Паламарчук О.

Науковий керівник – асист. Струк С.П.

РЕФАКТОРИНГ КОДУ ТА ЙОГО ВИКОРИСТАННЯ У ПРОЦЕСІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Рефакторинг – це процес такої зміни програмної системи, за якої не змінюється зовнішня поведінка коду, але покращується його внутрішня структура. Це спосіб систематичного впорядкування програми, при якому шанси появи нових помилок є мінімальними. При проведенні рефакторингу коду покращується дизайн вже готової програми.

Покращення коду після його написання може виявитись дещо дивним, оскільки в сьогоднішньому розумінні розробки програмного забезпечення спочатку робиться дизайн системи, а потім пишеться код. З часом програма модифікується, і цілісність системи, відповідність її структури початковому плануванню поступово погіршується.

З допомогою рефакторингу можна взяти проблемну програму і переробити її в добре спроектовану. Кожен крок цього процесу є надзвичайно простим. Переміщається поле з одного класу в інший, забирається частина коду з методу і поміщається в окремий метод, якийсь код переміщується по ієрархії в різних напрямках. Але сумарний ефект цих невеликих змін може радикально покращити проект. Це є протилежністю звичайному процесу розпаду програми.

При проведенні рефакторингу виявляється, що відношення різних етапів роботи змінюється. Процес проектування розтягнуто на весь період розробки, а не сконцентровано лише на її початковому етапі. В процесі роботи над проектом стає зрозумілим, як його можна покращити. Така взаємодія призводить до створення програми, якість програмного коду якої залишається високою під час доповнення та розвитку продукту.

Основна ціль рефакторингу – зробити код простішим і зрозумілішим. Причому, рефакторинг і оптимізація це далеко не одне і те ж. В результаті оптимізації код працює швидше, але не обов'язково є простішим і зрозумілішим. Рефакторинг – це спрощення і покращення «читабельності» коду.

Рефакторинг можна застосовувати до будь-якої програми. Частіше за все це робиться інтуїтивно. Якщо якийсь код є не дуже вдалим, або через певний період часу незрозуміло, що робить та чи інша частина програми, – потрібно здійснити рефакторинг. Можна виокремити деякі ознаки коду, якому потрібна переробка:

Якщо у програмі є дублювання коду, то необхідно проводити рефакторинг. Код, який дублюється в програмі, є основним джерелом помилок. Якщо якась дія виконується у декількох різних місцях, але однаковою послідовністю команд, то цей код потрібно винести в окрему функцію і викликати її. Інакше висока ймовірність того, що при виправленні помилки в одному місці дубляж виявиться не відредагований, і у ньому залишиться помилка.

Якщо у програмі дуже довгі функції. Як правило, людина не може повністю сприймати і оцінювати правильність коду, якщо він займає більше ніж 2-3 десятки рядків. Такі методи і функції потрібно розбивати на декілька дрібніших і робити одну загальну функцію, що буде послідовно викликати ці методи. Ніколи не варто скорочувати розмір функції, вписуючи декілька операторів в один рядок. Ймовірність допустити помилку в такому коді зростатиме.

Довгий список параметрів функції. Велика кількість параметрів зазвичай не тільки ускладнює розуміння того, що робить даний метод чи функція, але і ускладнює розуміння коду, що використовує цю функцію. Якщо ж дійсно є потреба у великій кількості параметрів функції, то їх необхідно винести в окрему структуру або клас і передавати вказівник чи посилання на об'єкт цієї структури чи класу.

Великі класи. Якщо у програмі є один або декілька великих класів, то їх необхідно розділити на менші, та включити об'єкти цих класів в один загальний клас.

Велика кількість тимчасових змінних також є ознакою поганого коду, якому необхідний рефакторинг. Як правило, багато тимчасових змінних зустрічаються у занадто великих функціях, після рефакторингу яких, кількість змінних в них стане меншою і код стане значно зрозумілішим і зручнішим.

Багато даних, що хаотично зберігаються, але є логічно пов'язані, і їх можна об'єднати в структуру або клас. Логічно пов'язані дані завжди слід зберігати в структурах або класах, навіть якщо їх всього лише дві чи три – від цього код тільки покращиться.

Якщо об'єкти одного класу занадто часто звертаються до даних об'єкта іншого класу, потрібно переглянути функціонал об'єктів. Можливо, було прийняте невірне архітектурне рішення, і його потрібно поміняти якомога раніше, щоб помилка не поширилась по всьому коду.

Не доцільно створювати занадто багато глобальних змінних. Якщо ж така необхідність все-таки є, то потрібно спробувати згрупувати їх в структури, класи, або хоча б винести їх в окремий простір імен. Тоді шанс помилкового використання якоїсь із змінних стане значно нижчим.

Основними стимулами для проведення рефакторингу є:

- Необхідність додати нову функцію, яка недостатньо вкладається в прийняте архітектурне рішення.
- Необхідність виправити помилку, причини виникнення якої не є очевидними.
- Необхідність долати труднощі командної розробки, які зумовлені складною логікою програми.

Основними методами рефакторингу є:

- зміна сигнатури методу;
- інкапсуляція поля;
- виділення класу;
- виділення інтерфейсу;
- виділення локальної змінної;
- виділення методу;
- генералізація типу;
- вбудовування;
- уведення фабрики;
- уведення параметра;
- підйом поля (методу);
- спуск поля (методу);
- заміна умовного оператора поліморфізмом.

Значно спростити рефакторинг можуть IDE, плагіни до IDE, а також системи контролю версій. IDE і плагіни до нього, наприклад, можуть підсвічувати розробнику рядки коду, які він поміняв з початку останнього запису вихідного коду на диск. Таким чином, програмісту стає візуально зрозумілим те, над яким кодом він вже працював, а який ще не редагувався. Це дозволяє більше зосередитись на змінах, а не на всій програмі при перевірці правильності написаного коду. Якщо навіть відкритий файл містить більше 1000 рядків коду, за допомогою IDE програміст легко зможе знайти ті декілька рядків програми, які він тільки що змінив, навіть якщо вони знаходяться в різних частинах файлу.

Системи контролю версій дозволяють програмісту порівнювати код із попередніми його версіями. Найчастіше вони можуть візуально оформити різницю між поточною та попередніми версіями програми. Наприклад, підсвічення нового коду зеленим, видаленого код червоним, перегляд історії розвитку коду, перегляд коментарів, які робились при записі апдейтів в систему контролю версій та багато іншого.

Рефакторинг потрібно проводити завжди, коли є можливість або необхідність. Він покращує код і зменшує затрати часу на пошук помилки у майбутньому. Найпростіший спосіб застосування рефакторингу такий: щоразу, коли щось міняється в програмі, додається нова функція, клас, декларується змінна, або навіть міняється кілька стрічок якоїсь функції – потрібно подивитись на сусідній код (вище і нижче ділянки над якою ведеться робота). Потрібно перевірити, чи не можна внести в цей код зміни, щоб він став ще більш простим та зрозумілим. Таким чином можливо усунути логічні помилки, якщо знайдеться варіант роботи програми, за якої вона дасть збій. Але не потрібно повністю аналізувати весь код. Спочатку потрібно зробити те, що було потрібне, оскільки можна взагалі не вирішити поточну проблему.

ЛІТЕРАТУРА:

1. Фаулер М. Рефакторинг: улучшение существующего кода. – Пер. с англ. –СПб: Символ-Плюс, 2003. – 432 с.
2. Скотт В. Эмблер, Прамодкумар Дж. Садаладж Рефакторинг баз данных: эволюционное проектирование — М.: «Вильямс», 2007. — 368 с.

3. http://dev.mindillusion.ru/refactoring/