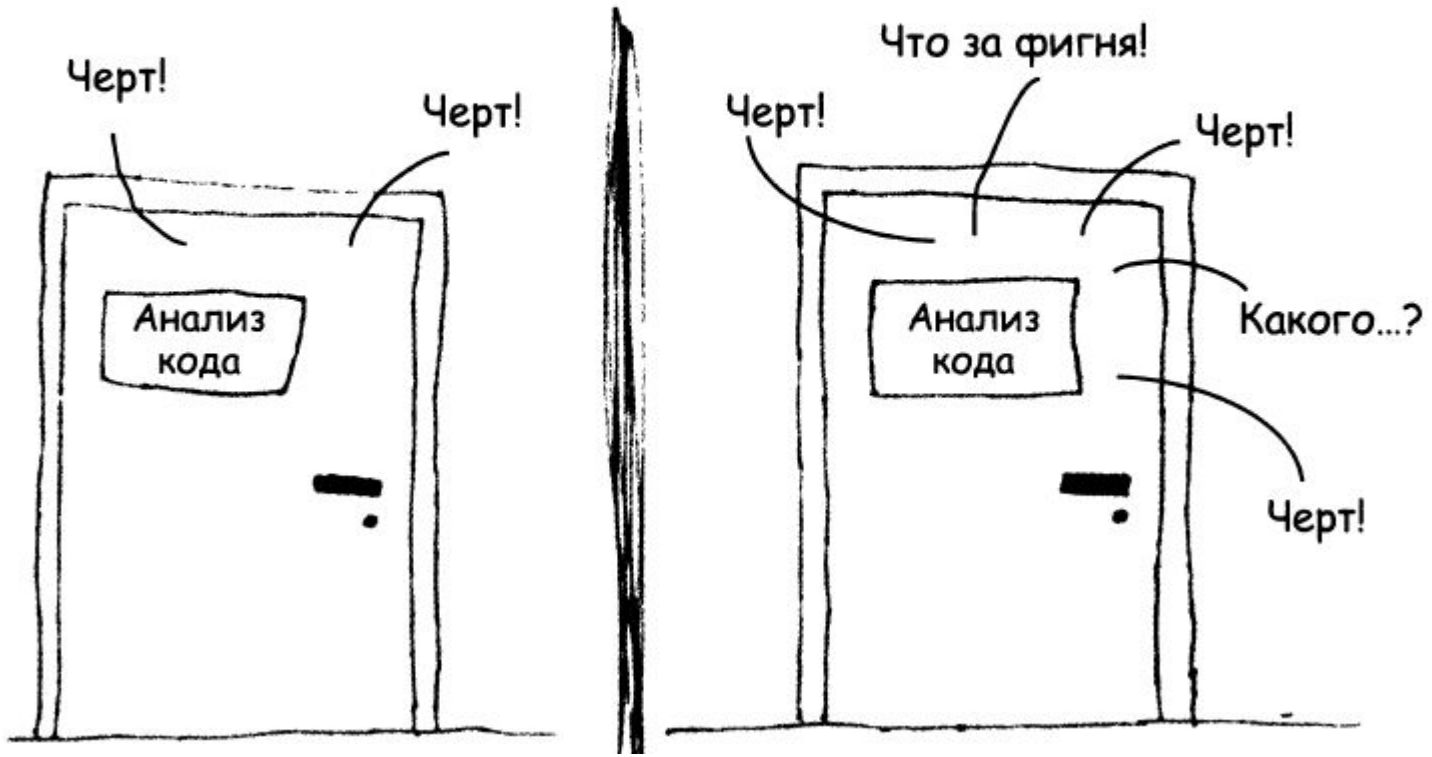
Чистый код: создание, анализ и рефакторинг.

Единственная надежная метрика качества кода:
количество «чертей» в минуту



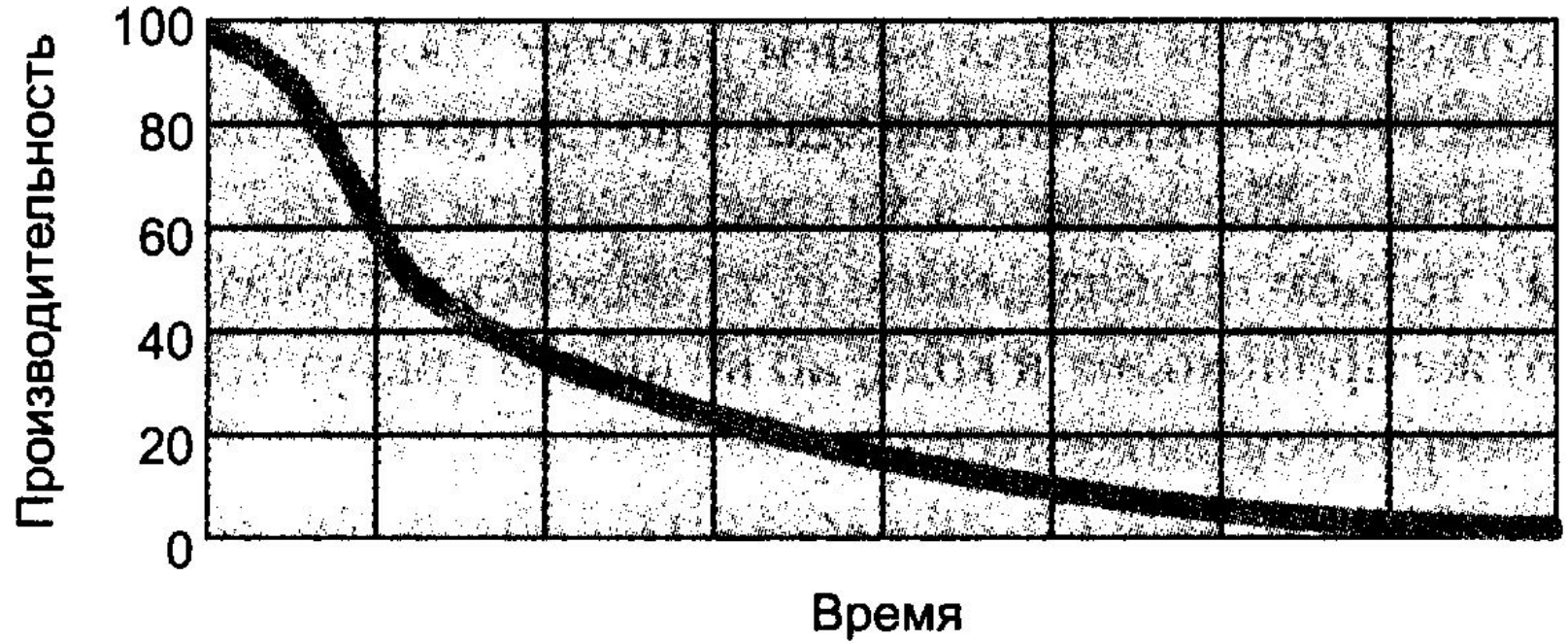


Рис. 1.1. Зависимость производительности от времени

Какой код называют “чистым”?

- Код приятно читать
- Удобно и возможно сопровождать
- Код не содержит дублирования
- Содержит простые абстракции
- Прямолинеен и компактен

Содержательные имена

- Не надо так: int d; //прошедшее время
- Лучше так:
  - int daysSinceCreation;
  - int fileAgeInDays;

Пример. Что делает эта функция?

```
public List<int[]> getThem()
    {
        List<int[]> list1 = new List<int[]>();
        foreach (int[] x in theList)
            if (x[0] == 4)
                list1.Add(x);
        return list1;
    }
```

Улучшение 1.

```
public List<int[]> getFlaggedCells()
    {
        List<int[]> flaggedCells = new List<int[]>();
        foreach (int[] cell in gameBoard)
            if (cell[STATUS_VALUE] == FLAGGED)
                flaggedCells.Add(cell);
        return flaggedCells;
    }
```

Улучшение 2.

```
public List<Cell> getFlaggedCells()
    {
        List<Cell> flaggedCells = new List<Cell>();
        foreach (Cell cell in gameBoard)
            if (cell.isFlagged())
                flaggedCells.Add(cell);
        return flaggedCells;
    }
```

Имена классов

- Существительные и их комбинации
  - Customer, WikiPage, Account, AddressParser, …

Имена методов

- Глаголы или глагольные словосочетания
  - PostPayment, DeletePage, Save, …

Функции

- Компактность (примерно 20 строк)
- Функция должна выполнять одну операцию и ничего другого
- Один уровень абстракции на функцию
- Чтение кода сверху вниз

Разделение команд и запросов

```
if (set(“username”, “unclebob”))


 if (AttributeExists("username"))
{
        SetAttribute("username", "unclebob");
        ...
}
```

Форматирование кода

```
public class MyClass
{
    private readonly int _somePrivateField;
    private readonly int _someOtherPrivateField;

    private const int SomeConst = 5;

    public MyClass(int parameter)
    {
        SetParameter(parameter);
    }



    public int Parameter { get; protected set; }



    public void SetParameter(int parameter)
    {
        Parameter = parameter;
    }


    private void SomePrivateMethod()
    {
    }
}
```

Объекты и структуры данных. Процедурный код.

```
public class Geometry
{
    public readonly double PI = 3.141592653589793;

    public double Area(object shape)
    {
        if (shape is Square)
        {
            Square s = (Square)shape;

            return s.side * s.side;

        }
        else if (shape is Rectangle)
        {
            Rectangle r = (Rectangle)shape;

            return r.height * r.width;
        }
        else if (shape is Circle)
        {
            Circle c = (Circle)shape;

            return PI  c.radius * c.radius;
        }

        throw new Exception("No such shape");
    }
}
```

```
public class Square
{
    public Point topLeft;

    public double side;
}
```

```
public class Rectangle
{
    public Point topLeft;
    public double height;
    public double width;
}
```

```
public class Circle
{
    public Point center;
    public double radius;
}
```

Объекты и структуры данных. Объектно-ориентированный код.

```
public abstract class Shape
{
    public abstract Area();
}
```

```
public class Square : Shape
{
    private Point topLeft;
    private double side;

    public override double Area()
    {
        return side * side;
    }
}
```

```
public class Circle : Shape
{
    private Point center;
    private double radius;
    private double PI = 3.141592653589793;

    public override double Area()
    {
        return PI * radius * radius;
    }
}
```

```
public class Rectangle : Shape
{
    private Point topLeft;
    private double height;
    private double width;

    public override double Area()
    {
        return height * width;
    }
}
```

Обработка ошибок

```
public List<Point> GetPoints()
{
    IList<Point> points = new List<Point>();

    try
    {
        points = GetPoints();

        return points;
    }
    catch(Exception exception)
    {
        Log(exception);

        throw new Exception("Exception", exception);
    }
}
```

SOLID

- S – SRP – Принцип единой ответственности
- O – OSP – Принцип открытости/закрытости
- L – LSP – Принцип подстановки Барбары Лисков
- I – ISP – Принцип разделения интерфейса
- D – DIP – Принцип инверсии зависимостей

KISS

- Keep it short and simple (keep it simple, stupid)

DRY

- Don’t repeat youserlf

- Антоним: WET – Write Everything Twice

YAGNI

- You ain’t gonna need it

Рефакторинг