2. Пелещишин А.М. Формування суспільного авторитету ВНЗ шляхом комплексного подання в системі Вікіпедія / А.М. Пелещишин, Ю.Й. Пероганич, О.В. Марковець, Н.О. Думанський // Вісник Нац. ун-ту «Львівська політехніка». – 2009. – № 653. – С. 180-187

УДК 004.415

РЕФАКТОРИНГ ЯК ПРОЦЕС УДОСКОНАЛЕННЯ ЯКОСТІ ПРОГРАМНОГО ПРОДУКТУ

Ладоха А. В., здобувачка вищої освіти 3 курсу
Петренко Д. О., здобувач вищої освіти 6 курсу
Науковий керівник: **Білоус І.В.,** доцент, к.т.н
Національний університет «Чернігівська політехніка»

Рефакторинг (або його інколи називають реорганізацією коду) – це процес зміни внутрішньої структури програмного продукту, який не зачіпає його зовнішню поведінку й має на меті полегшення розуміння програмного коду й оптимізацію продуктивності. Рефакторинг повинен відбуватися постійно під час розробки програмного продукту. Навіть якщо на початку розробки код був ідеальний, в процесі розробки повинен відбутися рефакторинг, який удосконалює якість програмного продукту. Без рефакторингу не обходиться жоден дійсно складний і довготривалий проект. Адже якщо цей проект постійно використовується, то він постійно дороблюється, відповідно з'являється додатковий код, який може призвести до суцільного хаосу, якщо не буде проведено рефакторинг.

Якщо розробники хочуть мати добре структурований код, який легко читається та швидко допрацьовується, тоді слід постійно проводити рефакторинг. Код, який пройшов рефакторинг, тобто «чистий код», значно легше й дешевше підтримувати, ніж заплутаний код [1]. Але рідко вдається мати такий код постійно, адже розробники поспішають, у процесі можуть змінюватися вимоги до поставленої задачі, тестувальники можуть знаходити помилки в роботі програми, які потрібно швидко виправити, або виникають термінові доопрацювання, які доводиться робити нашвидкуруч.

Як зрозуміти, що коду потрібен рефакторинг. У першу чергу, «чистий код» повинен проходити всі тести, навіть якщо проходить 95% тестів, значить десь існує «брудний код». Код, який не потребує реорганізації, повинен бути очевидним для інших розробників. Погане іменування змінних, завеликі класи й методи – це все впливає на очевидність коду. Постійне дублювання коду – це також є ознакою наявності «брудного коду». Й останньою ознакою «брудного коду» є кількість класів та інших рухомих частин. Чим менше класів та й самого коду, тим менше вірогідність того, що програмний продукт може «зламатися» при черговій зміні коду. Звісно, якщо код занадто заплутаний і потребує великої кількості спроб рефакторингу, простіше буде написати програмний продукт з нуля.

Яким чином проводиться рефакторинг. Під час рефакторингу проводяться маленькі послідовні покращення коду. У заплутаному програмному коді, можна чистити все, але варто почати в першу чергу з деяких проблем [2]:

1. Мертвий код. Видаленню підлягають змінні, методи або класи, які не використовуються й навряд чи знадобяться. Гірше всього, якщо мертвий код зустрінеться в умовно складній конструкції, яка не виконується через помилку або зміну вимог.

2. Дублювання. Якщо один й той самий код зустрічається не один раз у програмі, то його потрібно винести в окрему функцію.

3. Неправильні назви змінних, функцій, класів, які не передають суті їх призначення. Імена повинні повідомляти, навіщо даний елемент коду існує, що він робить і як

використовується. Якщо існують такі змінні, навіть без коментарів, тоді вони підлягають рефакторингу.

4. Занадто довгі функції, методи та класи. Оптимальний розмір методів і функцій – 20-30 рядків. Якщо виходить більше, то слід розділити функцію на декілька маленьких і додайте до однієї спільної. Так само і з класами, якщо клас більше 20-30 рядків, то слід його розбити на декілька маленьких і додати їх об'єкти до одного спільного класу.

5. Багато коментарів. «Поганий код» часто стараються замаскувати за великою кількістю коментарів, щоб усе було зрозуміло. Тому спочатку потрібно спробувати переписати потрібну ділянку коду так, щоб стало зрозуміло сторонньому програмісту (який ніколи не бачив даний проект) про що йдеться в функції чи методі без коментарів.

6. Довгий список параметрів функції чи методу тільки більш заплутує, ніж допомагає. У разі потрібності всіх цих параметрів, їх можна винести в окрему структуру або клас із зрозумілим ім'ям, а до функції передати посилання на нього [3].

Звісно, після кожної правки слід перевіряти сусідні ділянки коду, адже можна випадково заплутати код ще більше.

Чим може бути небезпечний рефакторинг. Через те, що зміна проходить у робочому коді можна не тільки все спростити, а зробити код взагалі непрацюючим. Такий рефакторинг може зупинити виконання проекту на декілька днів. Проводити реорганізацію коду потрібно постійно, а не від випадку до випадку [4].

Таким чином, рефакторинг це не просто примха «чистого» коду, а велика допомога в удосконаленні якості програмного продукту, в тому числі щодо оптимізації продуктивності програмному продукту (чим важче код, тим довше він виконується). Тому рефакторингу повинно бути місце в будь-якій розробці програмного продукту.

Список використаних джерел

1. Чистий код [Електронний ресурс] // Refactoring.Guru. – 2021. – Режим доступу до ресурсу: https://refactoring.guru/uk/refactoring/what-is-refactoring.

2. Что такое рефакторинг кода? Основные принципы и правила рефакторинга [Електронний ресурс] – Режим доступу до ресурсу: https://analytics.infozone.pro/basic-principles-and-rules-of-refactoring/.

3. Рефакторинг — это неизбежный процесс [Електронний ресурс] – Режим доступу до ресурсу: https://web-creator.ru/articles/refactoring.

4. Что такое рефакторинг кода и зачем он нужен [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://skillbox.ru/media/code/chto_takoe_refaktoring_ koda_i_zachem _on_nuzhen/.