворюються на вектори властивостей, де кожен вимір вектора відповідає ідентифікатору події, а значення - її частоті або послідовному індексу.

Формування вектора частот формується через створення вектору, де значенням є фактична частота появи кожного шаблону події у цільовому лозі.

Диференціальний аналіз та локалізація включає в себе обчислення розбіжності, коли обчислюється метрика відмінності або між очікуваними значеннями з еталонної моделі. Це може бути косинусна відстань (Cosine Distance), що використовується для порівняння семантичного напрямку виконання.

Статистичний критерій, тобто використання критерію для визначення того, чи є спостережувана частота події у статистично значущим відхиленням від її очікуваної частоти.

Визначення критичної послідовності відбувається шляхом ідентифікації послідовності подій, у якій вперше було зафіксовано відхилення, або де накопичена розбіжність є найбільшою.

Локалізація помилки, коли визначається критична подія - подія, для якої розбіжність перевищує встановлений поріг статистичної значущості. Ця подія

вважається найбільш імовірною точкою виникнення помилки або її безпосереднім наслідком.

На заключному етапі відбувається генерація звіту та адаптація. Створення звіту передбачає генерування структурованого звіту, що включає: контекстний фрагмент логу, що містить цю подію, та оцінку впевненості (Confidence Score), яка показує ймовірність коректності локалізації (на основі розміру статистичного відхилення).

Адаптація базової лінії грунтується на тому, що якщо помилка призвела до появи нового, невідомого раніше шаблону події, цей шаблон додається до словника парсера. Якщо помилка є варіантом вже відомого успішного сценарію, може бути використаний для уточнення дисперсії еталонної моделі, що забезпечує постійну адаптацію алгоритму до мінливої поведінки системи.

На рисунку 2.2 зображено алгоритм роботи програмної системи на основі методу автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

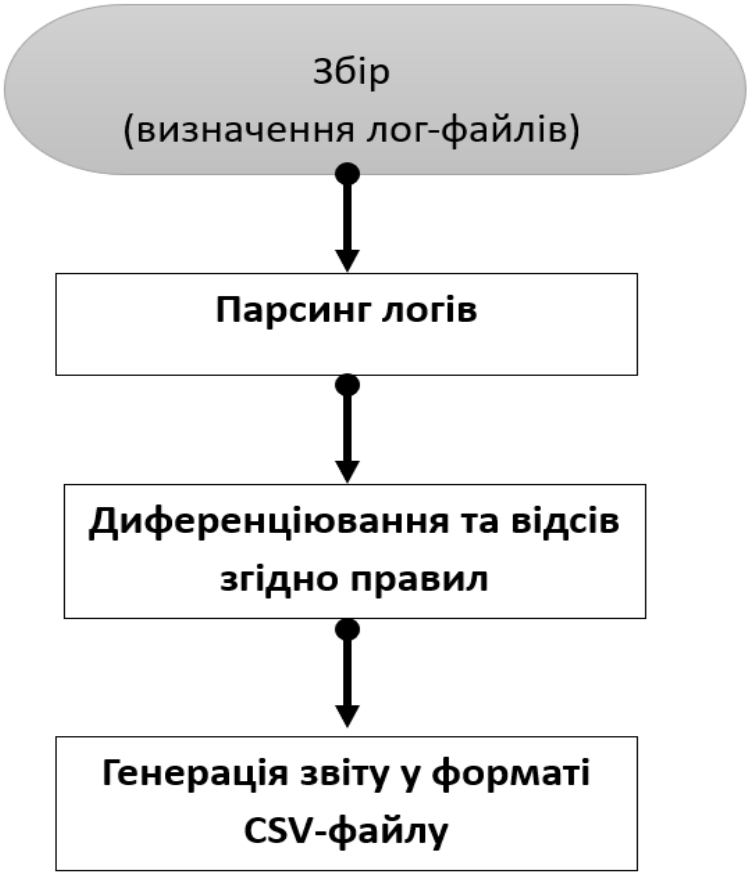


Рисунок 2.2 – Алгоритм

Отже, як бачимо даний алгоритм є досить привабливим та зручним у використанні для виявлення помилок у програмному забезпеченні.

2.4. Висновки до другого розділу

Отже, у другому розділі було здійснено вдосконаленння методу автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання.

Сформовано концептуальну модель автоматизованого виявлення помилок у програмному забезпеченні, що ґрунтується на диференційному аналізі логів виконання, є методом проактивної діагностики та аналізу аномалій у складних

розподілених системах. Суть методу полягає у метричному порівнянні поточного вектора поведінки системи, представленого у логах із референтною моделлю або базовою лінією нормальної роботи.

Суть алгоритму полягає у роботі таких етапів:

– етап збору, нормалізації та шаблонізації;
– формування профілю базової лінії;
– диференційний аналіз та виявлення аномалій;
– ідентифікація, локалізація та ранжування помилок.

Процес починається зі збору сирих логів виконання, які можуть надходити з множини гетерогенних компонентів програмного забезпечення. Для забезпечення можливості машинного аналізу ці сирі дані підлягають парсингу та нормалізації, трансформуючись у структурований формат, де чітко виділені метадані, такі як ідентифікатор транзакції, компонента та мітка часу. Наступний критичний крок – це генерація шаблонів логів, який є формою абстракції даних. Тут варіативні текстові повідомлення зводяться до стандартизованих синтаксичних шаблонів, де змінювані параметри замінюються на маркери. Це дозволяє перейти від лінгвістичного аналізу до кількісного векторного представлення поведінки системи.

Етап формування базової лінії є визначенням еталонного коректного стану. Збирається референсна множина логів з періоду, коли система функціонувала без відомих дефектів. На основі цієї множини будується статистичний профіль нормальної поведінки. Цей профіль інкапсулює типовий розподіл частоти кожного шаблону логу, типову послідовність подій та нормативні часові параметри виконання. Кожен шаблон логу та його частота утворюють векторне представлення базової лінії, що є багатовимірною моделлю нормальної поведінки системи.

Центральний етап моделі – це диференційний аналіз. Логи, отримані у поточному вікні виконання, наприклад, після нового розгортання, проходять ту ж саму шаблонізацію для створення поточного профілю за аналогічним векторним представленням. Далі обчислюється диференціал – метрична відстань (косинусна відстань або відносна ентропія) між поточним вектором і вектором базової лінії.

Значна зміна у векторі логування (високе значення диференціалу) класифікується як аномалія. Аномалії можуть проявлятися у вигляді появи нових шаблонів, несподіваного зникнення типових подій або різкого статистичного зростання частоти критичних шаблонів логів.

Останній етап зосереджений на перетворенні виявлених аномалій на дієздатну інформацію для інженерів. Схожі аномальні події підлягають кластеризації для ідентифікації кореневої причини. Помилки проходять ранжування, де вищий пріоритет надається тим аномаліям, що асоційовані з шаблонами високої критичності або мають найбільше відхилення від нормативних показників. Використовуючи ідентифікатор транзакції та ідентифікатор компонента, система забезпечує трасування та локалізацію помилки, вказуючи на конкретний компонент або блок коду, де відбулося відхилення. Фінальним результатом є автоматичний звіт, що містить не лише факт аномалії, але й кількісний диференціал та контекстні шаблони логів.

Отже, даного роду інструмент порівнює очікувану поведінку системи та спостережувану. Результатом порівняння є вирівняні траєкторії, згруповані на основі їх відмінностей. Також є можливість візуалізувати траєкторії, що відхиляються, за допомогою структури даних, що робить результат зрозумілим для розробника. Додаткова інформація про відхилення траєкторій надається у візуалізації для подальшого дослідження різниці.

Для методів порівняння великих обсягів даних спочатку слід зібрати всі можливі методи для будь-якої кількості даних. Таким чином, у даному розділі показано аналіз методів для вирішення конкретної проблеми та визначено, що порівняння неефективне для великих обсягів даних. Дуже важливим результатом цієї роботи є модифікація алгоритмів вирівнювання послідовностей, які зазвичай використовуються в біоінформатиці.

3 АРХІТЕКТУРА СИСТЕМИ АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ПОМИЛОК У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ НА ОСНОВІ ДИФЕРЕНЦІАЛЬНОГО АНАЛІЗУ ЛОГІВ ВИКОНАННЯ

3.1 Формування та аналіз вимог програмної реалізації програмної системи

Формування та аналіз вимог до системи, що фокусується на диференціальному аналізі логів, є специфічним завданням, оскільки фокус зміщується з простого пошуку помилок на порівняння еталонної поведінки системи з відхиленнями. Вимога не використовувати списки змушує структурувати цей матеріал у вигляді зв'язного наративу, де кожен аспект логічно випливає з попереднього.

Нижче наведено структурований опис формування та аналізу вимог до такої системи (рисунок 3.1).

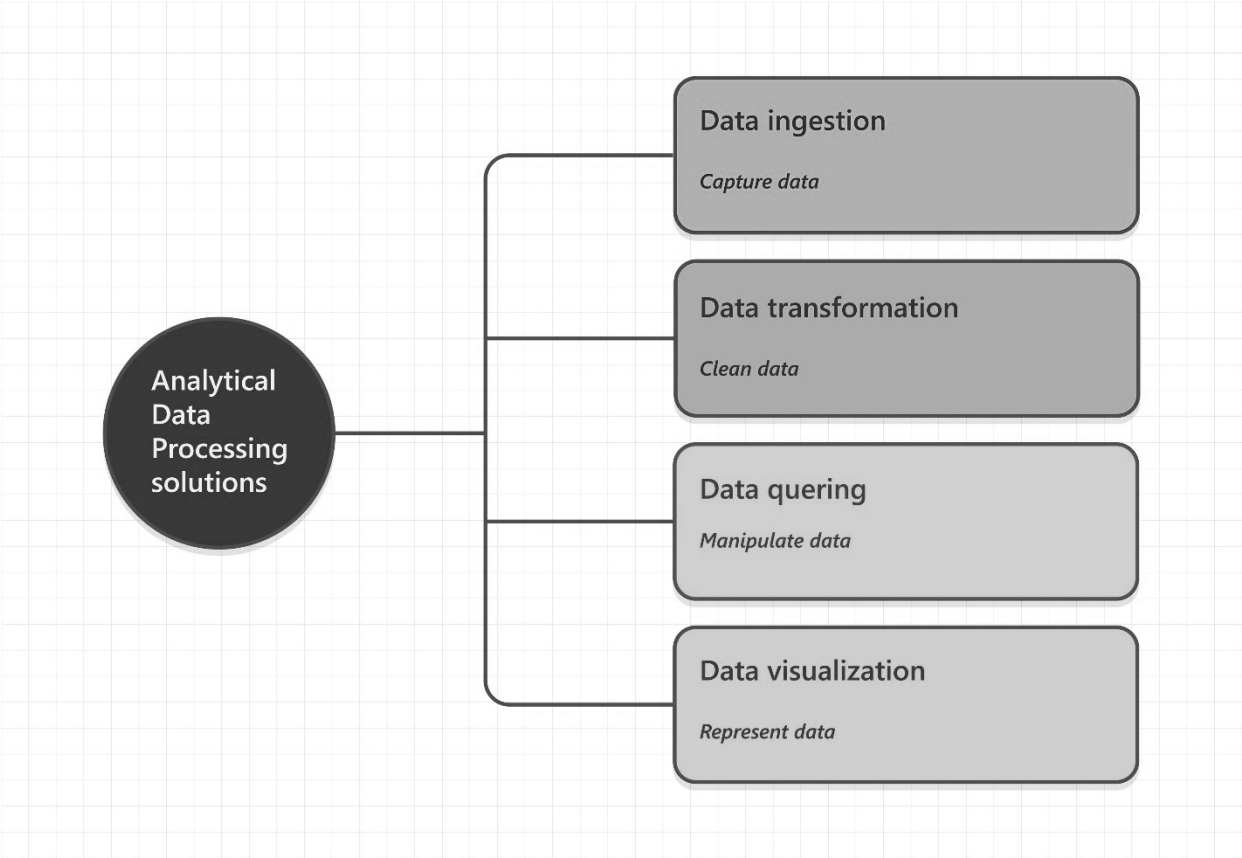


Рисунок 3.1 – Структурований опис формування та аналізу вимог до розроблюваної системи

Фундаментом системи є здатність коректно обробляти вхідні дані. Вимоги до підсистеми збору даних повинні передбачати підтримку різнорідних джерел. Система має вміти працювати як зі структурованими форматами, наприклад, JSON, XML, так і з неструктурованим текстом, оскільки сучасні додатки часто генерують змішані логи. Критично важливою вимогою є забезпечення потокової обробки даних у реальному часі, а також можливість пакетного завантаження історичних логів для навчання моделей або створення базових ліній.

Однією з найскладніших частин диференціального аналізу є відділення значущих змін від так званого шуму. Тому ключовою функціональною вимогою стає розробка механізму глибокої нормалізації логів. Система повинна автоматично виявляти та маскувати динамічні змінні, такі як часові мітки, унікальні ідентифікатори сесій, IP-адреси та випадкові числа, які змінюються при кожному запуску програми навіть за відсутності помилок. Без цього етапу диференціальний аналіз буде неможливим, оскільки кожен рядок логу буде вважатися зміненим. Вимоги мають чітко визначати правила токенізації, де змінні частини логу замінюються на узагальнені токени , перетворюючи унікальні записи на шаблони подій.

Нормалізація не повинна втрачати контекст помилки. Вимога має гарантувати, що при маскуванні змінних система зберігає структуру повідомлення про помилку, щоб алгоритм міг розрізнити критичний збій та звичайне інформаційне повідомлення.

Основою системи є логіка порівняння, яка вимагає реалізації алгоритмів знаходження різниці між еталонним набором логів, а саме успішний запуск та цільовим набором. Вимоги повинні описувати не лише просте текстуальне порівняння, але й статистичний аналіз частоти подій. Система має вміти виявляти три типи аномалій: поява нових типів повідомлень, зникнення очікуваних повідомлень та суттєва зміна частоти появи певних подій. Наприклад, якщо повідомлення про успішну транзакцію зазвичай з'являється 100 разів, а в новому запуску лише 5, система має кваліфікувати це як потенційну помилку, навіть якщо текст логу не змінився.

Вимоги до продуктивності та масштабованості є критичними через великий обсяг текстових даних. Архітектура повинна дозволяти горизонтальне масштабування для обробки гігабайтів логів за прийнятний час, що часто передбачає використання розподілених обчислень або оптимізованих сховищ часових рядів. Також важливо сформулювати вимоги до інтеграції з CI/CD пайплайнами (процесами безперервної інтеграції та розгортання). Система має функціонувати як автоматичний бар'єр якості (Quality Gate), який блокує реліз, якщо диференціальний аналіз виявляє критичні відхилення від норми.

Кінцевий користувач (розробник або QA-інженер) потребує зрозумілого представлення результатів. Вимоги до інтерфейсу повинні передбачати відображення логів у зручному форматі, де чітко підсвічуються нові помилки або аномальні блоки. Система повинна групувати схожі помилки, щоб користувач не переглядав тисячі ідентичних рядків, а бачив один згрупований інцидент із зазначенням кількості повторень. Це вимагає розробки механізму кластеризації результатів аналізу перед їх відображенням на дашборді.

Цей підхід дозволяє створити цілісну картину вимог без розбиття на атомарні пункти списків, зберігаючи при цьому логічну структуру та технічну глибину.

3.2 Розробка архітектури для системи автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Для того, щоб якісно проаналізувати розроблену методику, необхідно розробити програмний застосунок. Для цього потрібно описати архітектуру даного програмного рішення. Визначено, що це не буде один монолітний скрипт, а інструмент, що працює у трьох режимах:

- навчання (learn) з метою створення якогось еталону;
- аналітична складова (analyze);
- звіт (report).

Архітектура інструменту *Log-Narok*, передбачає, що передумовою його роботи є те, що інструмент очікує на вході структуровані логи (JSON/LTSV/тощо). Якщо на вході є велика кількість тексту, першим кроком завжди буде парсер (наприклад, Drain), щоб витягти шаблони. Для простоти, припустимо, у нас вже є шаблони. Тоді мають працювати наступні модулі, що описано нижче.

Модуль 1: Режим learn (Створення *Золотого еталону*)

Завданням є побудувати статистичну модель нормальної поведінки системи.

На початковому етапі є певний масив файлів логів (*log_file_1.json, log_file_2.json, ... log_file_N.json*) зі стабільного, програмного забезпечення, отриманих після певної кількості тестових прогонів.

Алгоритм побудований на тому, що спочатку відбувається ініціалізація, яка в свою чергу передбачає створення порожньої хеш-таблиці (словника) *ProfileStorage*.

ProfileStorage = Map<TemplateHash, ProfileData>

Далі необхідно здійснити парсинг та агрегацію із циклом по певній кількості файлів, тобто для кожного log_file з певної кількості вхідних файлів створити тимчасовий лічильник для цього файлу

FileCounter = Map<TemplateHash, Count>.

Далі відповідно здійснювати читання файлу рядок за рядком.

Для кожного рядка логу (log_entry) необхідно:

- визначити *TemplateHash* (хеш від MessageTemplate);
- визначити LogLevel (INFO, WARN, ERROR);
- інкрементувати *FileCounter[TemplateHash]*.

Після обробки файлу, для кожного *TemplateHash* у *FileCounter*:

- якщо *TemplateHash* ще не в *ProfileStorage*, створити для нього запис;
- додати Count з FileCounter до списку спостережень ProfileStorage[TemplateHash].Observations.Add(Count);
- зберегти LogLevel, можна припустити, що він не змінюється для одного шаблону).

Далі відбувається статистичний розрахунок, тобто фіналізація:

Проітерувати по кожному *TemplateHash* у *ProfileStorage*.

Для кожного запису *ProfileData* необхідно:

– розрахувати математичне сподівання μ на основі списку Observations;

– розрахувати стандартне відхилення σ на основі списку Observations;

– якщо $\sigma == 0$ і $\mu > 0$ (подія стабільна, але трапляється), встановити σ в якесь мале значення (наприклад, 0.1), щоб уникнути ділення на нуль, або позначити її як IsStable = true.

Вихід із збереженням, коли необхідно серіалізувати *ProfileStorage* (зі всіма μ та σ) у файл *golden_profile.dat.*

В результаті буде отримано так званий золотий еталон або статистичний портрет здорової системи.

Модуль 2 включає режим analyze або власне сам пошук відхилень. Завданням на такому етапі буде порівняння нового, підозрілого логу із золотим еталоном та знаходження аномалії.

Вхід: golden_profile.dat та один файл логу candidate_log.json з нового білду.

Алгоритм передбачає завантаження еталону, шляхом десеріалізації golden_profile.dat у пам'ять (ProfileStorage).

Ініціалізація передбачає створення порожнього списку *Anomalies = [].*

Парсинг та підрахунок кандидата передбачає такі дії:

– створити лічильник для кандидата: *CandidateCounter = Map<TemplateHash, {Count, LogLevel}>;*

– прочитати candidate_log.json рядок за рядком.

– для кожного log_entry:

– визначити TemplateHash та LogLevel.

– інкрементувати CandidateCounter[TemplateHash].Count та зберегти LogLevel.

Наступним кроком є диференціальний аналіз, що входить у головний цикл. На цьому кроці необхідно проітерувати по кожному *TemplateHash* у *CandidateCounter* (тобто по подіях, що реально трапились):

candidate_data = CandidateCounter[TemplateHash]

Перевірка *Привида* за типом 1):

– знаходження TemplateHash у ProfileStorage;

– у випадку відсутності, тоді - це нова подія;

– *Anomalies.Add({ Type: NewEvent, Hash: TemplateHash, Level: candidate_data.LogLevel })*

– якщо так, то подія відома, і можна переходити до аналізу.

– profile_data = ProfileStorage[TemplateHash]

Перевірка *Зміни рівня* (Тип 2):

– відповідь на запитання чи candidate_data.LogLevel != profile_data.LogLevel?

– якщо відповіль так, то подія змінила серйозність.

– *Anomalies.Add({ Type: LevelShift, Hash: TemplateHash, From: profile_data.LogLevel, To: candidate_data.LogLevel })*

Перевірка *Частоти* (Тип 3):

Використовуємо μ = profile_data.μ та σ = profile_data.σ.

– *current_count = candidate_data.Count.*

– якщо σ > 0 (або наш мінімальний поріг), то

z_score = (current_count - μ) / σ

якщо |*z_score*| > *3* (або інший поріг чутливості):

Anomalies.Add({ Type: "FrequencyAnomaly", Hash: TemplateHash, Count: current_count, Expected: μ, ZScore: z_score })

– якщо σ == 0 (подія була стабільною):

– якщо current_count != μ:

Anomalies.Add({ Type: "StableAnomaly", Hash: TemplateHash, Count: current_count, Expected: μ })

Перевірка *Зникнення* (Тип 4 - зворотний цикл):

– Проітерувати по кожному TemplateHash у ProfileStorage (тобто по подіях, які мали б трапитись):

– чи є TemplateHash у CandidateCounter?

– якщо ні, тоді подія зникла.

profile_data = ProfileStorage[TemplateHash]

– якщо *profile_data.μ > 0* (подія очікувалася):

– Anomalies.Add({ Type: "MissingEvent", Hash: TemplateHash, Expected: profile_data.μ })

Вихід (збереження) заключається у тому, щоб серіалізувати *Anomalies* у *analysis_report.json.*

Модуль 3. Режим report (Вердикт) під час якого виконується завдання, що полягає у перетворенні звіту про аномалії на чіткий вердикт для CI/CD.

Вхід analysis_report.json, алгоритм якого полягає у тому, що відбувається завантаження звіту, тобто читається analysis_report.json.

Розрахунок *Рахунку болю* (Score):

TotalScore = 0

Для кожної anomaly у звіті:

Якщо

anomaly.Type == NewEvent і anomaly.Level == "ERROR": TotalScore += 100

Якщо

anomaly.Type == MissingEvent і anomaly.Expected > 10: TotalScore += 50

Якщо

anomaly.Type == FrequencyAnomaly: TotalScore += |anomaly.ZScore| * 5

Винесення вердикту або попереднього рішення включає такі кроки:

Задати поріг (наприклад, *FailureThreshold = 50*).

Якщо *TotalScore > FailureThreshold*, то тоді необхідно вивести звіт аномалій у консоль (можна червоним кольором).

exit(1) (це зламає білд у CI/CD).

Якщо *TotalScore > 0 (але < Threshold),* то тоді необхідно вивести звіт аномалій (жовтим кольором, як попередження).

exit(0)

Якщо TotalScore == 0, то

Вивести повідомлення *Все чисто. Білд стабільний.*

exit(0)

Парсер шаблонів - це 90% успіху, оскільки у випадку його поганого групування схожих логів, вся статистика буде сміттям.

Налаштування чутливості. Z-score > 3 - це лише початок, оскільки ще буде необхідність аналізу порогів щоб відфільтрувати шум від реальних проблем.

На рисунку 3.1 показано архітектуру розроблюваної системи.

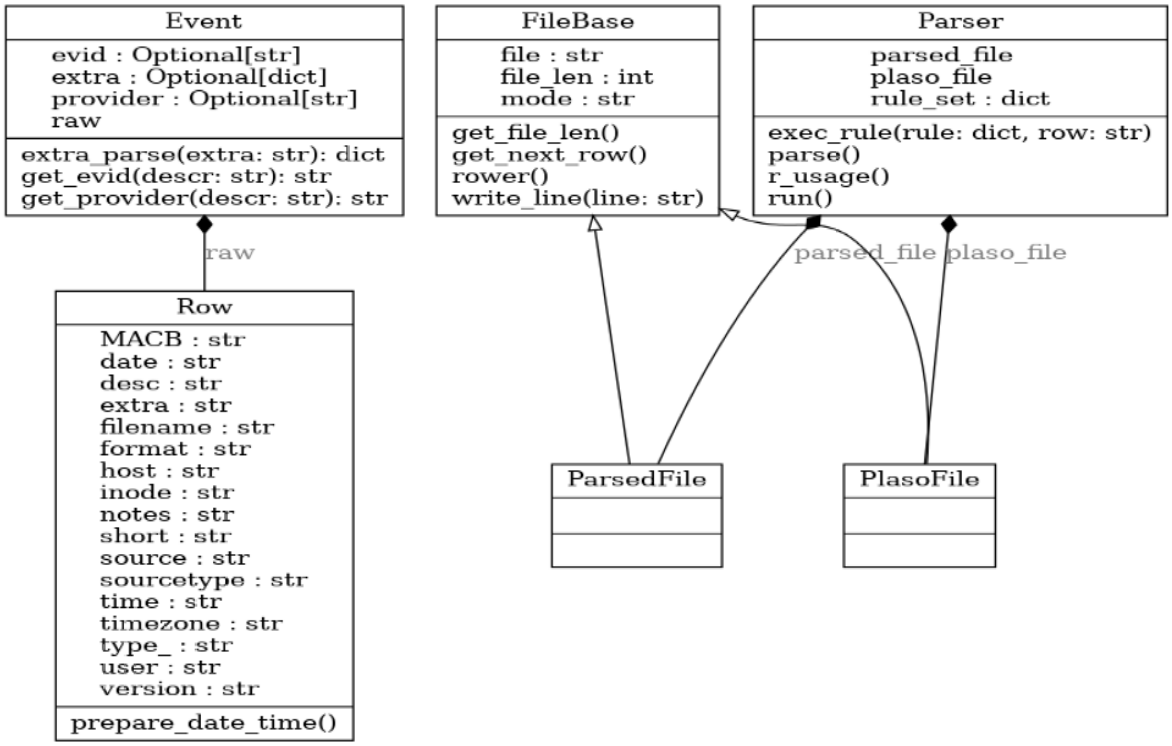


Рисунок 3.1 – Архітектура розроблюваної системи

Отже, визначена архітектура для програмної системи містить ряд модулів, а саме:

- навчання (learn), що має на меті створення якогось еталону;
- аналітичний модуль складова (analyze);
- звіт (report). Завданням цього модулю є перетворення інформації про аномалії на чіткий вердикт для CI/CD.

3.3. Висновки до 3-го розділу

У третьому розділі здійснено аналіз вимог до програмного забезпечення, а також розробка архітектури програмної реалізації.

Фундаментом системи є здатність коректно обробляти вхідні дані. Вимоги до підсистеми збору даних повинні передбачати підтримку різнорідних джерел. Система має вміти працювати як зі структурованими форматами, наприклад, JSON, XML, так і з неструктурованим текстом.

Однією з найскладніших частин диференціального аналізу є відділення значущих змін від так званого шуму. Тому ключовою функціональною вимогою стає розробка механізму глибокої нормалізації логів. Система повинна автоматично виявляти та маскувати динамічні змінні, такі як часові мітки, унікальні ідентифікатори сесій.

Основою системи є логіка порівняння, яка вимагає реалізації алгоритмів знаходження різниці між еталонним набором логів, а саме успішний запуск та цільовим набором. Вимоги повинні описувати не лише просте текстуальне порівняння, але й статистичний аналіз частоти подій. Система має вміти виявляти три типи аномалій.

4 ОЦІНКА СИСТЕМИ ВИЯВЛЕННЯ ЛОГІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

4.1 Методологія оцінювання системи на основі диференційного аналізу логів

Оцінка системи, що використовує метод диференційного аналізу логів виконання, є кількісною та якісною процедурою, спрямованою на підтвердження надійності, стабільності та відповідності поточної поведінки системи її еталонному стану. Оцінка відбувається шляхом обчислення відхилення від встановленої базової лінії.

Оцінка системи за даним методом інтегрована в процес моніторингу і включає такі ключові етапи:

- порівняння векторів логування;
- визначення критеріїв оцінки якості системи;
- верифікація та висновок оцінки.

Порівняння векторів логування є основним механізмом оцінки є обчислення метричної відстані (диференціалу) між вектором логування поточного стану та вектором логування базової лінії. Для кількісної оцінки відхилення часто використовується косинусна відстань або відносна зміна частоти шаблонів.

Якщо відстань дорівнює нулю або перебуває в межах визначеного довірчого інтервалу, система оцінюється як стабільна та коректна.

Якщо відстань перевищує порогове значення, то система оцінюється як аномальна та потенційно несправна.

Метрика Ступінь аномальності є прямим результатом диференційного аналізу і кількісно виражає рівень відхилення поточного профілю логів від базової лінії, високе її значення слугує індикатором деградації або помилки.

Також мають формуватись критерії оцінки якості системи, оскільки оцінка системи не обмежується лише виявленням аномалій, але й класифікує тип проблеми.

Коректність послідовності виконання полягає у тому, що система оцінюється на предмет несподіваної зміни порядку виклику компонентів або функцій. Якщо послідовність, що часто спостерігалася в базовій лінії, порушується, наприклад, зникає очікуваний крок автентифікації, це оцінюється як порушення цілісності процесу.

Наявність нових невідомих подій полягає у тому, що відбувається поява шаблонів логів, які не були присутні в базовій лінії, оцінюється як висококритична аномалія. Це часто свідчить про виникнення нової непередбаченої помилки (наприклад, винятку, який раніше не оброблявся) або про несанкціоновану зміну поведінки системи.

Зміна інтенсивності логування полягає у різкому зростанні частоти шаблонів з рівнем WARN або ERROR оцінюється як деградація продуктивності або збільшення кількості помилок, навіть якщо самі помилки були присутні в базовій лінії. І навпаки, несподіване зниження кількості логів може свідчити про зависання або зупинку компонента.

На етапі верифікації та формулювання висновку оцінки відбувається завершальна частина оцінки, що включає підтвердження результатів та генерацію висновку.

Під час трасування та локалізації встановлено, якщо оцінка виявляє аномалію, вона вважається підтвердженою лише після успішного трасування відхилення до конкретного ідентифікатора транзакції та компонента. Це перетворює абстрактну аномалію на конкретний дефект.

На заключному етапі відбувається формування висновку, який визначає, що на основі ранжування помилок та їхньої критичності, наприклад, виявлено 5 нових шаблонів помилок з високою частотою, то система робить висновок про загальний стан стабільності. Отже, висновок може бути таким, що система не відповідає базовій лінії, оскільки виявлено 78% відхилення у векторі логування через появу нових винятків у компоненті X.

Цей метод оцінки надає об'єктивний, кількісний підхід до діагностики, що зменшує залежність від ручного перегляду логів та підвищує операційну ефективність виявлення дефектів.

4.2 Метрика для оцінки системи автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Якщо система, заснована на диференційному аналізі логів, уже розгорнута й працює, аналітичне завдання полягає у витягуванні кількісних показників оцінки та локалізації дефектів.

На рисунку 4.1 подано концептуальну структуру та зміст вихідного файлу .csv, який може генерувати ця система для аналітика. Структура та зміст файлу CSV для оцінки системи в даному випадку слугує звітом про аномалії та дефекти, ідентифіковані під час диференційного аналізу, надаючи ключові метрики для подальшого Root Cause Analysis (RCA). У таблиці 4.1 показано орієнтовні заголовки стовпців, а також їхнє призначення.

Таблиця 4.1 - Заголовки стовпців та їхнє призначення

Стовпець	**Тип даних**	**Призначення**
Anomaly_ID	INTEGER	Унікальний ідентифікатор виявленої аномалії/дефекту.
Timestamp_Start	DATETIME	Початок часового вікна, в якому було зафіксовано аномалію.
Component	STRING	Компонент або мікросервіс, де було виявлено відхилення (наприклад, AuthService, PaymentGateway).

Продовження таблиці 4.1

Стовпець	Тип даних	Призначення
Log_Template_ID	STRING	Унікальний ID шаблону логу, який спричинив аномалію.
Anomaly_Type	STRING	Тип аномалії: New Pattern, Frequency Increase, Sequence Violation, Pattern Disappearance.
D_Score	FLOAT	Обчислена косинусна відстань або відносний диференціал від базової лінії. Чим вище, тим критичніше.
Baseline_Freq	FLOAT	Середня частота шаблону в референсній (нормальній) базовій лінії.
Current_Freq	FLOAT	Частота шаблону в поточному (аномальному) часовому вікні.
Frequency_Change	FLOAT	Відсоткова зміна частоти
Severity_Rank	INTEGER	Ранжування критичності для RCA (від 1 – найбільш критичний, до N).
Sample_Log_Message	STRING	Зразок сирого повідомлення логу, пов'язаного з аномалією, для контексту.

Для демонстрації того, як може виглядати вихідний файл, наведемо приклади записів:

Приклад вмісту CSV-файлу подано нижче.

Anomaly_ID,Timestamp_Start,Component,Log_Template_ID,Anomaly_Type,D_Score, Baseline_Freq,Current_Freq,Frequency_Change,Severity_Rank,Sample_Log_Message

101,2025-11-30 18:00:00,PaymentGateway,TPL_007,"Frequency Increase",0.89,5.2,45.8,780.77,1,"ERROR: Payment processor timeout on request ID: 87693."

102,2025-11-30 18:05:30,UserService,TPL_NEW_01,"New Pattern",0.95,0.0,12.5,Inf,2,"FATAL: Unhandled exception in DbContext initialization: Connection refused."
103,2025-11-30 18:15:00,AuthService,TPL_012,"Pattern Disappearance",0.65,150.0,0.0,-100.0,5,"User authentication success log missing after 18:15."
104,2025-11-30 18:20:15,ReportingService,TPL_055,"Sequence Violation",0.42,12.0,25.0,108.33,3,"WARN: Report generation started before data cache refresh complete."

Даний файл буде використовуватись із аналітичною метою, тобто буде відбуватись аналітична інтерпретація результатів і вже потім будуть прийматись певні рішення в залежності від складності виявлених аномалій.

Сортування полягає у тому, що першочергова увага приділяється записам з найвищими значеннями D_Score (наприклад, ID 102: 0.95) та Severity_Rank (ID 101: 1), оскільки вони вказують на найбільше відхилення або найбільш критичну проблему.

Виявлення кореневої причини (RCA), тобто аналіз *Anomaly_Type* та *Log_Template_ID* допомагає визначити джерело проблеми. Наприклад, ID 102 є *New Pattern* або новий шаблон із критичним повідомленням *Connection refused* у UserService, що прямо вказує на проблему з підключенням до бази даних або неправильне розгортання конфігурації.

Оцінка стабільності відбувається тоді, коли частка записів відносно загальної кількості проаналізованих вікон, є кількісною метрикою нестабільності системи після останньої зміни.

Ця структура забезпечує трасованість від абстрактного відхилення до конкретного рядка логу та компонента, що є кінцевою метою автоматизованого диференційного аналізу.

На рисунку 4.1 продемонстровано результат впровадженого методу.

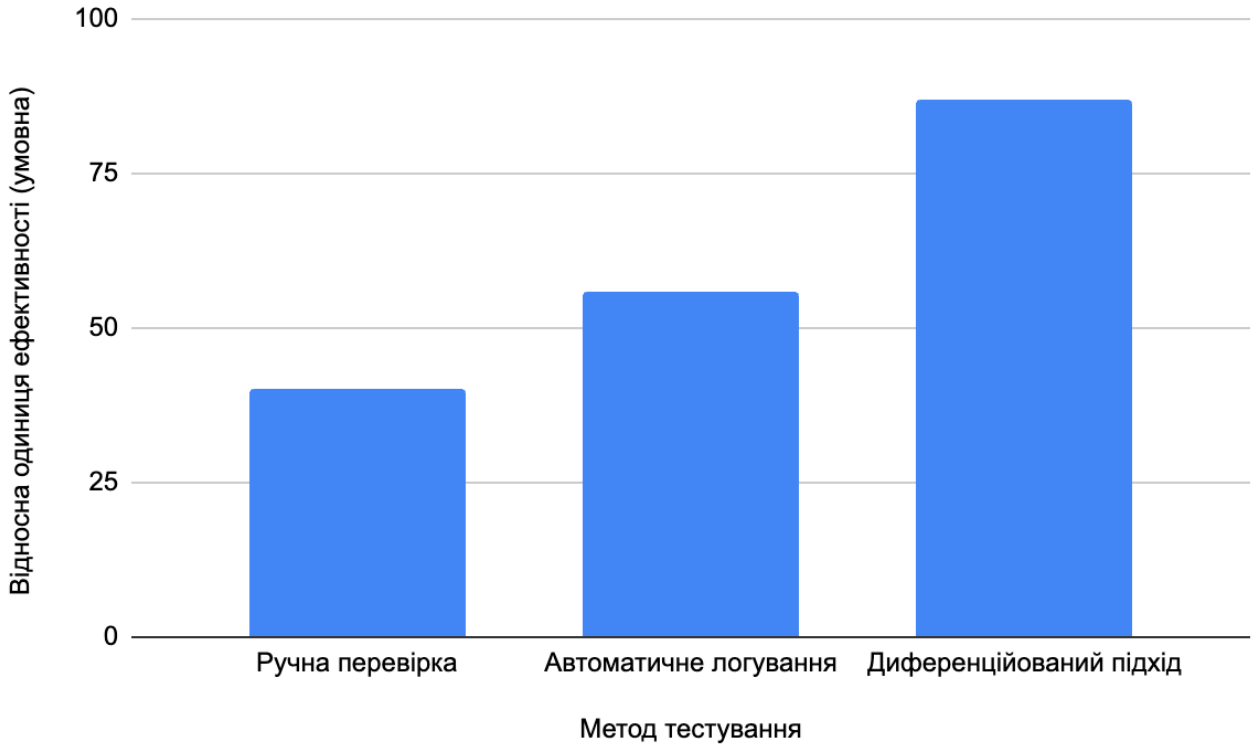


Рисунок 4.3 – Результат впровадження методу

Отже, як видно із описаних вище даних при використанні удосконаленого методу для логування, буде отримано такі результати, що майже у два рази перевищують перегляд та визначення логів при ручному використанні.

4.3 Висновки до 4-го розділу

Отже, у четвертому розділі виявлено та встановлено, що оцінка системи, що використовує метод диференційного аналізу логів виконання, є кількісною та якісною процедурою, спрямованою на підтвердження надійності, стабільності та відповідності поточної поведінки системи її еталонному стану. Оцінка відбувається шляхом обчислення відхилення від встановленої базової лінії.

Оцінка системи за даним методом інтегрована в процес моніторингу і включає такі ключові етапи:

– порівняння векторів логування;

– визначення критеріїв оцінки якості системи;

– верифікація та висновок оцінки.

Якщо система, заснована на диференційному аналізі логів, уже розгорнута й працює, аналітичне завдання полягає у витягуванні кількісних показників оцінки та локалізації дефектів.

Також у даному розділі показано подано концептуальну структуру та зміст вихідного файлу .csv, який може генерувати ця система для аналітика.

ВИСНОВКИ

Отже, в результаті проведеного дослідження можна зробити наступні висновки.

У першому здійснено введення поняття логування та його значення під час реалізації та користування програмним забезпеченням.

Виявлено, що логування - це процес запису та зберігання інформації про події, що відбуваються під час роботи комп'ютерної системи, програмного забезпечення або мережі. По суті, логування створює хронологічний запис (журнал, лог-файл) усіх дій, операцій, помилок і станів системи. Цей запис використовується для діагностики, моніторингу, аудиту та забезпечення безпеки.

Логування є життєво необхідним інструментом для:

– діагностики та налагодження, що полягає у виявленні та виправленні помилок, збоїв або несподіваної поведінки програмного забезпечення;

– моніторингу продуктивності, а саме відстеження часу виконання функцій, використання ресурсів (пам'ять, процесор) так званих вузьких місць у системі;

– аудиту безпеки, що забезпечує відстеження доступу користувачів, спроб авторизації, критичних змін у конфігурації чи системі (наприклад, у журналі безпеки Windows).

– аналізу поведінки користувачів, наприклад, у вебзастосунках відстежуються шляхи користувачів, популярні функції та конверсії.

У другому розділі було здійснено вдосконаленння методу автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання.

Сформовано концептуальну модель автоматизованого виявлення помилок у програмному забезпеченні, що ґрунтується на диференційному аналізі логів виконання, є методом проактивної діагностики та аналізу аномалій у складних розподілених системах. Суть методу полягає у метричному порівнянні поточного

вектора поведінки системи, представленого у логах із референтною моделлю або базовою лінією нормальної роботи.

У третьому розділі здійснено аналіз вимог до програмного забезпечення, а також розробка архітектури програмної реалізації.

Фундаментом системи є здатність коректно обробляти вхідні дані. Вимоги до підсистеми збору даних повинні передбачати підтримку різнорідних джерел. Система має вміти працювати як зі структурованими форматами, наприклад, JSON, XML, так і з неструктурованим текстом.

Однією з найскладніших частин диференціального аналізу є відділення значущих змін від так званого шуму. Тому ключовою функціональною вимогою стає розробка механізму глибокої нормалізації логів. Система повинна автоматично виявляти та маскувати динамічні змінні, такі як часові мітки, унікальні ідентифікатори сесій.

Основою системи є логіка порівняння, яка вимагає реалізації алгоритмів знаходження різниці між еталонним набором логів, а саме успішний запуск та цільовим набором. Вимоги повинні описувати не лише просте текстуальне порівняння, але й статистичний аналіз частоти подій. Система має вміти виявляти три типи аномалій.

У четвертому розділі виявлено та встановлено, що оцінка системи, що використовує метод диференційного аналізу логів виконання, є кількісною та якісною процедурою, спрямованою на підтвердження надійності, стабільності та відповідності поточної поведінки системи її еталонному стану. Оцінка відбувається шляхом обчислення відхилення від встановленої базової лінії.

В рамках кваліфікаційної роботи було здійснено розробку інструменту, який зможе розрізняти журнали файли з метою виявлення та візуалізації потенційних аномалій у роботі програмного забезпечення. Основна складність полягає в пошуку правильних методів для порівняння даних журналів, оскільки вимоги системи передбачають обробку великих обсягів даних.

Порівняння даних журналів, загалом, не є сферою, яка ретельно досліджується, проте результати доводять, що поведінка програмного

забезпечення багато разів відрізняється від очікуваної. У найпростішому випадку, порівняння поведінки тестового середовища з реальними транзакціями може продемонструвати, що існує набагато більше відмінностей, ніж очікувалось більшістю. Покриття тестами можна покращити, порівнюючи тестове середовище з реальними транзакціями та навчаючись на їхніх відмінностях (виявлено невиявлені тестові випадки). Крім того, процедуру моніторингу релізів можна покращити, порівнюючи журнали стабільної версії програмного забезпечення з новим представленим релізом.

Метод автоматизованого виявлення помилок на основі диференційного аналізу логів виконання є актуальним та корисним, особливо в сучасних складних, розподілених та мікросервісних архітектурах, оскільки він забезпечує проактивне виявлення дефектів та підвищення операційної ефективності.

Метод підходить для інтеграції в конвеєри безперервної інтеграції та розгортання. Система може автоматично порівнювати логи виконання нової версії програмного забезпечення із базовою лінією попередньої стабільної версії. Якщо швидкість нового розгортання перевищує порогове значення, конвеєр може бути автоматично відкочений. Це забезпечує надійний запобіжний клапан у DevOps-процесах. Крім того, диференційний аналіз ефективно виявляє непередбачені побічні ефекти та каскадні збої, які виникають у непов'язаних компонентах системи після внесення змін.

Аналіз логів не обмежується лише помилками, він також надає інформацію про ефективність. Раптове зростання швидкості, спричинене появою великої кількості інформаційних або налагоджувальних логів, може вказувати на недоцільне використання ресурсів через надмірне логування. Це дозволяє команді оптимізувати обсяг логів, зменшуючи витрати на зберігання та обробку даних. Також зміна послідовності або частоти певних шаблонів може свідчити про повільну деградацію продуктивності або неефективне використання ресурсів, ще до того, як це відобразиться на метриках затримки.

В кінцевому підсумку, практичне значення методу полягає у переході від реактивного усунення наслідків до проактивного запобігання інцидентам

управління стабільністю програмного забезпечення, що є критичним для підтримання високої якості обслуговування.

Відповідно до теми кваліфікаційної роботи опубліковані тези «Аналіз методів та засобів виявлення логів у програмному забезпеченні» на конференції «Актуальні проблеми комп’ютерних наук (АПКН-2025)».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is a Web Designer? URL: https://brainstation.io/career-guides/what-is-a-web-designer (дата звернення 25.09.2025).

2. Evers P. A large-scale evaluation of tracing back log data to its origin with static analysis. Master's thesis. Delft University of Technology. 2018.

3. Amar H., Bao L., Busany N., Lo D. and Maoz S. Using finite-state models for log differencing. *In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.. ACM. 2018. P. 49-59.

4. Saif Gunja. What is DevOps? Unpacking the purpose and importance of an IT cultural revolution URL: https://www.dynatrace.com/news/blog/what-is-devops/ (дата звернення 26.09.2025).

5. Log analytics, log mining, anomaly detection. URL: https://www.xenonstack.com/blog/data-science/log-analytics-log-mining-anomalydetection (last accessed: 01.10.2025).

6. What is DevOps? The History, Definition, Best Practices, Use Cases + Examples of DevOps. URL: https://www.puppet.com/blog/what-is-devops (last accessed: 01.10.2025).

7. Хто такі DevOps-інженери та чим вони займаються. URL: https://robotdreams.cc/uk/blog/ (дата звернення: 01.10.2025).

8. Лог-файли як невід'ємна частина процесу розробки. URL: https://foxminded.ua/shcho-take-loh-faily/ (дата звернення: 01.10.2025).

9. Package java.util.logging. Java™ Platform, Standard Edition 8 API Specification. URL: https://docs.oracle.com/javase/7/docs/api/java/util/logging/packagesummary.html (last accessed: 02.10.2025).

10. Smart T. Logging and Testing. In: Serverless Beyond the Buzzword. *Logging and Testing. In: Serverless Beyond the Buzzword.* Berkeley : Apress, 2023. URL: https://doi.org/10.1007/978-1-4842-8761-3_8.

11. Doan Q. T. A Framework for Integrating IoT Streaming Data from Multiple Sources (PhD thesis). La Trobe University. 2021. URL: https://opal.latrobe.edu.au/articles/thesis/A_Framework_for_Integrating_IoT_Streaming_Data_from_Multiple_Sources/17211713 (last access: 07.10.2025).

12. Рівні логування. URL: https://qamania.org/blog/rivni-loguvannya/ (дата звернення: 10.10.2025).

13. Завдання ELK Stack: логування та аналіз даних в DevOps-середовищах. URL: https://itedu.center/ua/blog/review/elk-stack-tasks-logging-data-analysis/?srsltid=AfmBOoqVC3aZvv0Pdw8GIWVwsyKQoPfh6NFXDgs7iajVZK8ikbeBonEp (last access: 07.10.2025).

14. Logging Wisdom: How to Log. URL: https://medium.com/unomaly/logging-wisdom-how-to-log-5a19145e35ec (last access: 07.11.2025).

15. Configuring Logging. https://docs.nginx.com/nginx/admin-guide/monitoring/logging/ (last access: 07.11.2025).

16. Лог-файли як невід'ємна частина процесу розробки. URL: https://foxminded.ua/shcho-take-loh-faily/ (дата звернення 11.11.2025).

17. Дзен логування. Як полюбити свої логи та почати жити. URL: https://dou.ua/lenta/columns/about-logging/ (дата звернення 11.11.2025).

18. Grafana Loki для логування: З чого почати? URL: https://robotdreams.cc/uk/blog/749-grafana-loki-lehkyy-start-lohuvannya-ta-monitorynhu-za-10-khvylyn (дата звернення 11.11.2025).

19. Завдання ELK Stack: логування та аналіз даних в DevOps-середовищах. URL:https://itedu.center/ua/blog/review/elk-stack-tasks-logging-data-analysis/?srsltid=AfmBOorh8qNsQPtyLkxaalmJPPwuk3gxnHbDkN2ZWfH8lvIcduBw3s-X (дата звернення 13.11.2025).

20. Pythonic посібник з логування. URL: https://devzone.org.ua/post/pythonic-posibnyk-z-lohuvannia (дата звернення 13.11.2025).

21. Що таке логування? URL: https://flip.ranok.cx.ua/ukraincyam/shho-take-loguvannya.html (дата звернення 13.11.2025).

22. Cybersecurity in working from home: An exploratory study. / M. Bispham et al. *N TPRC49: The 49th research conference on communication, information and internet policy*, Oxford: University of Oxford. 2021. P. 1–43. URL: https://doi.org/10.2139/ssrn.3897380.

23. Miner: A modular log data analysis pipeline for anomaly-based intrusion detection. / M. Landauer et al. *Digital Threats: Research and Practice*. 2023. Vol. 4, iss. 1. P. 1–16. URL: https://doi.org/10.1145/3567675.

24. Log in to HubSpot. URL: https://knowledge.hubspot.com/account-management/log-in-to-hubspot (дата звернення 17.11.2025).

ДОДАТОК А

(обов’язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

УДК 004.8

Червончук І.С.

Хмельницький національний університет

АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ВИЯВЛЕННЯ ЛОГІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

Розглянуто поняття журналів логів програмного забезпечення, їх важливість та актуальність. Проаналізовано існуючі задачі та проблеми, що стосуються логування, а також подано методи та засоби виявлення логів у програмному забезпеченні.

The concept of software log files and their importance and relevance are considered. Existing tasks and problems related to logging are analyzed, and methods and means of detecting logs in software are presented.

Величезні обсяги даних журналів логів щодня генеруються програмним забезпеченням. Разом з тим, ці дані містять цінну інформацію про поведінку та стан системи, яка рідко використовується через свій обсяг та неструктурованість. Ручне переглядання файлів журналів є трудомісткою та тривалою процедурою для розробників [3, 5].

Проте, інформація журналів може допомогти у виявленні проблемного виконання програмного забезпечення, навіть якщо кінцевий результат здається цілком нормальним. В сучасних умовах автоматичний аналіз файлів журналів має вирішальне значення для виявлення проблем, але головним чином для розуміння того, як поводиться програмне забезпечення, що було б корисним для запобігання збоям та вдосконалення самого програмного забезпечення. Тому досить актуальним є задача виявлення неочікуваних виконань програмного забезпечення та визначення корінної причини їх виникнення. Більш детально, очікувану поведінку програмного забезпечення можна апроксимувати за допомогою методів модельного висновку, а нові спостережувані дані можна проаналізувати, щоб перевірити, чи відповідають вони очікуваній поведінці.

Разом з тим існують такі проблемні питання, що доцільно розглядати в якості дослідницьких напрямків. До таких питань відносяться:

- об'єм даних;
- контекст;
- налаштування політики.

Нові способи логування включають перехід від текстів до структури, а саме централізоване логування (Centralized Logging) та структуроване логування (Structured Logging).

Концепція централізованого логування полягає у тому, що замість того, щоб зберігати лог-файли локально на кожному сервері чи в контейнері, усі логи збираються та агрегуються в єдиному місці. Перевагами є спрощення пошуку, а також кореляція події, що відбуваються в різних мікросервісах, і значно покращує моніторинг розподілених систем [2].

Концепція структурованого логування полягає у тому, що логи записуються у форматі, який легко читається машиною, найчастіше JSON або XML. Замість неструктурованого рядка, кожен запис є об'єктом із чітко визначеними полями. Перевагами є забезпечення високої швидкості обробки та аналізу, а також можливість створювати складні фільтри та запити, а також легко інтегрується із системами моніторингу та аналітики. Це є стандартом для сучасних хмарних і мікросервісних архітектур.

Трасування відноситься до розширеного методу логування, що сфокусований на відстеженні повного шляху виконання запиту через усі мікросервіси у розподіленій системі. Перевагами є можливість виявити «вузькі місця» та затримки у складному ланцюжку викликів, що є неможливим при звичайному логуванні. Стандарти, такі як OpenTelemetry, стають ключовими для реалізації трасування.

Найпопулярнішим набором інструментів для централізованого, структурованого логування є стек ELK або Elastic Stack.

Аналоги та хмарні рішення:

– Splunk - це комерційний лідер у сфері SIEM та логування, відомий потужними можливостями аналізу та кореляції даних.

– Loki (Grafana Labs) - альтернатива, орієнтована на ефективне зберігання та індексацію логів, де індексація відбувається лише за метаданими, а не за повним текстом, що робить її економічною.

– хмарні сервіси, наприклад, AWS CloudWatch, Azure Monitor, Google Cloud Logging, що надають повністю керовані сервіси для збору, аналізу та зберігання логів.

Сучасні технології виходять за рамки простого пошуку та візуалізації. В сучасних реаліях широко використовуються інтелектуальний аналіз логів, що включає:

– аномальне виявлення, що полягає у використанні машинного навчання для аналізу шаблонів логування та автоматичного виявлення нетипової поведінки, яка може свідчити про збої, атаки або несподівані зміни;

– автоматичне усунення неполадок, що полягає в інтеграції систем логування з інструментами автоматизації, наприклад, системами оповіщення та автоматичним перезапуском сервісів, для мінімізації часу простою;

– Log-as-Code полягає в управлінні конфігурацією логування, фільтрами та дашбордами за допомогою коду, наприклад, Terraform, що забезпечує узгодженість та можливість відстеження змін.

Впровадження цих засобів і способів дозволяє IT-фірмам не лише діагностувати проблеми, а й активно їх запобігати, перетворюючи логи з архіву даних на оперативний інтелект.

Перелік посилань

1. Peter Evers. A large-scale evaluation of tracing back log data to its origin with static analysis. Master's thesis, Delft University of Technology, 2018.
2. Hen Amar, Lingfeng Bao, Nimrod Busany, David Lo, and Shahar Maoz. Using finite-state models for log differencing. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 49-59. ACM, 2018.
3. Log analytics, log mining, anomaly detection. URL: https://www.xenonstack.com/blog/data-science/log-analytics-log-mining-anomalydetection (last accessed: 11.10.2025).
4. Лог-файли як невід'ємна частина процесу розробки. URL: https://foxminded.ua/shcho-take-loh-faily/ (дата звернення: 21.10.2025).

.

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Кафедра інженерії програмного забезпечення

Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Виконав:

студент групи ІПЗм-24-1 Червончук І. С.

Керівник роботи:

канд.пед.наук., доцент Онишко О. Г.

Актуальність

Генерування великих обсягів даних журналів логів;

Ручне переглядання файлів журналів є трудомісткою та тривалою процедурою для розробників.

Автоматичний аналіз файлів журналів має вирішальне значення для виявлення проблем і розуміння того, як поводиться програмне забезпечення.

Мета: удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Завдання дослідження:

1.Провести аналітичне дослідження предметної області досліджуваної проблеми.
2.Здійснити аналітичний огляд методів логування.
3.Здійснити удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Об’єкт дослідження: процес автоматизованого виявлення помилок у програмному забезпечення із використанням автоматизації на основі диференціального аналізу логів виконання.

Предмет дослідження: сучасні методи логування, що використовуються для виявлення помилок.

Використані методи під час проведення дослідження:

Аналізу, синтезу;

Диференціального та статистичного аналізу;

Абстрагування;

Моделювання систем;

Узагальнення даних.

Наукова новизна

1.Удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

2.Розроблено алгоритм роботи методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Практичне значення

У цій кваліфікаційній роботі показано удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

Результати цієї роботи можуть в подальшому бути використаними програмістами для автоматизації роботи із журналами логів, їх аналізу та перегляду.

Розділ 1. Аналіз відомих методів

централізоване логування;

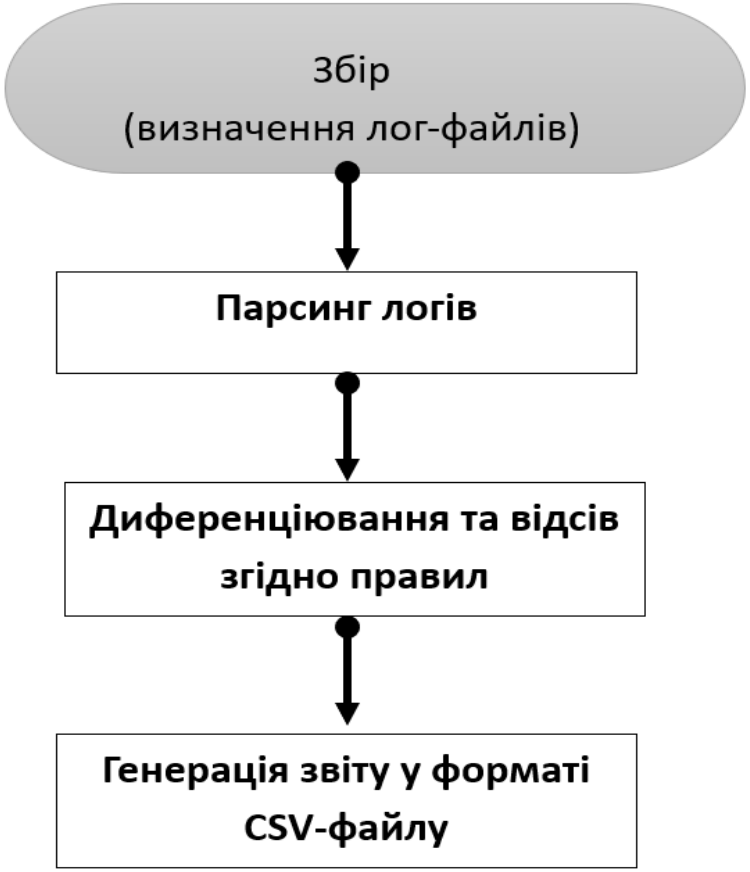
структуроване логування;

трасування.

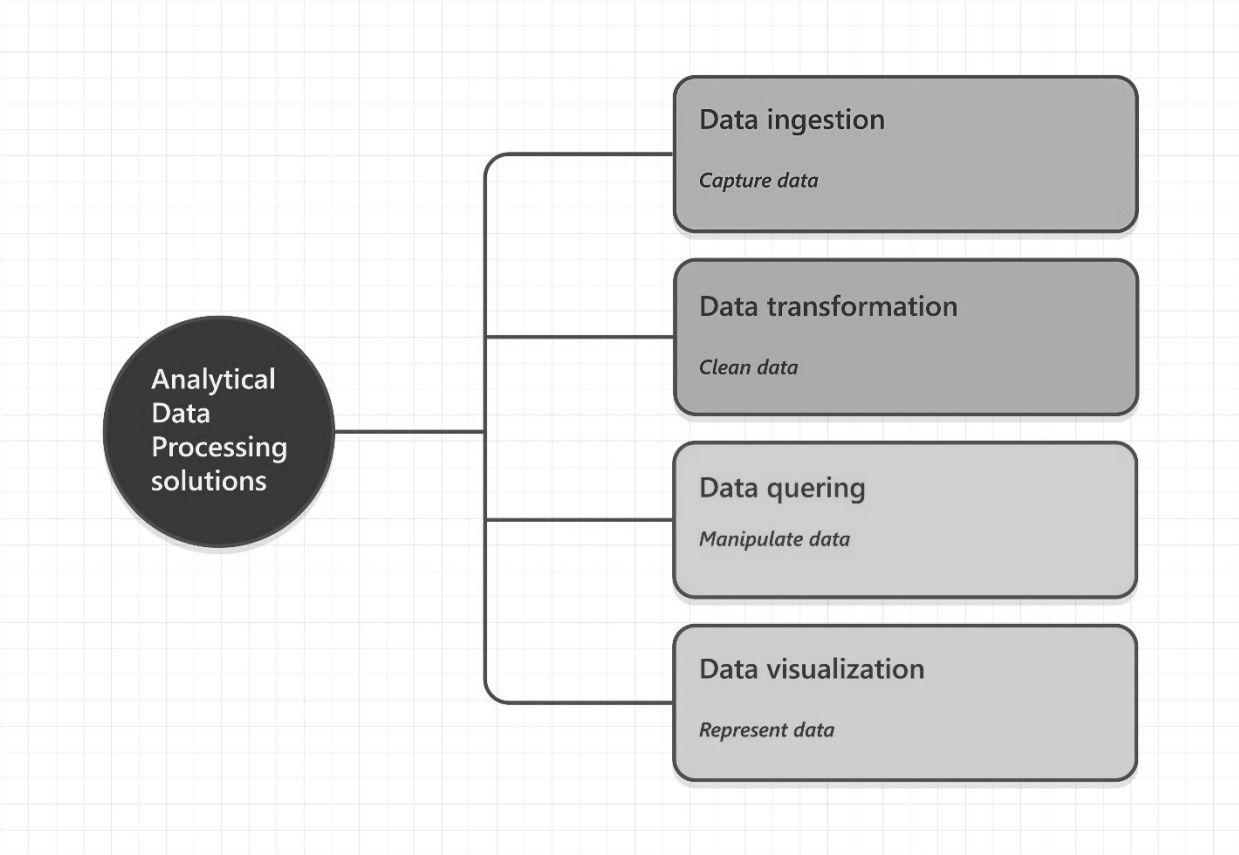
Розділ 2. Метод автоматизації знаходження помилок у програмному забезпеченні на основі аналізу логів

Метод автоматизації знаходження помилок на основі аналізу логів, удосконалено тим, що здійснюється перевірка відповідності та використовується повторення. Вхідні траси будуть відтворені на графіку, і в момент їх неперевірки діятиме алгоритм вирівнювання. Тобто два методи шукають найкраще вирівнювання на певному радіусі навколо проблемного вузла. Даний метод базується на відомому алгоритмі Нідлмана-Вунша для послідовностей.

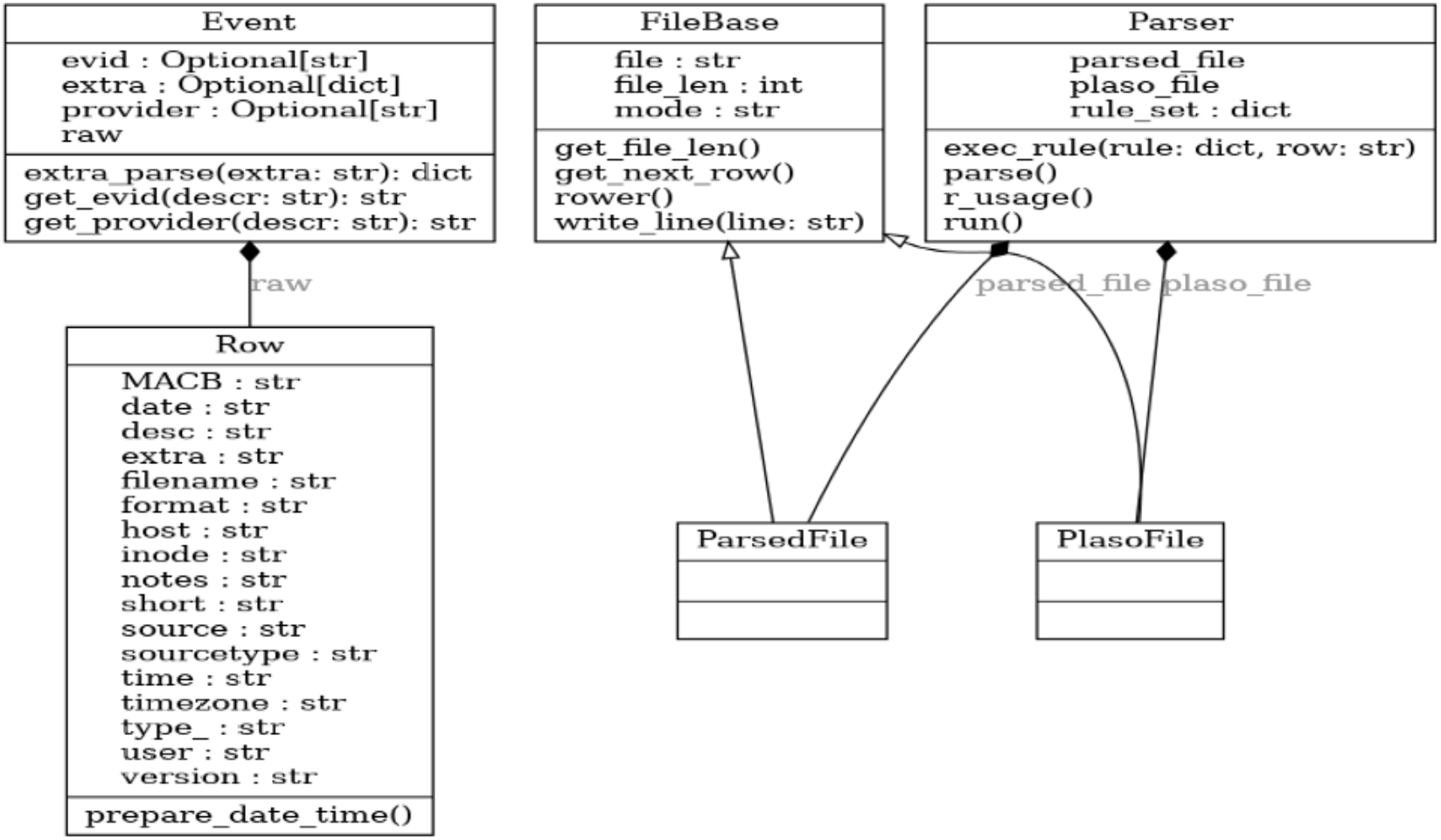
Розділ 2. Алгоритм автоматизації знаходження помилок у програмному забезпеченні на основі аналізу логів



Розділ 3. Аналіз вимог до програмного застосунку



Розділ 3. Структура програмного застосунку



Розділ 4. Результати

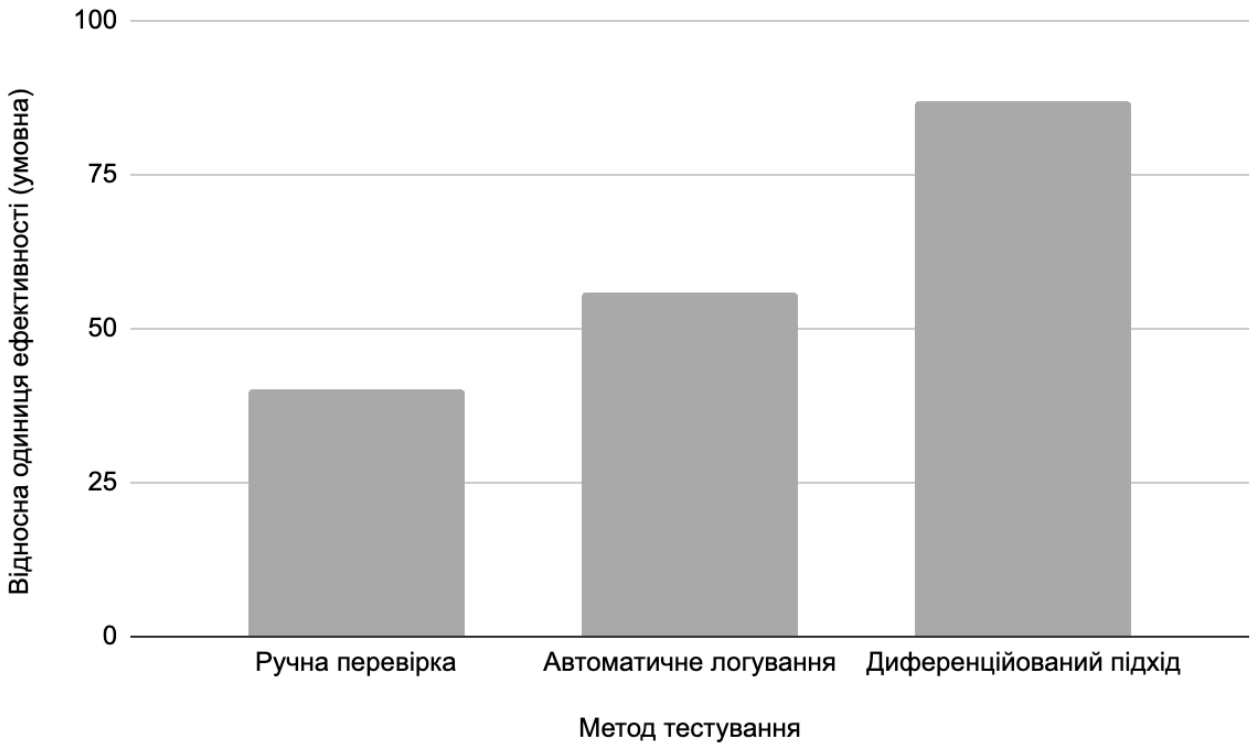
```
"rules": [
    {
        "regexp": r"(.*)",
        "rule_name": "user login failure",
        "level": "low",
        "output": {
            "regexp": r"(?<=Strings: \[\')(.*?)(\'\] Computer Name:)",
            "replace": {
                "' '": " | ",
                '0xc0000064': 'user name does not exist',
                '0xc000006a': 'user name is correct but the password is wrong',
                '0xc0000234': 'user is currently locked out',
                '0xc0000072': 'account is currently disabled',
                '0xc000006f': 'user tried to logon outside his day of week or time of day restrictions',
                '0xc0000070': 'workstation restriction', '0xc00000193': 'account expiration',
                '0xc0000071': 'expired password',
                '0xc0000133': 'clocks between dc and other computer too far out of sync',
                '0xc0000224': 'user is required to change password at next logon',
                '0xc0000225': 'evidently a bug in windows and not a risk',
                '0xc000015b': 'the user has not been granted the requested logon type (aka logon right) at this machine'

            }
        }
    }
```

Розділ 4. Результати

Після проведення збору даних, їх фільтрації та обробки програмним продуктом результати подаються у форматі csv-файла із обробленими лог-записами.

Розділ 4. Результати



Публікації за темою роботи

За темою дипломної роботи опубліковано тези доповіді конференції АПКН-2025 Хмельницького національного університету «Аналіз методів та засобів виявлення логів у програмному забезпеченні».

Висновки:

- **Мети роботи досягнуто:**

здійснено удосконалення методу автоматизації знаходження помилок у програмному забезпеченні із використанням диференціального аналізу логів виконання.

- Результати показують, що даний метод є ефективнішим, порівняно із ручним та автоматичним логуванням

Дякую за увагу!

СУПРОВІДНІ ДОКУМЕНТИ

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Ілля ЧЕРВОНЧУК

Співавтор:

Назва: Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Експерт: канд. пед. наук, доцент Оксана ОНИШКО

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:2.7%

Коефіцієнт подібності 2:0.5%

Мікропробіли: 34

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-12-14 19:16:28.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 15.12.2025 експерт (Онишко О.В.)

Sun Dec 14 18:20:23 EET 2025, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (UA) v-16.693

The maximum coincidence with one document 13.0%

Dictionaries check: UA, US, RU. **Errors in the documents: 30%**

	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
ID: 252778 Title: МКР_Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання Added in a DB: 2025-12-14 Authors: Ілля ЧЕРВОНЧУК Heads: канд. пед. наук, доцент Оксана ОНИШКО Consultants: Opponents:	96673	724	14573 (15%)	107 (15%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes
249053	Title: Переддипломна практика Added in a DB: 2025-10-14 Authors: Ілля ЧЕРВОНЧУК Heads: Оксана ОНИШКО, канд. пед.наук, доцент Consultants: Opponents:	12103 (13.0%)	98 (14.0%)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання»
Автор: Червончук Ілля Сергійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Онишко Оксана Григорівна, кандидат технічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Перевищено граничне значення рівня подібностей. Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення виявлені у роботі, є законними і не є плагіатом, оскільки:

1) У тексті кваліфікаційної роботи системою перевірки на плагіат StrickePlagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових сполучень у стандартних бланках (титулка, бланк завдань), у структурі змісту, у назвах розділів/підрозділів, у назвах публікацій, переліку джерел, посилань, тощо;
2) В якості запозичень системою StrickePlagiarism було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) Запозичення, виявлені в текстовій роботі, є фрагментарними або мають належним чином оформлені посилання;
4) Виявлені модифікації тексту не впливають на відсоток схожості;

Максимальний обсяг запозичень визначений системою Anti-Plagiarism, складає 13%. За системою StrickePlagiarism коефіцієнт подібності 1 складає 2.7% , коефіцієнт подібності 2 складає 0.5%.

Дата 17.12.2025

Завідувач кафедри — Леонід БЕДРАТЮК
Гарант освітньої програми — Оксана ЯШИНА
Керівник кваліфікаційної роботи — Оксана ОНИШКО

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «Магістр»

Дипломник Червончук Ілля Сергійович

Тема Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень немає; кількість сторінок записки 88

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі виконано комплексний аналіз проблеми автоматизації діагностики збоїв програмного забезпечення, зокрема у частині виявлення першопричин помилок через аналіз журналів виконання. Проведено дослідження існуючих методів, моделей та інструментів моніторингу лог-файлів, проаналізовано їх сильні та слабкі сторони щодо роботи з динамічними даними. На основі проведеного аналізу удосконалено метод виявлення помилок у програмному забезпеченні шляхом інтеграції підходу диференціального порівняння еталонних та аномальних трас виконання. Запропонована модель включає етапи нормалізації лог-повідомлень, структурного вирівнювання послідовностей подій, фільтрації шуму та локалізації відхилень. Розроблено алгоритм та архітектуру програмної системи, що реалізує запропонований метод. Продемонстровано результати роботи системи та її придатність до практичного використання для скорочення часу налагодження програмних продуктів.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота відповідає затвердженому завданню, а поставлені цілі досягнуті у повному обсязі.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи
У вступі наведено обґрунтування актуальності, мети, завдань, об'єкта та предмета дослідження.
У першому розділі проведено аналіз предметної області, визначено наукові підходи та сформульовано постановку задачі.
У другому розділі представлено модель та удосконалений метод формалізації вимог.
У третьому розділі описано архітектуру програмної системи.
У четвертому розділі наведено результати реалізації та оцінювання системи.

4. Позитивні сторони роботи
– Ґрунтовний аналіз предметної області, методів та засобів автоматизованого моніторингу програмного забезпечення.
– Обґрунтований та актуальний удосконалений метод виявлення помилок на основі диференціального аналізу логів.
– Реалізація програмного забезпечення, що демонструє придатність методу для використання в CI/CD процесах.
– Структурований та логічний виклад матеріалу, що послідовно розкриває етапи дослідження та розробки.

5. Негативні сторони роботи
– окремі аспекти алгоритмів нормалізації та фільтрації шуму подані узагальнено та потребують глибшого опису;
– математична модель оцінки схожості лог-файлів могла б бути більш детально формалізована;
– результати експериментальної перевірки системи подані стисліше, ніж це необхідно для повної оцінки ефективності (бракує кількісних метрик).

6. Оцінка графічного оформлення та пояснювальної записки Оформлення виконано акуратно та відповідно до вимог. Графічні матеріали подані коректно, хоча деяким з них не вистачає більш детальних пояснень.

7. Відгук про кваліфікаційну роботу в цілому Робота є завершеним дослідженням, яке демонструє володіння основними методами інженерії вимог та здатність автора застосовувати отримані знання на практиці. Незважаючи на окремі недоліки, робота має певну наукову та практичну цінність

8. Інші зауваження Бажано доопрацювати якість формулювань, логічні переходи між розділами та покращити стиль викладення. Експериментальну частину слід значно поглибити.

9. Оцінка кваліфікаційної роботи З огляду на переваги та недоліки, робота заслуговує оцінки “задовільно”.

РЕЦЕНЗЕНТ (прізвище, ім’я, по-батькові, посада, місце роботи) Кацеглин Марія Вікторівна, доцент каф. КІТС, ХНУ

“ 17 ” грудня 2025 р.
(підпис)

Завідувачу кафедри
інженерії програмного забезпечення
проф. Леоніду БЕДРАТЮКУ
студента групи ІПЗм-24-1
Чубончук І. С
Ім'я, ПРІЗВИЩЕ

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення»:

Метод автоматизованого виявлення помилок у програмному забезпеченні на основі диференціального аналізу логів виконання

(керівник кваліфікаційної роботи – Онищко Оксана Григорівна)
Ім'я, ПРІЗВИЩЕ

01.09.2025
Дата

Підпис здобувача

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Червончука Іллі Сергійовича
факультет ІТ, 2 курс, група ІПЗм-24-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений, та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1.09.2025
дата

підпис