Рефакторинг

Как рефакторить

Рефакторинг следует проводить серией небольших изменений, каждое из которых делает существующий код чуть лучше, оставляя программу в рабочем состоянии.

Чек-лист *правильно* проведённого рефакторинга:

- **Код должен стать чище**
- **В процессе рефакторинга не создаётся новая функциональность:** не смешивайте рефакторинг и непосредственную разработку новых функций. Пытайтесь разделять эти процессы хотя бы в рамках отдельных операций.
- **Все существующие тесты должны успешно проходить.** Есть два случая, когда после рефакторинга ломаются тесты:
  - **Вы допустили ошибку при изменении кода.** Следовательно необходимо исправить ошибку.
  - **Ваши тесты были слишком низкоуровневыми.** Чаще всего это случается из-за того, что ваши тесты проверяют работу приватных методов классов. В этом случае виноваты тесты и единственный способ это исправить — отрефакторить их или написать новые, более высокоуровневые.

Проблемы, возникающие при проведении рефакторинга

- **Использование базы данных**

Одной из областей применения рефакторинга служат базы данных. Большинство деловых приложений тесно связано с **поддерживающей их схемой базы данных**. Это одна из причин, по которым базу данных **трудно модифицировать**. Другой причиной является **миграция данных**. Даже если система тщательно разбита по слоям, чтобы уменьшить зависимости между схемой базы данных и объектной моделью, изменение схемы базы данных вынуждает к миграции данных, что может оказаться длительной и рискованной операцией.

- **Изменение интерфейсов**

Важной особенностью объектов является то, что они позволяют изменять реализацию программного модуля независимо от изменений в интерфейсе. Можно благополучно изменить внутреннее устройство объекта, никого при этом не потревожив, но интерфейс имеет особую важность – если изменить его, может случиться всякое. В рефакторинге беспокойство вызывает то, что во многих случаях интерфейс действительно изменяется. Такие простые вещи, как «Переименование метода» (Rename Method), целиком относятся к изменению интерфейса.

- **Изменения дизайна, вызывающие трудности при рефакторинге**

Когда рефакторинг не нужен?

- Основной пример – **необходимость переписать программу с нуля**. Иногда имеющийся код настолько запутан, что подвергнуть его рефакторингу, конечно, можно, но проще начать все с самого начала. А также явный признак необходимости переписать код – его неработоспособность.
- Другой случай, когда следует воздерживаться от рефакторинга, это **близость даты завершения проекта**. Приближение срока окончания работ – единственный случай, когда можно отложить рефакторинг, ссылаясь на **недостаток времени и дополнительная стоимость обслуживания** и расширения, обусловленная чрезмерной сложностью кода.

Запахи кода (Код с душком)

- **Запахи кода** — это индикаторы проблем, на которые нужно обращать внимание при рефакторинге. Часто их легко найти и исправить, но иногда они предвещают о глубинных проблемах с кодом.

Выделяют группы источников неприятных запахов:

1) *Раздувальщики*
2) *Нарушители объектного дизайна*
3) *Утяжелители изменений*
4) *Замусориватели*
5) *Опутыватели связями*

Запахи кода

Раздувальщики - представляют код, методы и классы, которые раздулись до таких больших размеров, что с ними стало невозможно эффективно работать. Все эти запахи зачастую не появляются сразу, а нарастают в процессе эволюции программы (особенно когда никто не пытается бороться с ними).

- *Длинный метод*
- *Большой класс*
- *Одержимость элементарными типами*
- *Длинный список параметров*
- *Группы данных*

Нарушители объектного дизайна. Все эти запахи являют собой неполное или неправильное использование возможностей объектно-ориентированного программирования.

- *Альтернативные классы с разными интерфейсами*
- *Отказ от наследства*
- *Операторы switch*
- *Временное поле*

Запахи кода

Утяжелители изменений. Эти запахи приводят к тому, что при необходимости что-то поменять в одном месте программы, вам приходится вносить множество изменений в других местах. Это серьезно осложняет и удорожает развитие программы.

- *Расходящиеся модификации*
- *Параллельные иерархии наследования*
- *Стрельба дробью*

Замусориватели являют собой что-то бесполезное и лишнее, от чего можно было бы избавиться, сделав код чище, эффективней и проще для понимания.

- *Комментарии*
- *Дублирование кода*
- *Класс данных*
- *Мёртвый код*
- *Ленивый класс*
- *Теоретическая общность*

Запахи кода

Опутыватели связями

Все запахи из этой группы приводят к избыточной связанности между классами, либо показывают, что бывает, если тесная связанность заменяется постоянным делегированием.

- *Завистливые функции*
- *Неуместная близость*
- *Неполнота библиотечного класса*
- *Цепочка вызовов*
- *Посредник*

Контрольные вопросы:

1) Как проводить рефакторинг?
2) Что значит правильный рефакторинг?
3) Какие проблемы могут возникнуть при проведении рефакторинга?
4) В каких случаях рефакторинг не нужен?
5) Что называют запахами кода?
6) Опишите группы источников неприятных запахов: пояснение и какие запахи к ним относятся

Задание: Заполните самостоятельно таблицу некоторых источников запахов кода (используя приложение файл **ЗапахиРефакторинга.pdf**):

Запах	Описание	Методы рефакторинга
дублирование кода		
длинный метод		
большой класс		
длинный список параметров		
операторы типы switch		
завистливые функции		
группы данных		
временное поле		
цепочки сообщений		
классы данных		
комментарии		

Основная:

Фаулер М. Рефакторинг: улучшение существующего кода. – Пер. с англ. – СПб: Символ-Плюс, 2018. – 432 с., ил.