УДК 004.9
DOI https://doi.org/10.32838/2663-5941/2019.5-1/28

Струзік В.А.
Національний університет харчових технологій

Грибков С.В.
Національний університет харчових технологій

Чобану В.В.
Національний університет харчових технологій

ВИЗНАЧЕННЯ МІСЦЯ РЕФАКТОРИНГУ В СУЧАСНИХ МЕТОДОЛОГІЯХ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ

Термін експлуатації програмного продукту неможливо передбачити на початку його створення, адже проблеми його використання можуть виникнути при виході нових операційних систем, фреймворків, нового обладнання. Розробники програмного забезпечення намагаються зберегти конкурентоздатність своєї продукції через постійну підтримку, додавання нових функцій та адаптацію наявних відповідно до нових вимог. Для гнучкого впровадження змін у власних програмних продуктах на різних етапах життєвого циклу розробники використовують рефакторинг програмного коду та баз даних. Залежно від обраної методології розробки програмного забезпечення потреба в проведенні рефакторингу виникає на різних етапах. Це зумовлює загальну ефективність створення та експлуатації кінцевого програмного продукту.

Після аналізу опублікованих видань на тему використання рефакторингу в методологіях розробки програмного забезпечення автори даної статті дійшли висновку про те, що задокументованої інформації недостатньо для повного коректного розуміння даної теми. Переважно рефакторинг описують при розборі класу методологій розробки програмного забезпечення "Agile", проте на практиці доцільно використовувати рефакторинг практично в будь-якій методології.

Дослідження проводились на основі аналізу робіт вітчизняних та зарубіжних авторів, в яких описано як рефакторинг програмного коду, так і рефакторинг баз даних, та на основі власного досвіду. Для розгляду в даній статті авторами були обрані такі методології: каскадна модель, V-модель, інкрементна модель, Rapid Application Development-модель, модель екстремального програмування, ітеративна модель, спіральна модель. Були виявлені місця проведення рефакторингу в методологіях розробки, які наведені вище, та його вплив на процес розробки програмного забезпечення, що декларується методологією. Крім цього, нами розглянуті потреби в проведенні рефакторингу та його використанні задля усунення недоліків у процесі розробки та експлуатації інформаційних систем.

Ключові слова: *методологія розробки програмного забезпечення, рефакторинг, розробка програмного забезпечення, функціональність, прототипування, цикл розробки, модель розробки.*

Постановка проблеми. Ефективність роботи команди розробників над проєктом залежить від складу та кваліфікації команди, а також від організації роботи. Під час створення програмного продукту важливо організувати процес розробки таким чином, щоб він був вигідним з погляду оптимального розподілення часу та фінансових витрат, тому важливо обирати методології розробки програмного забезпечення на початку розробки [1]. Попри склад та професіоналізм команди розробників, настає необхідність проведення рефакторингу програмного коду та баз даних. Це може бути зумовлено випуском нових версій операційних систем, фреймворків, інструментів розробки та апаратно-технічних засобів, які диктують умови використання створюваного програмного забезпечення [2]. Виявлення дефектів та їх усунення під час створення програмного забезпечення впливає на кінцеву вартість програмного продукту та загальний час на розробку, тому що залежить від етапу, на якому їх виявлено, а цей етап залежить від обраної методології командою розробників.

Отже, стаття присвячена дослідженню процесу рефакторингу в різних методологіях, вивченню особливостей його використання, а також наданню рекомендацій при виборі методологій розробки для різних проєктів.

Аналіз останніх досліджень і публікацій. Формалізація процесу рефакторингу Кентом Беком та Мартіном Фаулером у їх спільній роботі [3], що

вперше була опублікована у 1999 році, надала розробникам глибоке усвідомлення процесів адаптації програмного забезпечення до нових викликів сучасності. Названі вчені були співавторами у створенні методології екстремального програмування, що належить до комплексу ПЗ "Agile", та внесли рефакторинг як один з основних прийомів до розробки програмного забезпечення [4; 5].

У роботі [6] наведена середня вартість виявлення дефектів залежно від часу їх виявлення та усунення. У деяких розділах наводяться рекомендації щодо проведення рефакторингу у процесі розробки.

У роботах [7; 8] детально проаналізовані різні гнучкі методології розробки програмного забезпечення, їх характеристики, базові принципи та дані рекомендації щодо їх використання.

Робота [9] присвячена принципам сучасних методологій розробки, порівнянню їхніх переваг та недоліків у різних ситуаціях.

У роботі [10] викладена історія виникнення основних методологій, їх призначення, наведена інформація про основні роботи, присвячені різним методологіям та їх авторам.

Робота [11] присвячена рефакторингу та проєктуванню баз даних, теоретичним та практичним рекомендаціям щодо усунення помилок під час проєктування та покращення структур баз даних.

Робота [12] присвячена сучасним проблемам проведення рефакторингу та удосконалення цього процесу.

Хоча вітчизняними та зарубіжними авторами присвячено чимало наукових праць методологіям розробки та рефакторингу, однак існує досить мало інформації про місце та важливість рефакторингу на різних стадіях в кожній методології, а це впливає на якість та швидкість розробки. Варто зазначити, що більшість найуживаніших методологій була створена ще задовго до формалізації рефакторингу. Також необхідно відзначити складність розуміння та використання термінологій під час опису та використання методологій, які виникли в різні періоди та мають свої аспекти, що описані в сучасних джерелах інформації.

Постановка завдання. Метою даної роботи є дослідження процесу рефакторингу в різних методологіях, вивчення особливостей його використання, надання рекомендацій при виборі методологій розробки для різних проєктів.

Виклад основного матеріалу дослідження. Процесом розробки програмного забезпечення в програмній інженерії вважається розбиття процесу створення програмного продукту на фази задля вдосконалення дизайну, менеджменту продукту та проєкту. Такі фази називають циклами розробки програмного забезпечення [1]. Ці цикли формалізовані у методологіях розробки програмного забезпечення [7]. Методологія може містити аналіз попередніх результатів, а також артефактів, які розробляє та вдосконалює команда розробників.

Часто під час збільшення функціональності програмного забезпечення в процесі розробки виникає проблема сприйняття програмного коду. Для уникнення таких проблем командою розробників проводиться рефакторинг. Основними цілями рефакторингу вважаються такі: покращення програмного коду, пошук та виправлення помилок, забезпечення доступності сприйняття програмного коду усіма учасниками команди розробників, підвищення відмовостійкості проєкту в цілому, зміна програмного коду для забезпечення гнучкої модернізації при розширенні проєкту новими функціями та програмними модулями без зміни наявного семантичного значення програмного коду [3]. Окремо необхідно виділити рефакторинг баз даних, який полягає у простих змінах в схемі бази даних, що веде до її покращення при збереженні функціональності та інформаційної семантики [11]. Автори роботи [13] виділяють декілька таких категорій рефакторингу: рефакторинг структури, рефакторинг якості даних, рефакторинг посилальної цінності, рефакторинг архітектури та рефакторинг методів.

Методологія розробки програмного забезпечення – це сукупність методів, що мають спільний філософський підхід та дозволяють забезпечити найкращу ефективність процесів розробки на різних стадіях життєвого циклу [8].

Під час розробки програмного забезпечення командами розробників в усьому світі найчастіше використовуються такі методологічні моделі: каскадна модель, V-модель, інкрементна модель, RAD-модель, модель екстремального програмування, ітеративна модель, спіральна модель.

Найстаршою методологією розробки програмного забезпечення вважається каскадна модель, яку формалізував У. Ройс у роботі [11], після чого вона набула популярності. Відповідно до каскадної моделі проєкт реалізується поступово, проходячи чітку послідовність кроків, а довільні переходи з одного етапу до іншого є недопустимими. Дана модель широко використовувалась в 70-х і першій половині 80-х років, проте, оскільки каскадна модель має високу прозорість розробки і фаз проєкту, а також чітку послідовність виконання, вона досі не втрачає своєї актуальності при повторній розробці типового продукту або випуску нової

версії вже наявного продукту за умови, що зміни, які вносяться, чітко визначені та керовані. З часом недоліки каскадної моделі стали проявлятись частіше. Ці недоліки такі: потреба затвердження повного об'єму вимог вже на першій фазі; виникнення потреби внесення змін у вимоги означає повернення до першої стадії та перероблення всієї виконаної роботи; збільшення затрат коштів і часу у разі зміни вимог. Через специфіку методології не виникає потреби в проведенні рефакторингу, адже всі функціональні вимоги та технічні завдання мають бути описані та затверджені до безпосереднього написання програмного коду. Проте на практиці якщо після початку розробки виникла потреба внесення змін, то у разі повернення до стадії реалізації є можливість використати вже створений програмний код. Попри те, що такий підхід суперечить методології, він буде економічно вигідним. Саме в такій ситуації може бути проведений рефакторинг, адже з'являється необхідність змінити початковий код програмного продукту.

V-модель – це покращена версія класичної каскадної моделі [15]. Тут на кожному етапі проводиться контроль поточного процесу, щоб переконатись у можливості переходу на наступний етап. Таку модель доцільно використовувати під час розробки невеликих проєктів, де усі вимоги чітко визначені та описані. Кожна фаза даної моделі має конкретні результати. Порівняно з каскадною моделлю використання цієї моделі забезпечує економію часу, адже є можливість виявляти недоліки на поточній фазі розробки. У разі використання V-моделі у невідповідному їй проєкті можуть виникнути складнощі з підтримкою паралельних подій та неможливість внесення динамічних змін у вимоги на різних етапах. Рефакторинг в даній моделі може проводитись за тих самих умов, що і в каскадній моделі, проте через постійний контроль процесу розробки ймовірність виникнення потреби в рефакторингу зменшується.

Інкрементна модель базується на поетапній розробці, під час якої різні частини системи розробляються в різний час і різними темпами [16]. Якщо одна частина готова, її інтегрують в систему. Модель забезпечує побудову спочатку невеликої частини системи з подальшим її розширенням в декілька етапів. Поетапний підхід дає можливість легше отримати відгуки від замовників, адже вони можуть прокоментувати результати завершених етапів. Таким чином, частини системи, розроблені на перших етапах, є прототипом всієї системи в цілому. Проте основним недоліком є погіршення структури у разі додавання нових компонентів, що робить її важкою та дорогою для наступних змін, адже для її покращення необхідні додаткові кошти і час на рефакторинг. Проведення рефакторингу можливе як на етапі створення нового модуля, так і у разі виявленні помилок проєктування у вже наявній функціональності.

RAD-модель спрямована на отримання якісного програмного забезпечення у стислі терміни через прототипування та розробку програмного забезпечення з використанням однакових технологій та засобів [1]. Згідно з даною моделлю розробка програмного забезпечення виконується невеликою командою розробників за строк близько 3–4 місяців шляхом використання інкрементного прототипування з використанням інструментальних засобів візуального моделювання і розробки. RAD є досить популярним методом розробки завдяки своїм перевагам – високій швидкості розробки, низькій вартості та високій якості. Варто звернути увагу на те, що даний метод допускає, що перші версії системи не будуть повністю працездатні, тому використання цього підходу для розробки систем, від яких залежить безпека людей, є недопустимим. У RAD-моделі рефакторинг проводиться у процесі уточнення прототипів, адже під час зміни прототипу вносяться зміни в систему в тій самій частині. Завдяки використанню рефакторингу оновлення інформаційної системи проходить більш плавно.

Методологія екстремального програмування [4] є більш гнучкою, ніж інші, що дозволяє команді розробників швидко реагувати на зміни вимог з боку замовника та на конкретних етапах роботи легко підлаштовуватися під його вимоги. Вона належить до комплексу методологій ПЗ "Agile", що регламентується маніфестом гнучкої розробки програмного забезпечення, який вперше був опублікований у 2001 році [17]. Екстремальне програмування застосовується у проєктах, що мають незначні обмеження в часі на розробку, та направлене на ефективність процесу аналізу розвитку проєкту в цілому. В екстремальному програмуванні рефакторинг вже закладений як один з дванадцяти основних прийомів.

Ітеративна модель – це процес розробки програмного забезпечення, який проходить невеликими етапами, в ході яких ведеться аналіз отриманих проміжних результатів, висуваються нові вимоги та коректуються попередні етапи роботи [16]. Після аналізу кожної ітерації приймається рішення про можливість використання її результатів як вхідної точки для нової ітерації. Наприкінці процесу отримуємо результат, у якому були враховані всі вимоги. Використовувати дану модель

доцільно під час роботи з великими проєктами або коли вимоги до кінцевої системи заздалегідь невідомі, а також якщо основна задача має бути визначена, проте деталі реалізації можуть змінюватись під час розробки. Завдяки відсутності жорсткого планування процесу розробки ітеративна модель дозволяє будь-коли внести зміни у технічне завдання чи функціональні вимоги. Малоймовірно, що на першій ітерації розробки виникне потреба в проведенні рефакторингу, адже ще не існує сталої бази кінцевого продукту, а вже на другій ітерації така потреба може виникнути, тому що під час коригування вимог з'являється необхідність внести зміни у функції системи або доповнити їх. Ітеративна модель, регламентуючи можливість оперативно вносити зміни, та рефакторинг, що дозволяє гнучко впроваджувати необхідні зміни, водночас надають широкий спектр можливостей з виявлення та виправлення недоцільно прийнятих на ранніх етапах архітектурних рішень.

Спіральна модель є гібридом ітераційної та каскадної моделей [18; 16; 11]. Кожен цикл розробки в ітераційній моделі має послідовність операцій, що відповідають крокам в каскадній моделі. У кінці кожної ітерації розробники представляють для замовника нову версію програмного забезпечення з новим функціоналом. На кожній ітерації визначаються та уточнюються задачі та вимоги до програмного продукту, визначаються кількісні та якісні характеристики, а також планується завдання на наступну ітерацію. З кожною ітерацією проєкт стає складнішим, адже збільшується його функціонал завдяки деталізації вимог та зауважень від замовника. Використання даної моделі може бути доцільним під час розробки систем з високим рівнем можливих ризиків або у випадках, коли замовник не може надати достатньо конкретний перелік вимог до кінцевого продукту чи ці вимоги досить складні. Спіральна модель поділяється на чотири квадранти, в кожен з яких входять основні і допоміжні дії з розробки системи. Проведення рефакторингу доцільне саме у третьому квадранті, який включає розробки моделей та написання програмного коду на основі прототипів програмного забезпечення, що були розроблені у другому квадранті. Потреба у проведенні рефакторингу виникає під час виправлення помилок у прототипах, а також у разі конфлікту мети поточного циклу з наявними напрацюваннями, що отримувались у попередніх циклах. Використання рефакторингу можливе також і в другому квадранті за умови використання CASE-засобів у процесі прототипування.

Незалежно від використовуваної методології розробки програмного забезпечення з великою ймовірністю виникне потреба у рефакторингу, якщо проєктна робота виконується в рамках наявної інформаційної системи.

Висновки. На основі проведеного дослідження було виявлено місця виникнення потреби в проведенні рефакторингу в кожній розглянутій методології. Варто зазначити, що в жодній з методологій неможливо уникнути рефакторингу, адже в описаних циклах розробки програмного забезпечення важко виключити помилки при проєктуванні та конструюванні, хоча в класичних джерелах не виділяють місце рефакторингу. У кожній з моделей потреба в рефакторингу буде виникати в різних місцях з різною ймовірністю. Наприклад, у V-моделі рефакторинг виникає рідше, ніж у каскадній моделі, зокрема й через проведення контролю якості тестуванням на кожному етапі. В ітераційній моделі та в моделі екстремального програмування ймовірність виникнення рефакторингу найвища, коли функціональні вимоги та технічні завдання чітко не сформульовані замовниками. Також достатньо висока ймовірність рефакторингу виникає в інкрементній моделі, але тільки при впровадженні та на межі взаємодії модулів програмного забезпечення. За умови використання CASE-засобів для прототипування в RAD-моделі та спіральній методології потреба в рефакторингу виникає з однаковою імовірністю. Невикористання однакових інструментів для прототипування та розробки, що допускається спіральною методологією, тільки підвищує кількість операцій рефакторингу, адже їх необхідно виконувати і в прототипі, і в безпосередньо наявній версії продукту. Обираючи методологію, за якою буде працювати команда розробників, не потрібно зважати на ймовірність проведення рефакторингу, найголовнішими критеріями вибору мають стати тип проєкту та кваліфікація команди. Перспективним є дослідження доцільного вибору методології відповідно до типу проєкту.

Список літератури:

1. Мирошниченко Е.А. Технологии программирования : учебное пособие. 2-е изд., испр. и доп. Томск : Изд-во Томского политехнического университета, 2008. 128 с.

2. Зінов'єв Є.А., Собінов О.Г. Програмне забезпечення системи реінжинірингу та рефакторингу програмного коду до платформи .NET. *Сучасні інформаційні технології та програмне забезпечення комп'ютерних систем* : збірник тез доповідей всеукраїнської науково-практичної конференції студен-

тів та магістрантів, м. Кіровоград, 27–29 березня 2013 року. Кіровоград : «КОД», 2013. С. 78—79. URL: http://it-kntu.kr.ua/conf/AITandSCS_Kirovograd_2013.pdf (дата звернення: 12.09.2019).

3. Фаулер М., Бек К., Брант Д., Апдайк У., Робертс Д. Рефакторинг. Улучшение существующего кода. Москва : Символ-Плюс, 2008. 473 с.

4. Статті і ресурси по XP і суміжним питанням, огляд книг / Сайт Рона Джеффриза. URL: https://ronjeffries.com/ (дата звернення 10.09.2019).

5. Мартин Р.С., Ньюкирк Д.В., Косс Р.С. Agile software development. Principles, Patterns, and Practices. Москва : Вильямс, 2004. 752 с.

6. Макконнелл С. Совершенный код. Мастер класс / пер. с англ. Санкт-Петербург : БХВ-Петербург, 2018. 896 с.

7. Фаулер М. Новые методологии программирования / MAXKIR.com. URL: http://www.maxkir.com/sd/newmethRUS.html#N447 (дата звернення: 12.09.2019).

8. Agile Methodology & Model: Guide for Software Development & Testing URL: https://www.guru99.com/agile-scrum-extreme-testing.html (дата звернення: 12.09.2019).

9. Selecting a development approach // Centers for Medicare & Medicaid Services (CMS) Office of Information Services, 2008. URL: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf (дата звернення: 12.09.2019).

10. Software development process // Wikipedia: вільна енциклопедія. URL: https://en.wikipedia.org/wiki/Software_development_process#cite_ref-13 (дата звернення: 12.09.2019).

11. Эмблер С.В., Садаладж П.Д. Рефакторинг баз данных: эволюционное проектирование, Москва : Вильямс, 2007. 368 с.

12. Santos B., Guzman I., Camargo I., Piattini M., Ebert M. Software Refactoring for System Modernization in IEEE Software. 2018. Vol. 35, № 06. P. 62–67. DOI: 10.1109/MS.2018.4321236.

13. Durelli S.R. Improving the Structure of KDM Instances via Refactorings, Proc. 31st Brazilian Symp. Software Eng. (SBES 17). 2017. P. 174–183.

14. Dr. Winston W. Royce. Managing the development of large software systems. Proc. IEEE WESCON, Aug 1970. Reprinted 9 th Intl. Conf. Softw. Eng.

15. IABG Information Technology V-Model Lifecycle Process Model. URL: http://www.v-modell.iabg.de/kurzb/vm/k_vm_e.doc (дата звернення: 03.06.2016).

16. Ларман К., Базили В. Итеративная и инкрементальная разработка: краткая история. URL: http://www.osp.ru/os/2003/09/183412/ (дата звернення: 03.09.2019).

17. Agile-маніфест розробки програмного забезпечення URL: http://agilemanifesto.org/iso/uk/manifesto.html (дата звернення: 12.09.2019).

18. Richard W. Selby. Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research. John Wiley & Sons, 2007. 834 c.

Struzik V.A., Hrybkov S.V., Chobanu V.V. DETERMINATION OF REFACTORING PLACE IN MODERN METHODOLOGIES OF INFORMATION SYSTEMS DEVELOPMENT

The lifetime of the software product cannot be predicted at the beginning of its creation, because problems of its use may arise when new operating systems, frameworks, new equipment are released. Software developers are trying to ensure their products competitive by constantly supporting, adding new features and adapting existing ones according new requirements. Developers use refactoring of code and database for flexible implementation of changes to their own software products at different stages of the lifecycle. Refactoring occurs at different stages development of software depending on the chosen methodology. It is determines the overall efficiency of creation and operation of the final software product.

The authors of this article concluded that the documented information about the use of refactoring in software development methodologies is not sufficient for a complete correct understanding of the theme after analyzing the publications scientific figures who worked on that topic. Typically, refactoring is described when described a group of Agile software development methodologies. However, in practice, it is advisable to use refactoring in any methodology.

The researches were carried out on the basis of the analysis of works of domestic and foreign authors, where described both the refactoring of the program code and the refactoring of databases. Also, the knowledge from personal experience was applied when analyzing data and summarizing in this article. The authors have selected next methodologies for consideration in this article: cascade model, V-model, incremental model, Rapid Application Development model, extreme programming model, iterative model, spiral model. The place of refactoring and its impact in the methodologies of development above was analyzed and identified by authors of the article. In addition, authors give consideration of needs of refactoring and its uses to address deficiencies in the development and operation of information systems.

Key words: *software development methodology, refactoring, software development, functionality, prototyping, development cycle, development model.*