DOI: https://doi.org/10.15276/ict.02.2025.43
УДК 004.451:502/504

Екодизайн програмного забезпечення: від вимог до експлуатації

Попова Марія Олександрівна[1)]
Канд. еконіч. наук, доцент каф. Інженерії програмного забезпечення
ORCID: https://orcid.org/0000-0001-6836-3690; popova.m.o@op.edu.ua
Любченко Віра Вікторівна[1)]
Д-р техніч. наук, професор каф. Інженерії програмного забезпечення
ORCID: https://orcid.org/0000-0002-4611-7832; lvv@op.edu.ua. Scopus Author ID: 56667638800
[1)] Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

АНОТАЦІЯ

У роботі розглянуто концепцію екодизайну програмного забезпечення як обов'язкової складової сучасної інженерії програмного забезпечення (ПЗ). Обґрунтовано, що зростання масштабів цифрових систем, хмарних обчислень та використання штучного інтелекту формує нові виклики, пов'язані з енергетичним слідом і викидами парникових газів. Навіть мінімальна оптимізація програмних рішень здатна забезпечити суттєвий ефект на глобальному рівні. Автори підкреслюють важливість переходу від декларативних підходів до вимірюваних метрик, таких як інтенсивність вуглецю, енергоспоживання та обсяг переданих даних. Метою дослідження є формування цілісної рамки екодизайну, яка охоплює всі етапи життєвого циклу ПЗ – від постановки вимог і архітектурних рішень до реалізації, тестування, розгортання та експлуатаційного моніторингу. Основна увага приділяється двом аспектам: раціональному вимірюванню сталості з використанням прозорих метрик і керованим компромісам між продуктивністю, зручністю користування та екологічною ефективністю. У роботі систематизовано ключові практики: включення сталості до нефункціональних вимог; архітектурні рішення зі зменшення передачі даних і використання обчислень на периферії мережі; оптимізація алгоритмів і ресурсів коду; фронтенд-підходи до скорочення «важкого» контенту та сторонніх скриптів; інтеграція енергопрофілювання у процеси CI/CD; використання автоскейлінгу, «нічних пауз» та carbon-aware scheduling на етапі експлуатації. Особливе місце відведено роботі з даними та штучним інтелектом, де ключовими є політика економного зберігання, стискання, оптимізація моделей та баланс між точністю й енергетичною вартістю. Практичний аналіз доводить, що найбільший ефект досягається на ранніх стадіях, коли сталість інтегровано у вимоги та архітектуру. Водночас автори підкреслюють необхідність уникати догматичного підходу, роблячи компроміси прозорими й аргументованими. У висновках визначено перспективи подальших досліджень: удосконалення методів оцінювання, створення довідників архітектурних патернів для різних доменів та інтеграція екодизайну в освітні програми інженерії ПЗ.

Ключові слова: екодизайн програмного забезпечення; сталий розвиток у програмній інженерії; архітектура програмних систем; нефункціональні вимоги; оптимізація алгоритмів; управління даними; штучний інтелект; обчислення з урахуванням викидів вуглецю

Актуальність. Тема екодизайну програмного забезпечення (ПЗ) стрімко виходить за межі «модного» тренду і перетворюється на обов'язкову координату інженерного мислення. Сучасні цифрові системи охоплюють мільярди користувачів, а це означає, що навіть невелике зменшення споживання енергії одним застосунком масштабовано дає відчутний ефект. Популярність хмарних обчислень, інтенсивне використання моделей штучного інтелекту, вибухове збільшення обсягу даних і потоків відео тягне за собою не лише технічні виклики, а й відповідальність за енергетичний слід. Програмні архітектури, алгоритми, дані та інфраструктура утворюють єдину екосистему, у якій рішення на рівні коду й архітектури прямо впливають на викиди парникових газів під час експлуатації [1]. Саме тому екодизайн, тобто проєктування з урахуванням енергетичної, обчислювальної та вуглецевої ефективності, стає невід'ємною частиною якісної інженерії ПЗ.

На практиці ми бачимо дві важливі тенденції:

– зміщення фокуса від суто інфраструктурної «зеленості» до програмної, коли увага приділяється не лише дата-центрам і мережам, а й самим застосункам, їхнім даним і алгоритмам [2–4];

– перехід від декларацій до вимірюваності, коли цілі сталості підкріплюються метриками на кшталт інтенсивності вуглецю, споживання енергії та обсягу переданих даних [5].

Водночас вимірюваність ставить перед інженерами низку компромісів: продуктивність проти енергоощадності, зручність користувача проти «важких» мультимедійних елементів,

точність модельних обчислень проти їхньої обчислювальної вартості [6]. Розв'язання цих компромісів потребує системного підходу – від етапу формулювання вимог до фаз розгортання та експлуатації.

Мета дослідження полягає у висвітленні принципів екодизайну ПЗ крізь увесь життєвий цикл розроблення програмного забезпечення і у формулюванні практичної рамки, яка допомагає вбудувати сталість у вимоги, архітектуру, реалізацію, тестування, розгортання та експлуатаційний моніторинг. Дослідження фокусує увагу на двох аспектах, а саме на раціональній інженерній практиці вимірювання з прозорими метриками та на керованих компромісах, що зберігають баланс між якістю обслуговування і екологічним слідом.

У сучасній літературі сформувалося кілька ліній аргументації. Перша підкреслює роль вимог як місця, де сталість слід зафіксувати як нефункціональні вимоги: енергоспоживання на транзакцію, обсяг переданих даних, цільові показники вуглецевого сліду у типовій сесії користувача [6]. Друга лінія стосується архітектурних рішень: вибір між монолітом і мікросервісами, розміщення обчислень ближче до користувача на периферії мережі, продумана кешувальна політика та контроль версій даних [3, 4]. Третя – дисципліна розроблення інтерфейсів: мінімізація «важкого» контенту, відповідальне завантаження ресурсів, продуманий дизайн, який зменшує зайві переходи і взаємодії [3, 4]. Четверта – керування даними і штучним інтелектом: економне збирання вимірювань, стиснення і зберігання даних, уважний підбір і оптимізація моделей, коли спрощена або стиснена модель задовольняє функціональні цілі без надмірної обчислювальної вартості [7]. Нарешті, важливою стає експлуатаційна перспектива: моніторинг метрик енергії та трафіку, «вуглецево-обізнане» планування пакетних завдань, розгортання в регіонах і часових вікнах із вищою часткою відновлюваної енергетики [1, 8].

Практичний аналіз показує, що найбільший вплив на зменшення сліду забезпечують ранні рішення. Розглянемо, які рішення можуть бути застосовані на різних етапах процесу розробки ПЗ, схему якого показано на рис. 1.

Рис. 1. **Схема процеса**

Коли сталість вводиться у вимоги як нефункціональні вимоги, команда має спільну мову для компромісів і критеріїв приймання. Умовно кажучи, коли «енергоінтенсивність на запит» чи «обсяг даних на сторінку» фіксуються поруч із показниками швидкодії, вони перестають бути «приємними доповненнями» і стають справжніми обмеженнями проєкту. Архітектурно найбільший ефект спостерігається від зменшення зайвих передач даних, від збільшення долі кешування і від свідомого проектування життєвого циклу ресурсів. На рівні коду стабільні виграші дають спрощення критичних алгоритмів, уникнення непотрібної серіалізації, ліниве завантаження ресурсів і усунення «гарячих точок» у шляхах виконання. У фронтенді суттєво допомагає оптимізація зображень і відео, відкладене завантаження, обережне використання анімацій та обмеження сторонніх скриптів, які непомітно роздувають сторінки. На етапі експлуатації додає ефекту коректне масштабування, використання автоскейлінгу за реальним навантаженням, вимкнення «нічних» середовищ, а також планування обчислювально важких завдань у вікна з кращим енергетичним міксом. Сукупно такі кроки знижують споживання енергії без відчутної втрати якості для

користувача, якщо з самого початку задати правильні рамки. Опис розподілу практик узагальнено в таблиці 1.

Таблиця 1. **Розподіл практик по етапах процесу розробки ПЗ**

№	Етап	Рекомендовані практики
1	Вимоги	1. Зафіксувати сталість як нефункціональні вимоги поряд із продуктивністю і надійністю. 2. Ввести вимірювані критерії приймання: «енергоінтенсивність на запит», «обсяг даних на сторінку». 3. Узгодити явні компроміси (якість/швидкодія vs енергоефективність) і «бюджети» ресурсів.
2	Архітектура	1. Мінімізувати зайві передачі даних; підвищувати кеш-хіти, продумати політику кешування. 2. Свідомо проєктувати життєвий цикл ресурсів (створення, повторне використання, звільнення). 3. Розміщувати обчислення ближче до користувача (edge), уникати «чатуночних» залежностей.
3	Реалізація коду	1. Спрощувати критичні алгоритми; прибирати непотрібну серіалізацію. 2. Застосовувати лініве завантаження ресурсів; усувати «гарячі точки» у шляхах виконання. 3. Тримати залежності легкими; уникати зайвих перетворень форматів даних.
4	Фронтенд	1. Оптимізувати зображення/відео; вмикати відкладене завантаження. 2. Обережно використовувати анімації; обмежувати сторонні скрипти, що «роздувають» сторінки. 3. Проєктувати сценарії з меншим числом кроків і переходів для користувача.
5	Тестування & CI/CD	1. Додавати перформанс/енерго-профілювання до тестів; відстежувати трафік і час виконання. 2. Впровадити «зелену нотатку» в Pull Request: опис впливу змін на дані/кеш/навантаження. 3. Запровадити «бюджети» сторінок/ендпоінтів; зупиняти зміни, що перевищують ліміти.
6	Розгортання & Експлуатація	1. Коректно масштабувати; вмикати автоскейлінг за реальним навантаженням. 2. Вимикати «нічні»/непотрібні середовища; прибирати «забуті» ресурси. 3. Планувати важкі задачі у вікна з кращим енергетичним міксом (carbon-aware scheduling).
7	Моніторинг & Поліпшення	1. Безперервно відстежувати енергію, трафік, затримки; порівнювати версії за метриками. 2. Мінімізувати вимірювання до корисного мінімуму; впроваджувати політику економного зберігання. 3. Проводити A/B-експерименти з урахуванням енерговитрат та впливу на UX.

У ході дослідження було виявлено, що екодизайн працює тоді, коли він не існує «поруч» із процесом, а вшитий у нього. У практиці доцільно мислити двома контурами. Перший – стратегічний, коли команда завчасно визначає орієнтири: які показники вимірюємо, як і де; як будемо приймати компроміси; хто і на якому етапі відповідає за сталість. Другий – тактичний, коли кожне рутинне рішення має «зелену» альтернативу за замовчуванням: у тестах, у рев'ю коду, в архітектурних радах, у налаштуваннях безперервної інтеграції та розгортання. Коли запит на новий віджет інтерфейсу розглядається не лише з погляду досвіду користувача, а й з точки зору обсягу додаткових запитів, ваги зображень, тривалості сценарію навантаження – екодизайн стає інтуїтивною звичкою, а не разовою ініціативою.

Водночас не варто перетворювати сталість на догму. В окремих задачах компроміс на користь якості чи безпеки цілком виправданий. Ключ у тому, щоб ці компроміси були явними, перевіреними даними й обговореними в команді. Тут допомагає прозоре вимірювання: порівняння версій застосунку за енергоспоживанням, обсягом трафіку і часовими профілями, а також відстеження трендів у процесі створення готового продукту.

Особливе місце в екодизайні посідає робота з даними і штучним інтелектом. Часто дані збираються «про запас», проте їх реальна цінність невисока, а вартість зберігання й обробки значна. Виважена політика зберігання, анонімізації та агрегації зменшує навантаження без втрати інсайтів. У сфері ШІ доцільно оцінювати не лише точність моделей, а й їхню енергетичну вартість у цільовому сценарії. Стиснення, квантована інференція, віддалене обчислення «за розкладом», кешування відповідей у передбачуваних кейсах – це способи досягти потрібного результату без надмірної ціни. У поєднанні з «вуглецево-обізнаним» плануванням пакетних задач це дає реальні скорочення експлуатаційного сліду.

Нарешті, важливо поєднувати екодизайн із етикою і доступністю. Коли інтерфейс не перевантажений, користувач робить менше зайвих дій, швидше досягає мети, менш стомлюється, а пристрій споживає менше енергії. Принцип «менше – краще» тут працює і для людей, і для навколишнього середовища. Додатковим бонусом стає економія коштів: скорочення трафіку і навантаження зменшує рахунки за хмарні ресурси, а це робить зелений підхід вигідним навіть поза екологічним контекстом.

Висновки. Таким чином нами обґрунтовано необхідність екодизайну програмного забезпечення як цілісної практики, показано, що найбільший ефект забезпечують ранні інженерні рішення у вимогах та архітектурі, а також наведено аргументи на користь вимірюваності й прозорих компромісів упродовж усього життєвого циклу розроблення програмного забезпечення. Зроблено акцент на практичних підходах до інтерфейсів, даних, алгоритмів і експлуатації, які спільно формують контрольований екологічний профіль системи.

Перспективи відкриваються в трьох напрямах. Перший – поглиблення методів оцінювання, щоби енергетичні та вуглецеві метрики стали такими ж звичними, як продуктивність і надійність. Другий – розроблення архітектурних патернів і довідників компромісів для типових доменів, від медіаплатформ до фінансових технологій й систем з ШІ. Третій – інтеграція екодизайну в освітні програми інженерії ПЗ, щоб майбутні фахівці сприймали сталість як базову інженерну компетенцію, а не разову ініціативу ентузіастів.

СПИСОК ЛІТЕРАТУРИ

1. “ISO/IEC 21031:2024. Information technology – Software Carbon Intensity (SCI) specification”. *International Organization for Standardization/* 2024. – URL: https://www.iso.org/standard/86612.html.

2.“Web Sustainability Guidelines (WSG)”. *W3C*. 2025. – URL: https://www.w3.org/TR/2025/DNOTE-web-sustainability-guidelines-20251002.

3. Greenwood T. “Sustainable Web Design”. *New York, NY, USA: A Book Apart*. 2021.

4. Calero C., Piattini M. “Green in Software Engineering”. *Cham, Switzerland: Springer*. 2015. DOI: https://doi.org/10.1007/978-3-319-08581-4.

5. ISO/IEC 30134-2:2016. “Information technology – Data centres – Key performance indicators”. Part 2: “Power usage effectiveness (PUE)”. *International Organization for Standardization,* 2016. – URL: https://www.iso.org/standard/63451.html.

6. West K. “Exploring the potential of carbon-aware execution for scientific workflows”. *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2503.13705.

7. Schwartz R., Dodge J., Smith N. A., Etzioni O. “Green AI.” *Communications of the ACM*. 2020; 63 (12): 54–63. DOI: https://doi.org/10.1145/3381831.

8. “HTTP Archive”. *Web Almanac. Part III. Chapter 15. Sustainability*. 2024. – URL: https://almanac.httparchive.org/en/2024/sustainability

DOI: https://doi.org/10.15276/ict.02.2025.43
УДК 004.451:502/504

Software ecodesign: from requirements to operation

Mariia O. Popova[1)]
PhD, Associate Professor of the Department of Software Engineering
ORCID: https://orcid.org/0000-0001-6836-3690; popova.m.o@op.edu.ua
Vira V. Liubchenko[1)]
Doctor of Engineering Sciences, Professor of the Department of Software Engineering
ORCID: https://orcid.org/0000-0002-4611-7832; lvv@op.edu.ua. Scopus Author ID: 56667638800
[1)] Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine

ABSTRACT

The paper examines the concept of software eco-design as a crucial component of modern software engineering. It argues that the growing scale of digital systems, cloud computing, and artificial intelligence creates new challenges related to energy footprint and greenhouse gas emissions. Even minor optimizations in software solutions can yield significant global effects. The authors emphasize the importance of shifting from declarative approaches to measurable metrics such as carbon intensity, energy consumption, and data transfer volume. The study aims to establish a comprehensive eco-design framework that encompasses all stages of the software lifecycle, from requirements specification and architectural decisions to implementation, testing, deployment, and operational monitoring. Particular attention is devoted to two aspects: rational sustainability measurement through transparent metrics, and managed trade-offs between performance, usability, and environmental efficiency. The paper systematizes key practices: incorporating sustainability into non-functional requirements; architectural strategies for reducing data transfers and employing edge computing; algorithm and code optimization for resource efficiency; frontend approaches to minimize heavy content and third-party scripts; integration of energy profiling into CI/CD processes; and operational techniques such as autoscaling, night-time shutdowns, and carbon-aware scheduling. A special focus is given to data management and artificial intelligence, where efficient storage policies, data compression, model optimization, and balancing accuracy with computational cost are critical. Practical analysis demonstrates that the most significant impact is achieved at early stages, when sustainability is embedded into requirements and architecture. At the same time, the authors emphasize the importance of avoiding a dogmatic approach, ensuring that trade-offs remain transparent and evidence-based. The conclusions outline several perspectives for further research: advancing assessment methods so that energy and carbon metrics become as common as performance and reliability; developing reference architectural patterns and trade-off guidelines for various domains, from media platforms to financial technologies and AI systems; and integrating eco-design principles into software engineering education, ensuring that future professionals perceive sustainability as a fundamental engineering competence rather than an isolated initiative.

Keywords: Software eco-design; sustainable software engineering; software architecture; non-functional requirements; algorithm optimization; data management; artificial intelligence; carbon-aware computing.