

Общие понятия и определения

"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure."

- Martin Fowler

Рефакторинг представляет собой процесс такого изменения программной системы, при котором не меняется внешнее поведение кода, но улучшается его внутренняя структура.

Это способ систематического приведения кода в порядок, при котором шансы появления новых ошибок минимальны.

В сущности, при проведении рефакторинга кода вы улучшаете его дизайн уже после того, как он написан.



Цель рефакторинга

сделать код программы более легким для понимания,

без этого рефакторинг нельзя считать успешным.

```
arror_mod = modifier_ob
  mirror object to mirror
mirror_mod.mirror_object
 peration == "MIRROR_X";
irror_mod.use_x = True
irror_mod.use_y = False
!rror_mod.use_z = False
 _operation == "MIRROR_Y"
irror_mod.use_x = False
 llrror_mod.use_y = True
 Mrror_mod.use_z = False
  operation == "MIRROR_Z"
  rror_mod.use_x = False
  rror_mod.use_y = False
  rror_mod.use_z = True
 election at the end -add
   ob.select= 1
   er ob.select=1
   text.scene.objects.action
  "Selected" + str(modified
   rror ob.select = 0
  bpy.context.selected_obj
  ata.objects[one.name].se
 int("please select exactle
     OPERATOR CLASSES
      mirror to the selected
    ect.mirror_mirror_x"
  ext.active_object is not
```

Рефакторинг VS Оптимизация

И рефакторинг и оптимизация обычно не изменяют поведение программы

Оптимизация часто затрудняет понимание кода. Рефакторинг – упрощает.

Оптимизация позволяет «ускорить» работу программы. Рефакторинг может «замедлять» работу программы.

Рефакторинг VS Расширение функциональности программного обеспечения

при рефакторинге не изменяется поведение программы.

его зачастую применяют **до и/или после** расширения функциональности систем

Преимущества применения рефакторинга

Улучшает композицию программного кода

Облегчает читабельность программного кода

Помогает обнаружить ошибки

Помогает писать код быстрее



Причины применения

необходимость добавить новую функцию, которая недостаточно укладывается в принятое архитектурное решение;

необходимость исправить ошибку, причины возникновения которой сразу не ясны;

преодоление трудностей в командной разработке, которые обусловлены сложной логикой программы.

Признаки кода, нуждающегося в рефакторинге

(Code smell, also known as bad smell)

Code Smells - не сама проблема, а её индикатор

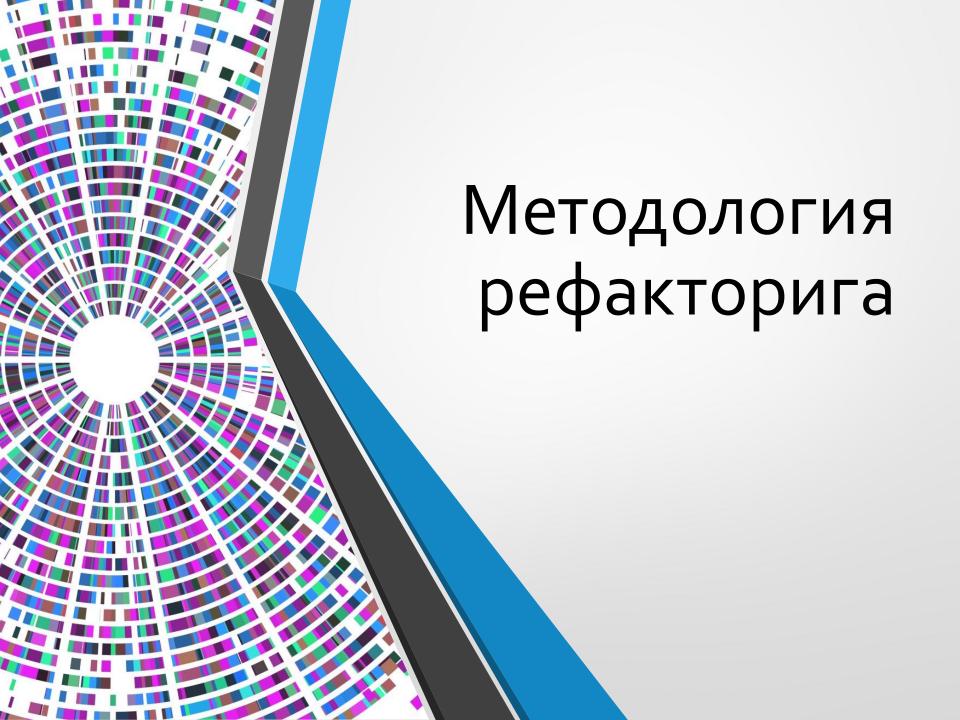
Существуют видимые проблемы в коде, требующие рефакторинга:

```
дублирование кода;
длинный метод;
большой класс;
длинный список параметров;
«завистливые» функции — метод, который
чрезмерно обращается к данным другого объекта;
«стрельба дробью»;
несгруппированные данные;
```

Проблемы, к которым приводит дублирование кода: Большое количество кода затрудняет его понимание.

Скрытое значение.

Аномалии обновления.



Наиболее применяемые методы рефакторинга	Изменение сигнатуры метода (Change Method Signature)
	Инкапсуляция поля (Encapsulate Field)
	Выделение класса (Extract Class)
	Выделение интерфейса (Extract Interface)
	Выделение локальной переменной (Extract Local Variable)
	Выделение метода (Extract Method)
	Генерализация типа (Generalize Type)
	Встраивание (Inline)
	Введение фабрики (Introduce Factory)
	Введение параметра (Introduce Parameter)
	Подъём метода (Pull Up Method)
	Спуск метода (Push Down Method)
	Переименование метода (Rename Method)
	Перемещение метода (Move Method)
	Замена условного оператора полиморфизмом (Replace Cond. with Polymorphism)
	Замена наследования делегированием (Replace Inheritance with Delegation)
	Замена кода типа подклассами (Replace Type Code with Subclasses)

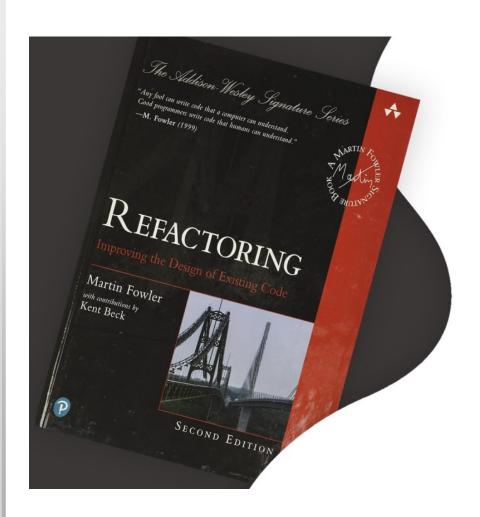
Список литературы



66666

Рефакторинг. Улучшение проекта существующего кода

Фаулер Мартинх и другие



Refactoring
Martin Fowler
2018



86666

Рефакторинг кода на JavaScript

Фаулер Мартин



Чистый код Роберт Мартин