УДК 004.4'2+004.4'416 **DOI:** 10.31673/2412-9070.2024.050575
І. В. ЗАМРІЙ, доктор техн. наук, доцент;
ORCID: 0000-0001-5681-1871
О. В. КУЛАКОВ, аспірант,
ORCID: 0009-0006-3456-0569
Державний університет інформаційно-комунікаційних технологій, Київ

ОГЛЯД ШЛЯХІВ АВТОМАТИЗАЦІЇ РЕФАКТОРИНГУ КОДОВИХ КЛОНІВ

Сучасний розвиток технологій у сфері програмної інженерії вимагає постійних зусиль з оптимізації та покращення якості коду. Однією з нагальних проблем є наявність кодових клонів, які можуть призводити до помилок та ускладнення подальшого обслуговування програмних систем. Автоматизація рефакторингу кодових клонів представляє значний інтерес для дослідників і практиків у цій області.

Основні задачі дослідження охоплюють декілька ключових аспектів. Перш за все, дослідження зосереджується на глибокому аналізі наявних методів і технік, які вже використовуються для автоматизації рефакторингу клонів. Це включає вивчення ефективності існуючих підходів та виявлення їхніх обмежень. Важливою частиною є аналіз алгоритмів, які можуть покращити точність виявлення клонів та ефективність їхнього рефакторингу. Крім того, дослідження аналізує вплив автоматизації на якість і обслуговуваність програмних систем, що допомагає визначити, як такі технології можуть бути інтегровані у реальні проєкти розробки програмного забезпечення.

Дослідження покликане виявити обмеження поточних підходів та запропонувати шляхи їх оптимізації, враховуючи сучасні технологічні можливості та потреби ринку програмного забезпечення. Дослідження включає критичний огляд літератури, вивчення звітів про застосування інструментів на практиці та аналіз результатів використання автоматизованих систем у реальних проєктах. Це дає можливість оцінити ефективність різних алгоритмів і методик виявлення та рефакторингу кодових клонів, а також їхній вплив на якість кінцевого продукту та ефективність розробки. Висновки, запропоновані в рамках дослідження, включатимуть рекомендації щодо усунення виявлених прогалин у поточних методиках, пропозицію нового підходу до автоматизації рефакторингу клонів, що зможе забезпечити більшу адаптивність та ефективність.

Ключові слова: кодові клони, автоматизація рефакторингу, машинне навчання, статичний аналіз, оптимізація коду, якість програмного коду.

Вступ

Автоматизація процесів у програмній інженерії постійно еволюціонує, пропонуючи нові шляхи для покращення ефективності розробки та підтримки програмного забезпечення. Однією з ключових проблем у цій галузі є наявність кодових клонів — дублікатів або майже ідентичних фрагментів коду, які можуть спричинити підвищення вартості обслуговування та зниження якості продукту. Ефективний рефакторинг клонів, який здатен визначити та оптимізувати ці повторювані блоки коду, є важливим для забезпечення сталості та ефективності програмних систем.

Основна мета дослідження — не лише провести глибокий аналіз існуючих технологій, але й виявити найбільш перспективні напрямки для подальшого розвитку автоматизації рефакторингу клонів, що може значно вплинути на майбутнє програмної інженерії. Висвітлення новітніх досліджень та інноваційних практик допоможе формувати рекомендації для вдосконалення робочих процесів і технологічних рішень, спрямованих на ефективний рефакторинг кодових клонів у різних умовах розробки програмного забезпечення.

Аналіз літературних даних і постановка проблеми

Існує значна потреба у покращенні існуючих методів та інструментів для автоматизації рефакторингу кодових клонів. Хоча багато сучасних методів забезпечують високу точність і ефективність, все ще залишаються виклики, пов'язані з інтеграцією цих інструментів у реальні проєкти, мінімізацією хибних позитивних виявлень та адаптацією до специфічних вимог різних галузей. Подальші дослідження повинні зосередитися на розробці універсальних підходів, які можуть легко інтегруватися у різні середовища розробки, підвищуючи загальну продуктивність і якість програмного забезпечення.

З метаю вирішення проблеми рефакторингу кодових клонів і виокремлення найбільш ефективних методик рефакторингу у статті проаналізовано наступні роботи:

Робота [1] присвячена порівняльному аналізу різноманітних технік рефакторингу кодових клонів, визначаючи ефективність кожної техніки на основі певних критеріїв. Використані підходи включають автоматизоване виявлення важливих клонів для рефакторингу, які часто мають вищий ризик дефектів або проблем з обслуговуванням.

У [8] дослідження фокусується на кодових клонах у програмному забезпеченні для програмованих логічних контролерів, з акцентом на їх рефакторинг у контексті промислових систем, зокрема використання нормалізації та методів фільтрації для підвищення релевантності клонів в рамках обслуговування.

Оглядове дослідження, що категоризує існуючі дослідження з рефакторингу клонів, виділяючи різні категорії підходів та обговорюючи можливості для подальшого дослідження в цій області проведено у [9] .

Робота [10], яка аналізує методи рефакторингу кодових клонів у великих програмних системах, зокрема розглядається використання різних стратегій детекції клонів, таких як текстове порівняння, AST (Abstract Syntax Tree) і токен-базовані методи.

Аналіз наукових робіт вказує на ряд істотних проблем, які залишаються актуальними у цій галузі. Основними викликами є висока частка помилкових позитивних результатів при виявленні клонів, що веде до ненеобхідних рефакторингів; недостатність інструментів для адекватного розпізнавання семантично різних, але функціонально схожих клонів; та обмежена здатність існуючих систем адаптуватися до специфічних вимог проєктів. Також важливою проблемою є велика обчислювальна вартість рефакторингу, особливо в контексті великих кодових баз, що обмежує можливість їх використання у реальному часі та вимагає значних ресурсів для обробки.

Мета і задачі дослідження

Метою цього дослідження є всебічний огляд існуючих методів та інструментів для автоматизації рефакторингу кодових клонів з акцентом на їх ефективність, обмеження та потенціал для подальшого розвитку. Це дослідження має на меті узагальнити поточний стан знань у цій галузі та надати рекомендації для майбутніх досліджень та практичного застосування.

Задачі дослідження:

- систематизація існуючих підходів;
- оцінка ефективності методів;
- ідентифікація обмежень, недоліків та викликів;
- надання рекомендацій щодо вибору найбільш ефективних підходів залежно від специфіки проєкту.

Результати дослідження

Систематизація існуючих підходів. Можна виділити кілька основних методів, які використовуються для рефакторингу кодових клонів. Ці методи включають текстові методи, методи на основі дерева синтаксичного аналізу (AST), методи на основі токенів, а також підходи, що використовують машинне навчання. Кожен з цих методів має свої особливості, переваги та недоліки, які слід розглянути більш детально.

Текстові методи [2, 10] порівнюють текстові рядки для виявлення дублікатів коду. Це найпростіший підхід, який ефективний для виявлення ідентичних або майже ідентичних клонів. Текстові методи не враховують структуру або семантику коду, але можуть бути корисними для швидкого виявлення повторюваних фрагментів.

Для визначення схожості між двома фрагментами коду x та y використовується функція перевірки:

$$\text{Check}(x,y) = \begin{cases} 1, \text{якщо } x=y; \\ 0, \text{в іншому випадку} \end{cases}$$

Регулярні вирази [2, 10] використовуються для виявлення повторюваних шаблонів у коді. Цей метод дозволяє підвищити точність порівняно з простим текстовим аналізом, оскільки регулярні вирази можуть враховувати певні шаблони і структури в коді. Регулярні вирази надають можливість більш гнучко знаходити клоновані фрагменти, але все одно не враховують повну семантику коду.

Методи на основі дерева синтаксичного аналізу (Abstract Syntax Tree, AST) [8, 10] використовують абстрактне синтаксичне дерево для аналізу структури коду. AST представляє структурну форму коду, яка дозволяє враховувати не лише текстові збіги, але й синтаксичні зв'язки між різними елементами коду. Це значно підвищує точність виявлення клонів, оскільки дозволяє виявляти фрагменти, які виконують однакові функції, навіть якщо вони мають різну синтаксичну структуру. AST-методи часто використовуються в інструментах статичного аналізу коду та можуть вимагати значних обчислювальних ресурсів для побудови і порівняння дерев.

AST-методи використовують абстрактні синтаксичні дерева для детального аналізу коду, дозволяючи розробникам точно оцінювати структурну схожість коду. Принцип роботи полягає у створенні дерев, де кожен вузол відображає конкретний синтаксичний елемент коду, такий як оператори, змінні або блоки виконання.

Функція схожості Sim(T1, T2) оцінює, наскільки два AST, T1 та T2 схожі один на одного. Ця функція порівнює вузли дерева, аналізуючи не тільки типи вузлів, але й їх взаємне розташування та відносини. Наприклад, чи є один вузол безпосереднім нащадком іншого, чи належать вузли до одного блоку коду. Це дозволяє не просто знаходити ідентичні фрагменти коду, але й виявляти схожі патерни з невеликими відмінностями.

Застосування цієї методики дуже ефективне для виявлення кодових клонів, де фрагменти коду можуть бути не просто копіями, а мати невеликі модифікації, такі як змінені імена змінних або легкі зміни у логічних умовах. Виявлення таких клонів критично для рефакторингу, адже вони можуть вказувати на потенційні точки для оптимізації або виправлення помилок.

Завдяки високій точності визначення схожості AST-методи допомагають підвищити ефективність рефакторингу, сприяючи поліпшенню якості коду та зниженню накопичення помилок і недоліків у програмному продукті. Це особливо важливо у великих проєктах, де повторення коду може призвести до збільшення витрат на його підтримку та розвиток.

Методи на основі токенів [1, 10] аналізують код на рівні найменших синтаксичних одиниць – токенів. Токени представляють собою елементи коду, такі як ключові слова, оператори, ідентифікатори та літерали. Порівняння на основі токенів дозволяє виявляти клоновані фрагменти з різною структурою, але схожим семантичним змістом. Це дозволяє виявляти більш гнучкі варіанти клонів, які можуть бути змінені лише частково.

Функція для обчислення кількості клонованих операторів (Number of Cloned Statements, NoCS):

$$\text{NoCS}(\text{stmtssi}) = \sum_{j=1}^{|\text{stmtssi}|} \sum_{r=j}^{|\text{stmtssi}|} \text{Check}(\text{stmtssi}[j],\text{stmtssi}[r])$$

У формулі застосовані наступні складові:

- NoCS(stmtssi): абревіатура для "Number of Clone Statements", що означає кількість клонованих заяв у вказаному сегменті коду. Функція NoCS обраховує загальну кількість подібних заяв між усіма можливими парами заяв у списку stmtssi.
- stmtssi: список або масив заяв у коді, що розглядається. Кожен елемент цього списку представляє окрему заяву в коді.
- j та r: індекси, що використовуються для ітерації по списку заяв stmtssi. Індекс j проходить від першого до останнього елемента списку, тоді як індекс r починається з j і також іде до кінця списку. Це забезпечує порівняння кожної заяви з іншими заявами, що слідують за нею в списку, уникнення повторного порівняння та порівняння заяви з самою собою.
- Check(stmtssi[j], stmtssi[r]): функція, що визначає чи є дві заяви, вказані як аргументи, подібними або ідентичними. Якщо заяви вважаються схожими або ідентичними, ця функція повертає 1; в іншому випадку вона повертає 0.

Методи машинного навчання [1, 6] використовують алгоритми для ідентифікації складних шаблонів та схожостей у коді. Ці методи враховують семантичні та контекстні фактори, що дозволяє виявляти клоновані фрагменти з високою точністю. Машинне навчання застосовується для класифікації та кластеризації фрагментів коду, що дозволяє автоматизувати процес виявлення клонів. Ці методи включають використання нейронних мереж, дерев рішень та інших моделей для аналізу коду, а саме:

- нейронні мережі використовуються для виявлення складних шаблонів у коді, які можуть бути важко виявленими традиційними методами. Глибокі нейронні мережі можуть навчатися на великих наборах даних, що дозволяє їм розпізнавати складніші схожості;
- дерева рішень використовуються для класифікації фрагментів коду на основі різних характеристик, таких як синтаксичні структури або семантичні особливості. Цей метод дозволяє створювати моделі, які можуть точно визначати чи є два фрагменти коду клонами;
- векторне представлення коду може бути отримане за допомогою методів, таких як Word2Vec або Doc2Vec, адаптованих для коду та проаналізованих за допомогою методів машинного навчання;
- токенізація та підготовленість коду для аналізу, що може включати нормалізацію змінних, видалення коментарів і форматування коду;
- навчання моделі на підготовлених даних з використанням обраного алгоритму машинного навчання;
- оцінка точності моделі за допомогою тестових даних;
- автоматизація процесу виявлення клонів у великих проєктах за допомогою використання навченої моделі для виявлення клонів у новому коді.

Гібридні методи [1, 6, 10] поєднують кілька підходів, таких як AST, токени та машинне навчання для досягнення максимальної точності та ефективності виявлення клонів. Ці методи використовують переваги кожного з підходів для забезпечення більш комплексного аналізу коду. Гібридні методи можуть комбінувати різні техніки для подолання обмежень окремих підходів, забезпечуючи більш точні та надійні результати.

Гібридні методи можуть використовувати комбіновані функції схожості.

Нехай $SimAST(T1, T2)$ — функція схожості для AST і $Simtoken(t1, t2)$ — функція схожості для токенів. Загальна функція може виглядати як:

$$Simhybrid(x, y) = \alpha SimAST(T1, T2) + \beta Simtoken(t1, t2),$$

де α і β - вагові коефіцієнти, що визначають вплив кожного методу на загальну схожість.

Техніка використання спеціалізованих інструментів [2, 3, 5, 7] для автоматизації процесу виявлення та рефакторингу кодових клонів дозволяє поєднувати кілька методів, таких як AST, токени та машинне навчання для забезпечення високої точності та ефективності. Інструменти, такі як CloneDR, CCFinder і NiCad забезпечують автоматичний аналіз коду і виявлення клонів, а також пропонують рефакторингові рішення.

Інструментальна підтримка для рефакторингу кодових клонів включає використання спеціалізованих інструментів для автоматизації процесу виявлення та рефакторингу кодових клонів. Ці інструменти можуть поєднувати кілька методів, таких як методи на основі абстрактного синтаксичного дерева (AST), методи на основі токенів, та підходи, що використовують машинне навчання для забезпечення високої точності та ефективності.

Інструменти для рефакторингу кодових клонів не лише виявляють дублікати коду, але й пропонують способи для їх автоматичного або напівавтоматичного усунення. Деякі з популярних інструментів включають:

- CloneDR — інструмент для виявлення клонів на основі AST, підтримує кілька мов програмування і дозволяє автоматично виявляти та рефакторити клони;
- CCFinder — інструмент, що використовує методи порівняння токенів для виявлення клонів, відомий своєю швидкістю та ефективністю;
- NiCad — інструмент, який використовує гібридний підхід для виявлення клонів, поєднуючи текстові та структурні методи.

Метод компонентного аналізу [1] спрямований на виявлення та рефакторинг кодових клонів на рівні компонентів або модулів, а не окремих функцій чи блоків коду. Це дозволяє підвищити масштабованість та ефективність рефакторингу у великих програмних системах. Компонентний аналіз дозволяє зменшити кількість дублікатів коду на рівні модулів і підвищити якість архітектури програмного забезпечення.

Контекстний аналіз [1, 4] враховує контекст, у якому використовується кодовий фрагмент для виявлення і рефакторингу клонів. Це дозволяє більш точно визначати клони, які мають однакову функціональність, але різні імплементації залежно від контексту використання. Контекстний аналіз може включати аналіз викликів функцій, змінних середовища та інших контекстуальних факторів, які впливають на поведінку коду.

Семантичний аналіз [1] враховує семантику коду для виявлення клонів. Це означає, що аналізуються не лише синтаксичні структури, але й значення та поведінка коду. Це дозволяє виявляти клоновані фрагменти, які виконують однакові операції, але мають різну синтаксичну реалізацію. Семантичний аналіз може включати перевірку типів, значень змінних і поведінки функцій для забезпечення більш точного виявлення клонів.

Оцінка ефективності методів. Розглянемо точність, продуктивність, складність реалізації та вплив на якість коду для кожного з методів у вигляді порівняльної таблиці.

Таблиця 1
Порівняльний аналіз методів рефакторингу кодових клонів

Назва методу	Точність	Швидкість	Складність реалізації	Вплив на якість коду	Особливості
Текстові методи	Середня, зазвичай	Висока	Низька	Обмежений	Не виявляють складні схожості
Регулярні вирази	Середня,	Висока	Середня	Обмежений	Не виявляють складні схожості
Методи на основі дерева синтаксичного аналізу (AST)	Висока, зазвичай перевищує	Середня	Висока	Значний	Вимагають знань про синтаксичний аналіз і структуру
Методи на основі токенів	Висока, досягає	Середня	Середня	Значний	Вимагають додаткової обробки для токенізації коду

Машинне навчання	Висока, більше	Низька	Висока	Дуже значний	Потребують спеціальних знань і великих наборів даних для навчання моделей
Гібридні методи	Висока, більше	Середня	Висока	Дуже значний	Вимагають інтеграції кількох методів, що може бути складним. Залежить від обраних методів.
Інструментальна підтримка для рефакторингу	Висока	Середня	Висока	Значний	Залежить від конкретного інструмента і його налаштувань
Компонентний аналіз	Висока, досягає	Середня	Висока	Значний	Потребує аналізу компонентів. Вимагає знань про архітектуру програмного забезпечення.
Контекстний аналіз	Висока, досягає	Середня	Висока	Значний	Вимагає аналізу контекстуальних факторів
Семантичний аналіз	Висока, більше	Низька	Висока	Дуже значний	Вимагає глибоких знань про семантику коду і відповідні інструменти

На основі аналізу різних методів виявлення кодових клонів наведених у табл. 1 можна зробити висновок, що методи на основі AST, токенів та машинного навчання виявляються найбільш перспективними через їх високу точність та здатність розпізнавати складні схожості у коді. Методи на основі AST та токенів забезпечують високу точність виявлення структурних та синтаксичних клонів, тоді як методи машинного навчання розширюють можливості аналізу за рахунок урахування семантичних та контекстуальних факторів. Незважаючи на високу складність та ресурсомісткість реалізації, ці методи пропонують значні переваги для підвищення якості коду.

Ідентифікація недоліків та викликів. Розглянемо основні проблеми та обмеження існуючих методів та інструментів.

1. Точність виявлення клонів

Методи, такі як текстові методи та регулярні вирази не враховують структурні та семантичні аспекти коду, що призводить до високої частки помилкових позитивних або негативних результатів. AST та токен-методи, хоча і більш точні, можуть не виявляти всі типи клонів, особливо якщо вони мають значні семантичні відмінності. Виявлено, що текстові методи досягають точності в межах 70-80%, тоді як методи на основі AST перевищують 90% точності. Машинне навчання може досягати більше 95% точності, але потребує великих обчислювальних ресурсів.

2. Продуктивність

Методи, що базуються на машинному навчанні та семантичному аналізі потребують значних обчислювальних ресурсів і часу на обробку, що робить їх менш придатними для великих кодових баз або інтеграції в реальному часі. Згідно з аналізом продуктивність текстових мето-

дів та регулярних виразів є високою, тоді як AST та токен-методи потребують більше ресурсів. Методи машинного навчання, хоча і дуже точні, є ресурсомісткими.

3. Складність реалізації

Сучасні методи, особливо ті, що базуються на машинному навчанні та семантичному аналізі, вимагають високого рівня експертизи та спеціалізованих знань для реалізації. Гібридні методи також можуть бути складними через необхідність інтеграції кількох різних підходів. Інструменти для автоматизації рефакторингу, такі як CloneDR, CCFinder, та NiCad, потребують значних зусиль для налаштування та інтеграції у процес розробки.

4. Вплив на якість коду

Методи можуть виявляти клони, але не завжди пропонують оптимальні рішення для рефакторингу. Це може призвести до рефакторингів, які покращують код лише частково або навіть створюють нові проблеми. Виявлено, що інструментальна підтримка може бути недостатньо гнучкою або налаштованою для специфічних потреб проєкту, що знижує її ефективність.

Пріоритетні напрямки досліджень

Під час аналізу методів для виявлення та рефакторингу кодових клонів ми зосередили увагу на кількох ключових параметрах: точності, продуктивності, складності реалізації та впливі на якість коду. Кожен метод був оцінений з цих аспектів, що дало можливість зрозуміти їхні переваги та недоліки у контексті різних сценаріїв використання. В результаті цього аналізу можна віддати пріоритет наступним методам:

- Методи на основі AST демонструють високу точність у виявленні структурних клонів, часто перевищуючи 90%. Вони ефективно ідентифікують схожості, які відображають логічну структуру коду, дозволяючи розрізнити клони, що мають суттєві синтаксичні відмінності. Цей метод є критично важливим для виявлення складних структурних патернів, які інші методи можуть пропустити.
- Токенізація забезпечує високу точність виявлення клонів на рівні синтаксичних одиниць, з показниками точності, що досягають 90-95%. Цей метод швидко обробляє великі обсяги коду і ефективний у виявленні текстових схожостей, що робить його ідеальним для первинного сканування кодової бази на наявність повторень.
- Машинне навчання дозволяє досягнути дуже високої точності, що може перевищувати 95%, особливо коли воно враховує семантичні та контекстуальні аспекти коду. Цей метод є найбільш комплексним і витратним з точки зору обчислювальних ресурсів та даних для навчання, але його можливість виявляти глибокі зв'язки та невидимі на перший погляд взаємозв'язки робить його незамінним для всебічного аналізу коду.

Задля максимізації ефективності існуючих методів і рефакторингу кодових клонів пропонується комбінування методів на основі AST, токенів і машинного навчання. Це поєднання дозволяє об'єднати сильні сторони кожного методу, мінімізувати їхні слабкі сторони та забезпечити більш глибокий і всебічний аналіз коду.

Використання AST дозволяє детально аналізувати структурну схожість коду, виявляючи клони, які можуть мати відмінності на рівні синтаксису, але схожі за своєю логічною структурою. Методи на основі токенів підвищують здатність виявляти схожості на детальному рівні, аналізуючи код на рівні найменших синтаксичних одиниць. Важливим доповненням є машинне навчання, яке розширює можливості аналізу за рахунок інтеграції семантичних та контекстуальних відомостей. Застосування машинного навчання дозволяє системі враховувати не лише текстові дані, а й глибші змістовні зв'язки коду. Це означає, що система може розуміти контекст, у якому використовується код, включаючи залежності між функціями та модулями, що веде до більш точного і ефективного рефакторингу.

Комбінація цих методів може бути реалізована через послідовне застосування різних технік аналізу під час одного проходу аналізу коду. Наприклад, можна спочатку застосувати AST для визначення структурної схожості блоків коду, після чого застосувати аналіз на рівні токенів для додаткового виявлення варіацій у цих блоках. Нарешті, машинне навчання може викори-

стовуватися для виявлення більш тонких семантичних схожостей або відмінностей, які не були виявлені попередніми методами.

Інтеграція цих технологій дозволяє виявляти не тільки текстові і структурні схожості, але й глибші семантичні зв'язки, що важливо для точного рефакторингу і оптимізації. Включення контекстуальних даних, таких як залежності функцій та онтологічні моделі збільшує точність і допомагає адаптувати аналіз до конкретних вимог проєкту, покращуючи якість і стабільність програмного продукту.

Ця інтегрована система забезпечить комплексне рішення, що може адаптуватися до різних стилів кодування і вимог проєкту, зменшить помилки та оптимізує процес розробки. Також вона може автоматично адаптуватись до нових умов і змін у проєкті, використовуючи здобуті дані для навчання та вдосконалення своїх алгоритмів.

Висновки

Процес автоматизації рефакторингу кодових клонів дозволяє зменшити дублювання коду, покращити його якість та зробити обслуговування проєктів більш ефективним. У результаті впровадження автоматизованих засобів для виявлення та виправлення клонів коду можна суттєво зекономити час та зусилля розробників. Шляхи автоматизації процесу рефакторингу кодових клонів включають в себе використання спеціалізованих інструментів та підходів, які дозволяють автоматично виявляти та усувати проблеми з клонуванням коду. Окрім цього, цей процес сприяє підвищенню надійності програмного забезпечення, зменшенню ризиків помилок та прискоренню процесу обслуговування проєктів.

Список літератури

1. Darshan Radadiya, Roopini Jayakumar, Saranya Haridass, "A Systematic Analysis of Code Clone Refactoring Techniques: A Comparative Review" DOI:10.13140/RG.2.2.33628.87685, March 2023.

2. M. Mondal, C. K. Roy, and K. A. Schneider, "Automatic Identification of Important Clones for Refactoring and Tracking," in 2014 14th IEEE International Working Conference on Source Code Analysis and Manipulation.

3. Z. Chen, M. Mohanavilasam, Y.-W. Kwon, and M. Song, "Tool Support for Managing Clone Refactorings to Facilitate Code Review in Evolving Software," in 2017 IEEE 41st Annual Computer Software and Applications Conference.

4. Y. Lin, Z. Xing, X. Peng, Y. Liu, J. Sun, W. Zhao, and J. Dong, "Clonepedia: Summarizing Code Clones by Common Syntactic Context for Software Maintenance," in 2014 IEEE International Conference on Software Maintenance and Evolution.

5. T. Hatano and A. Matsuo, "Removing Code Clones from Industrial Systems Using Compiler Directives," in 2017 IEEE 25th International Conference on Program Comprehension (ICPC).

6. Jiang, J., Liu, H., Wu, Z., & Zhang, H. (2017). Scalable and accurate large-scale code clone detection using deep learning. IEEE Transactions on Parallel and Distributed Systems.

7. Li, H., Zhang, W., & Liu, H. (2018). JAnalyzer: A tool for detecting and visualizing code clones in Java programs. Journal of Systems and Software.

8. Hannes Thaller, Rudolf Ramler, Josef Pichler and Alexander Egyed, "Exploring Code Clones in Programmable Logic Controller Software", Institute for Software Systems Engineering, Johannes Kepler University Linz, Austria, arXiv:1706.03934v1 [cs.SE] 13 Jun 2017.

9. Umberto Azadi, "Automation of duplicate code detection and refactoring", Università degli Studi di Milano-Bicocca, April 2020.

10. Jaweria Kanwal, Onaiza Maqbool, Hamid Abdul Basit, Muddassar Azam Sindhu, Katsuro Inoue, "Historical perspective of code clone refactorings in evolving software", Jacopo Soldani, University of Pisa, ITALY, December 1, 2022.

I. Zamrii, O. Kulakov

REVIEW OF AUTOMATED REFACTORING APPROACHES FOR CODE CLONES

The modern development of technologies in the field of software engineering requires constant efforts in optimizing and improving code quality. One of the pressing issues is the presence of code clones, which can lead to errors and complicate the further maintenance of software systems. Automating the refactoring of code clones is of significant interest to researchers and practitioners in this field.

The main tasks of the research cover several key aspects. First and foremost, the research focuses on an in-depth analysis of existing methods and techniques already used for automating clone refactoring. This includes studying the effectiveness of existing approaches and identifying their limitations. An important part is the analysis of algorithms that can improve the accuracy of clone detection and the efficiency of their refactoring. Moreover, the research analyzes the impact of automation on the quality and maintainability of software systems, helping to determine how such technologies can be integrated into real software development projects.

The study aims to identify the limitations of current approaches and propose ways to optimize them, considering the modern technological capabilities and needs of the software market. The research includes a critical review of the literature, the study of reports on the application of tools in practice, and the analysis of the results of using automated systems in real projects. This allows for the evaluation of the effectiveness of various algorithms and methods for detecting and refactoring code clones, as well as their impact on the quality of the final product and the efficiency of development. The conclusions proposed within the study will include recommendations for addressing identified gaps in current methods and suggesting a new approach to the automation of clone refactoring that can provide greater adaptability and efficiency.

Keywords: code clones, automation of refactoring, machine learning, static analysis, code optimization, code quality