http://www.moderntechno.de/index.php/meit/article/view/meit42-02-035
DOI: 10.30890/2567-5273.2025-42-02-035

УДК 004.2

ADAPTIVE STATIC ANALYSIS OF SOFTWARE CODE QUALITY

АДАПТИВНИЙ СТАТИЧНИЙ АНАЛІЗ ЯКОСТІ ПРОГРАМНОГО КОДУ

Blahyi A.I. / Благий А.І.
PhD Student /Аспірант
ORCID: 0009-0000-4172-6471
Taras Shevchenko National University of Kyiv,
Kyiv, 4d Akademika Glushkova Avenue, 02000
Київський національний університет імені Тараса Шевченка,
м. Київ, проспект Академіка Глушкова, 4д, 02000

Анотація. *У статті розглянуто проблеми сучасного статичного аналізу програмного коду, зокрема надмірну кількість хибнопозитивних результатів, складність роботи з мультимовними системами та недостатню контекстну чутливість існуючих інструментів. Запропоновано два напрями вдосконалення: контекстно-орієнтований розподіл правил аналізу за доменами коду та мовно-нейтральна модель залежностей компонентів, що інтегрує дані з різних аналізаторів у єдине представлення. Експериментальні результати показали зменшення хибнопозитивних попереджень на 30–40 % і підвищення узгодженості звітів без втрати здатності виявляти дефекти. Робота демонструє перспективність адаптивних підходів до статичного аналізу та окреслює можливості їх подальшого розвитку.*

Ключові слова: *статичний аналіз; хибнопозитивні результати; мультимовні системи; контекстно-орієнтований аналіз; графи залежностей; мовно-нейтральна модель; контроль якості коду; CI/CD.*

Вступ.

У сучасних програмно-інтенсивних системах якість коду визначає надійність і безпеку критично важливих цифрових сервісів. Зі зростанням масштабів і складності програмних продуктів посилюється потреба в методах раннього виявлення дефектів, зокрема таких, що не потребують виконання програм. До них належить статичний аналіз, який дає змогу виявляти помилки, вразливості та структурні невідповідності вже на ранніх етапах розробки.

Попри широкий розвиток технік статичного аналізу, на практиці його ефективність часто обмежується надмірною кількістю хибнопозитивних результатів, нечутливістю до архітектурного контексту та труднощами роботи з мультимовними кодовими базами. Це знижує практичну корисність інструментів і ускладнює їх інтеграцію в інженерні процеси.

У зв’язку з цим актуальною є потреба у методах, здатних адаптувати аналіз до структурних та організаційних особливостей системи. У даній роботі

зосереджено увагу на вирішенні цієї проблеми через контекстно-орієнтований підхід до зменшення хибнопозитивних результатів і побудову мовно-нейтральної моделі залежностей для мультимовних проєктів.

Основний текст

Сучасне використання статичного аналізу коду

У сучасному інформаційному суспільстві програмне забезпечення виконує критично важливу роль у функціонуванні технічних, соціальних, економічних та безпекових систем. Високі вимоги до його надійності та безпеки спонукають до пошуку ефективних методів їх забезпечення на всіх етапах життєвого циклу розробки. Одним із таких методів є статичний аналіз [1]. На відміну від тестування, яке вимагає повного або часткового виконання програми, статичний аналіз працює на рівні вихідного коду. Це дозволяє виявляти помилки, вразливості, порушення стилю коду, приховані залежності або архітектурні суперечності ще до етапу компіляції чи інтеграції. Метод може застосовуватися до окремих модулів або фрагментів, що робить його цінним для ранньої діагностики та безперервного контролю якості.

Різноманітність технік статичного аналізу охоплює як базову перевірку відповідності синтаксичним правилам, так і глибше дослідження потоків даних, структур управління, семантики, шаблонів і навіть поведінкових моделей. Таке різноманіття створює умови для побудови гнучких систем контролю якості, здатних враховувати специфіку мови, архітектури, масштабу і динаміку змін програмного продукту.

Загальний огляд методів статичного аналізу подано в роботах Stefanović D., Nikolić D., Dakić D., Spasojević I., Ristić S.: Static Code Analysis Tools: A Systematic Literature Review [2], Mweu, B., Ndia, J. Static Analysis Techniques for Secure Software: A Systematic Review (Journal of Cyber Security, 7(1), 2025) [3], та Christakis M., Bird C. – What developers want and need from program analysis (Automated Software Engineering, 2016) [4]. У цих дослідженнях розглянуто практичні аспекти застосування статичного аналізу – від промислових кейсів до факторів, що впливають на довіру розробників до результатів. У цих роботах

значну увагу приділено точності інструментів і проблемі хибнопозитивних результатів, які безпосередньо впливають на сприйняття аналітичних звітів користувачами.

Перевагою сучасного статичного аналізу (СА) є його здатність інтегруватися безпосередньо в процеси розробки, зокрема в безперервної інтеграції та доставки – Continuous Integration / Continuous Delivery (CI/CD), де регулярні автоматизовані перевірки дозволяють виявляти дефекти на ранніх етапах і забезпечують швидкий зворотний зв'язок для розробників [5]. У великих промислових системах, таких як Amazon або Meta, СА є обов'язковою частиною пайплайну – від pull request до продакшен-деплою, хоча ефективність його застосування залежить від розміру проєкту, структури команди та типу системи [6]. Натомість у невеликих або високодинамічних проєктах надмірна жорсткість регламентів може мати зворотний ефект, спричиняючи затримки або спонукання до обходу аналітики.

У відповідь на вищезгадані недоліки, деякі інструменти, зокрема Codacy та DeepSource, пропонують механізми гнучкого налаштування аналізу. При цьому якість оцінюється на основі історичних шаблонів проєкту, а попередження пріоритезуються залежно від частоти та тяжкості порушень. Такі рішення частково зменшують навантаження на розробників, проте не усувають системних обмежень статичного аналізу.

Попри значний розвиток технік, статичний аналіз залишається вразливим до кількох ключових проблем: складності налаштування, надмірної кількості хибнопозитивних результатів та труднощів інтеграції у складні мультимовні проєкти. Саме ці фактори стали підґрунтям для дослідження, метою якого є підвищення адаптивності статичного аналізу до особливостей реальних інженерних середовищ.

Вдосконалення структурного аналізу для складних мультимовних систем

Важливим інструментом статичного аналізу є побудова графів потоку керування (control flow graphs, CFG). Такі графи відображають послідовність

можливих переходів між операторами програми та дозволяють формалізувати логіку виконання на рівні потоку інструкцій [7]. Це створює підґрунтя для виявлення структурних дефектів, таких як зайві розгалуження, «мертвий код» або недосяжні гілки виконання.

Попри свою корисність, класичні CFG мають суттєві обмеження: вони не відображають динамічні властивості виконання, такі як виклики через рефлексію, динамічне завантаження модулів або взаємодію через зовнішні API. Це особливо проблематично для сучасних багаторівневих застосунків із розподіленою логікою. Для часткового подолання цих обмежень розроблено низку розширених структур, зокрема графи залежностей програм (Program Dependence Graphs, PDG) [8], які враховують інформаційні залежності між інструкціями та дозволяють глибше аналізувати взаємозв'язки у програмі.

Водночас навіть PDG не охоплюють повною мірою складність сучасних програмних екосистем, де логіка розподілена між модулями, сервісами та різними мовами програмування. У мультимовних проєктах (наприклад з використанням Java, JavaScript та Python) результати аналізу окремих мов залишаються фрагментованими, а дефекти, що виникають на межі між технологіями, часто залишаються непоміченими. Існуючі комбіновані рішення, у тому числі інструменти графового аналізу, частково інтегрують інформацію з різних джерел, але їхня ефективність обмежується глибиною реалізованих правил і рівнем підтримки конкретних мов.

З огляду на вищезгадані обмеження автором було розроблено мовно-нейтральну модель взаємозв'язків компонентів. Вона ґрунтується не на вихідних файлах чи класах, а на функціональних вузлах системи. Для побудови такої моделі результати аналізу з SonarQube, ArchUnit та спеціальних парсерів об'єднувалися у єдиний граф, який відображав взаємодію між підсистемами незалежно від мови реалізації. Це дозволило виявляти "вузли ризику" – області з високою складністю або великою кількістю залежностей, які було складно помітити при аналізі окремих мов.

Попереднє тестування показало, що запропонована модель допомагає

зменшити дублювання попереджень і виявляти проблеми, що раніше залишалися “невидимими”. Вона вже зараз демонструє потенціал для підвищення точності та узгодженості аналізу у багатомовних кодових базах, але для надійної верифікації ефективності моделі варто провести додаткові дослідження на більших і різноманітніших проєктах.

Контекстно-орієнтований метод зменшення хибнопозитивних результатів

Іншим практичним інструментом статичного аналізу є абстрактна інтерпретація. Це математично обґрунтована методика, що дозволяє будувати апроксимацію поведінки програми без її фактичного виконання [9]. Вона ґрунтується на тому, що реальні значення змінних та станів програми замінюються абстрактними представленнями, які зберігають достатню кількість інформації для виявлення дефектів, але знижують обчислювальну складність аналізу. Цей підхід широко використовується в промислових та академічних інструментах для виявлення типових помилок безпеки (наприклад, звернення до нульових покажчиків, вихід за межі масиву, використання неініціалізованих змінних) та порушень контрактів коду. Відомим прикладом є аналізатор Astrée, орієнтований на критичні системи. А у комерційних рішеннях абстрактна інтерпретація часто виступає складником багаторівневих «гібридних» перевірок.

Попри високу формальну строгість, абстрактна інтерпретація має низку характерних обмежень. Основна проблема полягає у балансі між точністю та продуктивністю: надто грубі абстракції пропускають дефекти, тоді як надто детальні – призводять до вибухового зростання обчислень. Крім того, сама природа абстрагування часто породжує хибнопозитивні результати: аналізатор сигналізує про ризики, що фактично не становлять загрози. Це ускладнює інтерпретацію висновків і поступово знижує довіру до інструмента. Існуючі методи зменшення хибнопозитивних результатів мають низку обмежень. Тонке налаштування правил у класичних інструментах, таких як SonarQube, потребує значних трудових витрат, тоді як підходи на основі машинного навчання залежать від великих маркованих дата-сетів, що звужує можливості їх

застосування.

Урахування цих недоліків мотивувало розроблення контекстно-орієнтованого підходу, у межах якого кодова база поділяється на домени – умовні зони, як-от зовнішні API, обробники подій, сервісні класи чи конфігураційні файли. Для кожного домену визначається окремий профіль перевірок: наприклад, у конфігураційних файлах можуть бути допустимі стилістичні відхилення, що в інших модулях вважаються помилками. Компоненти позначаються тегами, які під час Continuous Integration, безперервної інтеграції (CI) автоматично визначають набір правил для аналізу. Такий механізм дозволяє адаптувати перевірки до контексту без зміни глобальних налаштувань.

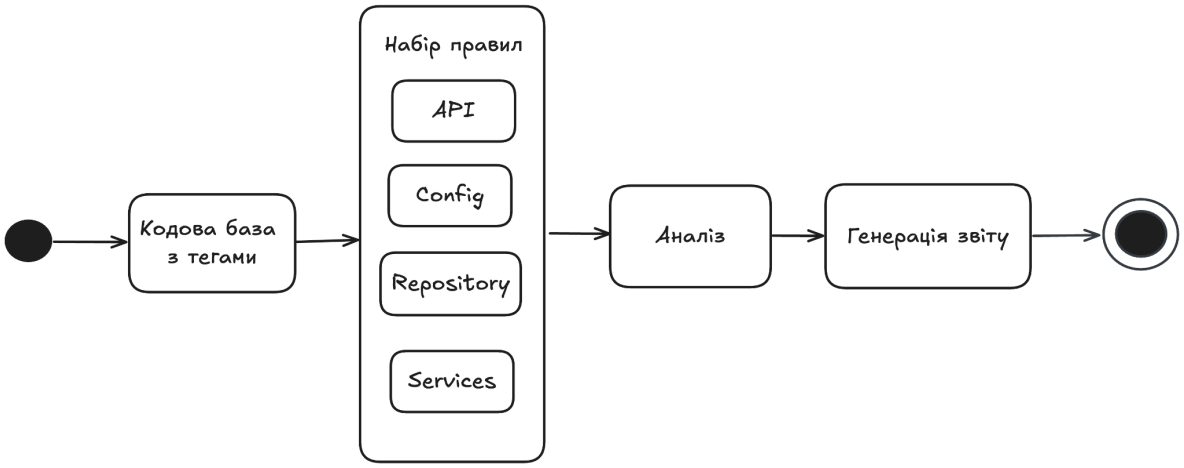

Рисунок 1 – Схема контекстно-орієнтованого підходу

Основні переваги даного рішення полягають у зменшенні “шуму” у звітах, збереженні повного набору важливих правил і простоті масштабування. Додавання нових доменів або зміна профілів не впливають на решту системи, а інтеграція через CI-теги дає змогу впроваджувати зміни поступово, без зупинки процесів розробки.

У тестовому використанні запропоноване рішення дозволило зменшити кількість хибнопозитивних попереджень на 30-40 %, причому без втрати здатності знаходити реальні дефекти. Одержані результати показали, що звіти стали компактнішими та містили менше другорядних сповіщень, завдяки чому

легше було знаходити дійсно важливі проблеми. Це свідчить про потенціал контекстного розподілу правил як способу підвищення користі від статичного аналізу, для повноцінних висновків потрібна перевірка на більшому обсязі коду та в умовах командної розробки.

Висновки

Основними результатами проведеного дослідження є розробка та апробація нових методик статичного аналізу. Було реалізовано:

- **Контекстно-орієнтований поділ кодової бази на домени** з окремими профілями правил; компоненти маркувалися тегами, що на етапі CI визначали релевантний набір перевірок.
- **Мовно-нейтральну модель взаємозв'язків компонентів**, побудовану шляхом інтеграції результатів SonarQube, ArchUnit та спеціальних парсерів у єдиний граф функціональних вузлів.
- **Проведено експериментальну перевірку методик** на практичних прикладах.

Отримані результати підтвердили ефективність запропонованих рішень: контекстний поділ значно зменшив кількість хибнопозитивних попереджень, а мовно-нейтральна модель дала змогу усунути дублювання сповіщень і виявити залежності, непомітні під час аналізу окремих мов. У сукупності це забезпечило компактніші та релевантніші звіти, а також засвідчило здатність методики масштабуватися без негативного впливу на інженерні процеси.

Запропонований підхід створює передумови для розробки нового класу контекстно-чутливих інструментів статичного аналізу, що враховують не лише синтаксис, а й архітектурний контекст, історію змін і стабільність модулів. Подальші дослідження спрямовуватимуться на автоматизацію профілювання, узагальнення моделей залежностей та розроблення інтегрованих систем аналізу, здатних поєднувати формальні методи з емпіричними спостереженнями.

Література:

1. Rival, X. and Yi, K. (2020). *Introduction to Static Analysis: An Abstract*

Interpretation Perspective. Mit Press. ISBN: 9780262043410

2. Stefanovic, D., Nikolic, D., Dakic, D., Spasojevic, I. and Ristic, S. (2020). Static Code Analysis Tools: A Systematic Literature Review. *DAAAM Proceedings*, (31), pp.0565–0573. doi:https://doi.org/10.2507/31st.daaam.proceedings.078.

3. Mweu, B. and Ndia, J. (2025). Static Analysis Techniques for Secure Software: A Systematic Review. *Journal of Cyber Security*, 7(1), pp.417–437. doi:https://doi.org/10.32604/jcs.2025.071765.

4. Christakis, M. and Bird, C. (2016). What developers want and need from program analysis: an empirical study. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016*. doi:https://doi.org/10.1145/2970276.2970347.

5. Gousios, G., Pinzger, M. and Deursen, A. van (2014). An exploratory study of the pull-based software development model. *Proceedings of the 36th International Conference on Software Engineering*. [online] doi:https://doi.org/10.1145/2568225.2568260.

6. Ohmann, P., Brown, D.B., Neelakandan, N., Linderoth, J. and Liblit, B. (2016). Optimizing customized program coverage. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp.27–38. doi:https://doi.org/10.1145/2970276.2970351.

7. Autili, M., Malavolta, I., Perucci, A., Scoccia, G.L. and Verdecchia, R. (2021). Software engineering techniques for statically analyzing mobile apps: research trends, characteristics, and potential for industrial adoption. *Journal of Internet Services and Applications*, 12(1). doi:https://doi.org/10.1186/s13174-021-00134-x.

8. Kiss, A. and Kiss, A. (2021). The Compressed Program Dependence Graph (13th Joint Conference on Mathematics and Computer Science (the 13th MaCS), on October 1-3, 2020). doi:https://doi.org/10.13140/rg.2.2.29229.41449.

9. Erhard, J., Saan, S., Tilscher, S., Schwarz, M., Holter, K., Vojdani, V. and Seidl, H. (2024). Interactive abstract interpretation: reanalyzing multithreaded C programs for cheap. *International Journal on Software Tools for Technology Transfer*. doi:https://doi.org/10.1007/s10009-024-00768-9.

Abstract. *The article examines key challenges of modern static code analysis, including the excessive number of false-positive results, the difficulty of working with multilingual systems, and the insufficient contextual sensitivity of existing tools. Two improvement directions are proposed: a context-oriented distribution of analysis rules across code domains, and a language-neutral dependency model that integrates data from different analyzers into a unified representation. Experimental results demonstrated a 30–40% reduction in false-positive warnings and improved consistency of reports without compromising defect-detection capabilities. The work shows the promise of adaptive approaches to static analysis and outlines opportunities for their further development.*

Key words: *static analysis, false positives, multilingual systems, context-oriented analysis, dependency graphs, language-neutral model, code quality assurance, CI/CD.*

Науковий керівник: к.ф.-м.н., доц. Шевченко В.П

Статтю надіслано: 16.12.2025 р.