DOI 10.31891/2307-5732-2022-309-3-44-49
УДК 004.056.5:621

ПРАВОРСЬКА Н. І.
Хмельницький національний університет
https://orcid.org/0000-0001-6001-3311
e-mail: margana2000007@gmail.com
ГРИПИНСЬКА Н. В.
Хмельницький національний університет
https://orcid.org/0000-003-0103-976X
e-mail: nvhrypynskaN@khmnu.edu.ua

ЕКСПЕРИМЕНТИ ТА ВИКОРИСТАНІ ОЦІНОЧНІ МЕТРИКИ ДОСЛІДЖЕНЬ, ЯКІ ПРОВОДИЛИСЬ ПРИ РОЗРОБЦІ МОВНО-НЕЗАЛЕЖНОГО ІНКРЕМЕНТНОГО ДЕТЕКТОРА

При розробці мовно-незалежного інкрементного детектора (МНІДП) важливу роль відіграють проведені експерименти та оціночні метрики, які дадуть змогу проаналізувати результати розробки та придатність розробленого алгоритму та пристрою. Експерименти дадуть також змогу відповісти на питання оцінювання продуктивності розробленого детектору та порівняти його з підходом комерційного рівня SIG з виявлення клонів, для вивчення переваг, які може запропонувати інкрементний підхід. Щоб отримати представлення про продуктивність МНІДП, запропоновано запускати його для п'яти програмних систем (з відкритими кодами), проводячи вимірювання вимог до часу та пам'яті. Також для відповіді на питання розширення та вдосконалення початкового підходу, шляхом використання локально-чутливого хешування (ЛЧХ), виникає необхідність в вимірюванні продуктивності запропонованого розширення на базі ЛЧХ та порівнянні його з продуктивністю МНІДП.

Ключові слова: мовно незалежний детектор, інкрементний підхід, локально-чутливе хешування, експеримент, оціночні метрики

Natalya PRAVORSKA, Nadiia HRYPYNSKA
Khmelnytskyi National University

EXPERIMENTS AND USED EVALUATION METRICS USED IN THE DEVELOPMENT OF A LANGUAGE-INDEPENDENT INCREASE DETECTOR

Experiments and evaluation metrics play an important role in the development of a language-independent incremental detector (MRIP), which will allow to analyze the results of the development and the suitability of the developed algorithm and device. The experiments will also provide an answer to the question of evaluating the performance of the developed detector and comparing it with the commercial SIG approach to clone detection, to explore the benefits that the incremental approach can offer. To get an idea of the performance of MNIDP, it is proposed to run it for five software systems (open source), measuring the requirements for time and memory. Also, to answer the question of expanding and improving the initial approach, by using locally sensitive hashing (LCH), there is a need to measure the performance of the proposed expansion on the basis of LCH and compare it with the performance of MNIP. The experiments conducted in the study provided some useful information based on the evaluation of the effectiveness of the proposed expansion on the basis of LCH. More specifically, in some cases, compared to the implementation of MNIDP, the stage of creating an index in the approach based on LCH was two, and in some cases three times slower. A possible reason for this may be the complexity of the MinHash operation, which is a significant part of the overall LCH scheme. This becomes obvious when you consider that the hashing of each tile for each set of tiles during MinHashing must be performed by k-based hash functions. It was assumed that the process of incremental implementation step on the basis of MNIDP will be much slower, due to the calculation of index records on the fly. However, in the course of the study, opposite results were obtained. In practice, this was justified by the fact that the similarity threshold used did not cause a large number of matches between the source files. To gain a better understanding of its behavior, there is a need for further research into the relationship of runtime required for the incremental implementation step flow based on the LCH and the similarity threshold.

Keywords: language independent detector, incremental approach, locally sensitive hashing, experiment, estimation metrics

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

У процесі розробки мовно-незалежного інкрементного детектору постає питання вибір оціночних метрик, та проведення на їх основі експериментів для подальшого аналізу і висновків стосовно придатності використання як запропонованого алгоритму, так і самого пристрою. В роботі [1] автори внесли пропозиції стосовно характеристик, на які необхідно звернути увагу при розробці алгоритмів.

Аналіз досліджень та публікацій

Принципи побудови МНІДП спираються на роботу [2], де авторами запропоновано використовувати індекс клонованої чистини коду, який і буде становити основну структуру даних. Також в роботах [3, 4] авторами не тільки розглянуто побудову мовно-незалежного інкрементного детектора, а саме процес виявлення блоків з повторами і надлишковістю при використанні пристрою та його розширення на базі ЛЧХ.

Виклад основного матеріалу

Для уникнення можливих помилок при розробці програмного забезпечення виникає необхідність виявлення дефектів на самих ранніх стадіях життєвого циклу. Саме дефекти можуть стати причиною збільшення витрат (як фінансових, часових і т.п.) при розробці та супроводу ПЗ, особливо, коли виявлення дефектів відбувається на останніх етапах життя програмного продукту. Помилки розробки програмного коду можуть бути припущені навіть досвідченими професіоналами-розробниками. Одними з таких помилок можуть бути клоновані блоки коду (тобто повтори та надлишковості). Через наявності великої кількості дубльованих частин коду з'являється можливість виникнення низки різноманітних проблем в вихідному коді системи. Збільшення розміру кодової бази, і відповідно, збільшення затрат на обслуговування, зокрема, є наслідком дублювання [1].

Важливим аспектом стає наявність інструментів для проведення аналізу вихідного коду. На сьогодні існує велика кількість подібних інструментів, але на жаль вони виконують дуже багато зайвих операцій та мають залежність від мови програмування. Прикладом може бути автоматизований інструмент, розроблений групою вдосконалення програмного забезпечення SIG. Детектор цієї компанії, використовуючи перелік заздалегідь визначених критеріїв, серед яких є складність коду та його обсяг, зв'язок між модулями і тому подібне, дає можливість оцінити ремонтопридатність розробленого програмного продукту. Згадані критерії покладено в основу роботи детекторів виявлення блоків з повторами та надлишковістю. Доля дубльованого коду в кодовій базі розробленого проекту може бути визначені спираючись на отримані від детектора результати. Детектор повторів із реальною застосовністю слів розроблено саме даною компанією, однак він не є незалежним від мови програмування.

В розробці мовно-незалежного інкрементного детектору основою якого і виступають деякі засади автоматизованого інструменту для виявлення дубльованого коду SIG. Алгоритм, за яким буде працювати запропонований МНІД базується на основі індексу повторів. Таким принцип добре висвітлений в роботі Хаммела [2]. Індекс клону в цьому випадку є основною структурою даних. Представляється такий індекс глобальною структурою даних і подібний до типового інвертованого індексу. Підхід Хаммела, хоч і має в використанні лексичний аналізатор, що дозволяє перетворити вихідний код в токени, але він найбільш наближений до інкрементного підходу незалежного від мови детектора. Принцип роботи МНІДП висвітлений в статті [3]. В роботі [4] представлене вдосконаленні мовно-незалежного інкрементного детектора шляхом застосування локально-чутливого хешування та експерименти, які проводились для п'яти проектів (а саме, Rippled, Kooboo, Tensorflow, Openjdk-jdk14u, Linux Kernel) з метою виявлення блоків з повторами та надлишковістю.

При дослідженні роботи МНІДП постає необхідність проведення експериментів, які зможуть відповісти на питання оцінювання продуктивності розробленого детектору та порівнянні його з підходом комерційного рівня SIG по виявленню клонів, для вивчення переваг, які може запропонувати інкрементний підхід.

Види експериментів

При експериментах розроблено для дві реалізації детектора виявлення повторів і надлишковості в програмному коді з потрібними функціями. Далі вимагається проведення кількісних експериментів для конкретних випадків.

Оцінка МНІДП

На питання, як діє підхід МНІДП, для початку треба визначити, принцип роботи запропонованого детектора, який було отримано в наслідок корегування підходу представленого Хаммелом [2]. Щоб отримати представлення про продуктивність МНІДП, його запускають для п'яти програмних систем (які зазначені вище), проводячи вимірювання вимог до часу та пам'яті. Тобто для кожної системи відбувається аналіз серії з п'ятдесяти комітів та виміри часу і пам'яті, які будуть потрібні для процесу створення індексу. Також проходить вимірювання середнього часу, потрібного для обробки п'ятдесяти комітів під час кроків інкрементного процесу. Для спостереження за поведінкою МНІДП по відношенню до виявлених повторів та надлишковості у програмному коді, постає потреба у проведенні додаткових експериментів. Зокрема, відбувається аналіз десяти самих останніх комітів для кожної з систем, існуючих в наборі даних і виявлення тих повторів (клонів), які додавалися та знищувалися.

Оцінювання МНІДП проти SIG

В ході експериментів відбувається порівняння ефективності запропонованого у дослідженні підходу МНІДП та сучасного традиційного підходу SIG, для виявлення повторів та надлишковості в програмному коді. Остаточна мета – це перевірка покращення, яке має забезпечувати інкрементний підхід, в тому разі, коли процес виявлення повторюється регулярно для кожного нового перегляду програмного проекту.

Порівнювання виявляється непростою справою, оскільки детектор клонів SIG відрізняється від МНІДП в багатьох відношеннях. Конкретніше, процес виявлення повторів в програмному коді SIG відпрацьовує в межах SAT (інструмент, для запуску окрім виявлення клонів, ще великої кількості додаткових процесів). Тому з'являється потреба в ізолюванні процесу виявлення клонів та виміру його окремо від інших процесів. Крім цього, відбувається виведення всіх клонів в версії програмного

забезпечення через те, що детектор SIG не є інкрементним. Напроти, МНІДП дає змогу виведення тільки нещодавно доданих або видалених блоків повторів (клонів) в визначеній версії системи. З цього випливає, що неможливо співставити напряму результати двох підходів, саме з точки зору виявлення повторів.

В даному дослідженні проводиться заміри і порівняння двох вищезгаданих підходів на основі часу, потрібного для процесу виявлення повторів та надлишковості в програмному коді. Зокрема, відбувається застосування детектору SIG для серії з півсотні комітів, заміри середнього часу, потрібного для обробки та порівняння результату з результатами індивідуальної оцінки МНІДП. Щодо виявлених кожним інструментом повторів (клонів коду) та надлишковості, та врахування описаної відмінності в вихідних даних кожного з підходів, на практиці відбувається перевірка на співпадіння клонів, виявлених цими детекторами. Однак, через те, що запропонований в дослідженні метод, та детектор SIG не являються ймовірнісними, заснованими на тексті методами (тобто необроблений вихідний код, не підлягає ніяким перетворенням, які можуть призвести до втрати інформації, що тягне за собою зниження точності та повноти), то можна очікувати співпадіння виявлених блоків повторень, виявлених обома підходами. Щодо пам'яті, то знову підходи виявляються неспівставними, через складність ізоляції процесу виявлення блоків повторів SIG та необхідності проведення вимірювання пам'яті тільки для цієї частини процесу.

Оцінка розширень, заснованих на ЛЧХ

Для відповіді на питання розширення та вдосконалення початкового підходу, шляхом використання ЛЧХ, виникає необхідність в вимірюванні продуктивності запропонованого розширення на базі ЛЧХ та порівнянні його з продуктивністю МНІДП. Так само, як і при оцінюванні мовно-незалежного інкрементного детектору повторів, досягається це виміром часу виконання та потреб в пам'яті запропонованого в дослідженні розширення на основі ЛЧХ. Порівнянні цих показників з відповідними результатами для підходу з використанням детектору МНІДП. Є можливість проведення індивідуального порівняння, оскільки обидва підходи являються інкрементними, та в їх основі полягає одна і та сама основна ідея. Відбувається порівняння метрик для кожного з двох процесів, процесу створення індексу та процесу інкрементного кроку.

Знову буде застосоване вище згадане експериментальне налаштування, де використані п'ять програмних систем і півсотні комітів для кожної з них. Важливо звернути увагу на те, що дослідження направлене лише на вивчення запропонованого розширення на базі ЛЧХ з точки зору ефективності. Тобто, запропонований метод може призвести до підвищення продуктивності в будь-якому з двох робочих процесів: створення індексу та інктементного кроку.

Логічно було б провести вимір відгуку реалізації заснованої на ЛЧХ, оскільки цей метод за визначенням включає ймовірність пропуску блоків коду з повторами (клонів) та надлишковістю, але такі дослідження проводитися не будуть. Після того, як буде визначено спочатку, чи достатньо ефективною виявилася реалізація на основі ЛЧХ для покращення підходу МНІДП, тільки потім має сенс отримати висновки, які містили б корисну інформацію, про втрату клонів. Однак результатом нашого дослідження стає вивчення ефективності підходу, а відповідні емпіричні експерименти по втраті блоків коду з повторами та надлишковістю залишаються в якості майбутньої роботи.

Вхідна перевірка

Для перевірки результатів, які будуть отримані після застосування розроблених детекторів, виникає потреба в еталонному тесті для обраних програмних систем. Тобто, потрібно отримати оцінку того, чи являються отримані виявлені блоки з повторами та надлишковістю дійсними та повними, а саме, чи були виявлені всі доступні клони. Під час розробки запропонованого дослідження подібний набір перевірочних даних не був доступним. В зв'язку з цим перевірка результатів запропонованого підходу на основі МНІДП та ЛЧХ в контексті даного невеличкого проекту відбувалася вручну. Було протестовано і перевірено декілька варіантів використання, які стосувалися знищення, оновлення, створення чи перейменування файлів. Вручну створено коміти для імітації оновлення реальної програмної системи. Звичайно, перевірка на повноту для детектора на основі ЛЧХ не проводилась, через характер даного методу. Зверніть увагу, хоча результат тестувався для невеликого проекту, очікується, що запропонований підхід буде відпрацьовувати так само і для великих кодових баз, якщо охоплюватимуться всі сценарії варіантів використання. Тести, які були проведені, відносяться до наступних варіантів використання:

1. *Додавання новостворених файлів.* Виконувалося тестування сценаріїв, в яких ці файли представляли один або декілька блоків з повторами (клонами) та надлишковістю, а інші сценарії, в яких дублювання не додавалося. У першому типі сценаріїв проводилася перевірка на правильність виявлення блоків з повторами та надлишковістю коду, тоді як в другому перевірялося, що клони не виводилися.

2. *Перейменування файлів.* Відбувалася перевірка, що блоки коду з повторами в перейменованих файлах були виявлені та зареєстровані з оновленим ім'ям файлу.

3. *Оновлення файлів.* Проводилося тестування декількох сценаріїв, в яких знищувалися та додавалися блоки коду в існуючі файли, перевіряючи виявлення знищених або доданих фрагментів програмного коду з повторами та надлишковістю.

4. *Знищення файлів.* Тестування стосувалося видалених файлів, в яких містилися блоки коду з повторами (клонами) та надлишковістю і підтвердження в цьому разі, що відбулося вірне виявлення видалених клонів.

Конфігурація інструментів.

Параметри конфігурації, які були використані в експериментах для реалізації детекторів клонів, заснованих на МНІДП та ЛЧХ, представлені в таблиці 1.

Таблиця 1

Конфігурація реалізацій МНІДП та ЛЧХ

LIICD	LIICD LSH-based		
CHUNK_SIZE	CHUNK_SIZE	PERMUTATIONS	THRESHOLD
6	6	64	0.2

Параметр CHUNK_SIZE – це кількість рядків у кожному блоці коду, який підлягає хешуванню. Значення його було обрано рівним 6, що означає мінімальний розбір блоку програмного коду з повторами та надлишковістю, та вказує на ідентичну мінімальну довжину клону, яку SIG визначає за замовчуванням на даний час.

Параметр PERMUTATIONS – це кількість функцій, в запропонованій реалізації на основі ЛЧХ, які використовуються для процесу MinHashing. З точки зору порівняння подібності, значення цього параметра рівне 64, призводить до рівня помилок, розрахованих за формулою (1)

$$\text{error} = 1/\sqrt{64} = 12{,}5\% \quad (1)$$

Тобто, коли відбувається ідентифікація двох фалів як подібних/не подібних, ймовірність того, що ця ідентифікація помилкова складає 12,5%.

Параметр THRESHOLD – поріг, який прирівнюється до 0,2, та перетворюється в 20%, вказуючи на самий низький поріг подібності, для якого визначається подібність двох файлів.

Інфраструктура

Для досягнення мети експериментів, при їх проведенні запускається дві категорії тестів на одній машині з налаштуванням апаратного забезпечення, представленого у таблиці 2.

Таблиця 2

Технічні характеристики забезпечення

	Специфікація
Пам’ять	32 GB
Процесор	Intel Xeon E5-2650 v2 @2.6GHz

Інформаційний фонд

В ході проведення досліджень для різних реалізацій детекторів, було використано в якості вихідних даних п’ять проектів з відкритим вихідним кодом. Вибір відбувався таким чином, щоб була змога охопити відносно широкий спектр проектів, які мають різний розмір та написаних на різних мовах програмування. Конкретніше, відбувалося вимірювання розміру кожного проекту з точки зору LOCs (Lines Of Code), з застосуванням інструменту CLOC[1]. Даний інструмент спроможний проводити підрахунок пустих рядків, рядків коментарів та фізичних рядків вихідного коду на багатьох мовах програмування. Вимірювання точної частки дублювання у вибраних системах неможливе, через характер запропонованих в дослідженні детекторів, спроможних виводити блоки коду з повторами (клони), які були додані або видалені в певній версії, замість того, щоб виводити всі клони і цій версії. Однак, спираючись на статистичні дані, які отримані з аналізу, проведеного в рамках SIG 192 програмних систем, виявилося, що середнє дублювання коду складає приблизно 13%, при стандартному ± 12%. В даному розділі роботи представлено процес вимірювання LOCs для кожної системи, а також процес фільтрації вихідних частин.

Початкова колекція інформаційного фонду

Завдяки вимірюванню великої кількості проектів з відкритим кодом, для створення колекції початкового інформаційного фонду, було обрано п’ять з них. При цьому їх LOCs коливаються в діапазоні приблизно від 300 000 до 23 000 000 рядків коду. На рис. 1 представлено лістінг, де вказані параметри конфігурації CLOC, які було використано на цьому етапі.

```
1  $ cloc --skip-uniqueness {target_project}
```

Рис. 1. Початкова конфігурація CLOC

[1] Інструмент доступний на Github: https://github.com/AlDanial/cloc

Обрані проекти зведені в таблицю 3, де також вказана мова програмування, загальна кількість файлів, та кількість рядків в програмному коді LOCs.

Причиною обрання програмних систем з LOCs нижче описаного діапазону являється те, що справжні переваги інкрементного підходу стають видимими лише для проектів з відносно великою кількістю рядків в програмному коді, коли повторне виявлення з використанням традиційних підходів стає доволі повільним.

Обране об'єднання програмних проектів, як видно з таблиці, охоплює широкий спектр мов програмування та кількості (LOCs) рядків. Надані значення у стовпці «Кількість стовпців LOCs» – є результатом додавання, що відповідає коментарям, і тих, які посилаються на фактичне джерело коду. До уваги не приймаються пусті рядки, оскільки їх буде знищено на етапі попередньої обробки кожного детектору.

Таблиця 3

Початковий інформаційного фонду проектів з відкритим кодом

	Мова програмування	**Кількість файлів**	**Кількість LOCs**
Linux Kernel	C	57.205	23.229.768
Openjdk-14	Java	60.444	12.045.316
Tensorflow	C++, Python	12.387	3.194.893
Kooboo	C#, JS, HTML, CSS	4.109	670.265
Ripple	C / C++	1.399	312.011

Фільтрація виконуваного коду

Для вимірювання різних показників ремонтопридатності, включаючи виявлення повторів та надлишковості у програмному коді SIG використовує такий інструмент, як SAT. За допомогою нього також є можливість виконання відповідного аналізу тільки тих частин кодової бази, що позначені як виконуваний код.

Ігнорування відбувається і коду, поміченого, як тестовий код, разом із іншими незначними файлами, такими як README.md або файли журналів. Після обробки SAT тільки частини всієї кодової бази, в результаті, що логічно, іде зменшення кількості LOCs, які піддаються обробці. Для об'єктивного порівняння детектора SAT та запропонованої в дослідженні реалізації, треба враховувати і таке знищення. Тому знову доведеться використовувати CLOC для оцінки LOCs для кожного проекту, включаючи цього разу неревалентні каталоги. Точна конфігурація представлена в лістінгу на рис. 2. Нові вимірювання для вищезгаданих проектів подані в таблиці 4.

```
$ cloc --skip-uniqueness --exclude-dir=test,tests,doc,examples,licences
,lib --not-match-f=^.*test.*$ {target_project}
```

Рис. 2. Конфігурація фільтрації CLOC

Таблиця 4

Відфільтрований інформаційний фонд проектів з відкритим кодом

	Мова програмування	**Кількість файлів**	**Кількість LOCs**
Linux Kernel	C	56.270	22.963.637
Openjdk-14	Java	23.905	7.498.482
Tensorflow	C++, Python	8.610	2.120.650
Kooboo	C#, JS, HTML, CSS	4.064	668.055
Ripple	C / C++	1.073	207.166

Фільтрація невірних файлів

Програмні проекти, в тому числі ті, які обрані в нашому інформаційному фонді, зазвичай складаються з ряду файлів у двійковому форматі. Обробка подібних файлів для виявлення, в потрібному контексті, блоків з повторами та надлишковістю позбавлена сенсу, оскільки не можна отримати цінної інформації. Отже при проведенні обробки кожного відповідного проекту, додатково будуть виключені двійкові байти та файли з неюнікодовими символами в обох реалізаціях МНІДП та ЛЧХ. В додатку Б представлено список з вказуванням усіх розширень файлів, які піддавалися фільтрації. Увага звертається на те, що, оскільки подібні файли утворює лише невелика частина всієї кодової бази, то видалення таких файлів, як і очікується, лише незначно вплине на кількість рядків (LOC) у коді, які згадуються у попередніх розділах.

Моделювання комітів

Постає необхідність моделювання процесу відправки комітів в репозиторій управляння версіями, де буде розміщено проект, який підлягає аналізу. Це забезпечуватиме спостереження та вимірювання того, як працюють детектори під час покрокового робочого процесу. Таке моделювання відбувалося з

використанням файлів конфігурації JSON. Подібні файли складаються за списків комітів, які піддаються аналізу, а також змін (створення, оновлення, знищення, перейменування), які кожний з цих комітів вносить.

В контексті експериментів, при проведенні досліджень в ході роботи, для кожного з програмних проектів відбувається аналіз п'ятдесяти останніх комітів, виключаючи злиття, та вимірюються показники, які відповідають кожній з описаних категорій експериментів. Самий останній коміт для кожного програмного проекту, показує снепшот (знімок файлової системи – копія файлів та каталогів файлової системи на даний момент часу), використаний під час аналізу та представлений в Додатку А. Оскільки детектори за визначенням виключать деякі з цих півсотні комітів, які можуть складатися із змінених файлів, то існує ймовірність обробки менше ніж п'ятдесяти. Наприклад, коміт буде пропускатися, якщо він включатиме лише модифікації тестових файлів (які пропускаються детекторами), то зрештою ніякі файли не оброблятимуться.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Проведені в дослідженні експерименти надали деяку корисну інформацію на основі оцінки ефективності запропонованого розширення на основі ЛЧХ. Конкретніше, в деяких випадках у порівнянні з реалізацією МНІДП, етап створення індексу в підході на основі ЛЧХ виявився в два, а в деяких випадках в три рази повільнішим. Можливою причиною, яка призводить до цього може бути складність операції MinHash, яка складає значну частину загальної схеми ЛЧХ. Це стає очевидним, якщо враховувати, що хешування кожної черепиці для кожного набору черепиць під час MinHashing має проводитися обумовленими k хеш-функціями. Навпаки, з точки зору використання пам'яті, що призвело до скорочення у п'ять разів для випадку Linux, ЛЧХ виявився набагато ефективнішим. Однак, тільки це не являється виправданням для використання даного підходу замість МНІДП. Вимірювання інкрементного кроку, крім цього, не відповідало нашим первинним очікуванням. Передбачалося, що процес інкрементного кроку реалізації на основі у порівнянні з МНІДП буде значно повільнішим, через обчислення записів індексу на льоту. Однак в ході проведеного дослідження було отримано протилежні результати. Практично, це виправдовувалося тим, що використаний поріг подібності не став причиною виявлення великої кількості збігів між вихідними файлами. Для отримання кращого розуміння його поведінки виникає необхідність проведення подальших досліджень зв'язку часу виконання, потрібного для потоку інкрементного кроку реалізації на основі ЛЧХ та порогу подібності.

Література

1. Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. IEEE Transactions on software engineering, 33(9):577–591, 2007.

2. Benjamin Hummel, Elmar Juergens, Lars Heinemann, and Michael Conradt. Indexbased code clone detection: incrementtal, distributed, scalable. In 2010 IEEE International Conference on Software Maintenance, pages 1–9. IEEE, 2010.

3. Праворська Н.І, Бармак О.В., Медзатий Д.М., Шестакевич Т.В. Процес виявлення блоків з повторами і надлишковістю при використанні мовно-незалежного інкрементного детектору. Вісник Хмельницького національного університету, серія Технічні науки, № 3, 2021, С. 39–45.

4. Праворська Н.І., Бедратюк Л.П., Форкун Ю.В. Яшина О.М. Мовнонезалежний детектор для виявлення і усунення повторів та надлишковостей програмного коду. Вимірювальна та обчислювальна техніка в технологічних процесах. Хмельницький, 2021. № 1, С. 56–61.

References

1. Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. IEEE Transactions on software engineering, 33(9):577–591, 2007.

2. Benjamin Hummel, Elmar Juergens, Lars Heinemann, and Michael Conradt. Indexbased code clone detection: incrementtal, distributed, scalable. In 2010 IEEE International Conference on Software Maintenance, pages 1–9. IEEE, 2010.

3. Pravorska N.I, Barmak O.V., Medzatyi D.M., Shestakevych T.V. Protses vyiavlennia blokiv z povtoramy i nadlyshkovistiu pry vykorystanni movno-nezalezhnoho inkrementnoho detektoru. Herald of Khmelnytskyi National University, seriia Tekhnichni nauky, № 3, 2021, s. 39-45.

4. Pravorska N.I., Bedratiuk L.P., Forkun Yu.V. Yashyna O.M. Movnonezalezhnyi detektor dlia vyiavlennia i usunennia povtoriv ta nadlyshkovostei prohramnoho kodu. Vymiriuvalna ta obchysliuvalna tekhnika v tekhnolohichnykh protsesakh. Khmelnytskyi, 2021. № 1, s. 56-61.