УДК 004.4

Д. Ю. Станішевський
О. В. Войцеховська

Оцінка ризиків технічного боргу та проведення рефакторінгу мовою програмування C#

Вінницький національний технічний університет

Анотація

У роботі проведений аналітичний розбір проблематики технічного боргу при розробці інформаційних систем. Наведенні негативні чинники, до яких призводить накопичення технічного боргу. Розглянуті підходи та методи боротьби з ним. Наведенні приклади вирішення популярних проблем, що виникають при розробці інформаційних систем мовою програмування C#.

Ключові слова: технічний борг, семантичне ускладнення, зростання вартості розробки, рефакторинг, винесення методу, ризики, декомпозиція кодової бази, програмний код що тестується, SonarQube, Veracode.

Abstract

The paper analyzes the issue of technical debt in the development of information systems. The negative factors that lead to the accumulation of technical debt. Approaches and methods of combating it are considered. Here are some examples of solutions to popular problems that arise when developing information systems in C# programming language.

Keywords: technical debt, semantic complication, increase in development cost, refactoring, method removal, risks, code base decomposition, software code under test, SonarQube, Veracode.

Вступ

З точки зору замовника та кінцевого користувача важливі стабільність, функціональність та швидкодія. Але у сучасних реаліях важливою характеристикою постала, якість коду та її похідні: розширюваність, пристосованість до тестування, здатність до повторного використання.

Сучасні інформаційні системи значно ускладнились за останніх два десятиріччя. Зараз навіть досить проста система для обліку клієнтської бази, може мати декілька рівнів логіки, з десяток інтеграцій і широку можливість для розширення. Іншим нововведенням стала тенденція до відмови терміну «завершений продукт», як такого – якщо певний проект успішний, то він буде розроблятись протягом довгого періоду часу з метою розширення функціональності або його покращення.

Такі зміни призвели до нових принципів в розробці коду. Один з був сформульований ще понад 50 років назад, але став максимально актуальним зараз – «програмний код пишеться не для користувача, а для розробника»[1]. Якщо раніше на невеликих проектах код розроблявся з ціллю оптимального виконання задач, оскільки після його написання та перевірки та коректну роботу потреба в його змінах відпадала, то зараз написаний код постійно перечитується, модифікується та повторно використовується.

Для проблем, які заважають цим діям з кодом, було дано новий термін – технічний борг. Збільшення технічного боргу призводить до зростання вартості та часу розобки.

Отже, в сучасних реаліях зросла важливість якості коду, для полегшення подальшої розробки.

Результати досліджень

Першої технологією, яку варто згадати в контексті технічного боргу, це формалізовані правила написання коду. Вони мають вигляд схожий на правила, які існують в наукових конференціях щодо робіт які публікуються. Задачею яку вирішують подібні правила, це приведення кодової бази інформаційної системи до єдиного виду. Оскільки значно важче працювати на проекті де кожен розробник має своє бачення щодо стилістики, практик та підходів до написання коду. Іншою проблемою що може вирішити створення єдиних правил, це загальна заборона шкідливих та непрактичних методів в написанні коду[2].

Іншої методологією підвищення якості коду – це написання автоматичних тестів різного рівня та виду. Зараз майже не можливо повноцінно протестувати сучасну інформацію систему силами людини, оскільки завжди є велика кількість варіантів відпрацювання логіки додатку. Також проблематично виконувати таку перевірку після кожної зміни в проекті. Цю проблему вирішує написання автоматичних тестів, які можуть виконати перевірку порушення логіки, яка була написана раніше. Такий вид тестування має назву «регресивного».

Ще важливим критерієм якості коду я відсутність «загроз» та «вразливостей», відомих проблем в бібліотеках, що використовуються, наприклад вразливість до хакерських атак, здатність до некоректної роботи або помилка в логікі роботи.

Для контролю дотримання всього вище сказаного існують спеціальні технології, а саме статичні аналізатори, які можуть проаналізувати розроблений код на достатнє покриття тестами, успішність їх проходження, відповідність правилам стилістики коду та наявності в них загроз та вразливостей. До найпопулярніших з них можна віднести SonarQube та Veracode. Перший відноситься до універсальних, може перевіряти покриття тестами, відповідність стилістиці, та відсутність популярних помилок в коді. Інший же вузькоспеціалізований та здатний до перевірки сторонніх бібліотек що використовуються на наявність вразливостей[3].

Але все ж зараз не може перевірити якість коду лише автоматичними методами. Оскільки окрім всього вище сказаного існує дотримання принципів та правил розробки, таких як SOLID, DRY, KISS. Відповідність до них може перевірити лише досвідчений розробник, за допомогою процесу перегляду коду – код рев'ю. В наслідок огляду коду в ньому знаходяться невідповідності та проблеми, які необхідно виправити. Процес виправлення невідповідності коду вимогам, має назву «рефакторінг»[4].

Проведемо рефакторинг невеликого блоку коду (Рис 1).

```
5 references
public class ImagesService
{
    IConfiguration configuration;
    ICloudStorage cloudStorage;

    2 references
    public ImagesService(IConfiguration config)
    {
        configuration = config;
        cloudStorage = new GoogleCloudStorage(config);
    }

    2 references
    public async Task<ImageInfo> SaveImagesAsync(IFormFile imageFile)
    {
        string filePath = null;
        string cloudLink = null;

        if (imageFile != null)
        {
            Guid id = Guid.NewGuid();
            string ext = Path.GetExtension(imageFile.FileName);

            filePath = id + ext;

            cloudLink = await cloudStorage.UploadFileAsync(imageFile, filePath);
        }

        return new ImageInfo(filePath, cloudLink);
    }
```

Рис 1. Код з порушеннями правил розробки

Перша проблема на яку варто звернути увагу це відсутність інтерфейсу в даного класу. Це може призвести до порушенню принципу інверсії залежностей з принципів SOLID. В такому випадку код який буде використовувати цей клас буде залежати від конкретної реалізації, а повинен – від абстракції. Виправляється дана проблема створенням інтерфейсу.

Друга проблема – передача в конструктор об'єкту інтерфейсу IConfiguration. Він належить до бібліотеки ASP .NET. Клас з логікою не повинен залежати від реалізації операцій введення та виведення. Це є порушенням принципу інакпсуляції рівнів та модулів. Виправляється ця проблема створенням нового класу для передачі параметрів конфігурацій, та використання його замість IConfiguration.

Третім проблемним місцем є застосуванням GoogleCloudStorage. В цьому місці існує одразу два порушення.

По-перше, це знову порушення інверсії залежностей, в даному випадку наш клас залежить від конкретної реалізації класу, а повинен залежати від інтерфейсу. Виправляється заміною конкретного класу на його інтерфейс.

По-друге, використання композиції залежності, це повністю унеможливлює написання юніт та модульних тестів для даного класу. Краще передавати залежність через параметр конструктора, тобто застосувати асоціацію

Висновки

Отже, на даному етапі еволюцію сфери розробки програмного забезпечення зросла важливість пякості кодової бази та зниження технічного боргу. Якість коду підвищується за допомогою написання тестів, дотримання стилю, виправлення вразливостей, дотримання принципів розробки. Існують автоматичні системи контролю якості коду, але потреба в перевірці людиною – код рев'ю залишається. Процес виправлення проблем в коду має назву рефакторинг. В даній роботі був наведенний приклад рефакторингу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Р. Мартін, Чиста архітектура. Харків, Україна : Фабула, 2021, 368 с. ISBN: 978-617-09-5286-8.
2. Офіційний стандарт стилю написання коду мовою C# [Електроний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions
3. Офіційний сайт SonarQube [Електроний ресурс] – Режим доступу до ресурсу: https://www.sonarqube.org/
4. Code refactoring best practices [Електроний ресурс] – Режим доступу до ресурсу: https://www.altexsoft.com/blog/engineering/code-refactoring-best-practices-when-and-when-not-to-do-it/.

Станішевський Дмитро Юрійович – студент групи 2КІ-18б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: stanishevskiy.dmitro@gmail.com

Войцеховська Олена Валеріївна – кандидат технічних наук, доцент кафедри обчислювальної техніки, Вінницький національний технічний університет, Вінниця

Stanishevskiy Dmytro Y. – students, 2KI-18b, Faculty of information Technologies and Computer Engeneering, Vinnytsa National Technical University, email: stanishevskiy.dmitro@gmail.com

Voytsekhovska Olena V. — PhD, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical 0University