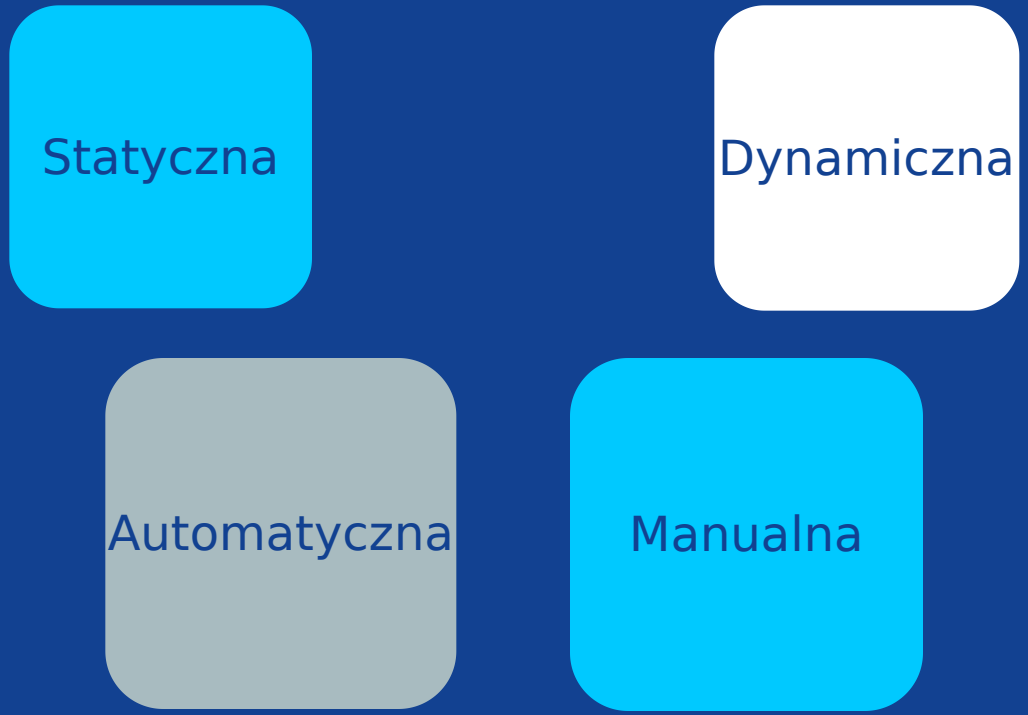
PARO - Środowisko Programisty

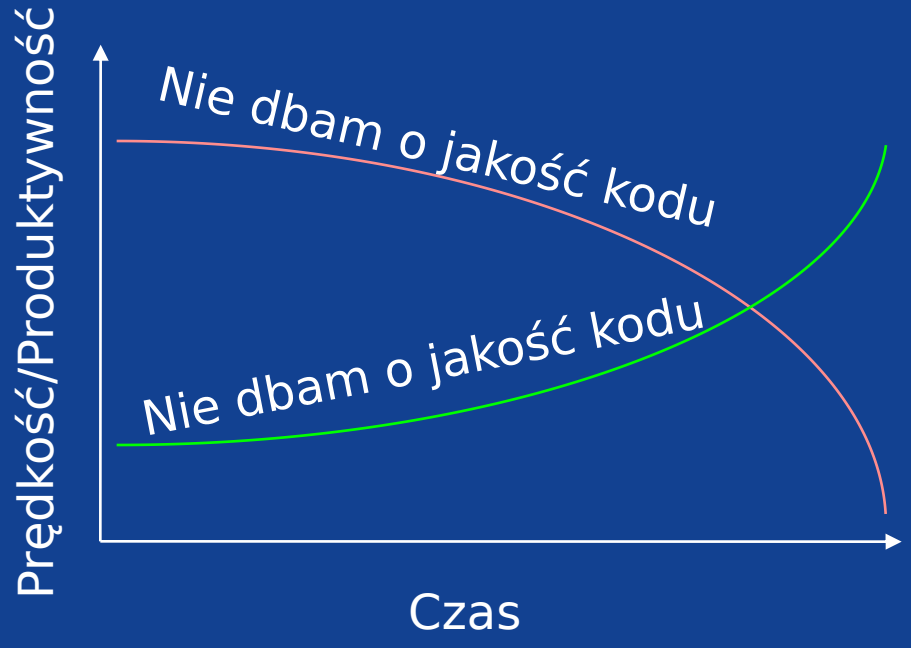
Narzędzia potrzebne do wytwarzania dobrej jakości oprogramowania

Marcin Zawadzki

Rodzaje analizy kodu



Dlaczego?



Kiedy?
Natychmiast!

Statyczna

Bez uruchamiania programu

Jak?

Statyczna analiza - niech piękno będzie wokół nas

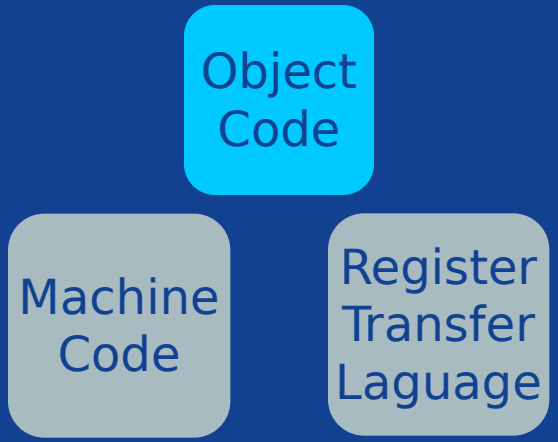
Style Checkers

IDE
Check while you type

CI/Delivery
C++ - uncrustify
JS - JSCS

Kompilator

Program komputerowy (lub zestaw programów) który przekształca kod źródłowy (źródłowy język programowania), w kod wynikowy (wynikowy język programowania).



Zadania kompilatora

- Preprocesing
- Parsowanie
- Analiza semantyczna
- OPTYMALIZACJA KODU
- Generacja kodu

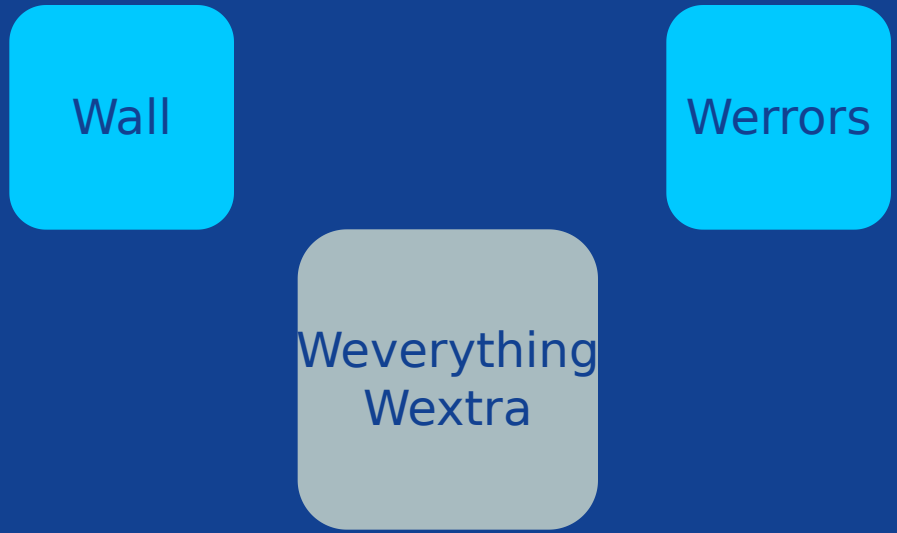
Kompilacja

```
clang++ -c  -g3  -O0 -Wall -frtti -fexceptions -IIncludeDir
-ILinkLibrary -LlinkLibraryDir -o object.o sourceFile.cpp

clang++ main.cpp -o binaryName

clang++ main.cpp
```

Kompilator - Obrońca - C++



Clang kontra gcc

gcc

- Wsparcie dla wielu architektur

Clang

- Czytelne błędy kompilacji
- Integracja z IDE (libclang)
- dodatkowe narzędzia

Statyczna analiza kodu

Służy do wykrywania błędów normalnie nie wykrywalnych przez kompilator.

C++

include-what-you-use
Klocwork
Coverity
clang-tidy

scan build/scan view
clang-tidy
Ćwiczenie

clang-modernize
clang-format

Kiedy?
To zależy :)

Dynamiczna

Podczas wykonywania programu

Dynamiczna analiza - zatem kiedy?

Ciągle w CI!

Przed dostarczeniem!

Gdy aplikacja jest zbyt wolna

Gdy pojawią się błędy!

Sprawdź, czy testujesz swój kod

- Function
- Statement
- Branch
- Condition

```
int foo (int x, int y)
{
    int z = 0;
    if ((x>0) && (y>0))
    {
        z = x;
    }
    return z;
}
```

gcov/bcov/kcov
Ćwiczenie

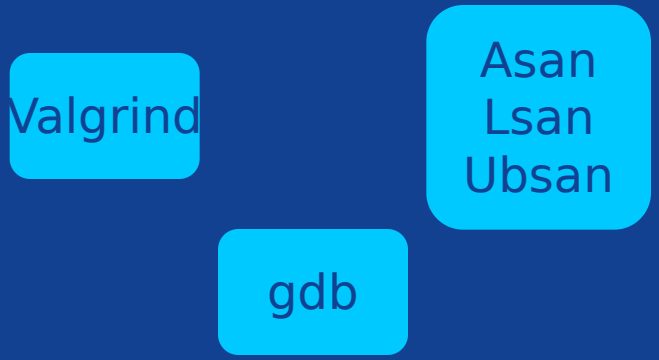
Code coverage - czy to wystarczy?

1. Czy 100% pokrycia kodu mówi że kod działa poprawnia?
2. Czy mogę zwiększyć pokrycie kodu nie pisząc prawdziwych testów?

Testy mutacyjne - bo pokrycie kodu to nie wszystko

Polegają na automatycznym wprowadzaniu błędów do kodu źródłowego i sprawdzaniu, czy testy wykryły wprowadzony błąd.

Mój kod nie działa



Have no fear cause gdb is here

Stan programu

1. Działa i wiem dlaczego
2. Nie działa i wiem dlaczego
3. **Nie działa i nie wiem dlaczego**
4. **Działa i nie wiem dlaczego**

gdb ./myApp
Ćwiczenie

gdb - watchpoints

Przydatne gdy chcemy zatrzymać wykonywanie programu jeśli wynik pewnego wyrażenia zmieni swoją wartość. Przy ustawianiu watchpoint'a nie podajemy dokładnego miejsca w kodzie w którym pułapka ma zostać ustawiona. Wyrażeniem może być zarówno pojedyńcza zmienna lub wiele zmiennych połączonych operatorami. Ustawianie watchpoint'ów mocno spowalnia wykonywanie programu.

gdb - watchpoints

- watch [-l|-location] expr [thread threadnum] [mask maskvalue]
- rwatch [-l|-location] expr [thread threadnum] [mask maskvalue]
- awatch [-l|-location] expr [thread threadnum] [mask maskvalue]
- info watchpoints

Ćwiczenie

gdb - core dump

Gdb umożliwia odtworzenie stanu aplikacji przy posiadaniu zrzutu aplikacji w czasie np. wystąpienie wyjątku.

Ćwiczenie

Gdb – Recording/Reverse debugging

Od wersji 7.0 gdb wspiera “rejestrowanie” i wsteczne debug'owanie programu. aby można było używać opcji wstecznego debug'owania należy wydać polecenie record. Do zatrzymania “rejestrowania” służy polecenie record stop.

Valgrind

Valgrind jest framework'iem służącym do budowania narzędzi do dynamicznej analizy kodu

- **memory error detector**
- thread error detector - Helgrind
- **call graph profiler**
- heap profiler - Massif

Spowalnia wykonanie programu

Ćwiczenie

Sanitizer'y jeżeli Valgrind jest dla Ciebie zbyt wolny :(

- Address sanitizer
- Leak sanitizer
- Undefined behaviour sanitizer

Ćwiczenie

Code profiling

Pomaga zadecydować czy już czas na refaktoring. I jeżeli tak to jakie miejsce w kodzie wymaga refaktoringu.

Text vs Call Graph vs Flame Graph