УДК 004.415:004.738.5:005.8

ВПЛИВ @LAYER, @SCOPE ТА CONTAINER QUERIES НА ЯКІСТЬ CSS-КОДУ В AGILE-ОРІЄНТОВАНІЙ FRONTEND-РОЗРОБЦІ

Р. В. Олійник, О. Р. Чабан, В. І. Плеша, І. В. Заяць

Львівський торговельно-економічний університет
вул. Туган-Барановського, 10, м. Львів, 79005, Україна
oliynyk@lute.lviv.ua

У статті розглянуто проблему організації CSS-коду в Agile-орієнтованій frontend-розробці, за якої зміни інтерфейсу вносяться ітеративно, а навіть незначні правки стилів можуть спричиняти неочікувані візуальні регресії. Обґрунтовано, що CSS доцільно розглядати не лише як засіб візуального оформлення, а і як складову frontend-архітектури, яка впливає на модифікованість коду, локальність змін, повторне використання компонентів і тестованість інтерфейсу. Досліджено роль трьох нативних механізмів CSS: @layer, @scope та Container Queries. Правило @layer проаналізовано як засіб підвищення передбачуваності каскаду через розділення reset-стилів, дизайн-токенів, базових правил, макетів, компонентів, утиліт і контрольованих перевизначень. Механізм @scope розглянуто як спосіб обмеження зони дії CSS-правил і зменшення ризику випадкового впливу на суміжні елементи інтерфейсу. Container Queries визначено як інструмент опису адаптивної поведінки компонента залежно від розміру або властивостей його контейнера, а не лише глобального viewport.

Запропоновано систему критеріїв оцінювання якості CSS-коду в умовах ітеративної розробки інтерфейсів. До них віднесено специфічність селекторів, кількість глобальних правил, використання !important, кількість перевизначень, залежність від DOM-структури, локальність змін, формалізацію інтерфейсних об'єктів, повторне використання компонентів, потенційні точки візуальної регресії та придатність до visual regression testing. Показано, що поєднання @layer, @scope і Container Queries може зменшити приховані залежності між компонентами, підвищити передбачуваність CSS-змін і створити умови для безпечного рефакторингу під час спринтів. Водночас наголошено, що ефективність цих механізмів залежить від архітектурної дисципліни, правил організації стилів, code review, автоматизованих перевірок і включення CSS-змін до Definition of Done Agile-команди.

***Ключові слова**: @layer, @scope, Container Queries, Object-Oriented CSS, frontend-розробка, Agile, тестування ПЗ, модифікованість, візуальна регресія.*

Постановка проблеми. Сучасна frontend-розробка характеризується швидким зростанням складності вебінтерфейсів, активним використанням компонентних архітектур, дизайн-систем, адаптивних макетів і динамічних сценаріїв взаємодії з користувачем. Якщо на ранніх етапах розвитку вебтехнологій CSS-код переважно виконував роль засобу візуального оформлення сторінки, то сьогодні він є важливою частиною архітектури frontend-додатка. Від способу організації стилів залежить не тільки зовнішній вигляд інтерфейсу, а й можливість його подальшого розвитку, тестування і безпечного внесення змін. Основна проблема полягає в тому, що зі зростанням кількості компонентів CSS-код може втрачати структурованість: у ньому накопичуються глобальні правила, надмірно специфічні селектори, неконтрольовані перевизначення та залежності від конкретної DOM-структури. У результаті навіть незначна зміна одного компонента може спричинити побічні візуальні зміни в інших частинах інтерфейсу. В умовах Agile-орієнтованої розробки ця проблема посилюється, оскільки зміни вносяться ітераційно, а вимоги до інтерфейсу можуть уточнюватися після sprint review, аналізу поведінки користувачів, тестування прототипів або оновлення дизайн-системи. Тому CSS-код повинен бути не лише працездатним у поточній версії продукту, а й достатньо модифікованим, передбачуваним і придатним до регресійного тестування.

Традиційні підходи до організації CSS-коду, зокрема глобальні таблиці стилів, BEM-методологія, CSS Modules, препроцесори або utility-first CSS, частково вирішують проблему структурування стилів. Однак розвиток самого стандарту CSS привів до появи нативних механізмів, які безпосередньо впливають на керування каскадом, областю дії правил і адаптивністю компонентів. До таких механізмів належать @layer, @scope та Container Queries. Незважаючи на практичну значущість нових правил специфікації, їх вплив на модифікованість і тестованість CSS-коду потребує окремого дослідження. Необхідно визначити, як їх використання впливає на кількість прихованих залежностей, локальність змін, складність перевизначення стилів і ризик візуальних регресій у процесі Agile-розробки. Отже, науково-практична проблема полягає у відсутності формалізованого підходу оцінювання того, як нові нативні CSS-механізми впливають на модифікованість, локальність змін і тестованість компонентного CSS-коду. У зв’язку з цим виникає потреба сформувати систему критеріїв, яка дозволить не лише описати можливості

@layer, @scope та Container Queries, а й оцінити їхній вплив на якість CSS-архітектури в умовах Agile-орієнтованої розробки.

Аналіз останніх досліджень і публікацій. Технічною основою для аналізу сучасних CSS-механізмів є специфікації W3C. У CSS Cascading and Inheritance Level 5 описано каскадні шари, що дає підстави розглядати @layer як засіб явного керування порядком застосування стилів [1]. У CSS Cascading and Inheritance Level 6 подано механізм @scope, призначений для обмеження області дії CSS-правил [2]. Специфікація CSS Containment Module Level 3 формує основу для Container Queries, за яких адаптивність компонента може залежати не лише від розміру вікна браузера, а й від параметрів контейнера [3]. У документації MDN Web Docs цей механізм розглядається як практичний інструмент створення компонентів, що можуть адаптуватися до різних контекстів відображення [4].

Окремо варто відзначити українські дослідження, присвячені тестуванню вебзастосунків. У них розглядаються особливості сучасного тестування вебдодатків, автоматизація тестування, а також поєднання ручних і автоматизованих підходів до перевірки якості програмних продуктів [5; 6]. Такі напрацювання є суттєвими, оскільки зміни в CSS-коді безпосередньо впливають на користувацький інтерфейс. Тому їх доцільно перевіряти не лише з погляду синтаксичної коректності, а й за фактичним візуальним результатом. У дослідженнях також наголошується, що швидкість розробки не повинна досягатися за рахунок зниження якості, тому перевірка змін після кожної ітерації є важливою частиною Agile-процесу [9]. Для frontend-частини це означає необхідність тестування візуальних регресій, оскільки навіть невелика зміна CSS-коду може змінити вигляд компонента або вплинути на суміжні елементи інтерфейсу [10].

Дослідження Agile-методологій також акцентують увагу на потребі в гнучкій архітектурі, яка може розширюватися без радикальної перебудови вже реалізованої частини системи [7]. Для CSS-архітектури це положення є не менш важливим, оскільки інтерфейс у frontend-додатку належить до найбільш змінюваних частин програмного продукту. Так окремого значення набувають дослідження автоматизованого тестування адаптивних вебінтерфейсів, де підкреслюється потреба в моделюванні та перевірці адаптивної поведінки [8]. Це безпосередньо стосується Container Queries, адже вони переносять логіку адаптивності з рівня всього екрана на рівень окремого контейнера. Такий підхід підвищує повторне використання компонентів, але водночас вимагає перевірки їхньої поведінки в різних контейнерних умовах.

Окремо слід відзначити праці, присвячені метрикам підтримуваності програмного коду [11]. Для CSS-коду такі підходи не можуть застосовуватися безпосередньо, оскільки вони не повністю враховують специфіку каскаду, специфічність селекторів, глобальну область дії правил і ризик візуальних регресій. Саме тому дослідження, присвячені підтримуваності CSS через структуру та рефакторинг, є важливими для формування критеріїв оцінювання якості стилів [12].

Отже, аналіз джерел показує, що в науковій і технічній літературі окремо розглядаються сучасні CSS-механізми, автоматизація тестування вебзастосунків, Agile-підходи та метрики підтримуваності програмного коду. Водночас недостатньо опрацьованим залишається питання комплексного оцінювання впливу @layer, @scope та Container Queries на модифікованість і тестованість CSS-коду саме в Agile-орієнтованій frontend-розробці.

Метою дослідження є аналіз впливу CSS-механізмів @layer, @scope та Container Queries на модифікованість і тестованість CSS-коду в Agile-орієнтованій frontend-розробці, а також формування системи критеріїв для оцінювання їхнього впливу на якість компонентних вебінтерфейсів.

Досягнення поставленої мети передбачає: аналіз сучасних підходів до організації CSS-коду в компонентно-орієнтованій frontend-розробці; дослідження @layer, @scope та Container Queries з позиції керування каскадом, областю дії стилів і адаптивністю компонентів; визначення критеріїв оцінювання модифікованості й тестованості CSS-коду; розроблення методики порівняння традиційної CSS-організації з підходом, що використовує сучасні нативні CSS-механізми; оцінювання потенційного впливу зазначених механізмів на локальність змін, кількість перевизначень, специфічність селекторів і ризик візуальних регресій; формулювання практичних рекомендацій щодо використання @layer, @scope та Container Queries у процесах Agile-орієнтованої frontend-розробки.

У роботі використано методи системного аналізу, порівняльного аналізу, структурного аналізу CSS-коду, метричного оцінювання, а також елементи експериментального моделювання frontend-компонентів. Системний підхід застосовано для розгляду CSS-коду як частини архітектури frontend-додатка. Порівняльний аналіз використано для зіставлення традиційної організації CSS із підходом, що базується на @layer, @scope та Container Queries. Метричний підхід застосовано для визначення показників модифікованості й тестованості, зокрема специфічності селекторів, кількості глобальних правил, кількості перевизначень, кількості змінених файлів і потенційних точок візуальної регресії.

Об'єктом дослідження є процес організації CSS-коду в компонентно-орієнтованій frontend-розробці вебінтерфейсів. Предметом дослідження є вплив CSS-механізмів @layer, @scope та Container Queries на модифікованість, підтримуваність і тестованість CSS-коду в умовах Agile-орієнтованої розробки.

Виклад основного матеріалу дослідження. У межах цього дослідження CSS-код розглядається не лише як набір правил візуального оформлення, а як складник frontend-архітектури, що описує структуру, стани, варіанти відображення та адаптивну поведінку інтерфейсних компонентів. Такий підхід є важливим для Agile-орієнтованої розробки, оскільки інтерфейс програмного продукту змінюється ітераційно, а отже CSS-код повинен підтримувати безпечне внесення змін, повторне використання компонентів і можливість автоматизованої перевірки візуального результату. CSS не є об'єктно-орієнтованою мовою програмування в класичному розумінні, однак у frontend-інженерії він часто організовується за компонентно-об'єктною логікою. Це проявляється в тому, що стилі описують повторно використовувані об'єкти інтерфейсу: кнопки, картки, форми, навігаційні елементи, повідомлення, модальні вікна або dashboard-віджети. Кожен такий об'єкт має базову відповідальність, допустимі модифікатори, стани, контексти використання та очікувану адаптивну поведінку. Наприклад, кнопка може мати базовий клас .button, модифікатори .button--primary, .button--secondary, а також стани .is-loading чи .is-disabled.

Саме з позиції компонентно-об'єктної логіки доцільно розглядати @layer, @scope та Container Queries. Ці механізми не просто розширюють синтаксис CSS, а формалізують різні властивості інтерфейсного об'єкта. Директива @layer визначає архітектурний рівень стилів об'єкта та його місце в каскаді. Директива @scope задає межі дії стилів і наближає CSS-код до ідеї інкапсуляції, оскільки зменшує ризик впливу правил одного компонента на інші частини інтерфейсу. Container Queries описують адаптивну поведінку об'єкта залежно від розміру контейнера, тобто дозволяють компоненту реагувати на власний контекст використання, а не лише на глобальну ширину viewport. У традиційній організації CSS така компонентно-об'єктна структура часто залишається неформалізованою. Компоненти можуть мати спільні глобальні класи, стилі можуть впливати один на одного через каскад, а нові вимоги часто реалізуються шляхом додавання контекстних селекторів або несистемних перевизначень. Унаслідок цього кожна нова user story, пов'язана зі зміною інтерфейсу, може підвищувати специфічність селекторів, збільшувати кількість override-правил і створювати ризик візуальних регресій.

Методологія дослідження ґрунтується на порівнянні трьох підходів до організації CSS-коду. Перший підхід передбачає використання традиційної організації стилів із глобальними класами, media queries та порядком підключення файлів як основним механізмом пріоритетності. Другий підхід базується на застосуванні @layer, що дозволяє формалізувати каскад через поділ стилів на архітектурні шари. Третій підхід поєднує @layer, @scope та Container Queries, тобто охоплює одночасно архітектурне розміщення CSS-об'єкта, локалізацію області його дії та опис адаптивної поведінки в межах контейнера.

Для формалізації дослідження запропоновано розглядати типовий набір frontend-компонентів: кнопку, картку, форму, інформаційний блок, навігаційний елемент і dashboard-віджет. Кожен із цих компонентів розглядається як інтерфейсний об'єкт, що має базове представлення, модифікатори, стани та адаптивні варіанти. Такий набір дозволяє змоделювати поширені сценарії змін, характерні для Agile-розробки: додавання нового стану компонента, зміну теми оформлення, адаптацію компонента до іншого layout-контексту, зміну поведінки форми або оновлення службових класів.

Застосування @layer дозволяє перейти від неявного керування каскадом до явного визначення пріоритетності груп стилів. Наприклад, CSS-код може бути структурований за такими шарами:

@layer reset, tokens, base, layout, objects, components, utilities, overrides;

У цьому випадку кожен шар виконує окрему роль. Шар reset відповідає за нормалізацію стилів, tokens — за змінні дизайн-системи, base — за базове оформлення HTML-елементів, layout — за сітки й контейнери, objects — за повторно використовувані стилістичні об'єкти, components — за конкретні UI-компоненти, utilities — за службові класи, а overrides — за контрольовані перевизначення. На відміну від хаотичного порядку підключення CSS-файлів, така структура робить взаємодію між рівнями стилів більш передбачуваною. Разом з тим @layer не вирішує проблему локалізації дії стилів. Якщо правила залишаються глобальними, вони можуть впливати на інші частини інтерфейсу. Для зменшення такого ризику доцільно використовувати @scope, який дозволяє обмежити дію стилів певним фрагментом DOM.

Container Queries вирішують іншу проблему — залежність адаптивності компонента від глобальної ширини viewport. У традиційному підході media queries описують поведінку інтерфейсу залежно від розміру вікна браузера, однак один і той самий компонент може розміщуватися в різних контекстах: основній області сторінки, sidebar, modal window або dashboard-grid. У таких випадках компоненту важливо реагувати не лише на ширину екрана, а й на

розмір власного контейнера. Такий підхід підвищує повторне використання компонента, оскільки його адаптивна поведінка стає прив'язаною до контейнера, а не до всієї сторінки. У контексті Agile-розробки це важливо, оскільки під час зміни layout або появи нових user stories компонент може використовуватися в новому місці без необхідності переписувати глобальні media queries.

Оцінювання впливу @layer, @scope та Container Queries на CSS-код потребує не лише загального опису їхніх технічних можливостей, а й формування системи критеріїв, за допомогою яких можна визначити, які саме зміни ці механізми спричиняють у структурі, супроводі та перевірці стилів, які доцільно висувати з урахуванням причинно-наслідкового зв'язку між способом організації CSS-коду та якістю подальшої роботи з ним.

Agile-орієнтована frontend-розробка передбачає ітераційне внесення змін до інтерфейсу. Нова user story або уточнення вимог може спричинити додавання нового стану компонента, зміну теми оформлення, адаптацію до іншого layout-контексту або коригування візуальної поведінки. Якщо CSS-код має нечітку структуру, високу специфічність селекторів і значну кількість взаємних залежностей, то кожна така зміна потребує додаткових перевизначень і підвищує ризик регресії. З цієї причини у доцільно розглядати критерії, які дають змогу оцінити, наскільки легко локалізувати зміну, яку кількість правил потрібно модифікувати, чи виникає потреба у додаткових перевизначеннях і чи обмежується вплив зміни межами конкретного компонента або шару.

Окремого формалізованого оцінювання потребує тестованість CSS-коду, оскільки зміни в стилях безпосередньо впливають на візуальний результат роботи інтерфейсу. Синтаксична коректність CSS не гарантує, що компонент після зміни відображатиметься стабільно, не порушить суміжні елементи та збереже очікувану поведінку в різних контейнерних умовах. Тому до системи критеріїв включено показники, які дозволяють визначити можливість ізольованої перевірки компонента, придатність стилів до visual regression testing, стабільність відображення в різних контекстах і рівень ризику неочікуваних візуальних змін.

Отже, у критерії оцінювання впливу @layer, @scope та Container Queries формуються як інструмент причинно-наслідкового аналізу: спосіб організації каскаду, області дії стилів і контейнерної адаптивності безпосередньо впливає на модифікованість, локалізованість змін і тестованість CSS-коду. Модифікованість характеризує можливість безпечного внесення змін без значного ускладнення структури стилів; локалізованість впливу показує, чи залишається зміна в межах конкретного компонента, шару або контейнера;

тестованість визначає, наскільки об’єктивно можна перевірити візуальний результат після зміни CSS-коду (табл. 1).

Таблиця 1

Критерії оцінювання модифікованості та тестованості CSS-коду

№	Критерій	Позначення	Що характеризує	Зв’язок із CSS-механізмами
1	Специфічність селекторів	C1	Складність перевизначення стилів і ризик накопичення технічного боргу	@layer зменшує потребу в підвищенні специфічності
2	Кількість глобальних правил	C2	Ризик побічного впливу стилів на інші компоненти	@scope обмежує область дії правил
3	Кількість !important	C3	Рівень примусового втручання в каскад	@layer дозволяє керувати пріоритетами без !important
4	Кількість override-правил	C4	Обсяг перевизначень, необхідних для внесення змін	@layer і @scope роблять перевизначення контрольованішими
5	Локальність змін	C5	Кількість файлів, шарів або блоків, які потрібно змінити	@scope і компонентна структура локалізують зміни
6	Залежність від DOM-структури	C6	Наскільки стилі прив’язані до вкладеності HTML-елементів	@scope зменшує потребу в довгих контекстних селекторах
7	Формалізація CSS-об’єкта	C7	Наявність базового класу, модифікаторів, станів і меж компонента	@layer, @scope і Container Queries формалізують властивості об’єкта

8	Повторне використання компонента	C8	Здатність компонента працювати в різних layout-контекстах	Container Queries підвищують незалежність компонента від viewport
9	Потенційні точки візуальної регресії	C9	Кількість компонентів, які можуть бути зачеплені CSS-зміною	@scope зменшує зону можливого побічного впливу
10	Придатність до visual regression testing	C10	Можливість ізольовано перевірити компонент після зміни стилів	@scope і контейнерна адаптивність спрощують сценарії перевірки

Запропоновані критерії дозволяють оцінити не лише наявність або відсутність певного CSS-механізму, а його фактичний вплив на архітектурну якість стилів. Наприклад, використання @layer доцільно оцінювати через зменшення потреби у високоспецифічних селекторах, !important і хаотичних override-правилах. Використання @scope доцільно оцінювати через зменшення кількості глобальних правил, підвищення локальності змін і зниження ризику побічного впливу на інші компоненти. Container Queries, своєю чергою, оцінюються через здатність компонента зберігати коректну адаптивну поведінку в різних контейнерних умовах. А у випадку Agile-розробки така система критеріїв дозволяє оцінювати CSS-код не лише після завершення великого рефакторингу, а й у межах окремих ітерацій. Наприклад, після реалізації user story, пов'язаної зі зміною інтерфейсу, можна перевірити, чи залишилася зміна локальною, чи не збільшилася специфічність селекторів, чи не з'явилися нові глобальні правила і чи не зафіксовано візуальних регресій під час автоматизованого тестування.

Щоб система критеріїв не залишалася суто описовою, кожен критерій доцільно пов'язати з певним способом його оцінювання, що дозволить перейти від загальних тверджень про вплив @layer, @scope та Container Queries до формалізованого аналізу, у якому можна простежити, за якими ознаками визначається модифікованість, локалізованість змін і тестованість CSS. Запропонований підхід до оцінювання критеріїв наведено в табл. 2.

Таблиця 2

Спосіб операціоналізації критеріїв оцінювання CSS-коду

Критерій	Спосіб оцінювання
C1 Специфічність селекторів	Середнє або максимальне значення специфічності селекторів у CSS-блоці
C2 Кількість глобальних правил	Кількість правил, що не належать до компонента, scope або визначеного шару
C3 Кількість !important	Прямий підрахунок входжень !important
C4 Кількість override-правил	Кількість правил, що перевизначають попередні стилі компонента
C5 Локальність змін	Кількість файлів, шарів або CSS-блоків, змінених для реалізації однієї user story
C6 Залежність від DOM-структури	Середня довжина селектора або кількість вкладених частин селектора
C7 Формалізація CSS-об'єкта	Наявність базового класу, модифікаторів, станів, scope та контейнерної поведінки
C8 Повторне використання компонента	Кількість layout-контекстів, у яких компонент працює без додаткових override-правил
C9 Потенційні точки візуальної регресії	Кількість компонентів або сторінкових зон, на які може вплинути CSS-зміна
C10 Придатність до visual regression testing	Можливість перевірити компонент ізольовано в базовому, модифікованому та адаптивному станах

Для інтегрального оцінювання якості CSS-архітектури пропонується використовувати зважену адитивну модель

$$Q = \sum_{i=1}^{n} \omega_i \cdot c_i \,, \tag{1}$$

де Q — інтегральний показник якості CSS-архітектури; ω_i— ваговий коефіцієнт і-го критерію; c_i — оцінка за і-м критерієм; n — кількість критеріїв. Для коректного використання моделі вагові коефіцієнти мають бути нормалізовані

Оцінювання за кожним критерієм може здійснюватися за шкалою від 1 до 5, де 1 означає низький рівень відповідності критерію, а 5 — високий рівень відповідності. За умови нормалізації ваг інтегральний показник Q також набуває значень у межах від 1 до 5. У межах цієї роботи вагові коефіцієнти можуть бути прийняті рівними для демонстрації роботи моделі; у подальших дослідженнях доцільно визначати їх методом експертного ранжування.

Для демонстрації застосування запропонованої моделі в табл. 3 наведено приклад експертного оцінювання трьох підходів до організації CSS-коду за шкалою від 1 до 5. Наведені оцінки мають демонстраційний характер і показують спосіб застосування моделі; для отримання статистично надійних результатів необхідна апробація на реальних frontend-проєктах.

Таблиця 3

Приклад демонстраційного оцінювання підходів до організації CSS-коду

Критерій	**Традиційний CSS**	**CSS з @layer**	**CSS з @layer, @scope та Container Queries**
C1 Специфічність селекторів	2	4	4
C2 Кількість глобальних правил	2	3	4
C3 Кількість !important	2	4	4
C4 Override-правила	2	4	4
C5 Локальність змін	2	3	5
C6 Залежність від DOM	2	3	4
C7 Формалізація CSS-об'єкта	2	3	5
C8 Повторне використання	2	3	5
C9 Візуальні регресії	2	3	4
C10 Visual regression testing	2	3	5
Середній показник	**2,0**	**3,3**	**4,4**

Результати демонстраційного оцінювання свідчать, що найбільше зростання інтегрального показника очікується при комплексному використанні @layer, @scope та Container Queries. Це пояснюється тим, що такий підхід одночасно впливає на три групи проблем: керування каскадом, локалізацію області дії стилів і адаптивність компонента до контейнера. Водночас отримані значення не слід розглядати як універсальний емпіричний результат, оскільки вони залежать від структури конкретного проєкту, правил іменування класів, прийнятої дизайн-системи та рівня автоматизації тестування. Для якісного узагальнення очікуваного впливу розглянутих підходів до організації CSS-коду доцільно використовувати порівняльну характеристику, наведену в табл. 4.

Таблиця 4

Узагальнення впливу CSS-механізмів на модифікованість і тестованість

Критерій	**Традиційний CSS**	**CSS з @layer**	**CSS з @layer, @scope та Container Queries**
Передбачуваність каскаду	Середня або низька	Вища	Вища
Локальність змін	Обмежена	Середня	Висока
Ризик глобального впливу	Високий	Середній	Нижчий
Залежність від DOM-структури	Висока	Середня	Нижча
Потреба в override-правилах	Висока	Нижча	Нижча
Придатність до тестування	Середня	Вища	Вища
Повторне використання компонентів	Обмежене	Середнє	Високе
Адаптивність у різних контейнерах	Переважно через viewport	Переважно через viewport	Через контейнер компонента

Таблиця 4 відображає очікуваний вплив кожного підходу на визначені критерії. Використання @layer найбільше впливає на передбачуваність каскаду, оскільки дозволяє явно визначити пріоритети між групами стилів. Використання @scope посилює локальність змін, оскільки обмежує область дії правил. Container Queries підвищують адаптивність і повторне використання компонентів, оскільки компонент може змінювати власну структуру залежно від контейнера, у якому він розміщений.

У випадку застосування Agile-розробки ці механізми доцільно розглядати як зміни, що виникають у процесі спринтів. Наприклад, під час sprint review може бути сформульована вимога додати compact-версію картки товару, змінити вигляд кнопок, адаптувати форму до sidebar або додати новий стан помилки. У традиційному CSS такі зміни часто потребують створення додаткових селекторів або перевизначень. У структурованому CSS з @layer, @scope та Container Queries такі зміни можуть бути локалізовані в межах відповідного шару, області дії або контейнерного правила. Варто зауважити, що для Agile-команди важливо, щоб кожна зміна стилів проходила перевірку не лише візуально розробником, а й через формалізовані механізми тестування. До Definition of Done для CSS-змін доцільно включити такі умови: зміна локалізована в межах відповідного компонента або шару; не додано нових !important, якщо це не обґрунтовано; не збільшено максимальну специфічність селекторів; компонент перевірено в основних контейнерних контекстах; виконано visual regression testing; не зафіксовано неочікуваних змін у суміжних компонентах.

Таким чином, @layer, @scope та Container Queries можуть бути інтегровані не лише в технічну структуру CSS-коду, а й у процеси командної розробки. Їх застосування має супроводжуватися правилами організації стилів, code review, автоматизованими перевірками та регресійним тестуванням. Лише за таких умов можна говорити про їхній реальний вплив на модифікованість і тестованість CSS-коду. Водночас необхідно зазначити, що ці механізми не є універсальним вирішенням усіх проблем CSS-архітектури. @layer може бути неефективним, якщо шари визначені хаотично або дублюють одне одного. @scope може ускладнювати підтримку, якщо використовується без єдиних правил локалізації стилів. Container Queries можуть призводити до надмірної

кількості адаптивних сценаріїв, якщо компонент не має чітко визначених меж використання. Тому їх позитивний ефект залежить від архітектурної дисципліни та системності застосування.

Висновки. У результаті проведеного дослідження встановлено, що сучасні нативні CSS-механізми @layer, @scope та Container Queries можуть бути використані як інструменти підвищення модифікованості й тестованості CSS-коду в Agile-орієнтованій frontend-розробці. Їх значення полягає не лише у розширенні синтаксису CSS, а й у створенні передумов для кращого керування каскадом, локалізації стилів і адаптації компонентів до різних контекстів використання.

Також запропоновано систему критеріїв для оцінювання модифікованості й тестованості CSS-коду, що включає специфічність селекторів, кількість глобальних правил, кількість !important, кількість override-правил, локальність змін, залежність від DOM-структури, повторне використання компонентів, кількість потенційних візуальних регресій і придатність до visual regression testing. Такий підхід дозволяє перейти від загального опису CSS-механізмів до формалізованого оцінювання їхнього впливу на якість frontend-коду.

Проведений аналіз показав, що найбільший практичний ефект може бути досягнутий не при ізольованому використанні одного з розглянутих механізмів, а при їх комплексному застосуванні. @layer доцільно використовувати для побудови шарової структури стилів, @scope — для обмеження області дії компонентних правил, а Container Queries — для створення адаптивних і повторно використовуваних компонентів. У сукупності ці механізми можуть зменшити приховані залежності між CSS-правилами та покращити умови для регресійного тестування інтерфейсу.

Водночас встановлено, що сам факт використання @layer, @scope та Container Queries не гарантує автоматичного підвищення якості CSS-коду. Їх ефективність залежить від наявності архітектурних правил, командних домовленостей, code review, тестування та включення CSS-змін до Definition of Done в Agile-команді. Без таких умов нові CSS-механізми можуть стати лише додатковим синтаксичним рівнем, який не усуває проблеми хаотичної організації стилів.

Перспективи подальших досліджень пов’язані з практичною апробацією запропонованої системи критеріїв на реальних frontend-проєктах, побудовою експериментального стенду для порівняння різних CSS-архітектур, а також використанням інструментів visual regression testing для кількісного оцінювання впливу CSS-змін на стабільність компонентних інтерфейсів.

Список використаних джерел

1. CSS Cascading and Inheritance Level 5. W3C URL: https://www.w3.org/TR/css-cascade-5
2. CSS Cascading and Inheritance Level 6. W3C. URL: https://www.w3.org/TR/css-cascade-6
3. CSS Containment Module Level 3. W3C. URL: https://www.w3.org/TR/css-contain-3
4. CSS Container Queries. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_container_queries
5. Багрій Р. О., Петровський С. С. Особливості сучасного тестування веб-додатків. Вісник Хмельницького національного університету. Технічні науки. 2022. № 3 (309). С. 70—74. DOI: https://doi.org/10.31891/2307-5732-2022-309-3-70-74.
6. Скідан В. В., Пилипенко В. І., Каленський Б. В. Автоматизація тестування веб-застосунків. Мехатронні системи: інновації та інжиніринг : тези доповідей VII Міжнародної науково-практичної конференції, м. Київ, 23 листопада 2023 р. Київ : КНУТД, 2023. С. 228—229. URL: https://er.knutd.edu.ua/handle/123456789/26070.
7. Яковенко В., Ульяновська Ю., Яковенко Т., Чупілко Т. Адаптація принципів Agile методології для управління проєктом розробки програмного застосунку. Інформаційні технології та комп'ютерна інженерія. 2021. № 3 (52). С. 44—52. DOI: https://doi.org/10.31649/1999-9941-2021-52-3-44-52
8. Іванюк В., Верлань А., Мястковська М., Косінов М. Структурно-функціональне моделювання системи автоматизованого тестування адаптивних веб-інтерфейсів. Математичне та комп'ютерне моделювання. Серія: Технічні науки. 2025. Вип. 28. С. 37—48. DOI: https://doi.org/10.32626/2308-5916.2025-28.37-48
9. Das S., Gary K. Regression Testing in Agile — A Systematic Mapping Study. Software. 2025. Vol. 4, No. 2. Article 9. DOI: https://doi.org/10.3390/software4020009
10. Johnsson C. Automated Generation of Visual Regression Tests Through State Exploration : master's thesis. Stockholm : KTH Royal Institute of Technology, 2025. 95 p. URL: https://kth.diva-portal.org/smash/get/diva2:1987983/FULLTEXT01.pdf
11. Heričko T., Šumak B. Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems. Applied Sciences. 2023. Vol. 13, No. 5. Article 2972. DOI: https://doi.org/10.3390/app13052972
12. Rydfalk V. Improving maintenance of CSS through structure and refactoring strategies : master's thesis. Linköping : Linköping University, 2022. 51 p. URL: https://liu.diva-portal.org/smash/record.jsf?pid=diva2:1646020

References

1. W3C. (2022). CSS Cascading and Inheritance Level 5. https://www.w3.org/TR/css-cascade-5
2. W3C. (2024). CSS Cascading and Inheritance Level 6. https://www.w3.org/TR/css-cascade-6
3. W3C. (2022). CSS Containment Module Level 3. https://www.w3.org/TR/css-contain-3/

4. MDN Web Docs. (n.d.). CSS Container Queries. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_container_queries
5. Bahrii, R. O., & Petrovskyi, S. S. (2022). Osoblyvosti suchasnoho testuvannia veb-dodatkiv [Features of modern web application testing]. Visnyk Khmelnytskoho natsionalnoho universytetu. Tekhnichni nauky, 3(309), 70–74. [in Ukrainian]. https://doi.org/10.31891/2307-5732-2022-309-3-70-74
6. Skidan, V. V., Pylypenko, V. I., & Kalenskyi, B. V. (2023). Avtomatyzatsiia testuvannia veb-zastosunkiv [Automation of web application testing]. Mekhatronni systemy: innovatsii ta inzhynirynh: tezy dopovidei VII Mizhnarodnoi naukovo-praktychnoi konferentsii, 228–229. Kyiv: KNUTD. [in Ukrainian]. https://er.knutd.edu.ua/handle/123456789/26070
7. Yakovenko, V., Ulianovska, Yu., Yakovenko, T., & Chupilko, T. (2021). Adaptatsiia pryntsypiv Agile metodolohii dlia upravlinnia proiektom rozrobky prohramnoho zastosunku [Adaptation of Agile methodology principles for software application development project management]. Informatsiini tekhnolohii ta kompiuterna inzheneriia, 3(52), 44–52. [in Ukrainian]. https://doi.org/10.31649/1999-9941-2021-52-3-44-52
8. Ivaniuk, V., Verlan, A., Miastkovska, M., & Kosinov, M. (2025). Strukturno-funktsionalne modeliuvannia systemy avtomatyzovanoho testuvannia adaptyvnykh veb-interfeisiv [Structural and functional modeling of an automated testing system for adaptive web interfaces]. Matematychne ta kompiuterne modeliuvannia. Seriia: Tekhnichni nauky, 28, 37–48. [in Ukrainian]. https://doi.org/10.32626/2308-5916.2025-28.37-48
9. Das, S., & Gary, K. (2025). Regression Testing in Agile — A Systematic Mapping Study. Software, 4(2), Article 9. https://doi.org/10.3390/software4020009
10. Johnsson, C. (2025). Automated Generation of Visual Regression Tests Through State Exploration [Master's thesis, KTH Royal Institute of Technology]. https://kth.diva-portal.org/smash/get/diva2:1987983/FULLTEXT01.pdf
11. Heričko, T., & Šumak, B. (2023). Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems. Applied Sciences, 13(5), Article 2972. https://doi.org/10.3390/app13052972
12. Rydfalk, V. (2022). Improving maintenance of CSS through structure and refactoring strategies [Master's thesis, Linköping University]. https://liu.diva-portal.org/smash/record.jsf?pid=diva2:1646020

Роботу виконано в межах наукової теми "Моделювання та застосування хаотичних динамічних систем для оптимізації алгоритмів штучного інтелекту, машинного навчання та вебтехнологій" (державний реєстраційний номер 0124U004447).

DOI 10.32403/2411-9210-2026-1-55-78-95

RESEARCH ON THE IMPACT OF @LAYER, @SCOPE AND CONTAINER QUERIES ON CSS CODE MODIFIABILITY AND TESTABILITY IN AGILE-ORIENTED FRONTEND DEVELOPMENT

R. Oliinyk, O. Chaban, V. Plesha, I. Zayac
Lviv University of Trade and Economics
oliynyk@lute.lviv.ua

The article addresses the problem of CSS code organization in Agile-oriented frontend development, where interface changes are introduced iteratively and even small modifications in styles may lead to unexpected visual regressions. The paper argues that CSS should be considered not only as a visual styling tool, but also as a part of frontend architecture that influences code modifiability, locality of changes, component reuse and testability. The study analyzes the role of three native CSS mechanisms: @layer, @scope and Container Queries. The @layer rule is considered as a way to make the cascade more predictable by separating reset styles, design tokens, base rules, layouts, components, utilities and controlled overrides. The @scope rule is analyzed as a mechanism for limiting the area of influence of CSS rules and reducing the risk of accidental impact on adjacent interface elements. Container Queries are considered as a tool for describing adaptive component behavior depending on the size or properties of the component container rather than only on the global viewport.

The article proposes a set of criteria for evaluating CSS code quality in the context of iterative interface development. These criteria include selector specificity, the number of global rules, the use of !important, the number of override rules, dependence on the DOM structure, locality of changes, formalization of interface objects, component reuse, potential visual regression points and suitability for visual regression testing. The paper shows that the combined use of @layer, @scope and Container Queries can reduce hidden dependencies between components, improve the predictability of CSS changes and create better conditions for safe refactoring during sprints. At the same time, the article emphasizes that these mechanisms do not automatically improve CSS quality. Their practical effect depends on architectural discipline, consistent style organization rules, code review, automated checks and the inclusion of CSS changes in the Definition of Done of an Agile team. Therefore, the proposed approach connects modern CSS features not only with

syntax, but also with the engineering practices required for maintainable and testable frontend systems.

***Keywords**:@layer, @scope, Container Queries, Object-Oriented CSS, frontend development, Agile, software testing, modifiability, visual regression.*

Стаття надійшла до редакції 08.04.2026
Received 08.04.2026