ІНФОРМАТИКА, ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА АВТОМАТИЗАЦІЯ

УДК 004.41
DOI https://doi.org/10.32782/2663-5941/2026.3.1/01

Абакумов Р.В.
https://orcid.org/0009-0004-2743-3815
Dispatch Health Management, USA

АВТОМАТИЗОВАНЕ ВИЯВЛЕННЯ АРХІТЕКТУРНИХ ПОРУШЕНЬ У JAVA-ПРОЄКТАХ ЗАСОБАМИ СТАТИЧНОГО АНАЛІЗУ

Актуальність дослідження зумовлена ускладненням сучасних програмних систем (далі – ПС), зростанням обсягів коду та тривалим життєвим циклом продуктів, що значно ускладнює забезпечення архітектурної узгодженості програмного забезпечення (далі – ПЗ). У масштабних Java-проєктах розвиток системи часто супроводжується порушеннями архітектурних правил: з'являються небажані залежності між модулями, циклічні зв'язки та деградація шарової структури. Це спричиняє накопичення технічного боргу, ускладнює супровід та ускладнює супровід ПЗ. За таких умов важливим є використання засобів статичного аналізу, які уможливлюють автоматизацію контролю архітектурних обмежень і своєчасне виявлення структурні відхилення. Метою статті є підвищення архітектурної узгодженості ПС шляхом використання засобів статичного аналізу коду для автоматизованого виявлення архітектурних порушень у Java-проєктах. У дослідженні застосовано методи системного аналізу архітектури ПС, порівняльний аналіз сучасних інструментів статичного аналізу коду, а також узагальнення практик їхнього застосування в процесі розроблення ПЗ. Досліджено особливості контролю архітектурних залежностей у Java-проєктах та можливості застосування таких інструментів, як SonarQube, ArchUnit, Structure101 та jQAssistant для автоматизованого виявлення порушень архітектурних правил. Проаналізовано сучасні підходи до забезпечення архітектурної цілісності ПС. Встановлено, що основними причинами архітектурної деградації є порушення правил залежностей між модулями, виникнення циклічних зв'язків та надмірна зв'язаність компонентів системи. Доведено, що застосування інструментів статичного аналізу дає змогу автоматично контролювати структуру коду, своєчасно виявляти архітектурні порушення та підтримувати модульність систем. Порівняльний аналіз інструментів підтвердив, що їхня інтеграція в процеси CI/CD дає змогу ефективно контролювати архітектурні обмеження під час розроблення. У висновках зазначено, що застосування інструментів статичного аналізу є ефективним механізмом автоматизованого контролю архітектурної структури ПС. Доведено, що інтеграція таких засобів у процеси розроблення ПЗ дає змогу своєчасно виявляти структурні відхилення, зменшувати технічний борг та підтримувати стабільну еволюцію архітектури Java-проєктів. Перспективи подальших досліджень пов'язані з удосконаленням методів автоматизованого аналізу архітектури програмного продукту та застосуванням інтелектуальних алгоритмів для прогнозування архітектурних порушень.

Ключові слова: *структура програмного коду, залежності між модулями, технічний борг, контроль архітектури, програмна інженерія, якість програмного забезпечення.*

Постановка проблеми. Сучасні ПС характеризуються високою складністю, значним обсягом коду та тривалою еволюцією програмних продуктів, що висуває підвищені вимоги до архітектурної узгодженості ПЗ. Архітектура ПС визначає структуру взаємодії її компонентів, регламентує залежності між модулями й забезпечує підтримуваність, масштабованість і надійність програмного продукту. Проте в процесі розвитку програмних проєктів, особливо у великих Java-системах із колективним розробленням, нерідко виникають відхилення від початково визначених архітектур-

них принципів. Такі порушення проявляються як небажані залежності між пакетами й класами, порушення шарової структури або формування циклічних зв'язків, що поступово ускладнює супровід системи й призводить до накопичення технічного боргу.

Виявлення таких архітектурних відхилень традиційними методами контролю якості ПЗ є складним, оскільки вони не пов'язані безпосередньо з функціональними помилками, а стосуються структурної організації коду. У цих умовах особливого значення набуває застосування статичного аналізу, що дає змогу аналізувати структуру ПС без виконання коду та автоматично перевіряти відповідність реалізації заданим архітектурним правилам. З огляду на це, розроблення та застосування підходів до автоматизованого виявлення архітектурних порушень у Java-проєктах є важливим науково-практичним завданням, спрямованим на підвищення якості ПЗ та ефективності його розроблення й супроводу.

Аналіз останніх досліджень і публікацій. Огляд сучасних досліджень дає змогу виокремити кілька груп наукових підходів, які формують методологічну основу дослідження. Значну частину робіт присвячено виявленню архітектурних антипатернів і структурних відхилень, які ідентифікуються за допомогою аналізу вихідного коду ПС. Зокрема, К. Трубіані та співавтори (C. Trubiani et al.) розглядають підхід до автоматизованого виявлення антипатернів продуктивності в Java-застосунках, що уможливлює визначення неефективних архітектурних рішень та потенційних структурних дефектів ПС [1]. Можливості автоматизованого виявлення та візуалізації архітектурних порушень у середовищах безперервної інтеграції аналізують С. Мендоса та колеги (C. Mendoza et al.), що сприяє забезпеченню раннього контролю відповідності коду встановленим архітектурним обмеженням [2]. Ефективність інструментів статичного аналізу у виявленні порушень, які традиційно ідентифікуються під час ручного перегляду коду досліджують С. Мехрпур (S. Mehrpour) та Т. Д. ЛаТоза (T. D. LaToza). Це підтверджує доцільність автоматизації контролю архітектурної дисципліни ПС [3]. Комплексне оцінювання інструментів статичного аналізу для пошуку вразливостей у Java-коді здійснюють М. Алкарадагі (M. Alqaradaghi) та Т. Козшик (T. Kozsik), демонструючи їхній потенціал для виявлення структурних і логічних дефектів ПЗ [4]. Автоматизований підхід до виявлення патернів проєктування в програмних проєктах пропонують К. К. Гупта та співавтори (K. K. Gupta et al.), що можна застосовувати для аналізу відповідності програмного коду типовим архітектурним рішенням і виявлення відхилень від них [5].

Інша група досліджень зосереджена на застосуванні статичного аналізу для виявлення вразливостей ПЗ та неправильного використання програмних бібліотек, що часто є наслідком порушення архітектурних принципів побудови ПС. Застосування software composition analysis для виявлення вразливостей у Java-проєктах шляхом аналізу залежностей сторонніх компонентів ПЗ досліджує колектив авторів на чолі з Л. Чжао (L. Zhao et al.) [6]. Проблему автоматичного виявлення неправильного використання криптографічних програмних інтерфейсів застосунків (application programming interface, API) у Java-програмах аналізують Ю. Чжан та співавтори (Y. Zhang et al.), демонструючи можливості статичного аналізу для контролю безпечного використання програмних бібліотек [7]. Емпіричне оцінювання інструментів статичного аналізу для виявлення дефектів коду на основі тестових наборів Juliet Test Suites, що підтверджує їхню ефективність у процесі автоматизованого аналізу ПЗ, здійснюють Р. Аманква та колеги (R. Amankwah et al.) [8]. Взаємозв'язок між архітектурними smells та попередженнями інструментів статичного аналізу досліджує колектив науковців під керівництвом М. Еспозіто (M. Esposito et al.). Автори доводять, що сигнали таких інструментів є індикаторами глибших архітектурних проблем ПС [9].

Окремий пласт досліджень присвячений розробленню методів автоматизованого виявлення та усунення дефектів коду на основі практичноорієнтованого аналізу та інтелектуальних алгоритмів. Новий підхід до автоматичного виявлення та виправлення вразливостей у Java-коді на основі прикладноорієнтованих програмних прикладів пропонують Ю. Чжан та співавтори (Y. Zhang et al.), що розширює можливості статичного аналізу від простого детектування до підтримки процесів корекції дефектів [10]. Метод виявлення ін'єкційних вразливостей у Java web-застосунках із застосуванням міжпроцедурного аналізу та методів глибинного навчання досліджують Б. Чжан та колеги (B. Zhang et al.), демонструючи перспективність поєднання статичного аналізу та інтелектуальних технологій програмного коду [11]. Підхід Darcy для автоматизованого виявлення й усунення архітектурних неузгодженостей у Java-проєктах пропонує колектив авторів на чолі з Н. Горбані (N. Ghorbani et al.), що дає

змогу підтримувати відповідність реалізації ПС визначеній архітектурній моделі [12]. Drift-аналіз на основі архітектурних подань, який сприяє ідентифікації відхилень між фактичними залежностями коду та запроєктованою архітектурною структурою ПС, досліджують Б. Узун (B. Uzun) та Б. Текінердоган (B. Tekinerdogan) [13].

Важливе значення для розвитку цієї проблеми мають також дослідження, присвячені методологічним основам застосування статичного аналізу коду для виявлення складних структурних і поведінкових дефектів ПС. Можливості використання статичного тестування безпеки застосунків для аналізу коду та виявлення потенційних дефектів ПЗ аналізують Л. Ма та співавтори (L. Ma et al.), демонструючи універсальність підходів статичного аналізу [14]. Методи виявлення помилок конкурентності в багатопотокових застосунках на основі статичного аналізу сирцевого коду, що підтверджує здатність інструментів аналізу виявляти складні міжкомпонентні залежності ПС, досліджують Д. Гебас (D. Giebas) та Р. Войшчик (R. Wojszczyk) [15].

Попри значний розвиток досліджень у сфері програмної інженерії, низка аспектів забезпечення архітектурної узгодженості ПС залишається недостатньо вивченою. Зокрема, обмежено узагальнено практичні можливості сучасних інструментів статичного аналізу коду для автоматизованого виявлення архітектурних порушень у Java-проєктах, а також недостатньо висвітлено проблеми контролю архітектурних залежностей у процесі еволюції ПС. Це зумовлює потребу в системному аналізі архітектурних обмежень, інструментів автоматизованого контролю та узагальненні практичних підходів до їхнього застосування. Запропоноване дослідження спрямоване на часткове усунення зазначених прогалин шляхом аналізу сучасних підходів до забезпечення архітектурної цілісності ПС, дослідження типових архітектурних залежностей у Java-проєктах та порівняльного аналізу інструментів статичного аналізу програмного коду. На основі отриманих результатів обґрунтовано рекомендації щодо автоматизованого виявлення архітектурних порушень, що сприяє підвищенню якості та підтримуваності ПЗ.

Постановка завдання. Метою статті є розроблення підходів до підвищення архітектурної цілісності ПС шляхом застосування засобів статичного аналізу для автоматизованого виявлення архітектурних порушень у Java-проєктах.

Для досягнення поставленої мети визначено такі завдання:

1) проаналізувати сучасні підходи до забезпечення архітектурної цілісності ПС та типові архітектурні залежності в Java-проєктах;

2) оцінити можливості інструментів статичного аналізу для виявлення порушень архітектурних правил у Java-проєктах;

3) виявити проблеми контролю архітектурної узгодженості ПЗ та обґрунтувати рекомендації щодо автоматизованого виявлення архітектурних порушень.

Виклад основного матеріалу. У сучасній програмній інженерії архітектурна цілісність систем розглядається як важлива передумова забезпечення їхньої підтримуваності, масштабованості та стабільності функціонування. У масштабних ПС, зокрема тих, що створюються мовою програмування Java, порушення архітектурних обмежень часто виникають у процесі еволюції коду, розширення функціональності та колективного розроблення. У дослідженнях підкреслюється, що такі відхилення призводять до появи небажаних залежностей між модулями, порушення шарової структури системи та поступового накопичення технічного боргу. З огляду на це значної уваги набуває застосування методів статичного аналізу коду (static code analysis, SCA), які дають змогу автоматично досліджувати структуру ПЗ без його виконання та виявляти архітектурні порушення на ранніх етапах розроблення (табл. 1).

Під час розроблення ПЗ зазначені підходи застосовуються комплексно та інтегруються в процеси керування якістю програмного коду. Архітектурні шаблони формують базову логіку структурування системи, а формалізовані правила залежностей визначають допустимі межі взаємодії між її компонентами. Наприклад, у типовій багатошаровій Java-архітектурі встановлюється правило, згідно з яким сервісний шар використовує рівень доступу до даних, однак він не має безпосередньо взаємодіяти з рівнем представлення. Порушення таких обмежень призводить до ускладнення структури та зниження її підтримуваності. На практиці контроль дотримання архітектурних обмежень дедалі частіше реалізується за допомогою інструментів статичного аналізу, які здатні автоматично перевіряти тисячі залежностей між класами та пакетами ПС. У масштабних Java-проєктах або на вебплатформах ці інструменти інтегрують CI, де аналіз коду виконується автоматично під час кожної збірки [1, с. 2877]. У разі виявлення небажаних циклічних залежностей або порушення архітектурних правил система формує відповідні повідомлення для розробників

або автоматично блокує інтеграцію змін до основної гілки проєкту. Це забезпечує підтримання архітектурної дисципліни в умовах інтенсивного розроблення, своєчасне виявлення структурних аномалій та запобігання накопиченню технічного боргу, що є особливо важливим для довготривалої еволюції складних ПС.

Архітектурні залежності в Java-проєктах визначають правила взаємодії між пакетами, модулями та класами ПС і є важливим механізмом підтримання її структурної узгодженості. У сучасній програмній інженерії такі залежності формалізуються як архітектурні обмеження, що регламентують допустимі напрями використання компонентів системи. Їхнє дотримання дає змогу зменшити зв'язаність між модулями, підвищити модульність коду та забезпечити стабільну еволюцію програмного продукту (табл. 2).

Так, у Java-проєктах, що реалізуються на основі багатошарової архітектури, правила залежностей дають змогу підтримувати чітку логіку взаємодії між компонентами системи. Наприклад, рівень представлення взаємодіє з сервісним шаром, де зосереджена бізнес-логіка, а доступ до бази даних забезпечується через окремий рівень доступу до даних [7, с. 293]. Порушення цього принципу, зокрема прямий виклик бази даних із рівня представлення, призводить до зростання зв'язаності між модулями та ускладнює подальшу модифікацію програмного коду.

У масштабних Java-системах, що містять сотні пакетів і тисячі класів, навіть окремі порушення залежностей поступово формують складну мережу взаємозв'язків, яка знижує модульність ПЗ. Зокрема, виникнення циклічних залежностей між пакетами ускладнює тестування, повторне використання компонентів та рефакторинг системи. З огляду на це в сучасних умовах контроль дотримання архітектурних обмежень дедалі частіше здійснюють за допомогою інструментів SCA, які автоматично перевіряють структуру залежностей під час збірки проєкту [3]. Це дає змогу своєчасно виявляти архітектурні відхилення, підтримувати структурну узгодженість системи та

Таблиця 1

Основні підходи до забезпечення архітектурної цілісності ПС та роль статичного аналізу

Підхід	Основний зміст	Практичне значення
Архітектурне проєктування на основі шаблонів	Використання усталених архітектурних моделей, зокрема модель – подання – контролер (model – view – controller, MVC) та мікросервісна архітектура (microservices architecture, MSA)	Забезпечує структуровану організацію програмного коду та чіткий розподіл функціональних відповідальностей
Формалізація архітектурних правил	Визначення допустимих залежностей між пакетами, модулями або шарами системи	Дає змогу обмежувати небажані залежності між компонентами ПЗ
Статичний аналіз програмного коду	Автоматизований аналіз структури програмного коду без його виконання	Дає змогу виявляти циклічні залежності, порушення архітектурних правил та інші структурні аномалії
Використання архітектурних метрик	Кількісна оцінка складності, зв'язності та рівня залежностей між компонентами системи	Забезпечує об'єктивне оцінювання стану архітектури ПЗ
Інтеграція перевірок у процес безперервної інтеграції та доставлення (continuous integration / continuous delivery, CI/CD)	Автоматичне виконання перевірок архітектури під час збірки програмного продукту	Забезпечує постійний контроль архітектурної структури ПС

Джерело: сформовано автором на основі [1, с. 2877; 2, с. 1826; 3; 7, с. 291; 9; 12, с. 1642]

Таблиця 2

Типові архітектурні залежності та обмеження в Java-проєктах

Тип залежності або обмеження	Характеристика	Значення для структурної якості
Шарова залежність	Взаємодія компонентів системи відповідно до рівнів архітектури	Забезпечує чітке функціональне розмежування модулів
Односпрямованість залежностей	Встановлення допустимого напряму використання пакетів або класів	Запобігає неконтрольованому поширенню залежностей
Відсутність циклічних залежностей	Заборона взаємних залежностей між пакетами або модулями	Підвищує модульність і спрощує модифікацію системи
Обмеження міжмодульної взаємодії	Регламентування використання компонентів одного модуля іншими	Зменшує зв'язаність між частинами системи
Інкапсуляція внутрішніх компонентів	Обмеження доступу до внутрішніх класів і пакетів	Підтримує стабільність архітектури системи

Джерело: сформовано автором на основі [1, с. 2879; 2, с. 1828; 3; 6, с. 963; 7, с. 293; 13]

запобігати накопиченню технічного боргу в процесі розвитку ПЗ.

Для кращого розуміння механізму контролю архітектурних залежностей у Java-проєктах доцільно розглянути узагальнену схему перевірки архітектурних обмежень засобами SCA.

На рис. 1 наведено логіку автоматизованого контролю архітектурних обмежень у сучасних Java-проєктах. На етапі проєктування визначають архітектурну модель системи, яка описує структуру модулів, шарів і допустимі залежності між ними. Ці правила фіксуються як формалізовані обмеження, які перевіряють інструменти статичного аналізу. Під час аналізу програмного коду система досліджує взаємозв'язки між класами та пакетами й порівнює їх із визначеними архітектурними правилами. У практиці розроблення ПЗ така перевірка часто інтегрується в процес CI/CD. Наприклад, під час автоматичної збірки Java-проєкту система аналізує структуру залежностей і визначає, чи не виникли заборонені зв'язки між модулями. У разі відповідності архітектурним правилам збірка завершується успішно, а з виявленням порушень розробники отримують повідомлення про помилку або система блокує інтеграцію змін в основну гілку проєкту. Такий механізм дає змогу підтримувати архітектурну дисципліну, своєчасно виявляти структурні відхилення та запобігати деградації архітектури ПС у процесі їхнього розвитку.

У сучасних Java-проєктах контроль дотримання архітектурних обмежень дедалі частіше здійснюють за допомогою інструментів SCA. Такі системи автоматично аналізують залежності між класами, пакетами та модулями ПЗ і дають змогу виявляти порушення архітектурних правил ще на етапі розроблення [13]. Сучасні інструменти аналізу відрізняються функціональністю, рівнем автоматизації та глибиною перевірки архітектурних залежностей, що зумовлює доцільність їхнього порівняльного аналізу (табл. 3).

У сучасних умовах розроблення ПЗ зазначені інструменти застосовуються як елемент системи управління якістю програмного коду. Найпоширенішим практичним рішенням у Java-проєктах є використання SonarQube, який під час автоматичної збірки проєкту аналізує структуру залежностей між пакетами та класами й формує звіт про якість коду. Наприклад, у корпоративних вебзастосунках цей інструмент дає змогу виявляти циклічні залежності між модулями або надмірну зв'язаність компонентів, що ускладнює подальший розвиток системи [15, с. 61303]. Такі порушення відображаються як показники технічного боргу та рекомендацій щодо рефакторингу.

Інструменти ArchUnit застосовуються для формалізації архітектурних правил безпосередньо в програмному коді. Розробники можуть визначити правило, згідно з яким класи рівня представлення не повинні напряму звертатися до компо-

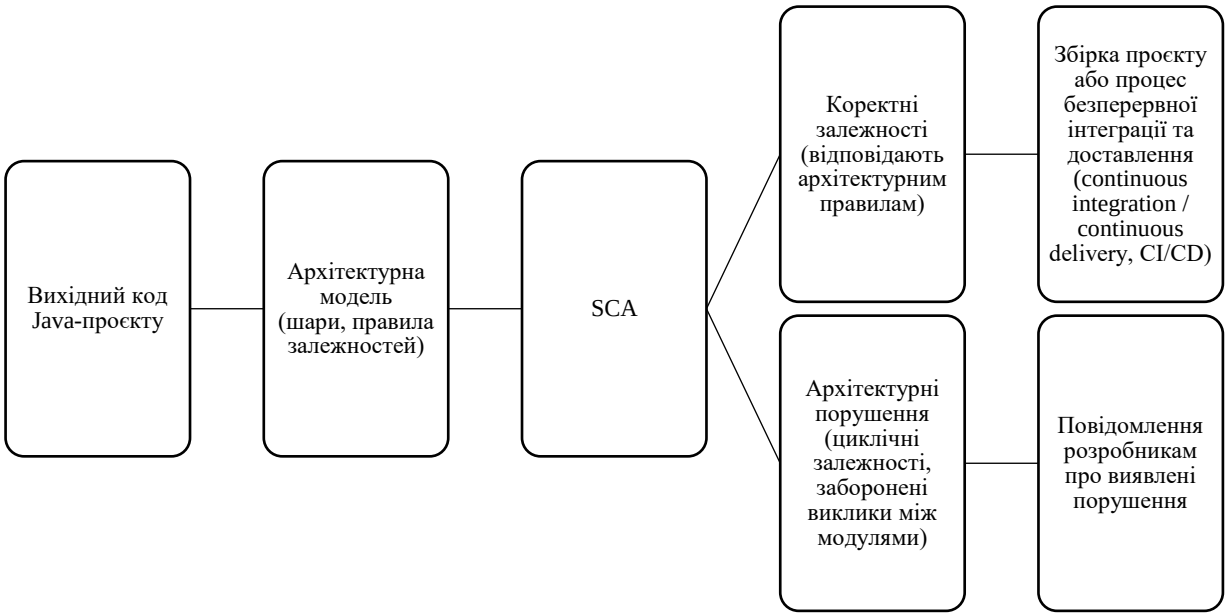

Рис. 1. Схема автоматизованого контролю архітектурних залежностей у Java-проєктах

Джерело: власна розробка автора

Таблиця 3

Порівняльна характеристика інструментів статичного аналізу для виявлення архітектурних порушень у Java-проєктах

Інструмент	Основні функціональні можливості	Виявлення архітектурних порушень	Особливості застосування
SonarQube	Аналіз якості коду, метрик складності, технічного боргу	Виявляє циклічні залежності, надмірну зв'язаність та структурні аномалії	Інтегрується в процес CI/CD
ArchUnit	Бібліотека для формалізації архітектурних правил у коді	Перевіряє дотримання залежностей між пакетами та модулями	Використовується як архітектурні тести
Structure101	Аналіз та візуалізація залежностей між компонентами	Виявляє архітектурну деградацію та складні мережі залежностей	Підтримує аналіз масштабних ПС
jQAssistant	Графовий аналіз структури програмного коду	Дає змогу перевіряти складні архітектурні правила та модульні залежності	Застосовується для аналізу масштабних ПС

Джерело: сформовано автором на основі [3; 4, с. 55829; 8, с. 1131; 9; 14, с. 1173; 15, с. 61303]

нентів рівня доступу до даних. Під час виконання тестів система перевіряє всі залежності в коді та сигналізує про порушення встановлених обмежень. У масштабних проєктах це дає змогу автоматично контролювати тисячі залежностей між пакетами.

Інструменти аналізу залежностей, такі як Structure101 або jQAssistant, застосовують переважно в складних ПС, де важливо оцінити загальну структуру архітектури. Вони дають змогу візуалізувати взаємозв'язки між модулями та виявляти ділянки архітектурної деградації, наприклад, надмірну концентрацію залежностей у певних компонентах системи. Застосування таких інструментів сприяє своєчасній ідентифікації проблемних ділянок програмного коду, підтримці модульності системи та запобіганню накопиченню технічного боргу.

Водночас у процесі супроводу ПС, створених мовою Java, забезпечення їхньої архітектурної узгодженості залишається складним науково-практичним завданням. Однією з основних проблем є поступова деградація архітектури, що виникає внаслідок частих змін коду, розширення функціональності та залучення до розроблення значної кількості фахівців. У таких умовах початкові архітектурні принципи системи часто порушуються, що проявляється появою небажаних залежностей між пакетами, формуванням циклічних зв'язків між модулями або порушенням визначених рівнів архітектури [4, p. 55829]. Додатковою проблемою є складність контролю архітектурних правил у масштабних ПС, де кількість взаємозв'язків між компонентами може досягати тисяч, що робить ручну перевірку структури коду практично неможливою. Унаслідок цього архітектурні порушення накопичуються поступово та формують так званий технічний борг, який ускладнює подальший розвиток ПС, підвищує складність її модифікації та знижує рівень підтримуваності ПЗ.

Розв'язання зазначених проблем потребує застосування автоматизованих механізмів контролю структури коду, зокрема інструментів SCA. Ці засоби дають змогу системно перевіряти відповідність програмної реалізації визначеним архітектурним правилам та оперативно виявляти порушення залежностей між компонентами системи. Практичні рекомендації щодо автоматизованого виявлення архітектурних порушень у Java-проєктах передбачають інтеграцію інструментів статичного аналізу в процес CI/CD, формалізацію архітектурних правил як перевірюваних обмежень між пакетами або модулями, а також регулярний аналіз структурних метрик ПС. Такий підхід дає змогу своєчасно виявляти небажані залежності, підтримувати модульність коду та забезпечувати стабільну еволюцію архітектури ПС у процесі їхнього тривалого використання й розвитку.

Висновки. У результаті дослідження встановлено, що забезпечення архітектурної узгодженості ПС є важливою умовою забезпечення їхньої якості, масштабованості та підтримуваності. Аналіз наукових підходів засвідчив, що ефективний контроль архітектури ПЗ ґрунтується на поєднанні архітектурного проєктування, формалізації правил залежностей між компонентами та використанні інструментів SCA. Доведено, що застосування таких засобів уможливлює автоматичне виявлення небажаних залежностей між модулями, циклічні зв'язки та інші структурні порушення ще на ранніх етапах розроблення. Встановлено, що основними проблемами забезпечення архітектурної узгодженості в Java-проєктах є поступова деградація архітектури внаслідок еволюції коду, складність контролю значної кількості залежностей між компонентами та накопичення технічного боргу.

Порівняльний аналіз сучасних інструментів статичного аналізу засвідчив, що застосування таких засобів, як SonarQube, ArchUnit, Structure101 та jQAssistant, дає змогу істотно підвищити ефективність автоматизованого контролю архітектурних обмежень. Обґрунтовано рекомендації з автоматизованого виявлення архітектурних порушень у Java-проєктах, які передбачають формалізацію архітектурних правил, інтеграцію інструментів статичного аналізу в процесі CI/CD, а також регулярний моніторинг структурних метрик системи.

Перспективи подальших досліджень пов'язані з розробленням інтелектуальних методів аналізу архітектури ПС та застосуванням алгоритмів машинного навчання для прогнозування архітектурних порушень.

Список літератури:

1. Trubiani C., Pinciroli R., Biaggi A., Fontana F. A. Automated detection of software performance antipatterns in Java-based applications. IEEE Transactions on Software Engineering. 2023. Vol. 49, № 4. P. 2873–2891. DOI: https://doi.org/10.1109/TSE.2023.3234321

2. Mendoza C., Bocanegra J., Garcés K., Casallas R. Architecture violations detection and visualization in the continuous integration pipeline. Software: Practice and Experience. 2021. Vol. 51, № 8. P. 1822–1845. DOI: https://doi.org/10.1002/spe.3004

3. Mehrpour S., LaToza T. D. Can static analysis tools find more defects? A qualitative study of design rule violations found by code review. Empirical Software Engineering. 2023. Vol. 28. Article 5. DOI: https://doi.org/10.1007/s10664-022-10232-4

4. Alqaradaghi M., Kozsik T. Comprehensive evaluation of static analysis tools for their performance in finding vulnerabilities in Java code. IEEE Access. 2024. Vol. 12. P. 55824–55842. DOI: https://doi.org/10.1109/ACCESS.2024.3389955

5. Gupta K. K., Shaik M., Kaveri P. R., Awasthi P., Naik S. A., Dhanaraj R. K. Developing an automated script for detecting design pattern in software project using Python. In: The Future of Business and Society. CRC Press, 2023. P. 283–291. URL: https://www.taylorfrancis.com/chapters/edit/10.1201/9781003778882-33 (дата звернення: 15.03.2026).

6. Zhao L., Chen S., Xu Z., Liu C., Zhang L., Wu J., Sun J., Liu Y. Software composition analysis for vulnerability detection: An empirical study on Java projects. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2023. P. 960–972. DOI: https://doi.org/10.1145/3611643.3616299

7. Zhang Y., Kabir M. M. A., Xiao Y., Yao D., Meng N. Automatic detection of Java cryptographic API misuses: Are we there yet? IEEE Transactions on Software Engineering. 2022. Vol. 49, № 1. P. 288–303. DOI: https://doi.org/10.1109/TSE.2022.3150302.

8. Amankwah R., Chen J., Song H., Kudjo P. K. Bug detection in Java code: An extensive evaluation of static analysis tools using Juliet Test Suites. Software: Practice and Experience. 2023. Vol. 53, № 5. P. 1125–1143. DOI: https://doi.org/10.1002/spe.3181.

9. Esposito M., Robredo M., Arcelli Fontana F., Lenarduzzi V. On the correlation between architectural smells and static analysis warnings. Software Quality Journal. 2025. Vol. 33. Article 33. DOI: https://doi.org/10.1007/s11219-025-09730-7.

10. Zhang Y., Xiao Y., Kabir M. M. A., Yao D., Meng N. Example-based vulnerability detection and repair in Java code. In: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension. 2022. P. 190–201. DOI: https://doi.org/10.1145/3524610.3527895

11. Zhang B., Zhi X., Wang M., Ren R., Dong J. Enhancing Java web application security: Injection vulnerability detection via interprocedural analysis and deep learning. IEEE Transactions on Reliability. 2025. Vol. 74, № 3. P. 3642–3656. DOI: https://doi.org/10.1109/TR.2024.3521381

12. Ghorbani N., Singh T., Garcia J., Malek S. Darcy: Automatic architectural inconsistency resolution in Java. IEEE Transactions on Software Engineering. 2024. Vol. 50, № 6. P. 1639–1657. DOI: https://doi.org/10.1109/TSE.2024.3396433

13. Uzun B., Tekinerdogan B. Detecting deviations in the code using architecture view-based drift analysis. Computer Standards & Interfaces. 2024. Vol. 87. Article 103774. DOI: https://doi.org/10.1016/j.csi.2023.103774

14. Ma L., Yang H., Xu J., Yang Z., Lao Q., Yuan D. Code analysis with static application security testing for Python program. Journal of Signal Processing Systems. 2022. Vol. 94, № 11. P. 1169–1182. DOI: https://doi.org/10.1007/s11265-022-01740-z

15. Giebas D., Wojszczyk R. Detection of concurrency errors in multithreaded applications based on static source code analysis. IEEE Access. 2021. Vol. 9. P. 61298–61323. DOI: https://doi.org/10.1109/ACCESS.2021.3073859

Abakumov R.V. AUTOMATED DETECTION OF ARCHITECTURAL VIOLATIONS IN JAVA PROJECTS USING STATIC ANALYSIS

The relevance of the study is determined by the increasing complexity of modern software systems, the growing volume of source code, and the long-term evolution of software products, which complicate maintaining architectural consistency of software systems. In large Java projects, architectural violations often arise during system evolution and functional expansion. Such violations manifest themselves in the emergence of undesirable dependencies between modules, cyclic relations between packages, and violations of layered architecture. These issues lead to the accumulation of technical debt, complicate software maintenance, and reduce system maintainability. Therefore, the use of static code analysis tools becomes increasingly important for automated control of architectural constraints and timely detection of structural deviations in software systems. The purpose of the article is to improve the architectural consistency of software systems through the use of static code analysis tools for automated detection of architectural violations in Java projects. The research applies methods of system analysis of software architectures, comparative analysis of modern static code analysis tools, and generalization of practical approaches to their application in software development. The features of controlling architectural dependencies in Java projects have been investigated, and the capabilities of such tools as SonarQube, ArchUnit, Structure101, and jQAssistant for automated detection of architectural rule violations have been analyzed. Results. The study analyzed modern approaches to ensuring architectural integrity of software systems. It has been established that the main causes of architectural degradation include violations of dependency rules between modules, the emergence of cyclic dependencies, and excessive coupling of system components. It has been proven that the use of static code analysis tools enables automated control of software structure, timely detection of architectural violations, and maintenance of modularity of software systems. The comparative analysis of modern tools demonstrated that their integration into the continuous integration and continuous delivery (CI/CD) process provides effective monitoring of architectural constraints during software development. The conclusions indicate that the use of static analysis tools is an effective mechanism for automated control of software architecture. It has been demonstrated that integrating such tools into software development processes enables the timely detection of structural deviations, reduces technical debt, and supports the stable evolution of Java project architectures. Prospects for further research are related to improving methods for automated analysis of software product architecture and the application of intelligent algorithms for predicting architectural violations.

Keywords: *code structure, module dependencies, technical debt, architecture governance, software engineering, software quality.*

Дата першого надходження статті до видання: 24.03.2026
Дата прийняття статті до друку після рецензування: 22.04.2026
Дата публікації (оприлюднення) статті: 19.05.2026