



Правила хорошего кода

Бернгардт Григорий, руководитель iOS направления

Хороший код?

ЛИСТИНГ 1

```
float isqrt2( float number )  
{  
    return 1 / sqrt(num);  
}
```

ЛИСТИНГ 2

```
float isqrt( float number )  
{  
    return 1 / pow(num,0.5);  
}
```

Листинг 3



ЛИСТИНГ 4

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                // evil floating point bit level
    hacking
    i  = 0x5f3759df - ( i >> 1 );        // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
    //y  = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be
    removed

    return y;
}
```

Какой лучше?

ЛИСТИНГ 3

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                // evil floating point bit level
    hacking
    i  = 0x5f3759df - ( i >> 1 );        // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
    //y  = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be
    removed

    return y;
}
```

https://ru.wikipedia.org/wiki/Быстрый_обратный_квадратный_корень

<http://imgtfy.com/?q=0x5f3759df>

Хороший код?

Код которого нет

Зачем писать код?

Производительность VS ПОНЯТНОСТЬ

```
int SimpleMultiplyBy2(int x)
{
    return x * 2;
}
```

```
int SimpleMultiplyBy2(int x)
{
    return x << 1;
}
```

Производительность VS ПОНЯТНОСТЬ

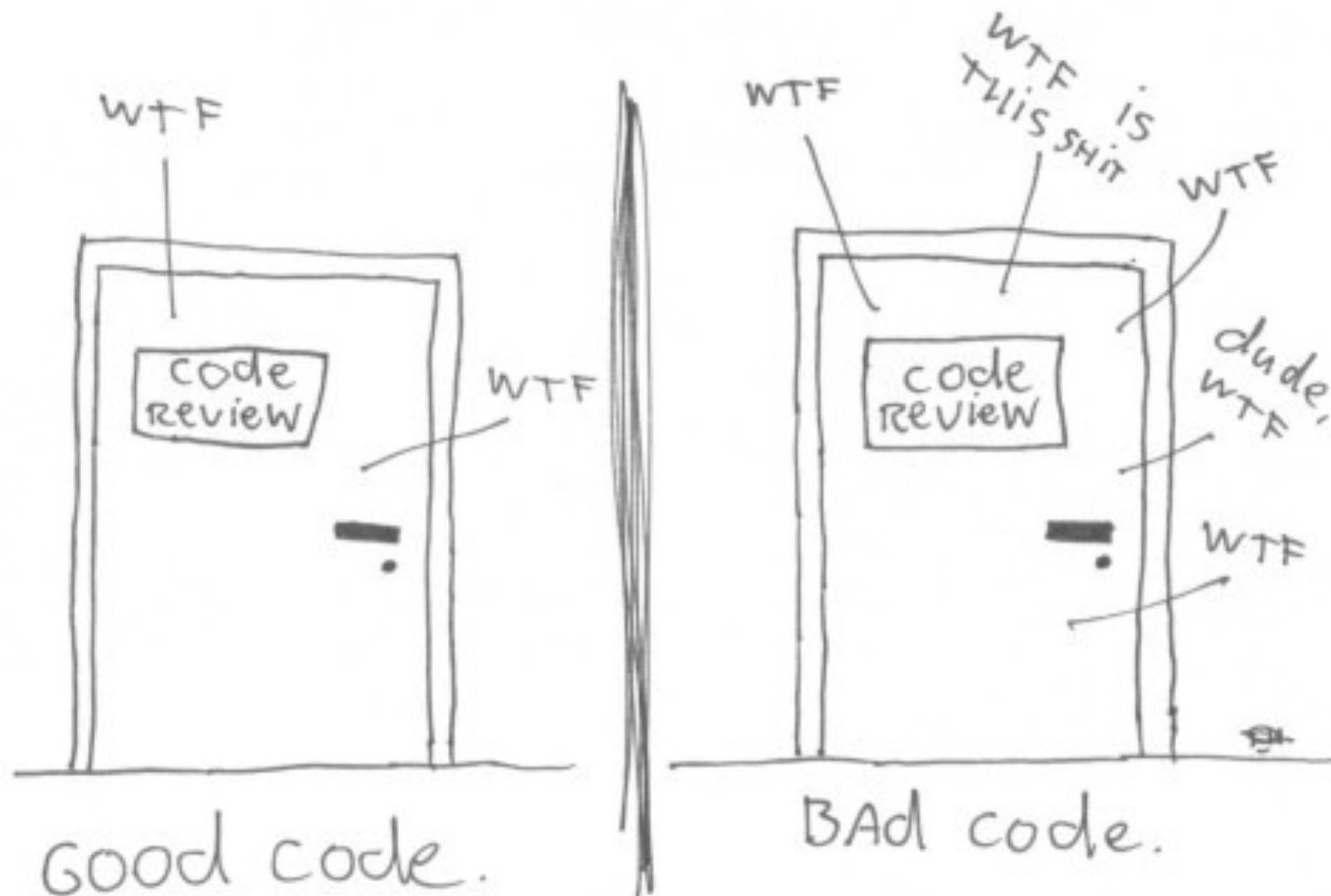
```
SimpleMultiplyBy2:  
    push rbp  
    mov rbp, rsp  
    mov DWORDPTR[rbp-4], edi  
    mov eax, DWORDPTR[rbp-4]  
    add eax, eax  
    pop rbp  
    ret
```

**“Programs must be written for people to read, and
only incidentally for machines to execute ”**

–Х. Абельсон, Д.Д. Сассман

“Структура и интерпретация компьютерных программ”

The ONLY valid MEASUREMENT
of code QUALITY: WTFs/minute



Хороший код?

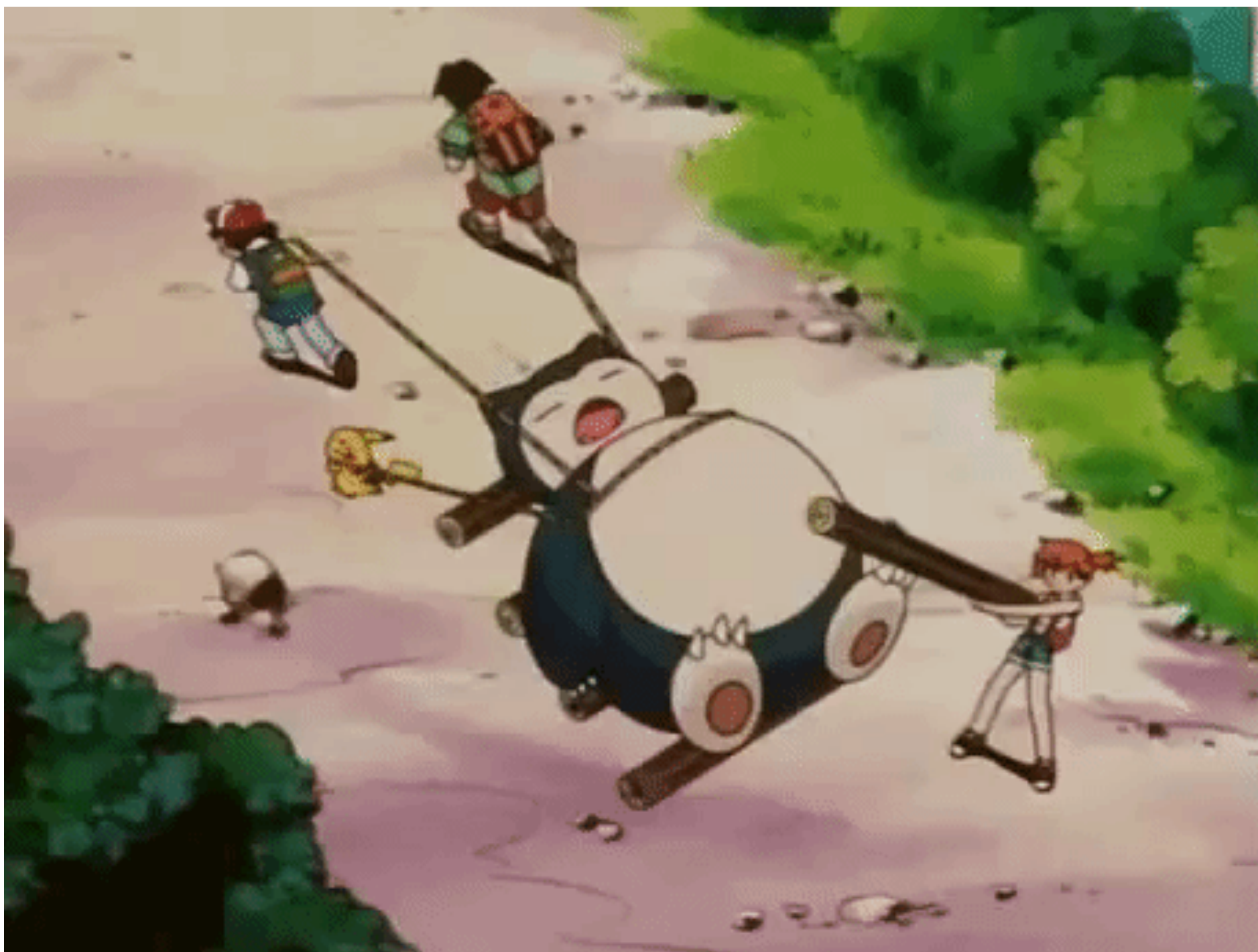
Понятный код

**Зачем писать
хороший код?**

Для себя

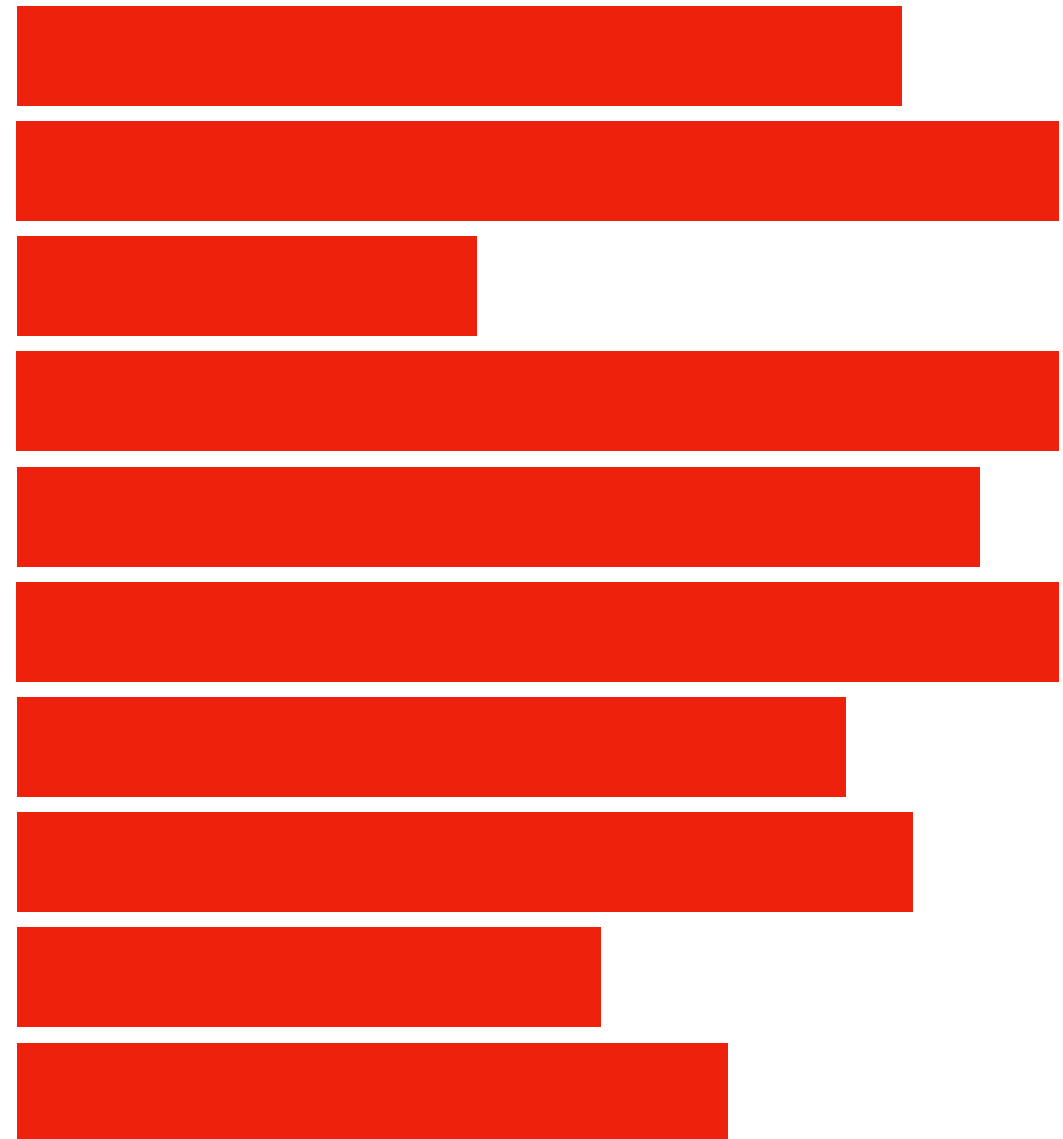
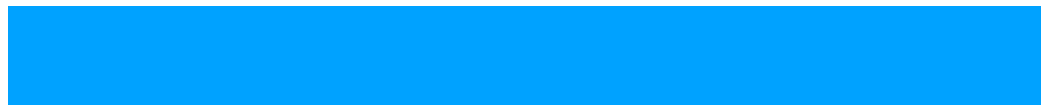
**“Ты помнишь зачем
написал этот метод в
прошлом месяце?”**

Мы не одни



Пишем > Читаем?

Пишем > Читаем?



– Р. Мартин “Чистый код”

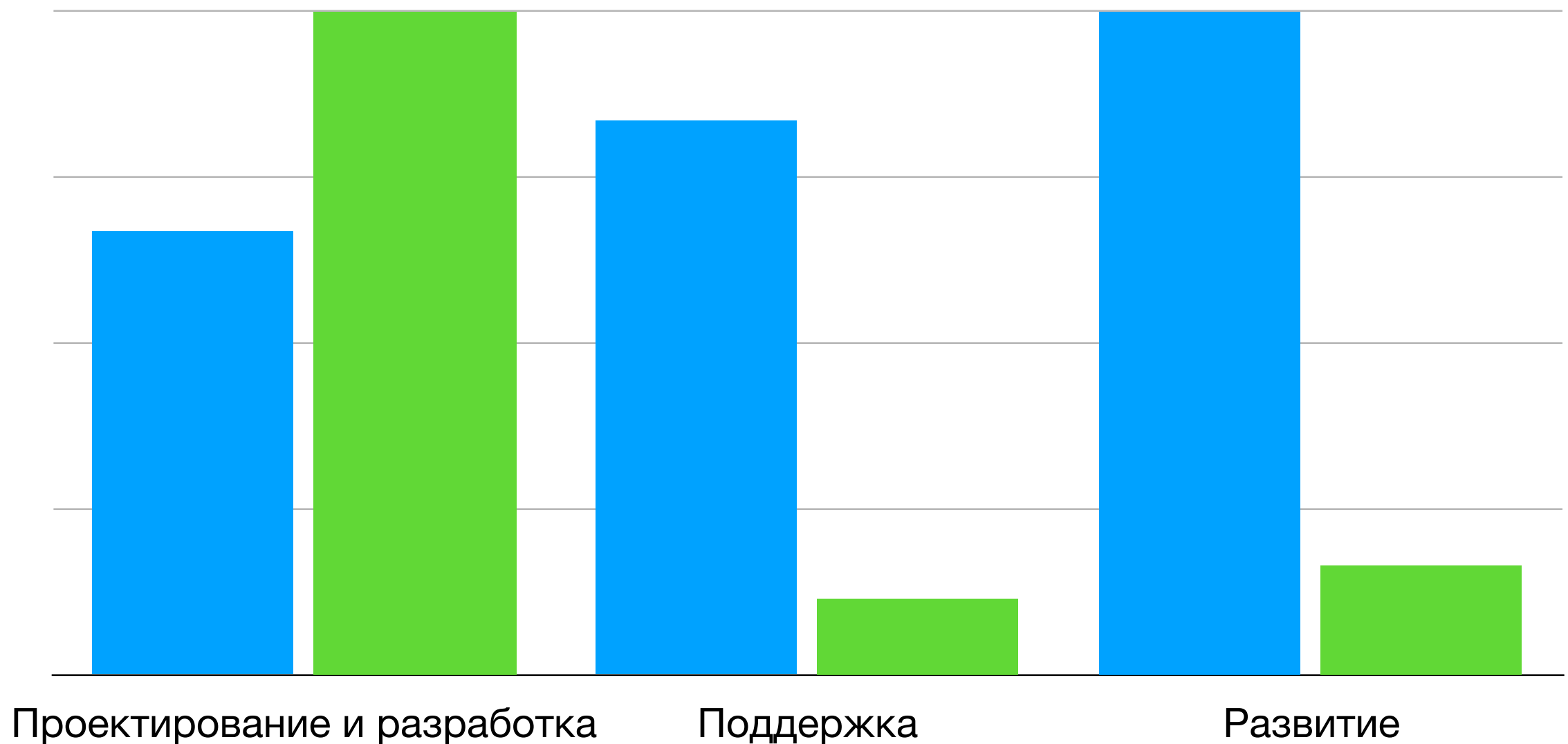
Сокращаем поддержку

Чем выше качество кода
тем меньше времени
на поддержку

Сокращаем поддержку

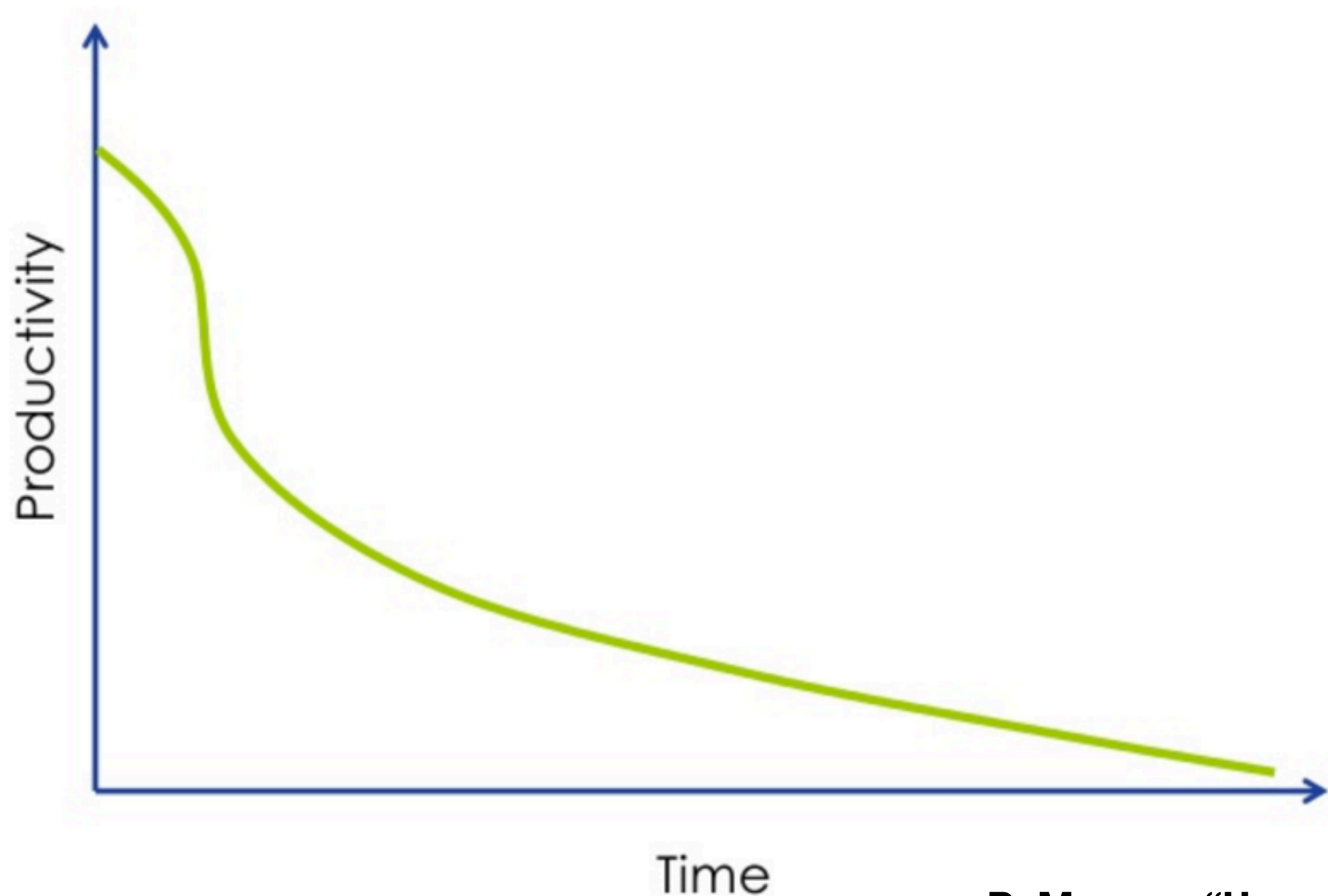
■ Плохой код

■ Хороший код



– С. Макконнелл “Совершенный код”

Взять хаос под контроль



– Р. Мартин “Чистый код”

Что такое хороший код

Зачем нужен хороший код

Как писать хороший код

Что такое хороший код

Зачем нужен хороший код

Как писать хороший код

Что такое хороший код

Зачем нужен хороший код

Как писать хороший код

Базовые принципы

- Написание кода – не цель
- Лучший код – не написанный
- Читаемый код важнее быстрого
- Шаблоны проектирования – не принципы программирования

Принципы проектирования

- KISS
- DRY
- YAGNI
- SOLID
- FIRST

KISS

Keep It Simple Stupid

DRY

Don't Repeat Yourself

Как же остальные?

- YAGNI
- SOLID
- FIRST
- ...

Признаки хорошего кода

- Легко читать
- Легко вносить изменения
- Сильная связность компонентов (cohesion)

Признаки плохого кода

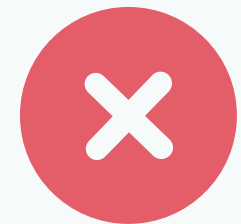
- На понимание тратится много времени
- Конструкции не помещаются в экран (как вертикально так и горизонтально)
- Код требует комментариев
- Высокая связанность между компонентами (coupling)
- Дублирование кода

Пример 1

```
const fetchUser = (id) => (  
  fetch(buildUri`/users/${id}`) // Get User DTO record from REST API  
    .then(convertFormat) // Convert to snakeCase  
    .then(validateUser) // Make sure the the user is valid  
);
```

Комментарии

```
const fetchUser = (id) => (  
  fetch(buildUri`/users/${id}`) // Get User DTO record from REST API  
    .then(convertFormat) // Convert to snakeCase  
    .then(validateUser) // Make sure the the user is valid  
);
```



```
const fetchUser = (id) => (  
  fetch(buildUri`/users/${id}`)  
    .then(snakeToCamelCase)  
    .then(validateUser)  
);
```



Комментарии



Пример 2

```
class Math1 {  
    public static Integer sum(List<Integer> numbers) {  
        Integer sum = 0;  
        for (Integer n : numbers) {  
            sum += n;  
        }  
        return sum;  
    }  
  
    public static Integer multiply(List<Integer> numbers) {  
        Integer product = 1;  
        for (Integer n : numbers) {  
            product *= n;  
        }  
        return product;  
    }  
}
```

Дублирование

```
class Math1 {  
    public static Integer sum(List<Integer> numbers) {  
        Integer sum = 0;  
        for (Integer n : numbers) {  
            sum += n;  
        }  
        return sum;  
    }  
  
    public static Integer multiply(List<Integer> numbers) {  
        Integer product = 1;  
        for (Integer n : numbers) {  
            product *= n;  
        }  
        return product;  
    }  
}
```



Дублирование

```
class Math1 {  
    public static Integer reduce(  
        List<Integer> numbers,  
        Integer unit, BinaryOperator<Integer> operator  
    ) {  
        Integer result = unit;  
        for (Integer n : numbers) {  
            result = operator.apply(result, n);  
        }  
        return result;  
    }  
  
    public static Integer sum(List<Integer> numbers) {  
        return reduce(numbers, 0, (x, y) -> x + y);  
    }  
  
    public static Integer multiply(List<Integer> numbers) {  
        return reduce(numbers, 1, (x, y) -> x * y);  
    }  
}
```



Пример 3

```
function showMessage(message) {  
  {  
    if(message != null){  
      show(message);  
    }  
    else{  
      show(defaultMessage); //This is the default action to do  
    }  
  }  
}
```


Golden Path

```
function showMessage(message) {  
  {  
    if(message != null){  
      show(message);  
    }  
    else{  
      show(defaultMessage); //This is the default action to do  
    }  
  }  
}
```



```
function showMessage(message) {  
  {  
    if(message != null){  
      show(message);  
      return;  
    }  
    show(defaultMessage);  
  }  
}
```



Пример 4

```
if index >= 0 && index < doughnuts.count {  
    print("Eating doughnut")  
}
```

Уточняющие переменные



```
if index >= 0 && index < doughnuts.count {  
    print("Eating doughnut")  
}
```



```
let needsToEat = index >= 0 && index < doughnuts.count  
if needsToEat {  
    print("Eating doughnut")  
}
```

Как поддерживать код

- Создать базу для хорошего кода
- Исправлять сразу
- Поддерживать качество кода

Создать базу

- Code Style
- Техническая документация
- Структура проекта

Code Style



Документ, в котором описаны правила работы с кодом

- <https://github.com/surfstudio/SwiftCodestyle>
- <https://github.com/surfstudio/android-code-style>
- <https://github.com/surfstudio/java-code-style>
- <https://github.com/surfstudio/objective-c-style-guide>

Техническая документация



Документ, в котором описаны...

- договоренности команды по работе над проектом
- технические требования и особенности проекта
- инструкции по настройке окружения
- неявные детали реализации

Структура проекта



Для всего кода есть свое место в проекте

Исправлять сразу


- Продумывать реализацию заранее
- Следить за качеством кода
- Использовать генераторы кода
- Практиковать ревью кода и/или парное программирование

Продумывать реализацию

🤔 Цена ошибки возрастает со временем


- Подумай, что хочешь реализовать
- Не пиши идеальный код
- Не копайся в деталях
- Не доверяй своему коду
- Используй шаблоны проектирования

Следить за качеством

 Использовать инструменты для слежения за качеством кода

- Статический анализ кода
- Автоматизированные тесты
- CI

Использовать генераторы кода

 Чем больше рутинных действий автоматизируется
тем больше сил остается на важное

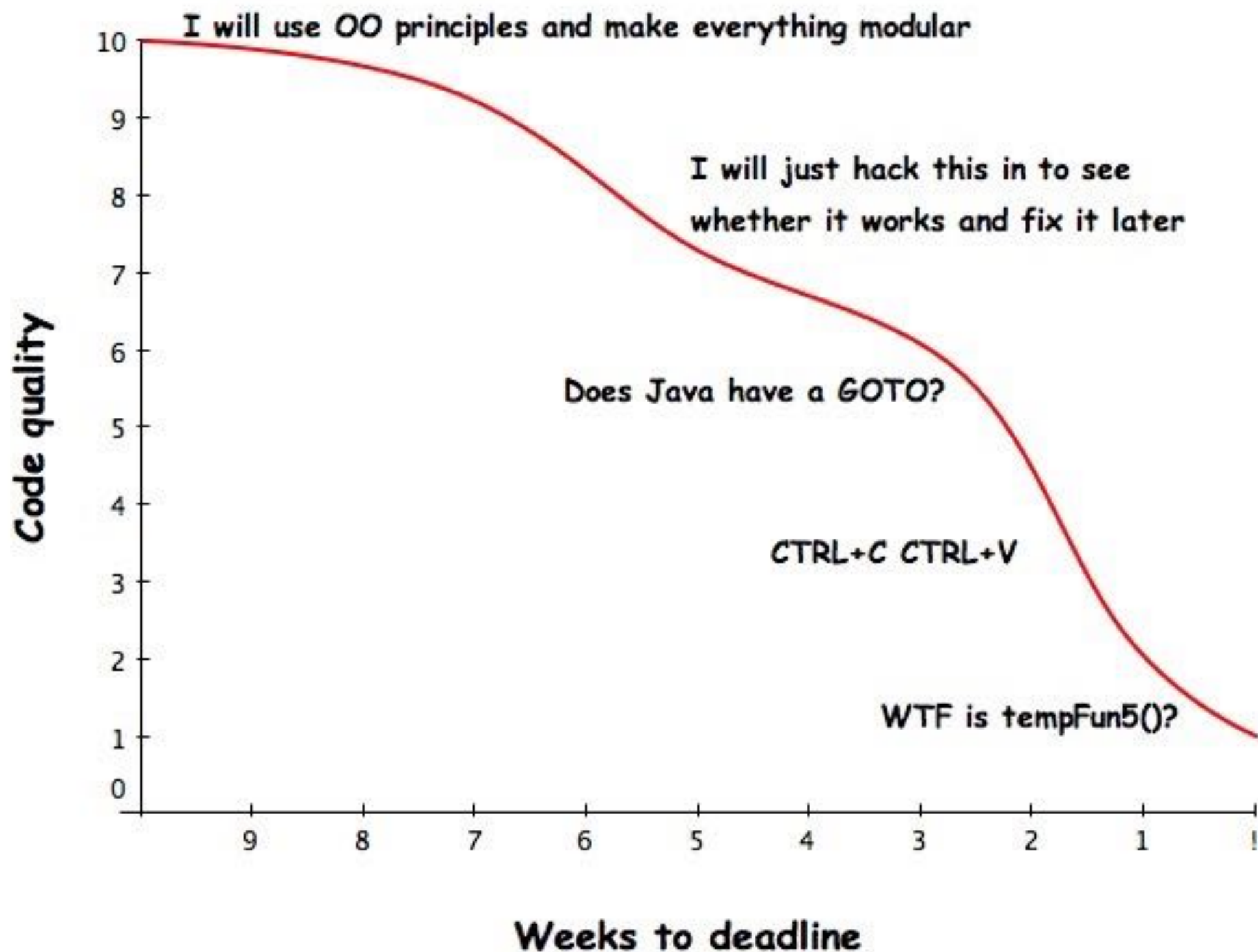
- <https://github.com/rambler-digital-solutions/Generamba>
- <https://developer.android.com/studio/projects/templates>
- <https://github.com/krzysztofzablocki/Sourcery>
- <https://github.com/SwiftGen/SwiftGen>
- ...

Ревью кода

👁️ Позволяет отловить большую часть проблем на раннем этапе

- Участвуют все
- Обязательно для попадания кода в основную ветку разработки

Поддерживать качество



Поддерживать качество

- Коллективное владение кодом
- Правило бойскаута
- Планировать рефакторинг постоянно

Правило бойскаута

Оставляй за собой лучше
чем было до тебя

Что такое хороший код

Зачем нужен хороший код

Как писать хороший код

Материалы

- Р. Мартин – «Чистый код»
- С. Макконнелл – «Совершенный код»
- Х. Абельсон, Д.Д. Сассман – «Структура и интерпретация компьютерных программ»
- М. Фаулер – «Рефакторинг. Улучшение существующего кода»

