

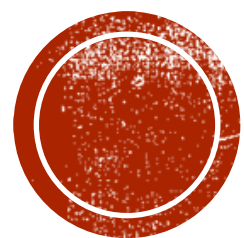
# ЯЗЫКИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ



A blue speech bubble with a black question mark inside is positioned in the upper left. Below it, a small, brown, blocky robot figure is visible. The background is a textured olive green with vertical stripes.

КАК НЕ НАПИСАТЬ  
УЖАСНЫЙ КОД?



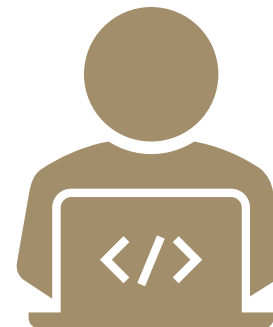


ПРИНЦИПЫ...

# DRY



Don't Repeat Yourself / Не повторяйтесь



Прежде чем что-либо писать, проявите прагматизм: посмотрите. Возможно, эта логика уже реализована в другом месте. Повторное использование кода – ~~всегда~~ разумное решение.





ПОЧЕМУ?

- Вам придется поддерживать одну и ту же логику и тестировать код сразу в двух местах, причем если вы измените код в одном месте, его нужно будет изменить и в другом.

# НО ЕСТЬ НЮАНС...

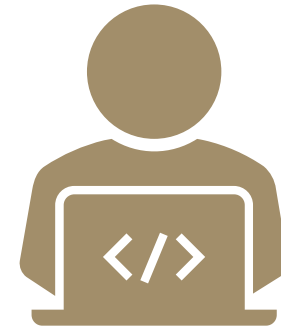
- Не всегда то, что кажется сейчас полным дублем – настоящий дубль.



# KISS



*Keep It Simple, Stupid / Будь проще*



Не придумывайте к задаче более  
сложного решения, чем ей требуется





ПОЧЕМУ?

- В простом классе/системе значительно легче обеспечить надежность



# ИЛИ

- «Понятный» код лучше «умного»

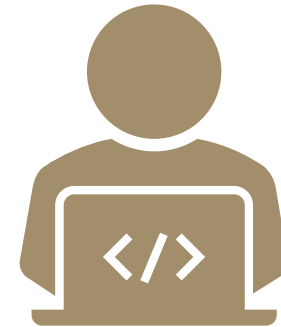


# YAGNI

---



You Aren't Gonna Need It / Вам это не понадобится



Если пишете код, то будьте уверены, что он вам понадобится. Не пишите код, если думаете, что он пригодится позже.

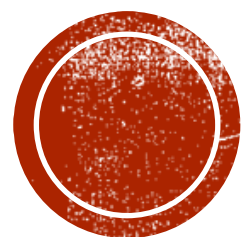




# ПОЧЕМУ?

- Тратится время, которое было бы затрачено на добавление, тестирование и улучшение необходимой функциональности.
- Новые функции должны быть отлажены, документированы и сопровождаться.
- Новая функциональность ограничивает то, что может быть сделано в будущем, — ненужные новые функции могут впоследствии помешать добавить новые нужные.
- ...





**SOLID**



# SOLID

- **SRP:** Single Responsibility Principle - Принцип единственной ответственности
  - **OCP:** Open-Closed Principle - Принцип открытости/закрытости
  - **LSP:** Liskov Substitution Principle - Принцип подстановки Барбары Лисков
  - **ISP:** Interface Segregation Principle - Принцип разделения интерфейсов
  - **DIP:** Dependency Inversion Principle - Принцип инверсии зависимости
- 
- Авторы:
    - Роберт Мартин
    - Майкл Физерс

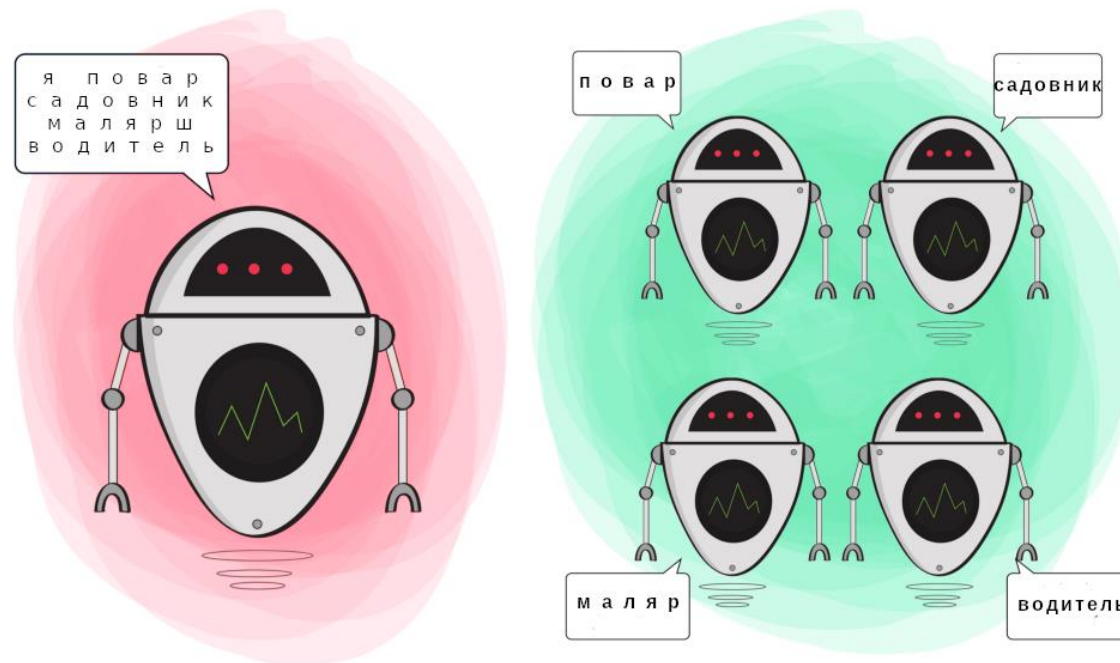
# SRP

- Модуль должен иметь одну и только одну причину для изменения
- Модуль - связный набор функций, классов, структур данных.
- Если у класса много обязанностей, это увеличивает вероятность возникновения ошибок, потому что внесение изменений в одну из его обязанностей, может повлиять на другую без вашего ведома.





# SRP



**X** Single Responsibility **✓**  
Принцип Единой Ответственности

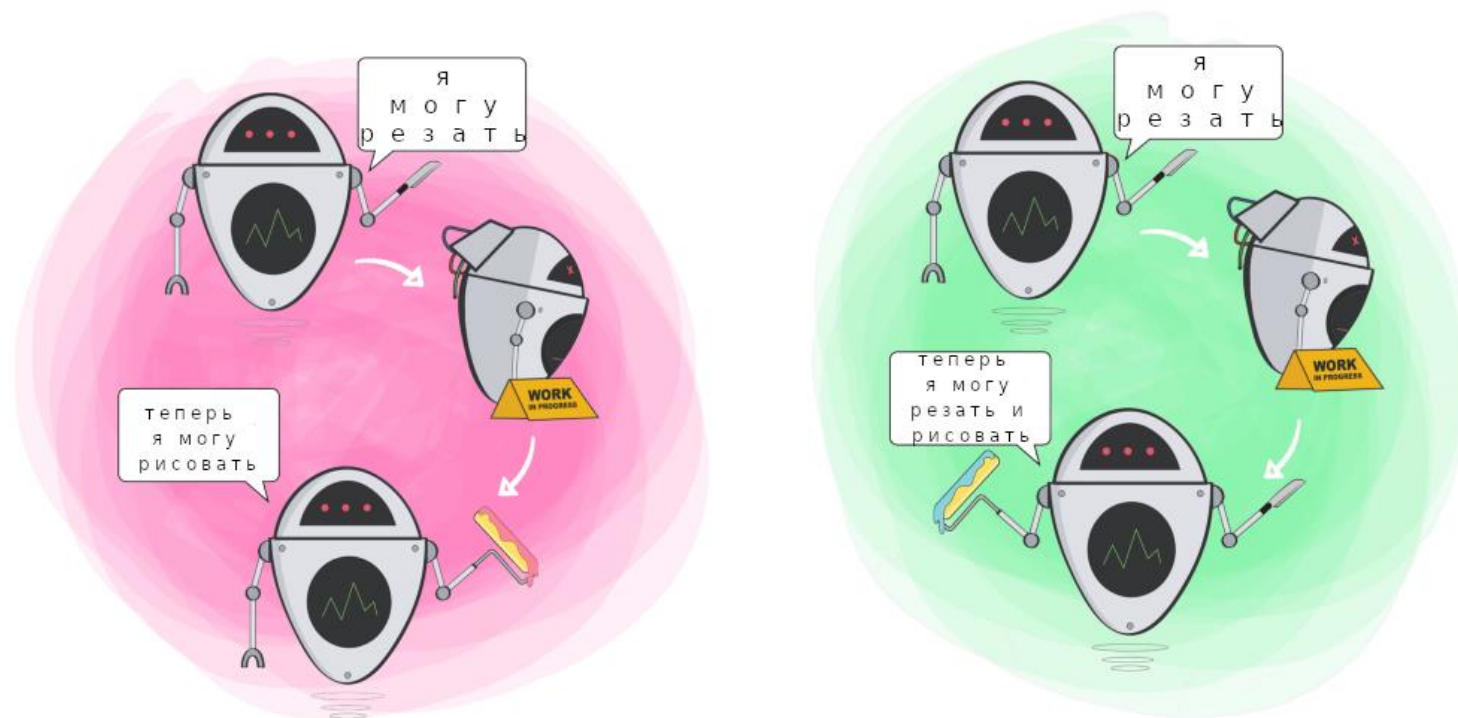


# ОСР

- Программные сущности должны быть открыты для расширения и закрыты для изменения.
- Изменение текущего поведения класса повлияет на все системы, использующие этот класс.
- Если вы хотите, чтобы Класс выполнял больше функций, то добавьте новый функционал НЕ изменяя существующие.



# ОСР



Оpen - Closed  
Принцип открытости/закрытости





# LSP

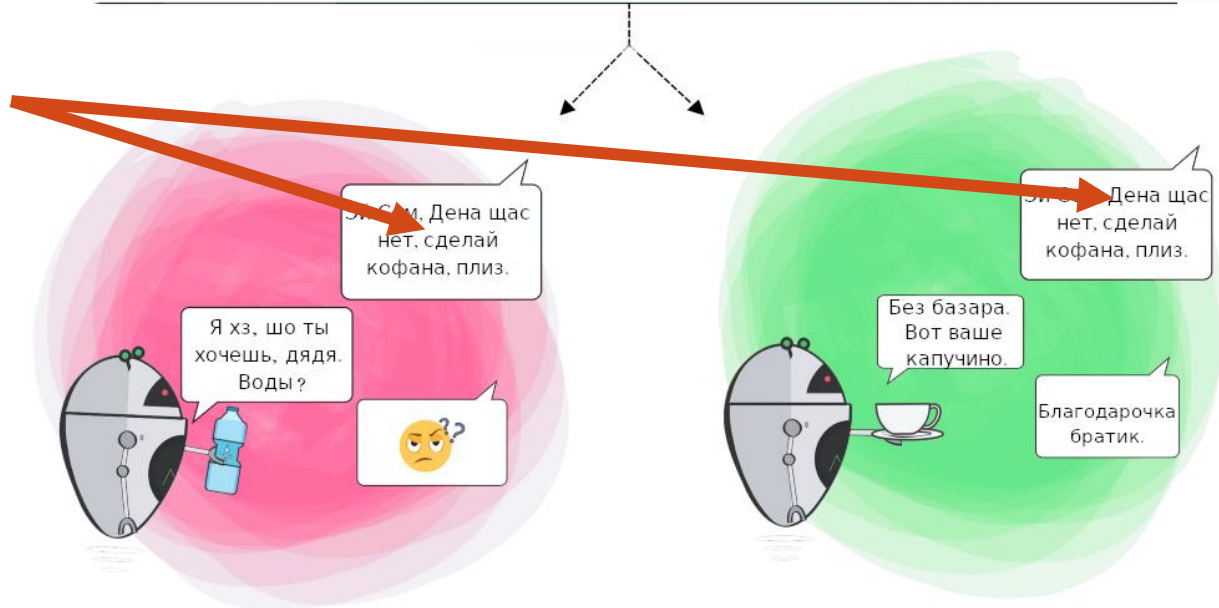
- Если  $S$  является подтипом  $T$ , то объекты типа  $T$  в программе могут быть заменены объектами типа  $S$  без изменения каких-либо дополнительных свойств этой программы.
- Цель - обеспечение последовательности:
  - родительский класс или его дочерний класс могут использоваться одинаковым образом, и взаимозаменяемы, без каких-либо ошибок.



# LSP



Тут просят  
Дена сделать  
кофе...



Liskov Substitution

Принцип подстановки Барбары Лисков

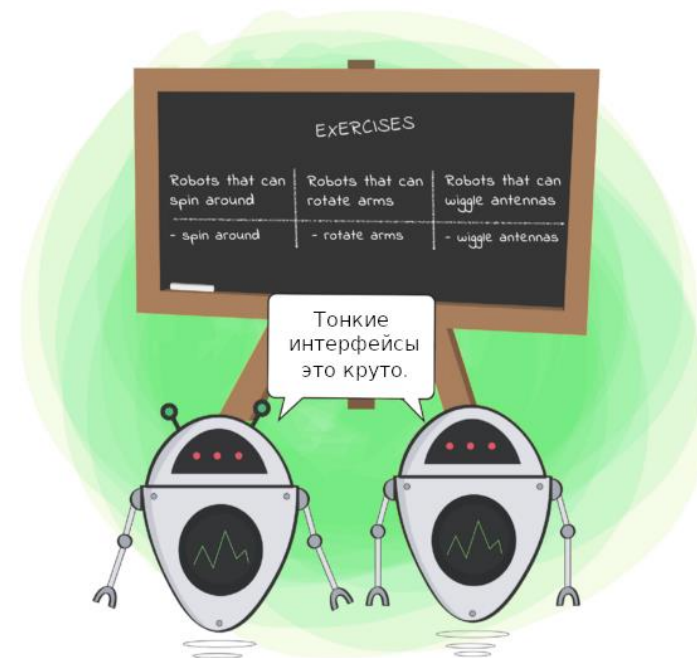
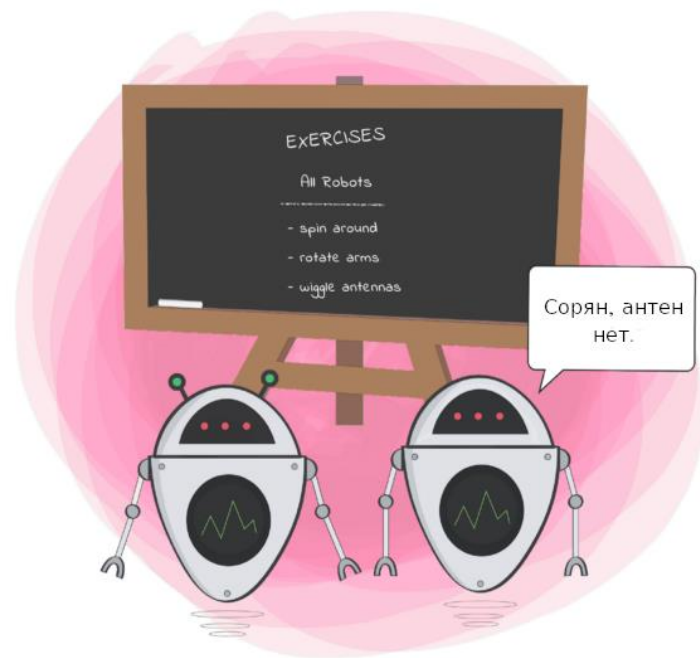


# ICP

- Клиенты не должны зависеть от методов, которые они не используют.
- Разделяйте «толстые» интерфейсы, которые "делают всё", на более узконаправленные интерфейсы, решающие узконаправленную задачу.



# ICP



Interface Segregation  
Принцип Сегрегации интерфейсов



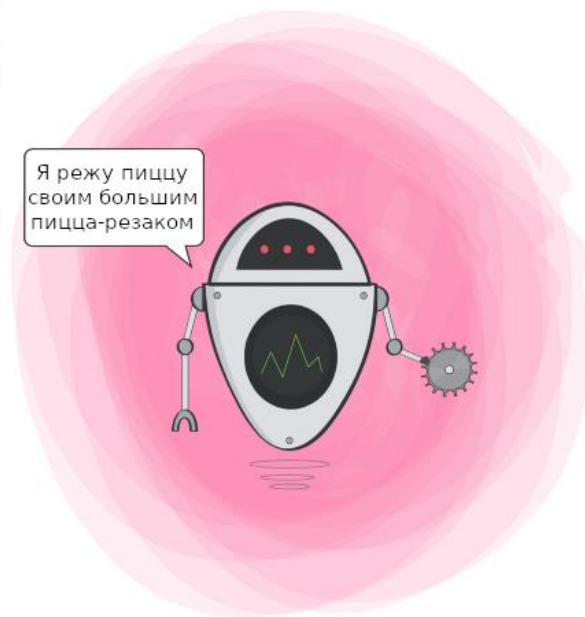


# DIP

- Высокоуровневые модули не должны зависеть от более низкоуровневых модулей. Оба должны зависеть от абстракции.
- Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.
- Набор правил:
  - Не ссылайтесь на изменчивые конкретные классы, ссылайтесь на абстрактные интерфейсы
  - Не наследуйте изменчивые конкретные классы
  - Не переопределяйте конкретные функции, делайте функции абстрактными и добавляйте несколько реализаций.
  - Не ссылайтесь на имена конкретных и изменчивых сущностей

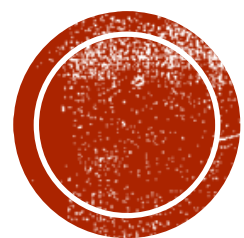


# DIP



Dependency Inversion  
Принцип инверсии зависимостей





ОПТИМИЗАЦИЯ



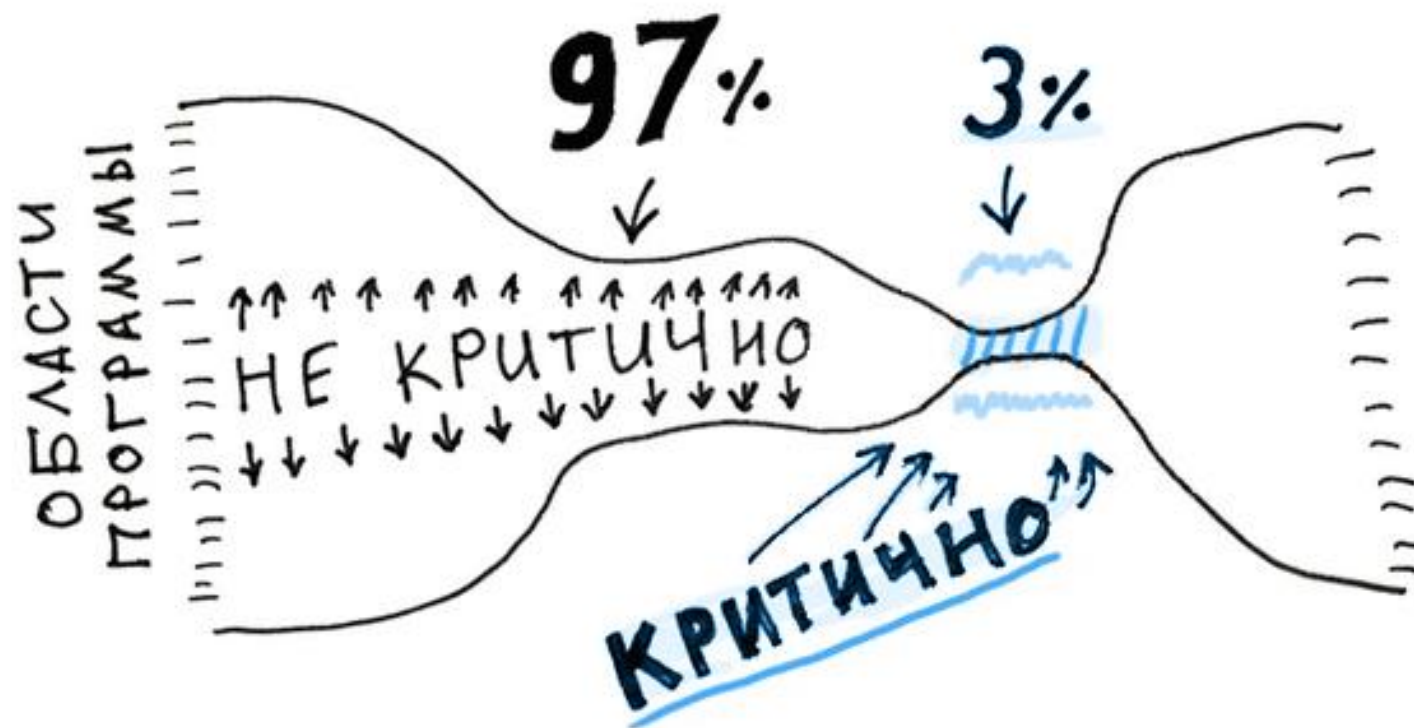


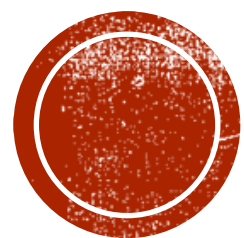
# ОПТИМИЗАЦИЯ

Программисты тратят огромное количество времени, размышляя и беспокоясь о некритичных местах кода, и пытаются оптимизировать их, что исключительно негативно сказывается на последующей отладке и поддержке. Мы должны вообще забыть об оптимизации в, скажем, 97% случаев; более того, поспешная оптимизация является корнем всех зол. И напротив, мы должны уделить все внимание оставшимся 3%.

*Дональд Кнут*







РЕФАКТОРИНГ



# РЕФАКТОРИНГ

- Рефакторинг – изменение кода программы без изменения поведения, исправления ошибок и т.п.
- Цель – улучшить код, а именно - сделать его более:
  - Читаемым и понятным
  - Расширяемым





КАК НИЧЕГО НЕ  
СЛОМАТЬ?



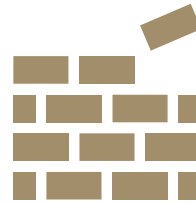


# КАК ПРАВИЛЬНО РЕФАКТОРИТЬ?

---



Отдельно от добавления  
нового функционала



Небольшими частями, а  
не все сразу



Проверяя, что логика не  
поменялась (тесты)





The image features a bright yellow background with a subtle grid pattern. In the upper left, there is a blue, rounded square icon containing a black question mark. In the lower center, a small, stylized wooden robot figure is visible, composed of various geometric shapes like spheres and cylinders. Overlaid on the center of the image is the Russian text 'КОГДА НУЖНО РЕФАКТОРИТЬ?' in a large, white, serif font.

КОГДА НУЖНО  
РЕФАКТОРИТЬ?



# КОГДА РЕФАКТОРИТЬ?

- Когда проблема встречается не первый раз
- Когда цена отсутствия рефакторинга выше его игнорирования
- Когда есть время на рефакторинг
- Когда рефакторинг «попутный» основной задаче
- **Когда есть мотивация (причина) рефакторить**







А ЧТО  
РЕФАКТОРИТЬ?





# КОД С «ДУШКОМ» (CODE SMELL)

- Мертвый код
- Дублирование (DRY)
- Длинный метод
- Большой класс
- Большой список параметров
- «Завистливый объект» - обращение к данным чужого объекта чаще, чем к своим
- Большое количество комментариев
- Данные которые всегда «ходят» вместе
- Большое количество **switch-case**
- «Циклический ад» - слишком много циклов
- ...







ЗАЧТО ПЛАТЯТ  
ХОРОШЕМУ  
РАЗРАБОТЧИКУ?





- Сразу сесть писать код — **ошибка**.
- Легко потратить уйму времени на работу в стол.
- Начинать надо с проблемы, которую код призван решать.

