

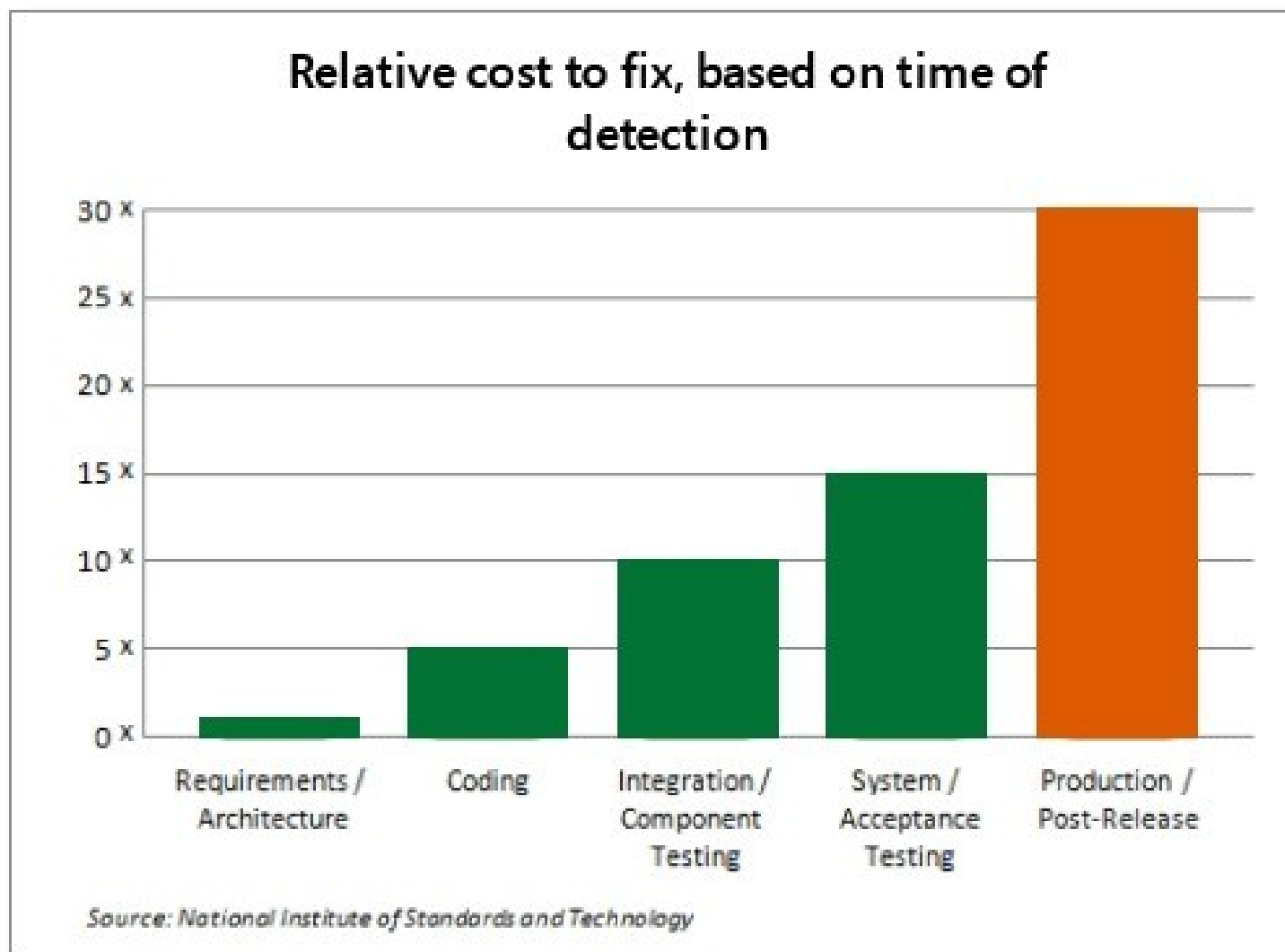
Błędy

jak pisać bezbłędne programy

Są dwa sposoby pisania bezbłędnego oprogramowania, ale niestety działa tylko ten trzeci.

Alan J. Perlis

Dlaczego chcemy uniknąć błędów?



Główne kategorie błędów

- **Składniowe** – są to najczęściej zwykłe literówki, z których większość jest wykrywana automatycznie w fazie kompilacji
- **Semantyczne** – związane ze sposobem działania programu (np. błędna inicjalizacja licznika pętli bądź niewłaściwa ilość jej powtórzeń)
- **Logiczne** – wynikające z błędnych założeń programisty odnośnie np. faktycznych potrzeb użytkownika, kontekstu działania programu, formatu danych źródłowych itd.

Klasyfikacja błędów

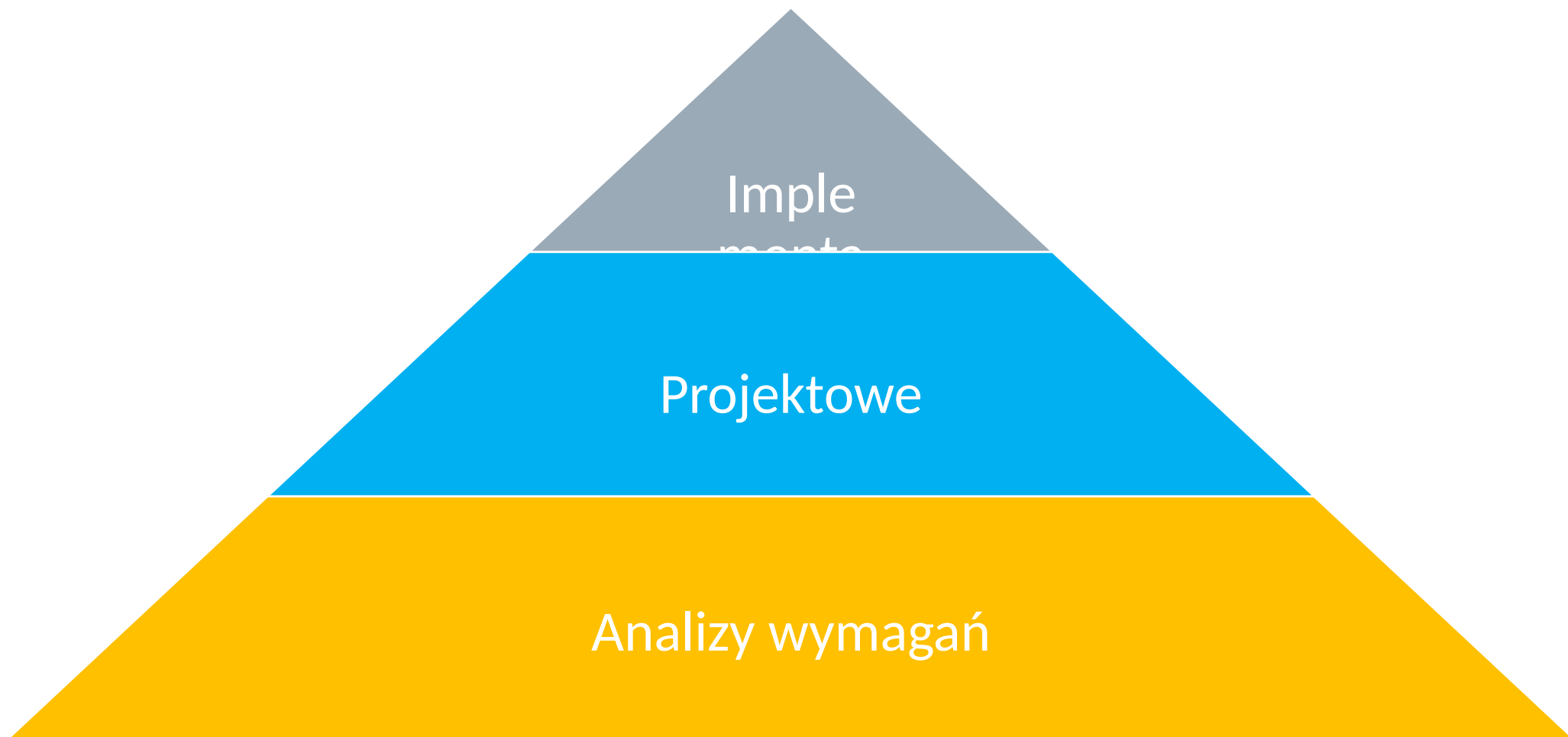
- **Funkcjonalny** – wskazuje na funkcjonalność, która jest niezgodna z ustalonymi wymaganiami
- **Systemowy** – wskazuje na nieprawidłowe zarządzanie zasobami, ich brak bądź zajętość
- **Przetwarzania** – dotyczy niewłaściwego przetwarzania danych
- **Użyteczności** – wskazuje na trudności z użytkowaniem systemu, problemy z ergonomicznością
- **Bezpieczeństwa** – wskazuje na naruszenie bezpieczeństwa systemu
- **Graficzny** – dotyczy interfejsu użytkownika
- **Kodowania** – niewłaściwe użycie języka programowania, błędy definicji zmiennych

Scenariusz wystąpienia błędu w programie



π

Rodzaje błędów



Wyszukiwanie błędów

- › Podpowiedzi IDE
- › Informacje kompilatora
 - Błąd składni
- › Informacje linkera
 - Brak biblioteki
- › Błędy podczas wykonywania programu
 - Błąd krytyczny
 - Błąd logiczny
- › Błędy użytkownika
 - Liczba spoza zakresu

Przykłady: błąd składni



```
while True print('Hello world')
```



File "[<ipython-input-3-2b688bc740d7>](#)", line 1

```
while True print('Hello world')
```

^

SyntaxError: invalid syntax

SEARCH STACK OVERFLOW

Przykłady: wyjątek – dzielenie przez 0



```
10 * (1/0)
```



```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-4-0b280f36835c> in <module>()  
----> 1 10 * (1/0)
```

```
ZeroDivisionError: division by zero
```

SEARCH STACK OVERFLOW

Przykłady: błąd nazwy zmiennej



```
4 + spam*3
```



```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-5-c98bb92cdcac> in <module>()  
----> 1 4 + spam*3
```

```
NameError: name 'spam' is not defined
```

SEARCH STACK OVERFLOW

Przykłady: niezgodne typy danych



'2' + 2



TypeError

Traceback (most recent call last)

[<ipython-input-6-d2b23a1db757>](#) in <module>()
-----> 1 '2' + 2

TypeError: must be str, not int

SEARCH STACK OVERFLOW

Przykłady: błąd składni – brak wcięcia



```
auta = ['polonez', 'syrena', 'warszawa', 'fiat']
```

```
for auto in auta:  
print(auto)
```



File ["<ipython-input-7-6d7dea596ffd>"](#), line 4

```
    print(auto)  
      ^
```

IndentationError: expected an indented block

SEARCH STACK OVERFLOW

Przykłady: błąd składni – błędne wcięcie



```
auta = ['polonez', 'syrena', 'warszawa', 'fiat']
```

```
for auto in auta:  
    print(auto)  
    print(len(auto))
```



File "<ipython-input-8-7f75424f4b6f>", line 5

```
    print(len(auto))  
    ^
```

IndentationError: unexpected indent

SEARCH STACK OVERFLOW

Zapobieganie błędom

- Narzędzia
 - IDE, kompilatory
- Analiza statyczna kodu
 - Analiza leksykograficzna
 - Metody formalne
 - › Modele matematyczne
 - › Reguły logiczne
 - › Analiza przepływu
 - Metryki kodu źródłowego
- Testowanie
 - testy jednostkowe, testy automatyczne
- Metodyki wytwarzania
 - Programowanie w parach
 - Code review
 - Refaktoring

Przykłady: brak nawiasu otwierającego {

The screenshot shows the Dev-C++ 5.11 IDE. The main window displays a C++ file named `main.cpp` with the following code:

```
1  #include <iostream>
2
3  int main(int argc, char** argv)
4  {
5      return 0;
6  }
```

The line `return 0;` is highlighted in red, indicating an error. The error message is: `[Error] expected identifier before numeric constant`.

The bottom panel shows the Compiler (5) window with the following errors:

Line	Col	File	Message
4	9	C:\Temp\pr1\main.cpp	[Error] expected identifier before numeric constant
4	2	C:\Temp\pr1\main.cpp	[Error] named return values are no longer supported
5	1	C:\Temp\pr1\main.cpp	[Error] expected '{' at end of input
28		C:\Temp\pr1\Makefile.win	recipe for target 'main.o' failed

The status bar at the bottom indicates: Line: 4, Col: 12, Sel: 0, Lines: 5, Length: 72, Insert, Done parsing in 0,359 seconds.

Kompilatory

Kompilatory oprócz sprawdzania składni również potrafią sygnalizować możliwość występowania błędu w kodzie

- -pedantic
tylko czyste ISO C i ISO C++
- -Wall
wszystkie ostrzeżenia
- -Werror
ostrzeżenia generują błąd kompilacji
- -Wextra
dodatkowe ostrzeżenia wykraczające poza Wall

Kompilatory – wyszukiwanie błędów

```
int main()
{
    char *x;
    int a[20];

    *x = 5;
    a[20] = 5;

    return 0;
}
```

```
cppcheck --enable=all test1.c
Checking test1.c...
[test1.c:4]: (style) Variable 'x' is not assigned a value.
[test1.c:8]: (style) Variable 'a' is assigned a value that is never used.
[test1.c:8]: (error) Array 'a[20]' accessed at index 20, which is out of bounds
[test1.c:7]: (error) Uninitialized variable: x
Checking usage of global functions..
```

Analiza statyczna

- Szybkość działania - szybkie wykrywanie błędów, których naprawa jest prosta i mało kosztowna
 - nie wymagają uruchamiania programu
 - łatwe do zrównoleglenia
- Łatwość użycia - proste we wdrożeniu w cykl wytwarzania oprogramowania
- Automatyzacja - integracja z narzędziami continuous integrations
- Możliwości rozszerzania: własne reguły, wtyczki, ...
- Integracja z innymi narzędziami: serwery automatyzacji, IDE, kontrola wersji

Wady analizy statycznej

- Wymagany dostęp do źródeł
- Reguły wykrywają zazwyczaj proste błędy i nie są w stanie wyeliminować ręcznego sprawdzenia kodu
- Dużo szumu, zbyt czułe - duże prawdopodobieństwo zaklasyfikowanie poprawnego fragmentu jako błędu (false positive)
- Każde narzędzie zazwyczaj pokrywa pewien zakres testów (do 14% błędów?!). Dlatego warto korzystać z kilku różnych skanerów kodu. Istnieją narzędzia integrujące wiele narzędzi SAST, np: CodeEx, Yast

Co możemy analizować

- analiza poprawności składni
- luki w bezpieczeństwie, także błędy które mogą pojawić się przy specyficznych danych wejściowych
- detekcja backdoors, niebezpieczne i nieaktualne funkcje, wycieki pamięci, przepełnienie bufora, używanie niezainicjowanych zmiennych, SQL Injections,
- jakość kodu, ocena stylu, powtórzenia kodu, nieużywane fragmenty kodu, ...
- wydajność, wykrywanie wąskich gardeł, niewydajne konstrukcje, sugestie dotyczące poprawienia wydajności
- zgodność z dobrymi praktykami, zachowanie standardów, norm nazewnicznych, problemy z przenośnością kodu

Co nam daje analiza kodu

- zwiększenie wydajności i stabilności poprzez zasady oparte na dobrych praktykach
- uniknięcie typowych błędów podczas programowania
- dostarczenie struktury do zarządzania standardami kodu
- wymuszenie zasad i standardów pisanie kodu
- zwiększanie bezpieczeństwa poprzez kolejny etap testowania
- analizując sygnalizowane błędy można się sporo nauczyć na temat dobrych praktyk bezpiecznego programowania

Testowanie

- kiedy analizować: przed/po commicie, po każdym buildzie, po zapisaniu pliku
- IDE i ich edytory sprawdzają składnię też (rozszerzenia ReSharper)
- testy po stronie developera lub po stronie repozytorium (serwera automatyzacji)
- skrypty uruchamiane po zatwierdzeniu zmiany + powiadomienia (email, inne alerty), gitlab, github
- uniemożliwienie zatwierdzenia kodu z błędami – gitlab, github
- serwery automatyzacji, np.: Jenkins, TemCity

Testy jednostkowe

- Test jednostkowy to nic innego jak kod wykonujący inny kod w kontrolowanych warunkach. Jego zadaniem jest weryfikacja (bez ingerencji programisty), że testowany kod działa poprawnie. Robi to w sposób dość banalny: autor testu dostarcza dane wejściowe (input), test wykonuje pewne instrukcje i sprawdza, czy rezultat działań (output) zgodny jest z oczekiwaniami.
- W świecie idealnym każdy test bada jedną ścieżkę wykonania jednej metody.

```
....  
-----  
Ran 4 tests in 0.004s  
  
OK  
<unittest.runner.TextTestResult run=4 errors=0 failures=0>
```


Testy jednostkowe

- › Poprawa architektury rozwiązania
- › Weryfikacja działania bez uruchamiania
- › Testy jako dokumentacja
- › Testy jako zabezpieczenie przed regression bugs
- › Testy jako narzędzie zabezpieczenia przed błędnym merge request
- › Testy jako narzędzie do nauki?

Refaktoring

- **usprawnienie projektu** – celem refaktoryzacji jest ochrona struktury oprogramowania przed rozkładem
- **przejrzystość oprogramowania** - każda osoba może zrozumieć założenie i proces myślowy autora
- **szybsze wyszukiwanie błędów** - podczas refaktoryzacji można lepiej zrozumieć program i jego działanie a przez to można szybciej znaleźć błędy i usprawnić go
- **szybsze programowanie** - do spowalniania pracy przyczynia się nieodpowiedni projekt – kod po refaktoringu łatwiej zrozumieć
- **eliminacja długu technologicznego** – wymiana starych bibliotek na nowsze, uproszczenie kodu

Trudne błędy

- Jest wiele typów błędów, które ciężko jest wykryć, np.: błędy związane z konfiguracją lub logiką biznesową
- Problemem są zależności, dodatkowe moduły, biblioteki, konteksty dodawane przez frameworki
- Skomplikowane algorytmy i architektury bardzo ciężko jest analizować



```
godziny = 144
```

```
# oblicz liczbę dni  
dni = godziny / 23  
print(dni)
```



```
6.260869565217392
```

Debugger

- Ustawianie punktu zatrzymania
- Praca krokowa
- Podgląd stanu programu
 - Podglądu bieżącej zawartości zmiennych
 - Podglądu bieżącej zawartości rejestrów procesora i obszaru pamięci pod wybranym adresem
 - Podglądu stosu wywołań funkcji i informacji o uruchomionych wątkach
- Możliwość zmiany stanu programu

π

Debugger C++ Code::Blocks

The screenshot displays the Code::Blocks IDE interface with the following components:

- Memory Panel (1):** Shows memory addresses and their corresponding values. Address 0x4009f4 contains the string "Hello world!....".
- Disassembly Panel (3):** Shows the assembly code for the current function. The function is `main()` and the assembly code includes instructions like `push rbp`, `mov rbp, rsp`, and `mov eax, edx`.
- Watches Panel (4):** Shows the values of variables being watched. The variables are `a` (value 1), `b` (value 1), `ret` (value 2), and `ret` (value 2).
- Program Console (6):** Shows the output of the program. The output is "Hello world!".
- Logs & others Panel (5):** Shows the logs of the debugger. The logs include the message "Setting breakpoints" and "Debugger name and version: GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1".

The main window shows the source code of `main.cpp` with a breakpoint at line 12. The code is as follows:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int dodaj(int a, int b) {
6     int ret = a + b;
7     return ret;
8 }
9
10 int main()
11 {
12     cout << "Hello world!" << endl;
13     for (int i = 0; i < 10; i++) {
14         int podwojny = dodaj(i, i);
15         cout << podwojny << endl;
16     }
17     return 0;
18 }
```

Colab – debugger



```
%debug
import pdb
from pdb import set_trace as bp

wymiar = [400, 100, 250, 300, 123]

wymiarWMetrach = [x / 100 for x in wymiar]
print("\nWymiar w m:")
for wymiar in wymiarWMetrach:
    bp()
    print(wymiar)
```

... ERROR:root:No traceback has been produced, nothing to debug.

```
Wymiar w m:
> <ipython-input-1-9ac7ddd044dc>(11)<module>()
-> print(wymiar)
(Pdb) c
4.0
> <ipython-input-1-9ac7ddd044dc>(10)<module>()
-> bp()
(Pdb) display wymiar
display wymiar: 1.0
(Pdb)
```

Obsługa błędów

```
[ ] while True:  
    try:  
        x = int(input("Proszę wprowadzić cyfrę: "))  
        break  
    except ValueError:  
        print("Oops! To nie jest numer. Spróbuj ponownie...")
```

Obsługa błędów – Python

```
try:
    with open(filename) as f_obj:
        contents = f_obj.read()
except FileNotFoundError as e:
    msg = "Plik " + filename + " nie istnieje pobieram z Internetu."
    pobierzPlik("https://github.com/rroszczyk/Python/raw/master/"+filename, filename)
    print(msg)
    try:
        with open(filename) as f_obj:
            contents = f_obj.read()
    except:
        print("Jest problem z plikiem")
        contents = None
else:
    print("Używam wersji pliku z lokalnego repozytorium")
```


Obsługa błędów – Python



```
import sys

try:
    f = open('plik.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("Błąd systemu operacyjnego: {0}".format(err))
except ValueError:
    print("Błąd konwersji danych do liczby.")
except:
    print("Niespodziewany błąd:", sys.exc_info()[0])
    raise
```

Obsługa błędów – Python

```
[ ] def count_words(filename):  
    """Obliczenie przybliżonej liczby słów w danym pliku."""  
    try:  
        with open(filename) as f_obj:  
            contents = f_obj.read()  
    except FileNotFoundError:  
        pass  
    else:  
        # Obliczenie przybliżonej liczby słów w danym pliku.  
        words = contents.split()  
        num_words = len(words)  
        print("Plik " + filename + " zawiera " + str(num_words) + " słów.")
```

Obsługa błędów – Python

```
▶ def divide(x, y):  
    try:  
        result = x / y  
    except ZeroDivisionError:  
        print("dzielenie przez zero!")  
    else:  
        print("wynikiem operacji jest", result)  
    finally:  
        print("kończymy działanie\n")  
  
divide(10, 5)  
divide(10, 0)
```

☞ wynikiem operacji jest 2.0
kończymy działanie

dzielenie przez zero!
kończymy działanie

Obsługa błędów – C++

```
try {  
    f();  
} catch (const std::overflow_error& e) {  
    // this executes if f() throws std::overflow_error (same type rule)  
} catch (const std::runtime_error& e) {  
    // this executes if f() throws std::underflow_error (base class rule)  
} catch (const std::exception& e) {  
    // this executes if f() throws std::logic_error (base class rule)  
} catch (...) {  
    // this executes if f() throws std::string or int or any other unrelated type  
}
```

```
try {  
    f();  
} catch (const std::exception& e) {  
    // will be executed if f() throws std::runtime_error  
} catch (const std::runtime_error& e) {  
    // dead code!  
}
```

π

Dekompilacja

JetBrains dotPeek

File View Navigate Inspect Tools Windows Help

Assembly Explorer

- PDFCombiner (1.8.0.0, x86, .Net Framework v4.0 C)
- Metadata
- References
- Resources
- PDFCombiner
 - App
 - Program
 - Base types
 - InitializeComponent():void
 - Main():void
 - OnResolveAssembly(object sender, ResolveEventArgs e):void
 - embeddedDlls:Dictionary<string, string>
 - PDFCombiner.Common
 - PDFCombiner.Decryptor
 - PDFCombiner.Dialogs
 - PDFCombiner.Locators
 - ViewModelLocator
 - Base types
 - AboutWindowViewModel:AboutWindowViewModel
 - PayPalDonateViewModel:PayPalDonateViewModel
 - PdfFileViewModel:PdfFileViewModel
 - ProgressReporterViewModel:ProgressReporterViewModel
 - payPalDonateViewModel:PayPalDonateViewModel
 - progressReporterViewModel:ProgressReporterViewModel
 - PDFCombiner.Models
 - PDFCombiner.Properties
 - PDFCombiner.Resources
 - PDFCombiner.ViewModels
 - PDFCombiner.Views

Type to search

PDFCombiner.cs Program.cs ViewModelLocator.cs

```
// Decompiled with JetBrains decompiler
// Type: PDFCombiner.Locators.ViewModelLocator
// Assembly: PDFCombiner, Version=1.8.0.0, Culture=neutral, PublicKeyToken=null
// MVID: C6E0BCB9-E403-4317-B38D-8D9E35F72148
// Assembly location: C:\Users\radek\AppData\Roaming\PDF Combiner\PDFCombiner.exe

using PDFCombiner.Common;
using PDFCombiner.ViewModels;

namespace PDFCombiner.Locators
{
    public class ViewModelLocator
    {
        private static ProgressReporterViewModel progressReporterViewModel;
        private static PayPalDonateViewModel paypalDonateViewModel;

        public PdfFileViewModel PdfFileViewModel
        {
            get
            {
                return new PdfFileViewModel((IProgressReporter) ViewModelLocator.ProgressReporterViewModel);
            }
        }

        public AboutWindowViewModel AboutWindowViewModel
        {
            get
            {
                return new AboutWindowViewModel();
            }
        }

        public static ProgressReporterViewModel ProgressReporterViewModel
        {
            get
            {
                return ViewModelLocator.progressReporterViewModel ?? (ViewModelLocator.progressReporterViewModel = new ProgressReporterViewModel());
            }
        }

        public static PayPalDonateViewModel PayPalDonateViewModel
        {
            get
            {
                return ViewModelLocator.paypalDonateViewModel ?? (ViewModelLocator.paypalDonateViewModel = new PayPalDonateViewModel());
            }
        }
    }
}
```

IL Viewer

```
// end of method ViewModelLocator::get_ProgressReporterViewModel

.method public hidebysig static specialname class PDFCombiner.ViewModels.PayPalDonateViewModel
get_PayPalDonateViewModel() cil managed
{
    .maxstack 8

    // [45 9 - 45 129]
    IL_0000: ldsfld      class PDFCombiner.ViewModels.PayPalDonateViewModel PDFCombiner.Locators.ViewModelLocator::paypalDonateViewModel
    IL_0005: dup

    IL_0006: brtrue.s      IL_0014
    IL_0008: pop
    IL_0009: newobj        instance void PDFCombiner.ViewModels.PayPalDonateViewModel::.ctor()
    IL_000e: dup
    IL_000f: stsfld      class PDFCombiner.ViewModels.PayPalDonateViewModel PDFCombiner.Locators.ViewModelLocator::paypalDonateViewModel
    IL_0014: ret

} // end of method ViewModelLocator::get_PayPalDonateViewModel

.method public hidebysig specialname rtspecialname instance void
ctor() cil managed
{
    .maxstack 8

    IL_0000: ldarg.0      // this
    IL_0001: call        instance void [mscorlib]System.Object::.ctor()
    IL_0006: ret

} // end of method ViewModelLocator::.ctor

.property instance class PDFCombiner.ViewModels.PdfFileViewModel PdfFileViewModel()
{
    .get instance class PDFCombiner.ViewModels.PdfFileViewModel PDFCombiner.Locators.ViewModelLocator::get_PdfFileViewModel
} // end of property ViewModelLocator::PdfFileViewModel

.property instance class PDFCombiner.ViewModels.AboutWindowViewModel AboutWindowViewModel()
{
    .get instance class PDFCombiner.ViewModels.AboutWindowViewModel PDFCombiner.Locators.ViewModelLocator::get_AboutWindowViewModel
} // end of property ViewModelLocator::AboutWindowViewModel

.property class PDFCombiner.ViewModels.ProgressReporterViewModel ProgressReporterViewModel()
{
    .get class PDFCombiner.ViewModels.ProgressReporterViewModel PDFCombiner.Locators.ViewModelLocator::get_ProgressReporterViewModel
} // end of property ViewModelLocator::ProgressReporterViewModel

.property class PDFCombiner.ViewModels.PayPalDonateViewModel PayPalDonateViewModel()
{
    .get class PDFCombiner.ViewModels.PayPalDonateViewModel PDFCombiner.Locators.ViewModelLocator::get_PayPalDonateViewModel
} // end of property ViewModelLocator::PayPalDonateViewModel

} // end of class PDFCombiner.Locators.ViewModelLocator
```

mgr inż. Radosław Roszczyk

Jednostka macierzysta: **Wydział Elektryczny**
Instytut Elektrotechniki Teoretycznej i Systemów Informacyjno-Pomiarowych
Zakład Elektrotechniki Teoretycznej i Informatyki Stosowanej

Stanowisko: **Asystent**



Informacje kontaktowe:

E-mail: **radoslaw.roszczyk@ee.pw.edu.pl**
Telefon: **22 234 75 43 (w godzinach konsultacji)**
Pokój **GE224**
Strona www: <https://www.roszczyk.net>
Adres
korespondencyjny: **IETiSIP - Radosław Roszczyk**
ul. Koszykowa 75 / GE 216
00-662 Warszawa

Nota biograficzna, profil działalności (jęz. polski)

Analiza i przetwarzanie obrazów biomedycznych. Obliczenia równoległe i wielkoskalowe. Metodyki wytwarzania oprogramowania. Języki i metody programowania.

PYTANIA ?



<https://github.com/rroszczyk/python>