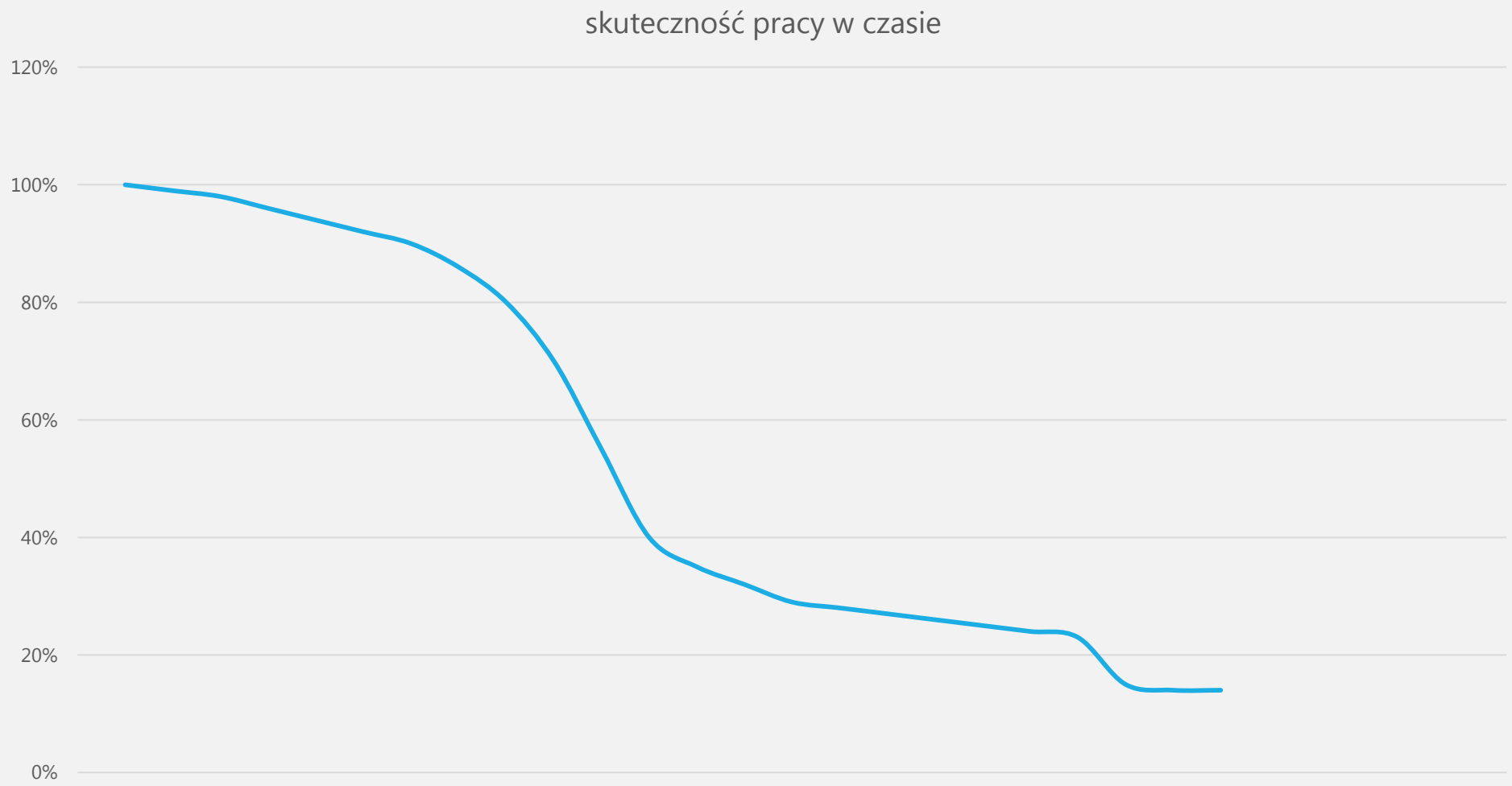
SOLIDnie śmierdzący kod

http://www.benedykt.net

abenedykt

produktywność



ile kosztuje pracownik

5 programistów x **2000PLN BRUTTO** x **24** miesiące = **240 000 PLN**

2000PLN Brutto =

Kwota netto	**Rodzaj zatrudnienia**
1420 zł	umowa o pracę
1480 zł	umowa zlecenie
1700 zł	umowa zlecenie - student do 26 lat
1800 zł	umowa o dzieło
1639 zł	faktura Vat

co można kupić za 240 000 ?

1.8 TFSI, 120KM
116 200 PLN BRUTTO

3.0 TDI quattro 245KM
215 600 PLN BRUTTO

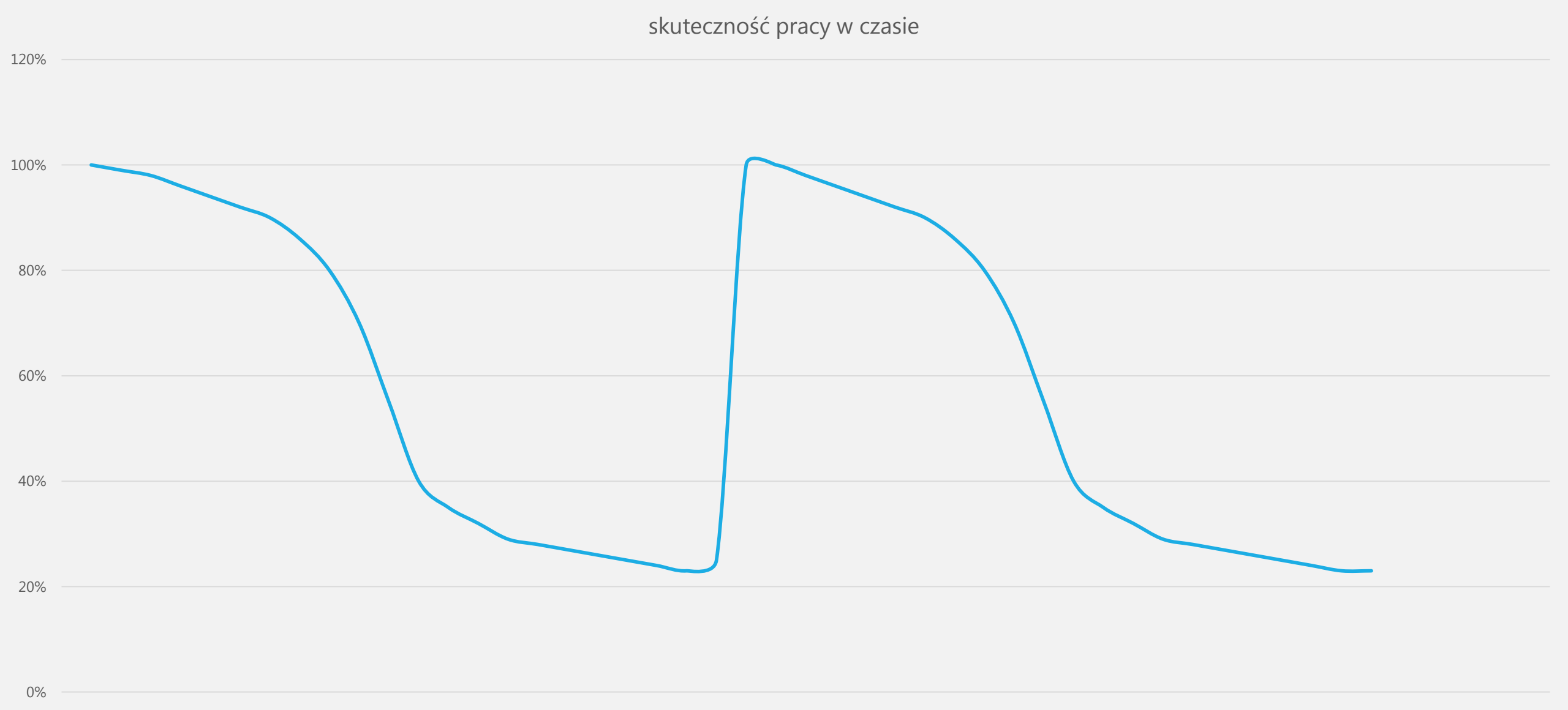
co można kupić za 240 000 ?

~240 000 PLN

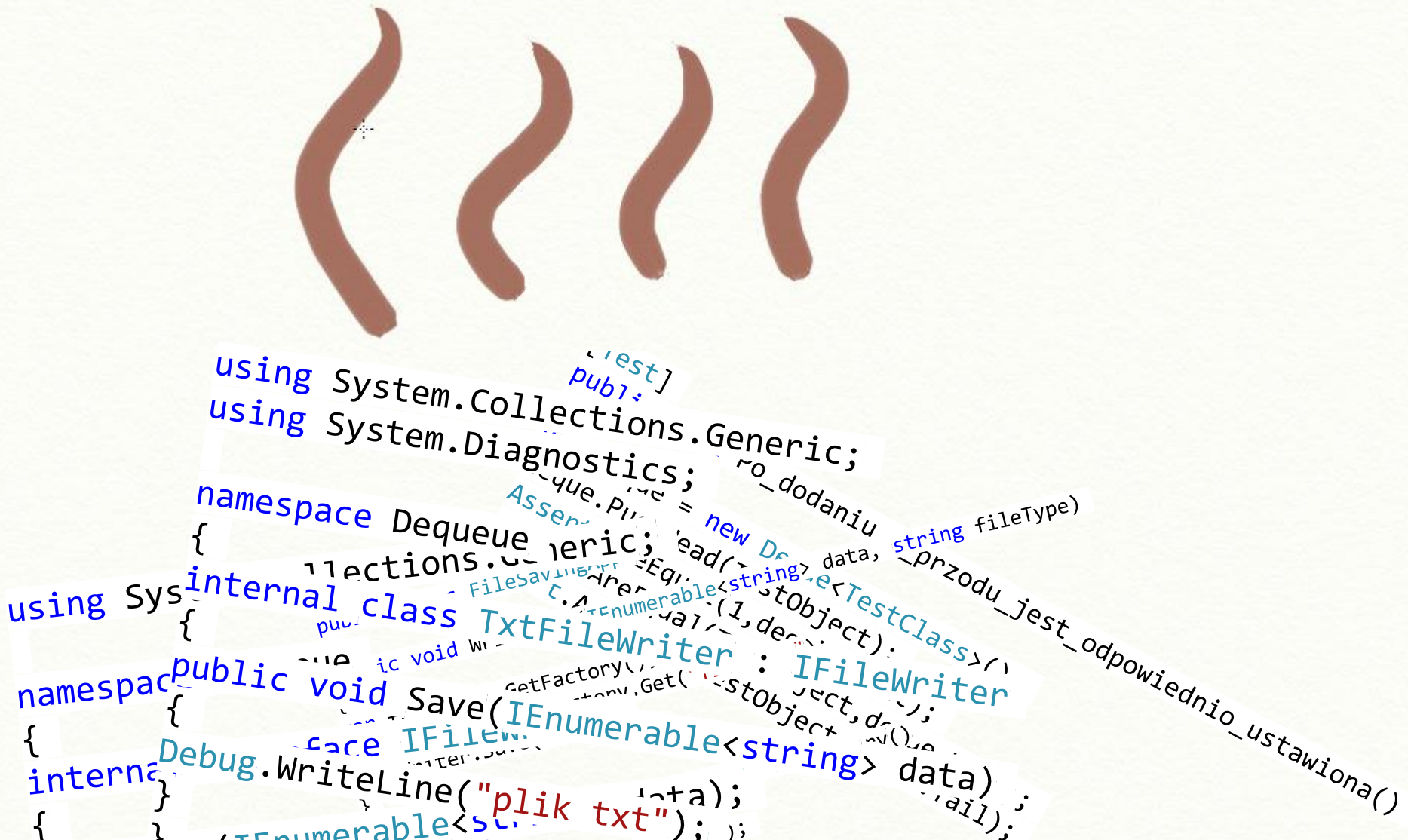
520i 6-man R4 / 1 997 135 (184) 8,9 / 5,5 / 6,8 157 141 382 **173 900**

535i 6-man R6 / 2 979 225 (306) 11,1 / 6,3 / 8,1 188 194 634 **239 400**

produktywność



smrodki


using System.Collections.Generic;
using System.Diagnostics;
namespace Dequeue
{
internal class TxtFileWriter : IFileWriter
public void Save(IEnumerable<string> data)
Debug.WriteLine("plik txt");
string fileType)
przodu_jest_odpowiednio_ustawiona()

źródło smrodu

```
var obj = new SomeClass();
```

panaceum

Fabryka

Rozwiązuje problem tworzenia -> zadanie fabryki to stworzenie obiektu i nic ponad to,

Single Responsibility Principle

jesteśmy jedno zadaniowi

Ludzie są tak skonstruowani że najlepiej robią jedną rzecz w danej chwili

Procesory robią jedną rzecz w jednej chwili
 1 core 1 akumulator 1 obliczenie w danej chwili

źródło smrodu

```
if( val<0 && val < 1000 | (person==„kierownik” &&
val>0 && val <= 5000)
```

podobne smrodki: **for**, **foreach**, **while** z operacjami **break** lub **continue**

panaceum 1

```
if(ValueCanBeAcceptedAutomaticly){
.....
```

```
bool ValueCanBeAcceptedAutomaticly(){

return ValueInRangeForEmployee ||
ValueInRangeForManager;
}
```

```
bool ValueInRangeForManager(){
reutrn person==„kierownik” &&  val>0 &&
val <= 5000;
}
```

demo

refaktoryzacja if-a

panaceum 2

zamiast

```
if(saveToXml){

//zapisz do xml

} else {

// zapisz do txt

}
```

```
var fileSaver= Factory.Get(fileFormat);

fileSaver.Save(data);
```

Fabryka zwraca odpowiedni obiekt dla zadanego formatu pliku.

demo

fabryka zamiast if

metoda pomiaru : ZŁOŻONOŚĆ CYLKOMATYCZNA

Idealna złożoność to złożoność = 1

„Przytaczane są poniższe wartości złożoności cyklomatycznej:
od 1 do 10 – kod dość prosty stwarzający nieznaczne ryzyko
od 11 do 20 – kod złożony powodujący ryzyko na średnim poziomie
od 21 do 50 – kod bardzo złożony związany z wysokim ryzykiem
powyżej 50 – kod niestabilny grożący bardzo wysokim poziomem ryzyka.”

Źródło: wikipedia (http://pl.wikipedia.org/wiki/Z%C5%82o%C5%BCono%C5%9B%C4%87_cyklomatyczna)

źródło smrodu

```
var a = 100;
var b = 100;
a = Add(a, ref b);

return a + b;
```

źródło smrodu

```
var a = 100;
var b = 100;
a = Add(a, ref b);

return a + b;
```

```
function int Add(int a, ref int b){
    b = a + b;
    return b;
}
```

demo

out i ref

źródło smrodu

Długie metody, bardzo długie metody
które mają dużo kodu,
Bardzo dużo kodu,
Nieczytelnego
Ciągnącego się kodu
Z wieloma linijkami
Funkcjami
Metodami
Zmiennymi
Przełącznikami
I innymi długimi niepotrzebnie zaciemniającymi

Arfetaktami rzekomo pokazującymi biegłość i przebiegłość żegomego super programisty
Który myśli że jak wrzuci wyszstko do jednego wielkiego pliku
A najlepiej do jednej długiej linii to jest mistrzem świata.
Wszak było trudno napisać to niech będzie trudno zrozumieć

SINGLE RESPONSIBILITY

PRINCIPLE

źródło smrodu

Kopiowanie kodu

Klasa A zależy od Klasy B, Klasa B zależy od klasy A

nie można użyć klasy A w innym projekcie (związane przez operator new) zatem kopiujemy kod

To powoduje że błędy są powielane, błąd rozwiązany w klasie A w jednym projekcie
nie oznacza rozwiązania go w projekcie B

panaceum

DEPENDENCY INVERSION PRINCIPLE

Klasy zależą od abstrakcji,

Od interfejsów i ewentualnie klas abstrakcyjnych

źródło smrodu

Nieprzewidywalny kod

Niepotrzebne, źle użyte dziedziczenie robi więcej złego niż dobrego.

Jeśli klasa nie robi tego czego się spodziewamy po niej to ….

```
if( obj is XmlWriter){

….
}
else if(obj is TxtWriter){
…
```

demo

kwadrat i prostokąt

panaceum

LISKOV SUBSTITUTION PRINCIPLE

Obiekty w programie powinny być wymienialne na instancje ich podtypów bez zmiany poprawności działania programu

S ingle responsibility principle

O pen/Close principle

L iskov substituton principle

I nterface segregation principle

D ependancy inversion principle

SOLID nie = śmierdzący kod

SOLID = nie śmierdzący kod

panaceum

Foto:Henryka Pakuła

droga na skróty

Test Driven Development

dziękuję

więcej na http://www.benedykt.net