УДК 004.4 DOI: 10.20998/2413-3000.2024.8.4

О. О. ЛИСЕНКО, І. В. КОНОНЕНКО

СПЕЦИФІКА ТА СКЛАДНІСТЬ МІГРАЦІЇ СТОРОННІХ ІНСТРУМЕНТІВ В ІТ ПРОЄКТАХ

Виконаний аналіз релевантності проблеми міграції сторонніх інструментів в ІТ проєктах, котрій притаманні регулярність та наявність викликів для розробників щодо прийняття рішень. Аналіз показує, що міграція сторонніх інструментів вимагає від розробників не лише технічних знань та навичок, але й глибокого розуміння стратегій управління міграцією, методів оцінки ризиків, та здатності інтегрувати нові інструменти в існуючі проєкти без негативного впливу на робочий процес. Розглянута безпосередньо сама необхідність міграції сторонніх інструментів у сфері ІТ, яка є важливим аспектом для забезпечення актуальності, ефективності, та інноваційності програмного забезпечення в умовах технологічного ландшафту, який швидко змінюється. Основна увага приділяється огляду сучасних досліджень та методологій, спрямованих на спрощення процесу міграції інструментів, зменшення витрат на розробку та підтримку, а також підвищення безпеки програмного забезпечення. Розглянуті пропозиції щодо комплексних підходів до управління міграціями, що охоплює використання автоматизованих систем для аналізу великих обсягів даних про історію змін у проєктах, оцінку ризиків, та ефективну комунікацію між усіма учасниками проєкту. Розглядається обґрунтування значущості міграції сторонніх інструментів для забезпечення сталого розвитку програмного забезпечення в умовах динамічного технологічного середовища. Зазначена необхідність подальших досліджень у цій галузі, спрямованих на розробку нових інструментів і методологій для оптимізації процесу міграції, з метою підвищення продуктивності розробників і забезпечення високої якості кінцевих програмних продуктів. Підкреслюється важливість систематичного та комплексного підходу до міграції сторонніх інструментів, що базується на детальному аналізі даних, глибокому розумінні ризиків, ефективній комунікації та застосуванні сучасних технологічних рішень.

Ключові слова: сторонні бібліотеки, сторонні інструменти, міграція сторонніх бібліотек, рекомендація сторонніх бібліотек, мультиметричний рейтинг, прийняття сторонніх бібліотек, вибір сторонніх бібліотек.

A. LYSENKO, I. KONONENKO

SPECIFICS AND COMPLEXITY OF THIRD-PARTY LIBRARY MIGRATIONS IN IT-PROJECTS

An analysis has been conducted on the relevance of the problem of migrating third-party tools in IT projects, which is characterized by regularity and presents challenges for developers in terms of decision-making. The analysis shows that the migration of third-party tools requires developers not only to have technical knowledge and skills but also a deep understanding of migration management strategies, risk assessment methods, and the ability to integrate new tools into existing projects without negatively impacting the workflow. The necessity of migrating third-party tools in the IT field, which is a crucial aspect for ensuring the relevance, efficiency, and innovation of software in a rapidly changing technological landscape, is directly considered. Main attention is given to the review of modern research and methodologies aimed at simplifying the process of tool migration, reducing development and support costs, and enhancing software security. Proposals regarding comprehensive approaches to managing migrations are considered, including the use of automated systems for analyzing large volumes of data about the history of changes in projects, risk assessment, and effective communication among all project participants. The justification for the significance of migrating third-party tools to ensure the sustainable development of software in a dynamically changing technological environment is discussed. The need for further research in this field is highlighted, aimed at developing new tools and methodologies to optimize the migration process, with the goal of enhancing developer productivity and ensuring the high quality of final software products. The importance of a systematic and comprehensive approach to migrating third-party tools, based on detailed data analysis, a deep understanding of risks, effective communication, and the application of modern technological solutions, is emphasized.

Keywords: library migration, third-party dependency migration, library recommendation, multi-metric ranking, library adoption, library selection.

Вступ. Сучасна ІТ індустрія характеризується швидкими темпами розвитку, що породжує необхідність у міграції сторонніх інструментів, включаючи технології, фреймворки, бібліотеки та API. Ці інструменти відіграють ключову роль у підвищенні ефективності та якості програмного забезпечення, однак постійна потреба в їх оновленні ставить перед фахівцями складні завдання. Проблема міграції тісно пов'язана з забезпеченням безпеки, оптимізацією витрат та підвищенням продуктивності розробників. Відповідно, ефективне управління процесом міграції є критично важливим для розвитку якісного програмного забезпечення.

Мета статті. Проведення огляду та аналізу сучасних підходів, методологій та інструментів у контексті міграції сторонніх інструментів у розробці ІТ проєктів. Стаття висвітлює ключові аспекти, проблеми та виклики, пов'язані з міграціями як процесами, а також узагальнює ефективні стратегії управління міграціями щодо підвищення продуктивності розробників, оптимізації витрат на розробку та забезпечення безпеки програмного продукту. Визначаються потенційні зони подальших досліджень.

Поширеність та необхідність міграцій сторонніх інструментів. На часі спостерігається висока динаміку розвитку у сфері технологій, особливо в ІТ індустрії, де однією із релевантних тем є постійні зміни та оновлення сторонніх допоміжних інструментів розробки ПЗ, що вже є звичайним явищем. До сторонніх інструментів відносяться технології, фреймворки, бібліотеки та API, котрим притаманне різноманіття та вирішення конкретних задач, використовуючи експертно перевірені підходи. У дослідженнях [1, 2] наголошується, що використання сторонніх інструментів з готовими до використання функціями є загальноприйнятою практикою в розробці програмного забезпечення вже тривалий час, що підвищує якість продукту та продуктивність розробки [3].

Використання сторонніх інструментів заохочує до практики повторного використання програмного коду та дає можливість розробникам зосередитись на

розробці функціональності програмного продукту. У роботі [2] зазначається, що використання інструментів є практикою для прискорення розробки програмних систем і, як наслідок, для зниження їх вартості [4]. Програмні інструменти надають розробникам спеціалізовану функціональність, яка є корисною під час розробки програмного забезпечення [5]. Розробникам не потрібно "винаходити колесо" кожного разу, а шукати інструменти, які відповідають їхнім цілям та очікуванням [6].

На практиці, сторонні інструменти інтегруються до проєктів у вигляді зовнішніх залежностей, котрі завантажуються на етапі компіляції проєкту через централізовані хостингові сервіси (наприклад, Maven Central [7], GitHub [8], NPM [9], PyPI [10]).

За ініціативи дослідників [1], котрими було знайдено джерело із статистичними даними [7] щодо числа публікацій нових інструментів у сервісі Maven Central, пропонується розглянути релевантний період - з 2015 до 2022 років на рис.1, де спостерігається зростання кількості інструментів з 364,961 до 2,251,027.

Published Packages by Year

2023	1,657,878
2022	2,251,027
2021	2,045,263
2020	1,435,839
2019	1,223,821
2018	929,167
2017	712,956
2016	542,799
2015	364,961

Рис. 1. Кількість опублікованих сторонніх інструментів у Maven Central сервісі [7] за період з 2015 по 2023 рр.

Такий показник підтверджує релевантність теми постійного характеру змін та оновлень сторонніх інструментів, а також підтеми – заміни сторонніх інструментів із часом. В той же час, статистичні дані очевидно свідчать про наявність конкуренції серед сторонніх інструментів, де колективи розробників та навіть компанії постачають інструменти у стані готовності до інтеграції у інші програмні проєкти.

У дослідженні [1] авторами визначається, наскільки поширеними є міграції інструментів. Статистичні дані на рис. 2 підтверджують широке застосування цієї практики.

За даними 19652 досліджених проєктів [1] на Java було виявлено, що для 41,04% проєктів характерним є принаймні одне видалення інструменту та для 28.72% проєктів - потенційно одна міграція інструменту. Міграції інструментів більш ймовірні серед проєктів з більшою кількістю зафіксованих змін у вигляді Git ревізій та набору залежностей. При розгляді середніх за розміром проєктів встановили, що проєкти з видаленнями інструментів мали одне видалення на 139 ревізій, а проєкти з міграціями мали від 2 до 4 міграцій в цілому.

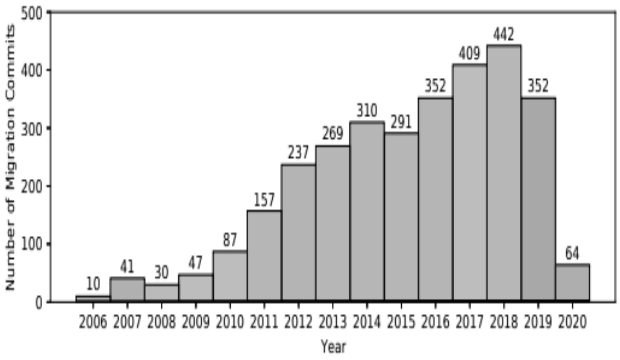

Рис. 2. Кількість міграцій сторонніх інструментів для проєктів [1] у період з 2006 по 2020 рр. [6]

Складність питання міграції сторонніх інструментів. Впровадження сторонніх інструментів створює унікальні виклики для розробників на всіх етапах життєвого циклу програмного забезпечення [11]. Зважаючи на широкий спектр доступних інструментів, навіть завдання вибору правильного інструменту для певної мети стає непростим [1, 2], в якому потрібно враховувати складні соціотехнічні фактори [12, 13]. При чому завдання вибору стороннього інструменту може бути при таких сценаріях:

- коли інструмент за призначенням впроваджується на проєкт вперше, та ще немає у проєкті альтернативних інструментів у вигляді залежностей;
- коли інструмент розглядається як заміна існуючому через застарілість останнього або соціотехнічні фактори.

Дослідницьким шляхом [2] було встановлено, що недостатньо якісний та суб'єктивний вибір стороннього інструменту може серйозно вплинути на програмний проєкт з точки зору витрат, часу та зусиль розробки, причому ступінь впливу залежить, серед іншого, від ролі інструменту в архітектурі програмного забезпечення. Та незважаючи на важливість слідування ретельному процесу вибору, на практиці вибір сторонніх інструментів здійснюється спонтанно, без належного та достатнього процесу, але, все ж таки, десятки факторів грають впливову роль у прийнятті рішення.

Може виникнути питання "Чи слід взагалі проводити заміни інструментів, якщо це не так легко та швидко?". Відповідь: так, безумовно, тому що за результатами дослідження [1] зростає стурбованість щодо ризиків використання застарілих інструментів, бо вони можуть містити або впливати на безпекові вразливості та невирішені проблеми або породжувати їх [14, 15, 16. 17], що викликає сумніви у звичній стратегії "Якщо все працює, не чіпайте". Також, відомо, що невдачі або невідповідності із інструментом можуть все ж таки відбуватися та не буде вважатися достатнім лише оновлення версії інструменту [1]. У таких випадках, інструмент необхідно повністю видалити та замінити іншим, що в

літературі називається міграцією інструментів [18, 19, 20, 21, 22].

Тобто, як висновок, існує щонайменше два напрямки щодо дослідницького руху:

- вивчення характеристик, причин та чинників щодо оновлення інструментів [23, 24, 25]. Вже існує велика кількість досліджень стосовно характеристик та розуміння прийняття інструментів [13, 26, 27, 28, 29, 30, 31, 32] та оновлень інструментів [17, 23, 25, 33, 34, 35, 36, 37, 38, 39, 40];

- пошук та пропонування промислових рішень [41, 42, 43], які прагнуть підтримувати актуальність та вільність від вразливостей. Дослідження та розвиток щодо міграції інструментів [20, 21, 22, 23, 44, 45] тривають, але отримані результати ще є фрагментарними та неповними, та потребують систематизації або доповнення чи покращення.

Комунікація як складова міграції сторонніх інструментів. У дослідженні [1] зазначається, що процес міграції сторонніх інструментів складається із трьох необхідних етапів робіт: обґрунтування необхідності міграції, знаходження найкращого інструменту та модифікація коду для використання нового інструменту. Невід'ємною частиною перших 2х етапів є проведення дискусій між розробниками в системах відстеження проблем [20], оскільки це дає змогу полегшувати роботу через оцінювання очікуваних витрат та обмін знаннями щодо переваг. Але такі обговорення можуть не призвести до міграції, якщо не досягнуто консенсусу або сприйняті переваги не переважають витрати. За результатами дослідження [20] було встановлено, що серед 49 спроб міграції інструменту логування в проєктах ASF, лише у 14 випадках є відмова від міграції через неможливість досягнення консенсусу серед розробників та менеджерів щодо необхідності міграції (у 4 проєктах із 14 або 28%) та через невиконання необхідних змін у коді проєкту протягом міграції (у 6 проєктах із 14 або 42%).

Фактори та причини міграцій сторонніх інструментів. Під час обговорення розробниками переваг кожного із кандидатів на міграцію інструменту, у дослідженні [1] приводять 14 різних факторів. Найчастішими вважаються відсутність обслуговування, функціональності, зручності використання, вимоги проєкту та спрощення залежностей. Міграція може відбуватись тому, що інструмент є застарілим та/або не має вже підтримки, і проєкт вирішує використовувати інший інструмент, який рекомендується та підтримується спільнотою. У дослідженні [21] згадується "застарілість" як причина для міграції, але автори [1] додатково підкреслюють, що основна турбота полягає у з'ясуванні, чи інструмент все ще підтримується, та визнанні відсутності підтримки (наприклад, поява оголошення про завершення терміну служби), яка часто стимулює міграції.

Найчастіше згадувана причина у заміні інструменту полягає в тому, що новий інструмент є зручнішим у використанні, має приємне API, призводить до більш чистого коду, зрозумілішого налаштування, читабельних тестів або є гнучким для розробників, щоб за потреби змінювати реалізації. Велика кількість міграцій також відбувається тому, що інструмент має специфічні функції, які попередній аналог не може надати. Поширені випадки міграції включають: міграцію з однієї версії на іншу версію [32], з одного API на інше API [46], з однієї мови програмування на іншу мову програмування [47], з однієї платформи на іншу платформу [48], або з одного інструменту на інший [19, 20, 21, 22].

Найпоширенішою причиною є необхідність інтеграції інструменту з іншими компонентами проєкту для досягнення конкретних цілей, уникнення проблем або забезпечення сумісності [1]. Деякі міграції відбуваються для спрощення використання інструментів в проєкті, досягнення консистентного стилю або очищення від непотрібних залежностей, як запобіжний захід для контролю складності проєкту, усунення технічного боргу та, можливо, зменшення зусиль, необхідних для майбутнього обслуговування [20]. Авторами досліджень [1, 2] було запропоновано згрупувати фактори у три категорії (див. табл. 1).

Таблиця 1– Факторні групи, які враховуються при заміні інструментів розробки

технічні	людські	економічні фактори
- функціональність - якісні атрибути - тип проєкту - процес випуску	- зацікавлені сторони - організація - індивідуальні - спільнота	- вартість володіння - ризик

Проте, часто непросто знайти гарний цільовий сторонній інструмент або вибрати серед численних кандидатів [47, 48, 49]. Списки, створені спільнотою, такі як awesome-java [50] та AlternativeTo [51], часто бувають неінформативними та містять бібліотеки низької якості, тоді як блоги та статті зазвичай опираються на особисті думки і застарілі [48]. Легкодоступні метрики, такі як популярність та частота оновлень, мають обмежену корисність, і вони варіюються залежно від сфери [49]. На практиці, проєкти в промисловості часто покладаються на думку експертів у галузі для прийняття рішень про міграцію [52], тоді як у відкритих проєктах міграція бібліотек відбувається лише тоді, коли основні розробники досягають консенсусу в обговореннях [53]. У будь-якому випадку, міграція не гарантує що буде відповідною, економічно вигідною або корисною для проєкту.

Підтримка рішення при міграції сторонніх інструментів. Дослідники запропонували декілька підходів для пошуку факту здійснення міграцій бібліотек, аналізуючи вибірки даних щодо розробки – історії змін коду [54, 55, 56] у вигляді ревізій, котрі розташовуються у GitHub сервісі. Спочатку

відбувається пошук фактів здійснених міграцій серед деякої кількості проєктів. На базі знайдених фактів далі формуються асоціативні правила, котрі у подальшому експертно фільтруються та фільтруються на основі частоти їх появи [54] або пов'язаних змін коду [56]. Однак, запропоновані підходи страждають від низького охоплення [54, 56], або низької точності [54, 55] з двох причин: визначається глобальний поріг фільтрації (котрий не є ефективним для всіх сценаріїв міграції) та цінні джерела інформації не враховуються в метриках фільтрації. Корисність наведених підходів обмежена, оскільки низька точність призводить до високих зусиль людини при інспектуванні, а низьке охоплення перешкоджає розробникам у прийнятті оптимальних рішень, оскільки деякі можливості міграції можуть бути пропущені.

Для покращення ефективності цих підходів [54, 55, 56], авторами роботи [57] було запропоновано новий підхід для автоматизованого рекомендування цільових бібліотек міграції з історій існуючої розробки програмного забезпечення. На відміну від фільтрації, автори зосереджуються на ранжуванні, оскільки відносні позиції ранжування більш стійкі до змін метрик та параметрів. Згідно підходу [57] відбувається пошук кандидатів на зміну бібліотеки серед послідовностей, котрі описують зміни залежностей у проєкті, аналізуючи велику кількість історій розробки програмного забезпечення. Після цього кандидати ранжуються на основі комбінації чотирьох ретельно розроблених метрик: підтримка правил, підтримка повідомлень, підтримка відстані та підтримка API. Метрики розроблені для виявлення різних джерел доказів з даних та визначення ймовірних цілей міграції на основі цих доказів. Нарешті, релевантні цільові бібліотеки, їх метрики та відповідні випадки міграції повертаються для людської інспекції.

Серед опублікованої значної кількості досліджень у сфері рекомендацій сторонніх бібліотек [58, 59, 60], переважно використовується метод пошуку асоціативних правил [61]. Ефективність цієї техніки у рекомендаціях бібліотек значно залежить від ймовірності їх спільного використання. Автори дослідження [60] розробили систему для рекомендації інструментів на основі їх спільного використання у певному наборі проєктів. Цей підхід передбачає, що команда розробників вже знайома з деякими необхідними бібліотеками і лише потребує визначення альтернатив до них. Однак на практиці вони можуть мати обмежені знання про сторонні бібліотеки.

Початкове припущення не завжди відображає реальність програмних проєктів. Не гарантовано, що команда розробників буде знайома з усіма типами сторонніх бібліотек. Навіть якщо вони мають деякі знання, вони можуть варіюватися за глибиною. Багато сторонніх бібліотек використовуються не тільки з однією метою або особливістю. Тому існує нагальна потреба у системі рекомендацій, заснованій на більш загальному припущенні, для допомоги розробникам.

Технологія пошуку тексту пропонує рішення цієї проблеми, висвітлюючи підхід дослідників [60]. Як об’єкти дослідження автори аналізували опис програмних продуктів та опис інструментів розробки через пошук текстів. Далі за знайденими вибірками встановлювались зв’язки, котрі створюють базу для покращення рекомендацій бібліотек. Тобто підхід використовує багатий інформаційний зміст описів для створення більш обізнаної та ефективної системи рекомендацій.

Метод автоматизованого аналізу тексту включає комбінацію Латентного розподілення Діріхле (LDA) [62] та колаборативної фільтрації [63, 64, 65] для формування рекомендацій щодо вибору сторонніх бібліотек у розробці програмного забезпечення. Латентне розподілення Діріхле є генеративною ймовірнісною моделлю для аналізу колекцій дискретних даних, наприклад, текстових корпусів. Ця модель представляє собою трьохрівневу ієрархічну баєсову модель, де кожен елемент колекції розглядається як скінченна суміш різних тем. З іншого боку, колаборативна фільтрація — це процес відбору інформації чи шаблонів, що здійснюється за допомогою методів, що базуються на взаємодії множини агентів. У рамках цієї фільтрації програмне забезпечення виступає як окрема сутність, яку порівнюють з іншими сутностями в наборі даних для створення списку найбільш подібних сутностей, заснованого на метриці відстані. На основі цього аналізу формуються рекомендації для цільової сутності, засновані на подібності з іншими сутностями.

Висновки. Сторонні інструменти є невід'ємною частиною сучасної розробки програмного забезпечення. Вони сприяють підвищенню якості продукту та ефективності розробки, дозволяючи розробникам зосередитися на створенні функціональності. Вибір та міграція сторонніх інструментів є складним процесом, що вимагає врахування різноманітних соціотехнічних факторів. Неправильний вибір може негативно вплинути на проєкт, збільшуючи витрати та зусилля на розробку.

Ефективна комунікація між розробниками є ключовою для успішної міграції сторонніх інструментів. Обговорення та обмін знаннями сприяють знаходженню найкращих рішень та зменшенню ризиків, пов'язаних з міграцією. Аналіз виявив потенційну зону щодо покращення процесу комунікації між розробниками інтегруючи деякі техніки гнучких методологій управління проєктами.

Використання автоматизованих систем рекомендацій, заснованих на аналізі історії змін коду та метриках, може поліпшити процес вибору та міграції сторонніх інструментів. Це дозволяє розробникам приймати більш обґрунтовані рішення. Аналіз виявив потенційну зону щодо впровадження альтернативних підходів, котрі ґрунтуються на використанні закритих баз знань та принципів їх підтримки.

У світі, де технології швидко розвиваються, розробникам необхідно постійно оновлювати свої знання та вміння, щоб ефективно вибирати та інтегрувати найкращі сторонні інструменти у свої проєкти. Існує впевненість щодо ефективності використання закритих баз знань, оскільки усі експертні знання проходять декілька етапів перевірок та як результат відповідають стандартам та практикам розробки корпорації.

Список літератури

1. He H., He R., Gu H., Zhou M. A Large-Scale Empirical Study on Java Library Migrations: Prevalence, Trends, and Rationales. *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)* (Athens, Greece). New York, NY, USA: Association for Computing Machinery, 2021. P. 478–490.
2. Larios Vargas E., Aniche M., Treude C., Bruntink M., Gousios G. Selecting Third-Party Libraries: The Practitioners' Perspective. *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020*, Virtual Event, USA. New York, NY, USA: ACM, 2020. 12 pages. Available from: https://doi.org/10.1145/3368089.3409711
3. Mohagheghi Parastoo, Conradi Reidar. Quality, Productivity and Economic Benefits of Software Reuse: A review of Industrial Studies. *Empir. Softw.* Eng. 2007. Vol. 12, No. 5. P. 471–516. Available from: https://doi.org/10.1007/s10664-007-9040-x
4. Mojica I. J., Adams B., Nagappan M., Dienst S., Berger T., Hassan A. E. A Large-Scale Empirical Study on Software Reuse in Mobile Apps. *IEEE Software. March 2014*. Vol. 31, No. 2. P. 78–86. Available from: https://doi.org/10.1109/MS.2013.142
5. Li M., Wang W., Wang P., Wang S., Wu D., Liu J., Xue R., Huo W. LibD: Scalable and Precise Third-Party Library Detection in Android Markets. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017. P. 335–346. Available from: https://doi.org/10.1109/ICSE.2017.38
6. Nguyen P. T., Di Rocco J., Di Ruscio D., Di Penta M. CrossRec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software*. 2020. Vol. 161. Article 110460. Available from: https://doi.org/10.1016/j.jss.2019.110460
7. MvnRepository. *Maven Central Repository* [Electronic resource]. 2021. Available from: https://mvnrepository.com/repos/central
8. *Github* [Electronic resource]. Available at: https://github.com/
9. *npm, Inc. npm | Build amazing things* [Electronic resource]. 2021. Available at: https://www.npmjs.com/
10. *Python Software Foundation. PyPI: the Python package index* [Electronic resource]. 2021. Available at: https://pypi.org/
11. Cox Russ. Surviving software dependencies. *Commun. ACM*. 2019. Vol. 62, No. 9. P. 36-43. Available from: https://doi.org/10.1145/3347446
12. Pano Amantia, Graziotin Daniel, Abrahamsson Pekka. Factors and actors leading to the adoption of a JavaScript framework. *Empir. Softw. Eng*. 2018. Vol. 23, No. 6. P. 3503–3534. Available from: https://doi.org/10.1007/s10664-018-9613-x
13. Larios Vargas Enrique, Aniche Maurício Finavaro, Treude Christoph, Bruntink Magiel, Gousios Georgios. Selecting third-party libraries: the practitioners' perspective. *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. ACM, 2020. P. 245–256. Available from: https://doi.org/10.1145/3368089.3409711
14. Alfadel Mahmoud, Costa Diego Elias, Shihab Emad. Empirical Analysis of Security Vulnerabilities in Python Packages. *28th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2021), Honolulu, HI, USA, March 9-12, 2021*. IEEE, 2021. P. 446–457. Available from: https://doi.org/10.1109/SANER50967.2021.00048
15. Decan Alexandre, Mens Tom, Constantinou Eleni. On the impact of security vulnerabilities in the npm package dependency network. *Proceedings of the 15th International Conference on Mining Software Repositories (MSR 2018), Gothenburg, Sweden, May 28-29, 2018*. ACM, 2018. P. 181–191. Available from: https://doi.org/10.1145/3196398.3196401
16. Pashchenko Ivan, Plate Henrik, Ponta Serena Elisa, Sabetta Antonino, Massacci Fabio. Vulnerable open source dependencies: counting those that matter. *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2018), Oulu, Finland, October 11-12, 2018*. ACM, 2018. P. 42:1–42:10. Available from: https://doi.org/10.1145/3239235.3268920
17. Zimmermann Markus, Staicu Cristian-Alexandru, Tenny Cam, Pradel Michael. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. *28th USENIX Security Symposium, USENIX Security 2019*, Santa Clara, CA, USA, August 14-16, 2019. USENIX Association, 2019. P. 995–1010.
18. He Hao, Xu Yulin, Cheng Xiao, Liang Guangtai, Zhou Minghui. MigrationAdvisor: Recommending Library Migrations from Large-Scale Open-Source Data. *43rd IEEE/ACM International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 2021. P. 9–12. Available from: https://doi.org/10.1109/ICSE-Companion52605.2021.00023
19. He Hao, Xu Yulin, Ma Yixiao, Xu Yifei, Liang Guangtai, Zhou Minghui. A Multi-Metric Ranking Approach for Library Migration Recommendations. *28th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021, Honolulu, HI, USA, March 9-12, 2021*. IEEE, 2021. P. 72–83. Available from: https://doi.org/10.1109/SANER50967.2021.00016
20. Kabinna Suhas, Bezemer Cor-Paul, Shang Weiyi, Hassan Ahmed E. Logging library migrations: A case study for the Apache Software Foundation projects. *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*. ACM, 2016. P. 154–164. Available from: https://doi.org/10.1145/2901739.2901769
21. Teyton Cédric, Falleri Jean-Rémy, Blanc Xavier. Mining Library Migration Graphs. 19*th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012*. IEEE Computer Society, 2012. P. 289–298. Available from: https://doi.org/10.1109/WCRE.2012.38
22. Teyton Cédric, Falleri Jean-Rémy, Palyart Marc, Blanc Xavier. A study of library migrations in Java. J. Softw. *Evol. Process*. 2014. Vol. 26, No. 11. P. 1030–1052. Available from: https://doi.org/10.1002/smr.1660
23. Bavota Gabriele, Canfora Gerardo, Di Penta Massimiliano, Oliveto Rocco, Panichella Sebastiano. How the Apache community upgrades dependencies: an evolutionary study. *Empir. Softw. Eng*. 2015. Vol. 20, No. 5. P. 1275–1317. Available from: https://doi.org/10.1007/s10664-014-9325-9
24. Kula Raula Gaikovina, Germán Daniel M., Ouni Ali, Ishio Takashi, Inoue Katsuro. Do developers update their library dependencies? - An empirical study on the impact of security advisories on library migration. *Empir. Softw. Eng.* 2018. Vol. 23, No. 1. P. 384–417. Available from: https://doi.org/10.1007/s10664-017-9521-5
25. Zerouali Ahmed, Constantinou Eleni, Mens Tom, Robles Gregorio, González-Barahona Jesús M. An Empirical Analysis of Technical Lag in npm Package Dependencies. *New Opportunities for Software Reuse - 17th International Conference, ICSR 2018*, Madrid, Spain, May 21-23, 2018, Proceedings. Springer, 2018. P. 95–110. Available from: https://doi.org/10.1007/978-3-319-90421-4_6
26. López de la Mora Fernando, Nadi Sarah. An Empirical Study of Metric-based Comparisons of Software Libraries. *Proceedings of the 14th International Conference on Predictive Models and*

Data Analytics in Software Engineering, PROMISE 2018, Oulu, Finland, October 10, 2018. ACM, 2018. P. 22–31. Available from: https://doi.org/10.1145/3273934.3273937
27. Kavaler David, Trockman Asher, Vasilescu Bogdan, Filkov Vladimir. Tool choice matters: JavaScript quality assurance tools and usage outcomes in GitHub projects. *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 2019. P. 476–487. Available from: https://doi.org/10.1109/ICSE.2019.00060
28. Lamba Hemank, Trockman Asher, Armanios Daniel, Kästner Christian, Miller Heather, Vasilescu Bogdan. Heard it through the Gitvine: an empirical study of tool diffusion across the npm ecosystem. *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. ACM, 2020. P. 505–517. Available from: https://doi.org/10.1145/3368089.3409705
29. Ma Yuxing, Mockus Audris, Zaretzki Russell, Bichescu Bogdan, Bradley Randy. A Methodology for Analyzing Uptake of Software Technologies Among Developers. *IEEE Transactions on Software Engineering*. 2020. P. 1–1. Available from: https://doi.org/10.1109/TSE.2020.2993758
30. Pano Amantia, Graziotin Daniel, Abrahamsson Pekka. Factors and actors leading to the adoption of a JavaScript framework. *Empir. Softw. Eng.* 2018. Vol. 23, No. 6. P. 3503–3534. Available from: https://doi.org/10.1007/s10664-018-9613-x
31. Xu Bowen, An Le, Thung Ferdian, Khomh Foutse, Lo David. Why reinventing the wheels? An empirical study on library reuse and re-implementation. *Empir. Softw. Eng.* 2020. Vol. 25, No. 1. P. 755–789. Available from: https://doi.org/10.1007/s10664-019-09771-0
32. Yin Likang, Filkov Vladimir. Team Discussions and Dynamics During DevOps Tool Adoptions in OSS Projects. *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 2020. P. 697–708. Available from: https://doi.org/10.1145/3324884.3416640
33. Cogo Filipe Roseiro, Oliva Gustavo Ansaldi, Hassan Ahmed E. An empirical study of dependency downgrades in the npm ecosystem. *IEEE Transactions on Software Engineering*. 2019. Available from: https://doi.org/10.1109/TSE.2019.2952130
34. Cox Joel, Bouwers Eric, van Eekelen Marko C. J. D., Visser Joost. Measuring Dependency Freshness in Software Systems. *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015*, Volume 2. IEEE Computer Society, 2015. P. 109–118. Available from: https://doi.org/10.1109/ICSE.2015.140
35. Decan Alexandre, Mens Tom, Constantinou Eleni. On the Evolution of Technical Lag in the npm Package Dependency Network. *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 2018. P. 404–414. Available from: https://doi.org/10.1109/ICSME.2018.00050
36. Dietrich Jens, Pearce David J., Stringer Jacob, Tahir Amjed, Blincoe Kelly. Dependency versioning in the wild. *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, Montreal, Canada, May 26-27, 2019*. IEEE / ACM, 2019. P. 349–359. Available from: https://doi.org/10.1109/MSR.2019.00061
37. Kula Raula Gaikovina, Germán Daniel M., Ishio Takashi, Inoue Katsuro. Trusting a library: A study of the latency to adopt the latest Maven release. *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015*. IEEE Computer Society, 2015. P. 520–524. Available from: https://doi.org/10.1109/SANER.2015.7081869
38. Kula Raula Gaikovina, Germán Daniel M., Ouni Ali, Ishio Takashi, Inoue Katsuro. Do developers update their library dependencies? - An empirical study on the impact of security advisories on library migration. *Empir. Softw. Eng*. 2018. Vol. 23, No. 1. P. 384–417. Available from: https://doi.org/10.1007/s10664-017-9521-5
39. Mirhosseini Samim, Parnin Chris. Can automated pull requests encourage software developers to upgrade out-of-date dependencies?. *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*. IEEE Computer Society, 2017. P. 84–94. Available from: https://doi.org/10.1109/ASE.2017.8115621
40. Soto-Valero César, Benelallam Amine, Harrand Nicolas, Barais Olivier, Baudry Benoit. The emergence of software diversity in Maven Central. *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, Montreal, Canada, May 26-27, 2019*. IEEE / ACM, 2019. P. 333–343. Available from: https://doi.org/10.1109/MSR.2019.00059
41. Snyk Limited. Snyk | Developer security | *Develop fast. Stay secure* [Electronic resource]. 2021. Available from: https://snyk.io/
42. *WhiteSource Software. WhiteSource: Open Source Security and License Management Solution* [Electronic resource]. 2021. Available from: https://www.whitesourcesoftware.com/
43. GitHub, Inc. *GitHub Advisory Database* [Electronic resource]. 2021. Available from: https://github.com/advisories
44. Alrubaye Hussein, Alshoaibi Deema, AlOmar Eman Abdullah, Mkaouer Mohamed Wiem, Ouni Ali. How Does Library Migration Impact Software Quality and Comprehension? An Empirical Study. *Reuse in Emerging Software Engineering Practices - 19th International Conference on Software and Systems Reuse, ICSR 2020, Hammamet, Tunisia, December 2-4, 2020*, Proceedings. Springer, 2020. P. 245–260. Available from: https://doi.org/10.1007/978-3-030-64694-3_15
45. Bartolomei Thiago Tonelli, Czarnecki Krzysztof, Lämmel Ralf, van der Storm Tijs. Study of an API Migration for Two XML APIs. *Software Language Engineering, Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009*, Revised Selected Papers. Springer, 2009. P. 42–61. Available from: https://doi.org/10.1007/978-3-642-12107-4_5
46. Alrubaye Hussein, Mkaouer Mohamed Wiem, Ouni Ali. On the use of information retrieval to automate the detection of third-party Java library migration at the method level. *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 2019. P. 347–357. Available from: https://doi.org/10.1109/ICPC.2019.00053
47. Teyton C., Falleri J., Blanc X. Mining library migration graphs. *19th Working Conference on Reverse Engineering, WCRE 2012, Kingston*, ON, Canada, 2012.
48. Chen C., Gao S., Xing Z. Mining analogical libraries in Q&A discussions - Incorporating relational and categorical knowledge into word embedding. *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016*. Volume 1. IEEE Computer Society, 2016. P. 338–348.
49. de la Mora F. L., Nadi S. An empirical study of metric-based comparisons of software libraries. *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2018, Oulu, Finland, October 10, 2018*. 2018. P. 22–31.
50. *Awesome java: A curated list of awesome frameworks, libraries, and software for the Java programming language* [Electronic resource]. Available from: https://github.com/akullpp/awesome-java
51. *Alternativeto: Crowd-sourced software recommendations* [Electronic resource]. Available from: https://alternativeto.net/
52. Larios-Vargas E., Aniche M., Treude C., Bruntink M., Gousios G. Selecting third-party libraries: The practitioners' perspective. *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020).* 2020.
53. Kabinna S., Bezemer C., Shang W., Hassan A. E. Logging library migrations: A case study for the Apache Software Foundation projects. *Proceedings of the 13th International*

Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016. 2016. P. 154–164.
54. Huang E.H., Socher R., Manning C.D., Ng A.Y. Improving word representations via global context and multiple word prototypes // *Proc. 50th Annual Meeting of the Association for Computational Linguistics: Long Papers*. vol.1. 2012. P. 873–882.
55. Koren Y. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*. 2010. Vol. 4, No. 1, P. 1.
56. Mccarey F., Cinnéide M.O., Kushmerick N. Rascal: A recommender agent for agile reuse. *Artificial Intelligence Review*. 2005. Vol. 24, No. 3-4. P. 253–276.
57. Zhao X., Li S., Yu H., Wang Y., Qiu W. Accurate Library Recommendation Using Combining Collaborative Filtering and Topic Model for Mobile Development. *IEICE Transactions*. 2019. Vol. 102-D, No. 3. P. 522–536.
58. Chen C., Gao S., Xing Z. Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding. *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol.1. IEEE, 2016. P. 338–348.
59. Chen C., Xing Z. Similartech: Automatically recommend analogical libraries across different programming languages. *Proc. 31st IEEE/ACM International Conference on Automated Software Engineering. ACM*, 2016. P. 834–839.
60. Thung F., Lo D., Lawall J. Automated library recommendation. *20th Working Conference on Reverse Engineering (WCRE 2013): Proceedings: Koblenz, Germany, 14-17 Oct. 2013*. 2013. P. 182–191.
61. Agrawal R., Srikant R. *Fast algorithms for mining association rules*. Proc. 20th Int. Conf. Very Large Data Bases, VLDB, vol.1215. 1994. P. 487–499.
62. Blei D.M., Ng A.Y., Jordan M.I. Latent dirichlet allocation. *Journal of Machine Learning Research*. 2003. Vol. 3, No. Jan. P. 993–1022.
63. Koren Y. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*. 2010. Vol. 4, No. 1, P. 1.
64. Burke R. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*. 2002. Vol. 12, No. 4. P. 331–370.
65. Terveen L., Hill W. Beyond recommender systems: Helping people help each other. *HCI in the New Millennium*. Vol. 1. 2001. P. 487–509.

Надійшла (received) 15.01.2024

Відомості про авторів / About the Authors

Лисенко Олександр Олександрович (Lysenko Alexander) – аспірант, Національний технічний університет «Харківський політехнічний інститут», кафедра стратегічного управління, м. Харків, Україна, e-mail: alexander.lysenko.dev@gmail.com; ORCID: 0009-0009-4962-5881.

Кононенко Ігор Володимирович (Kononenko Igor) – доктор технічних наук, професор, Національний технічний університет "Харківський політехнічний інститут", професор кафедри стратегічного управління, м. Харків, Україна, Азербайджанський університет архітектури та будівництва, запрошений професор, м. Баку, Азербайджанська Республіка, e-mail: igorvkononenko@gmail.com; ORCID: http://orcid.org/0000-0002-1218-2791.