

Jak zadbać o jakość w projekcie i nie zwariować?

Clang-Format, Clang-Tidy,
Cppcheck, CMake i dobre praktyki

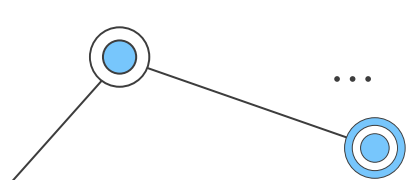
Mateusz Patyk

Gdzie mnie znaleźć

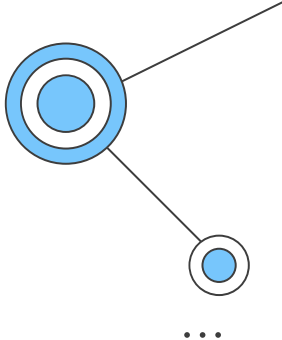


 [linkedin.com/in/mateusz-patyk1/](https://www.linkedin.com/in/mateusz-patyk1/)

 github.com/the-risk-taker



?

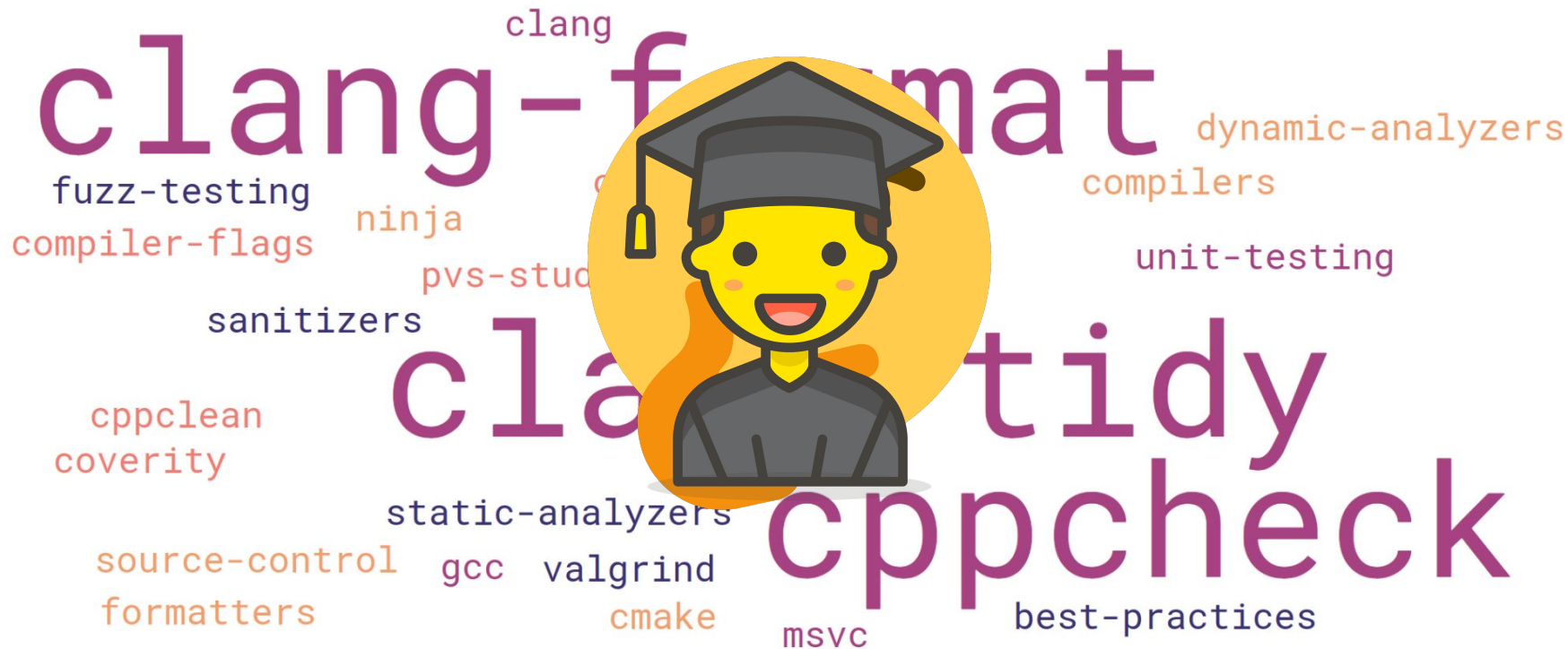
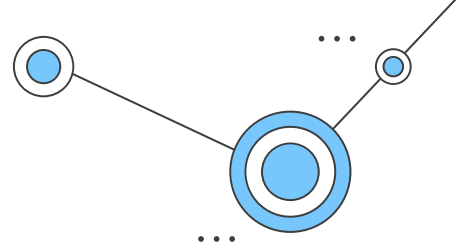


```
struct basic_pipebuf : std::basic_streambuf<CharT, Traits>S
{
    ...
    basic_pipebuf();
    basic_pipebuf(const basic_pipebuf & ) = default;
    basic_pipebuf(basic_pipebuf && ) = default;

    ~basic_pipebuf()
    try
    {
        if (basic_pipebuf::is_open())
            basic_pipebuf::overflow(Traits::eof());
    }
    catch (process_error & )
    {
    }

    basic_pipebuf(pipe_type && p);
    ...
};
```

Wstęp

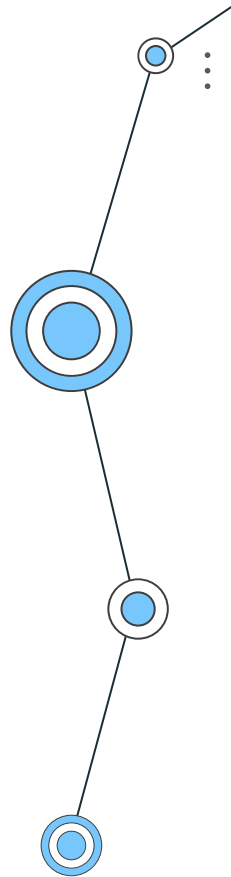
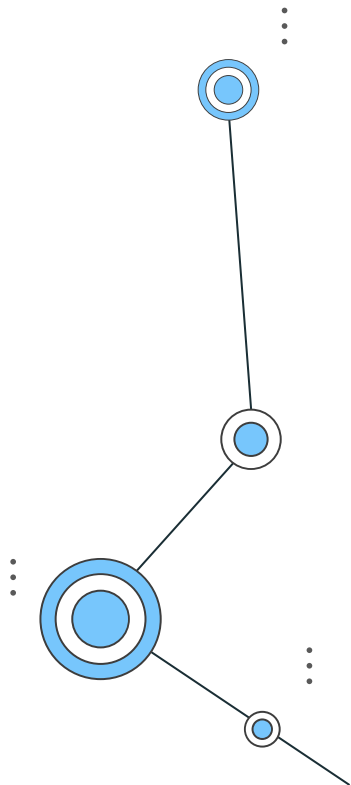


Moje **trzy** zasady stosowania narzędzi

I

łatwość użycia

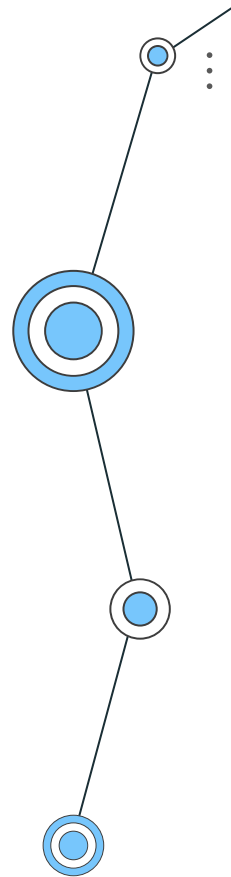
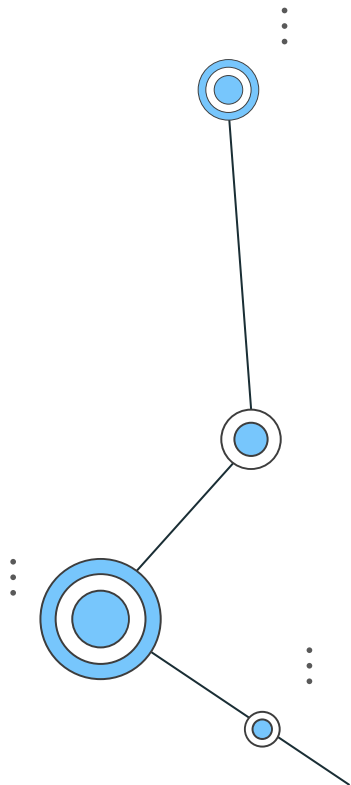
```
$ ninja -C build cppcheck-check
```



Moje **trzy** zasady stosowania narzędzi

II

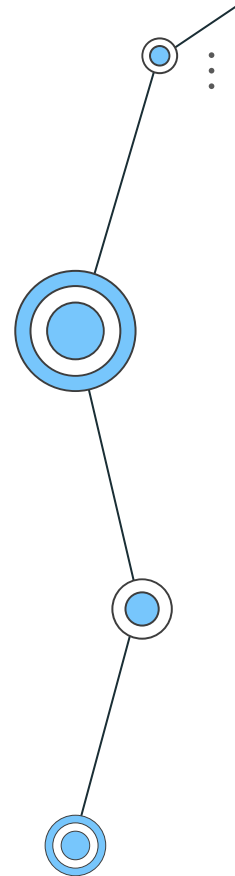
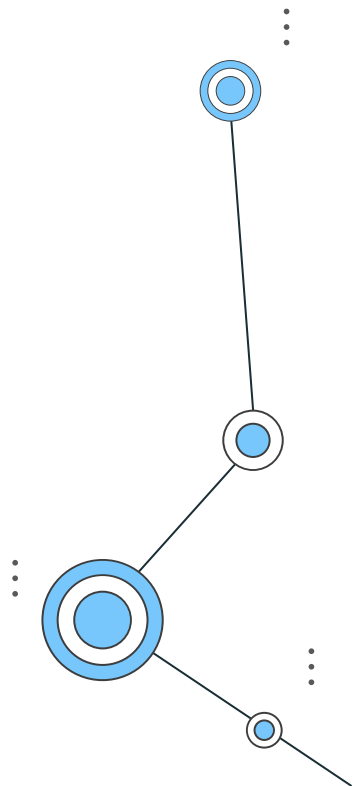
żadnych półśrodków

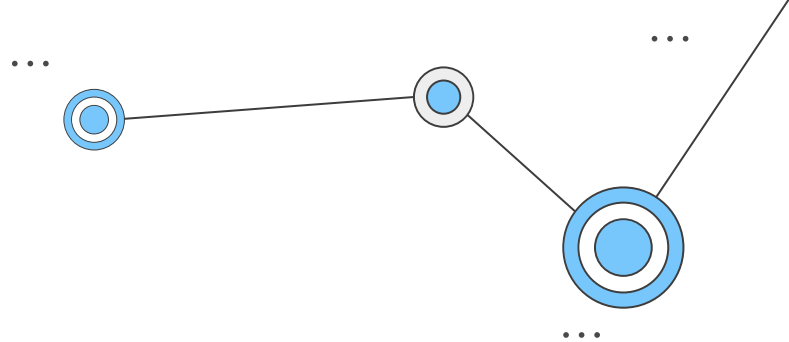


Moje **trzy** zasady stosowania narzędzi

III

poprawność wymuszona
przez pipeline CI/CD





Clang-Format



Clang-Format podstawy



Clang-Format to narzędzie do automatycznego formatowania kodu źródłowego dla C, C++, Java i wielu innych. Clang-Format jest częścią projektu LLVM.

- formatowanie plików:

```
$ clang-format -i [...files]
```

- formatowanie z użyciem predefiniowanych stylów:

```
$ clang-format -i -style=llvm [...files]
```

- sprawdzenie poprawności formatowania:

```
$ clang-format --dry-run --Werror [...files]
```



Clang-Format własny styl



Sposób formatowania przez Clang-Format można dowolnie skonfigurować. Domyślnie plik z konfiguracją ma nazwę **.clang-format**. Wszystkie opcje: [Clang-Format Style Options](#).

- drukowanie wszystkich opcji:

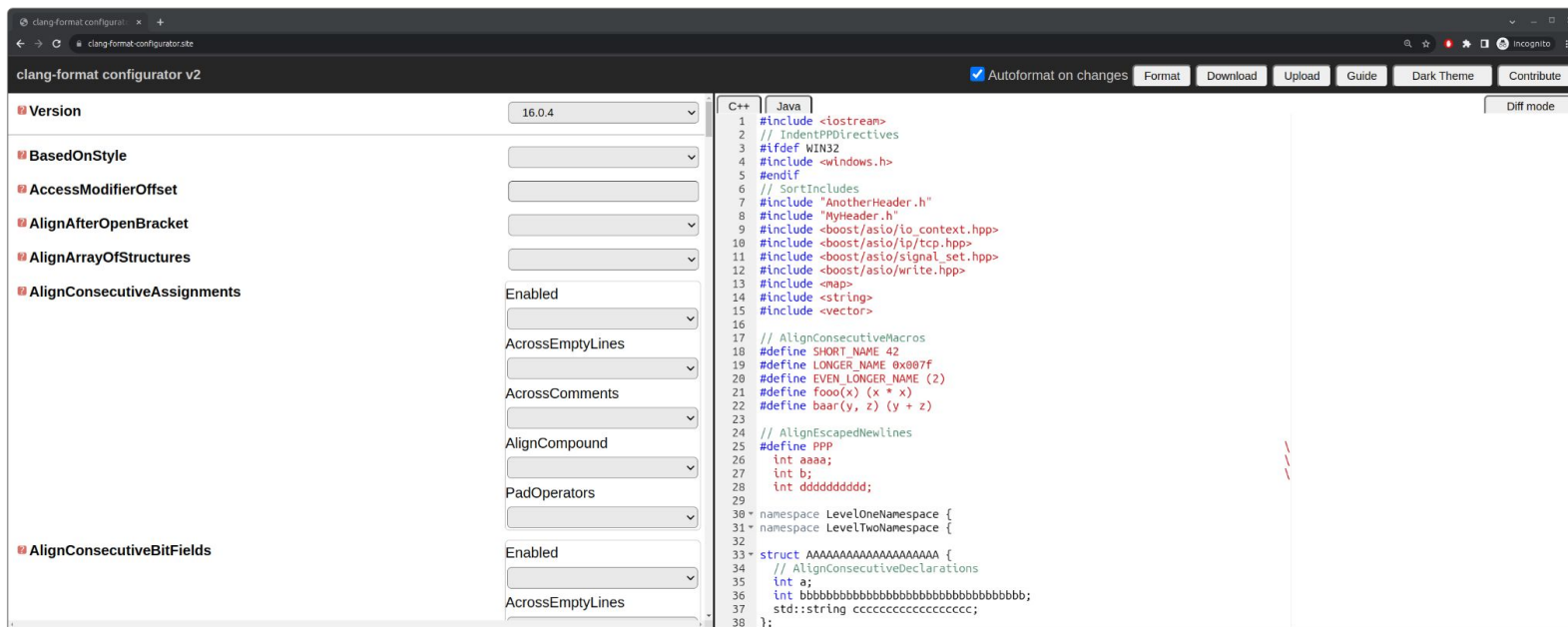
```
$ clang-format --dump-config > .clang-format
```

- otrzymamy config w formacie YAML:

```
---  
Language: Cpp  
AccessModifierOffset: -2  
AlignAfterOpenBracket: Align  
AlignArrayOfStructures: None  
AlignConsecutiveAssignments:  
  Enabled: false  
...
```

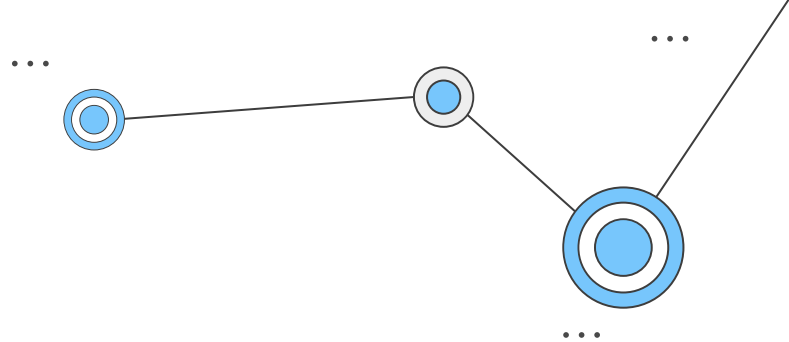
Clang-Format - konfigurator online

Przyjazny konfigurator online - [clang-format configurator](https://clang-format-configurator.site)



The screenshot displays the 'clang-format configurator v2' web application. On the left, a sidebar contains configuration options: Version (16.0.4), BasedOnStyle, AccessModifierOffset, AlignAfterOpenBracket, AlignArrayOfStructures, AlignConsecutiveAssignments (Enabled), and AlignConsecutiveBitFields (Enabled). The main area is split into two panes. The left pane shows C++ code with various formatting options like AcrossEmptyLines, AcrossComments, AlignCompound, PadOperators, and AcrossEmptyLines. The right pane shows the same code after formatting, with a 'Diff mode' button. The top of the interface includes a browser address bar, a title bar, and a toolbar with buttons for 'Format', 'Download', 'Upload', 'Guide', 'Dark Theme', and 'Contribute'. A checkbox for 'Autoformat on changes' is also present.

```
1 #include <iostream>
2 // IndentPPDirectives
3 #ifdef WIN32
4 #include <windows.h>
5 #endif
6 // SortIncludes
7 #include "AnotherHeader.h"
8 #include "MyHeader.h"
9 #include <boost/asio/io_context.hpp>
10 #include <boost/asio/ip/tcp.hpp>
11 #include <boost/asio/signal_set.hpp>
12 #include <boost/asio/write.hpp>
13 #include <map>
14 #include <string>
15 #include <vector>
16
17 // AlignConsecutiveMacros
18 #define SHORT_NAME 42
19 #define LONGER_NAME 0x807F
20 #define EVEN_LONGER_NAME (Z)
21 #define fooo(x) (x * x)
22 #define baar(y, z) (y + z)
23
24 // AlignEscapedNewLines
25 #define PPP
26 int aaaa;
27 int b;
28 int dddddddddd;
29
30 namespace LevelOneNamespace {
31 namespace LevelTwoNamespace {
32
33 struct AAAAAAAAAAAAAAAAAA {
34 // AlignConsecutiveDeclarations
35 int a;
36 int bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb;
37 std::string cccccccccccccccccc;
38 };
```



**Jak to wpiąć w
build system?**



Integracja w build systemie



Jeśli CMake  to korzystamy z `add_custom_target()` - target formatujący:

```
add_custom_target(clang-format
```

```
    COMMAND clang-format -i ${SOURCES}
```

```
    COMMENT "Formatting code with Clang-Format"
```

```
    WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}
```

```
    USES_TERMINAL
```

```
)
```

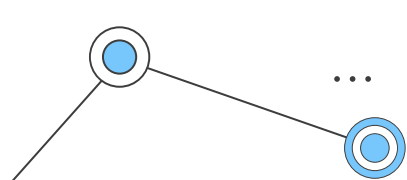
👉 nazwa targetu

👉 komenda wywołania

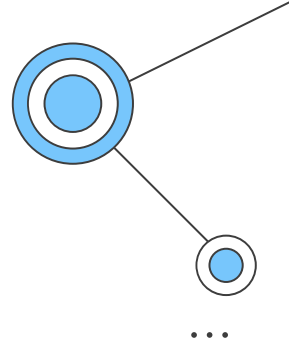
👉 komentarz

👉 uruchom w głównym katalogu

👉 uruchom jak w terminalu



Integracja w build systemie

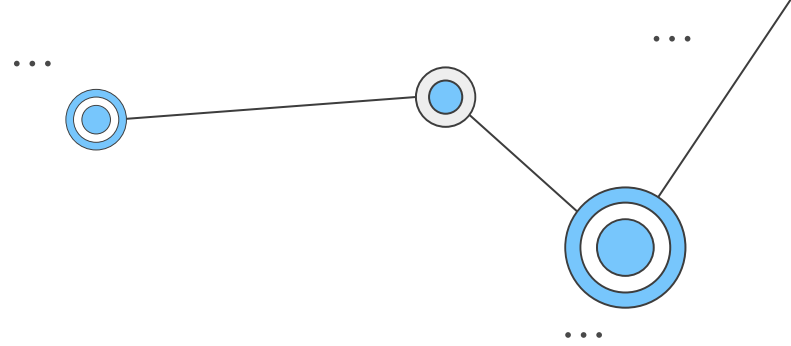


Target sprawdzający styl:

```
add_custom_target(clang-format-check
  COMMAND --dry-run --Werror ${SOURCES}
  COMMENT "Checking code with Clang-Format"
  ...
)
```

przykład





Baza kompilacji



Baza kompilacji



JSON Compilation Database Format Specification - format określający sposób odtworzenia pojedynczej jednostki kompilacji niezależnie od build systemu. Niektóre narzędzia lubią ten format.

```
[
  {
    "directory":"/home/user/llvm/build",
    "arguments":["/usr/bin/clang++", "-Irelative", "-c", "-o", "file.o", "file.cc"],
    "file":"file.cc"
  },
  {
    "directory":"/home/user/llvm/build",
    "command":"/usr/bin/clang++ -Irelative -c -o file.o file.cc",
    "file":"file2.cc"
  }
  ...
]
```

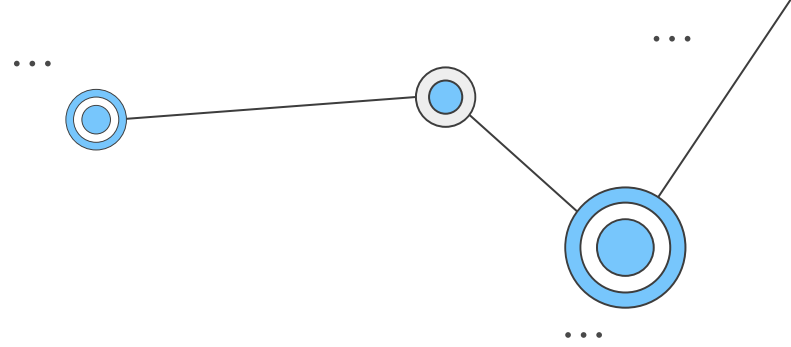



Baza kompilacji



CMake potrafi wygenerować bazę kompilacji, **Meson** robi to standardowo, **Bazel** posiada specjalną wtyczkę. A co jeśli mój build system nie potrafi...

- dla systemów Linux jest - **bear** - ʘ·ʘ·ʘ Build EAR - <https://github.com/rizotto/Bear>
- Qt + **Qmake** - QtCreator pozwala z IDE wygenerować bazę kompilacji
- projekty tworzone w **Visual Studio** - niektóre narzędzia potrafią przeczytać solucję
- jeżeli nic z powyższego nie działa, to powinieneś zastanowić się nad **zmianą** systemu budowania...



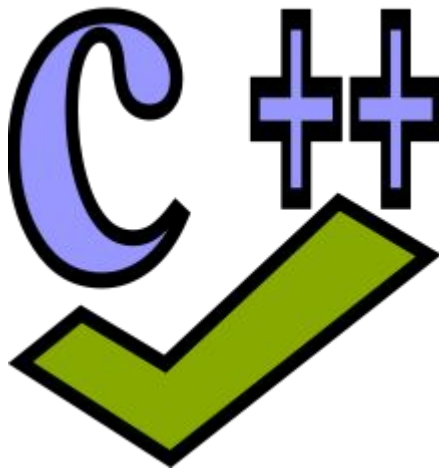
Cppcheck

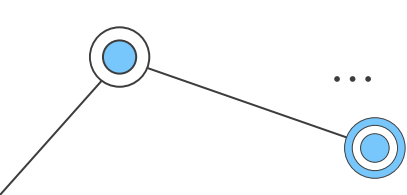


Cppcheck

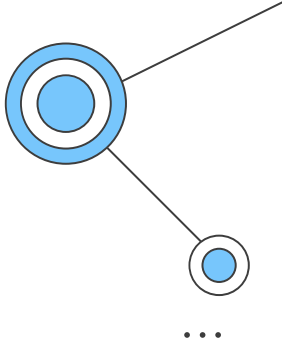


Cppcheck analizuje kod C/C++. Zapewnia unikalną analizę kodu - koncentruje się na wykrywaniu niezdefiniowanych zachowań i niebezpiecznych konstrukcji kodowania. Przy zachowaniu małej liczby false-positive'ów. Zaprojektowany tak, aby móc analizować kod, nawet jeśli ma niestandardową składnię (systemy wbudowane).





?



```
namespace third_party
{
    struct fancy_struct_interface
    {
        ~fancy_struct_interface() = default;

        virtual void do_something() = 0;
    };

    struct fancy_struct_impl : public fancy_struct_interface
    {
        void do_something() { /* ... */ };
    };
}
```



Cppcheck - podstawy



Najprostsze wywołanie to uruchomienie w głównym katalogu projektu z opcją `--enable`, gdzie podajemy jaki rodzaj checków chcemy stosować.

```
$ cppcheck --enable=all .
```

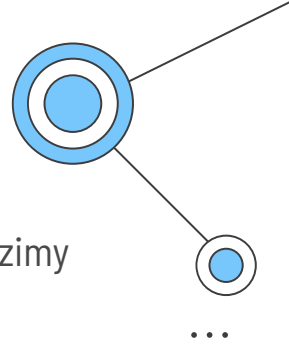
```
Checking app\app.cpp ...
1/3 files checked 64% done
Checking libmath\libmath.cpp ...
2/3 files checked 69% done
Checking libthirdparty\libthirdparty.cpp ...
libthirdparty\libthirdparty.hpp:12:14: style: The function 'do_something' overrides
a function in a base class but is not marked with a 'override' specifier.
[missingOverride]
    void do_something();
        ^
libthirdparty\libthirdparty.hpp:7:22: note: Virtual function in base class
    virtual void do_something() = 0;
...
...
```



...

Cppcheck - brak ścieżek do plików

Cppcheck raportuje problemy, gdy nie znajdzie plików nagłówkowych. Brakujące ścieżki sprawdzimy opcją: `--check-config`. Od Cppcheck 2.11 takie problemy raportowane są automatycznie.



```
$ cppcheck --enable=all --check-config .
```

```
Checking app\app.cpp ...
```

```
app\app.cpp:1:0: information: Include file: "libmath.hpp" not found.
```

```
[missingInclude]
```

```
#include "libmath.hpp"
```

```
^
```

```
app\app.cpp:2:0: information: Include file: "libthirdparty.hpp" not found.
```

```
[missingInclude]
```

```
#include "libthirdparty.hpp"
```

```
^
```



Cppcheck - baza kompilacji



W regularnym cyklu rozwoju projektu, chcemy korzystać z bazy kompilacji, a nie zarządzać konfiguracją. Wtedy korzystamy z opcji `--project`.

```
$ cppcheck --project=build/compile_commands.json --enable=all
```

```
Checking app\app.cpp ...
```

```
Checking app\app.cpp: _DEBUG=1;_DLL=1;_MT=1...
```

```
libthirdparty\libthirdparty.hpp:10:14: style: The function 'do_something' overrides  
a function in a base class but is not marked with a 'override' specifier.
```

```
[missingOverride]
```

```
    void do_something();
```

```
    ^
```

```
libthirdparty\libthirdparty.hpp:5:22: note: Virtual function in base class
```

```
    virtual void do_something() = 0;
```

```
    ^
```

```
...
```

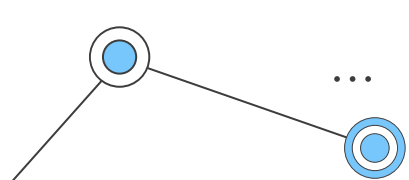


Cppcheck - suppressions...

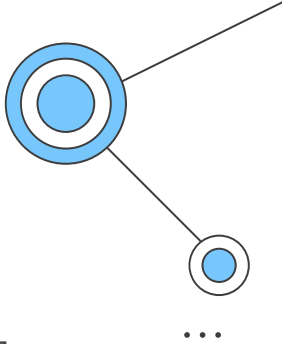


Cppcheck pozwala na wyciszenie problemów, np. z zewnętrznych bibliotek na kilka sposobów.

- opcja `--suppress=[error_id]:[filename]:[line]`
`$ cppcheck -I libmath --suppress=:*libthirdparty* .`
- wyłączenie katalogu z analizy:
`$ cppcheck -I libmath -i libthirdparty .`
- inline suppress przez komentarz:
`void do_something(); // cppcheck-suppress missingOverride`
`$ cppcheck -I libmath -I libthirdparty --inline-suppr .`



Cppcheck - suppressions pułapki



Uwaga na pułapki przy stosowaniu opcji `--suppress`, zwłaszcza dla zewnętrznego kodu.

Opcja ta **wyłączy wszystkie ostrzeżenia**, nawet takie, które skutkują przerwaniem analizy przez Cppcheck np. natknięcie się na branch z `#error` przy serii `#ifdef` z powodu braku jakiegoś `define`.

**Jeśli zaczynasz z Cppcheck to najpierw zrób pełną analizę bez wykluczeń
i sprawdź output, dopiero wtedy stosuj wykluczenia!**



Cppcheck - CI/CD



Jeśli Cppcheck wykryje jakiś defekt, to możemy kazać mu zwrócić określony exit code, służy do tego opcja: `--error-exitcode`.

```
> cppcheck --project=database.json --enable=all --error-exitcode=1
```

```
Checking app\app.cpp ...
app\app.cpp:1:0: information: Include file: "libmath.hpp" not found.
[missingInclude]
#include "libmath.hpp"
^
...
```

```
> echo Exit code = %errorlevel%
```

```
Exit code = 1
```



Cppcheck - pozostałe flagi



Flagi którym warto się przyjrzeć:

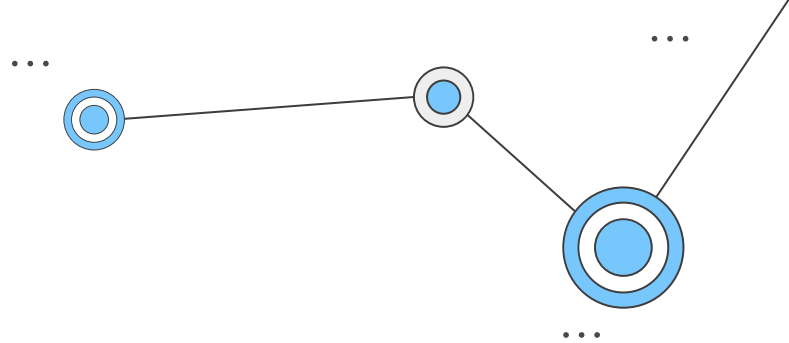
- **--addon** - dodatek np. checki MISRA
- **--cppcheck-build-dir** - katalog roboczy Cppcheck, w którym zapisuje informacje
- **--check-level** - poziom analizy np. *exhaustive*
- **--force** - sprawdzaj wszystkie konfiguracje
- **--inconclusive** - raportuj nawet jeśli analiza jest nierozstrzygająca
- **-j** - liczba wątków do analizy równoległej

Cppcheck - raporty

Cppcheck może wygenerować raport w postaci pliku XML. Z takiego XMLa można wygenerować raport HTML za pomocą skryptu: <https://github.com/danmar/cppcheck/tree/main/htmlreport>.
How-to tutaj: <https://dunterov.github.io/cppcheck/>.

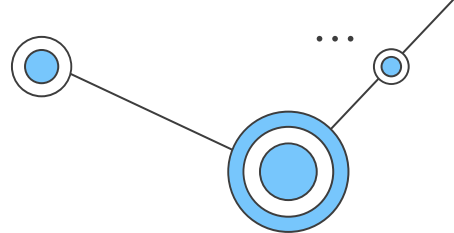
Cppcheck report - VIMs_SRC:

Defect summary:		Line	Id	CWE	Severity	Message	
Show # Defect ID		0	toomanyconfigs	398	information	Too many #ifdef configurations - cppcheck only checks 12 configurations. Use --force to check all configurations. For more details, use --enable=information.	
/src/os_unix.c							
✓	72	uninitStructMember	4916	preprocessorErrorDirective	error	failed to expand 'read_eintr', Wrong number of parameters for macro 'read_eintr'.	
✓	4	memleakOnRealloc				src/GvimExt/gvimext.cpp	
✓	3	uninitvar	1049	memleakOnRealloc	401	error	Common realloc mistake: 'cmdStrW' nulled but not freed upon failure
src/blowfish.c							
✓	1	preprocessorErrorDirec	44	syntaxError	error	syntax error	
src/ex_docmd.c							
✓	1	resourceLeak					
✓	1	sprintfOverlappingData	1100	uninitStructMember	908	error	Uninitialized struct member: cmd_loop_cookie.current_line
src/fold.c							
✓	1	syntaxError	2296	uninitStructMember	908	error	Uninitialized struct member: fp.fd_top
✓	1	toomanyconfigs	2296	uninitStructMember	908	error	Uninitialized struct member: fp.fd_len
src/gui_photon.c							
84 total		2160	memleakOnRealloc	401	error	Common realloc mistake: 'utf8_buffer' nulled but not freed upon failure	
src/gui_w32.c							
Statistics		8354	uninitStructMember	908	error	Uninitialized struct member: sign.uType	
src/libterm/src/keyboard.c							
		153	uninitStructMember	908	error	Uninitialized struct member: k.type	
src/misc1.c							
		7556	uninitStructMember	908	error	Uninitialized struct member: our_paren_pos.Inum	
		7566	uninitStructMember	908	error	Uninitialized struct member: our_paren_pos.Inum	
		7626	uninitStructMember	908	error	Uninitialized struct member: our_paren_pos.col	
src/normal.c							
		6720	uninitStructMember	908	error	Uninitialized struct member: start.Inum	
		6720	uninitStructMember	908	error	Uninitialized struct member: start.col	
		6720	uninitStructMember	908	error	Uninitialized struct member: start.coladd	
		6721	uninitStructMember	908	error	Uninitialized struct member: end.Inum	
		6721	uninitStructMember	908	error	Uninitialized struct member: end.col	

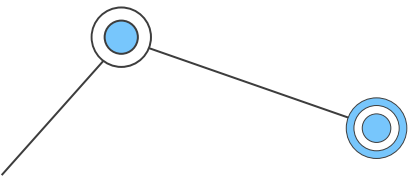


Clang-Tidy

Clang-Tidy

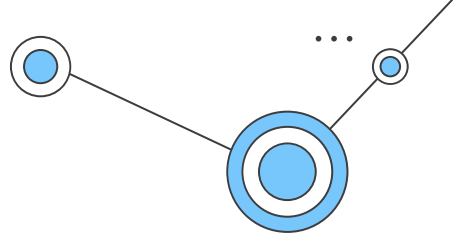


Clang-Tidy to oparte na Clang narzędzie do statycznej analizy kodu z możliwością automatycznej naprawy problemów. Projekt poddawany analizie musi budować się pod Clang'iem.



📁 przykłady pojawiające się później na podstawie repo: <https://github.com/the-risk-taker/cppcheck-clang-tidy-and-cmake>

Clang-Tidy - proste użycie



Clang-Tidy możemy uruchomić na projekcie lub pliku/plikach. Checki podajemy opcją `--checks`. Po znakach `--`` podajemy argumenty kompilacji.

```
$ clang-tidy app/app.cpp -checks=* -- -I libmath -I libthirdparty
```

```
22 warnings generated.
```

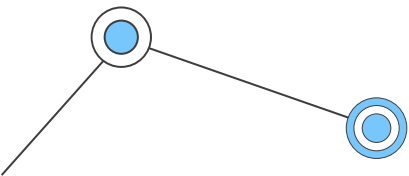
```
C:\devstuff\cppcheck-and-clang-tidy\app\app.cpp:9:1: warning: a trailing return  
type is disallowed for this function declaration [fuchsia-trailing-return]
```

```
auto main() -> int  
^
```

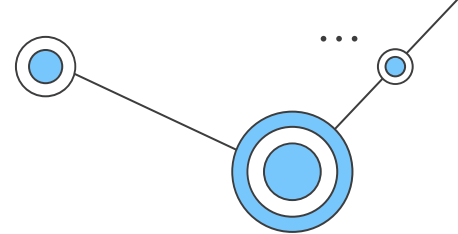
```
C:\devstuff\cppcheck-and-clang-tidy\app\app.cpp:9:6: warning: declaration must be  
declared within the '__llvm_libc' namespace [llvmlibc-implementation-in-namespace]
```

```
auto main() -> int  
^
```

```
...
```

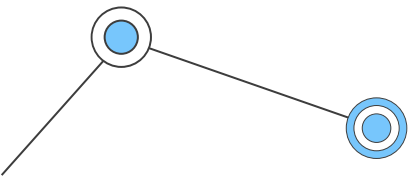


Clang-Tidy - grupy checków



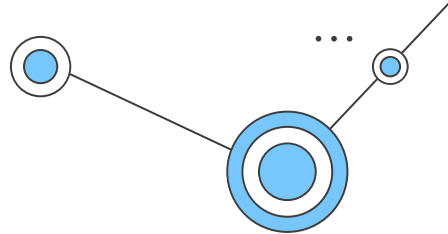
Najciekawsze grupy checków to:

- bugprone
- cert
- concurrency
- cppcoreguidelines
- hicpp
- misc
- modernize
- performance
- readability



Pełna lista tutaj: <https://clang.llvm.org/extra/clang-tidy/checks/list.html>

Clang-Tidy - definiowanie checków



Syntax definiowania checków jest następujący - przecinek jest separatorem:

-check=*

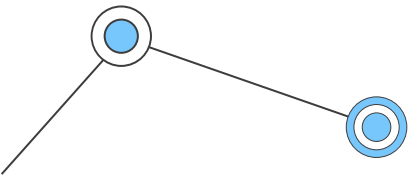
- włącza wszystkie checki (*)

-check=-*,modernize-*

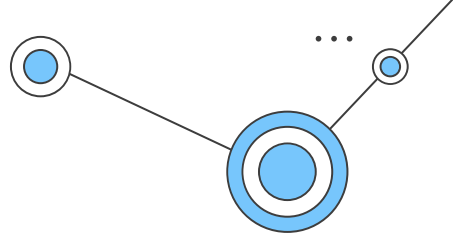
- wyłącza wszystkie (-*) ale włącza wszystkie z grupy modernize(modernize-*)

-check=-*,modernize-*,-modernize-avoid-c-arrays

- wyłącza wszystkie (-*) ale włącza wszystkie z grupy modernize(modernize-*) oraz wyłącza check na "surowe" tablice (-modernize-avoid-c-arrays)



Clang-Tidy - baza kompilacji



Clang-Tidy raczej chcemy uruchamiać korzystając z opcji bazy kompilacji. Potrzebujemy [listę] plików i zastosować opcję `-p`, `-project`, gdzie podajemy ścieżkę do katalogu z bazą kompilacji.

```
$ clang-tidy app/app.cpp libmath/libmath.cpp -p=build -checks=*
```

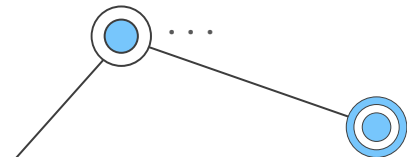
```
22 warnings generated.
```

```
27 warnings generated.
```

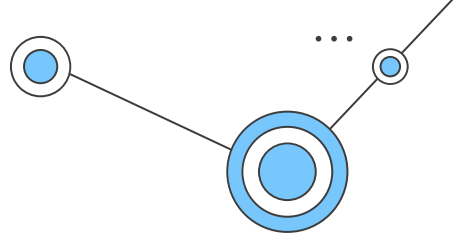
```
C:\devstuff\cppcheck-and-clang-tidy\app\app.cpp:4:6: warning: declaration must be  
declared within the '__llvm_libc' namespace [llvmlibc-implementation-in-namespace]  
void use_third_party(third_party::fancy_struct_interface& object)  
    ^
```

```
C:\devstuff\cppcheck-and-clang-tidy\app\app.cpp:9:5: warning: use a trailing return  
type for this function [modernize-use-trailing-return-type]  
int main()  
~~~ ^
```

```
...
```



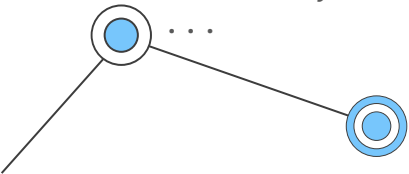
Clang-Tidy - plik konfiguracyjny



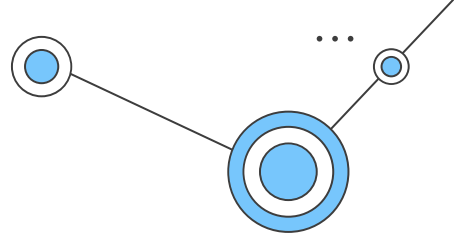
Clang-Tidy najlepiej skonfigurować plikiem **.clang-tidy**.

```
$ cat .clang-tidy
```

```
---
Checks: '-*',
        bugprone-*,
        -bugprone-exception-escape,
        cert-*,
        clang-analyzer-*, ...,
        '
WarningsAsErrors: '*'
HeaderFilterRegex: '.*'
FormatStyle: file
...
```

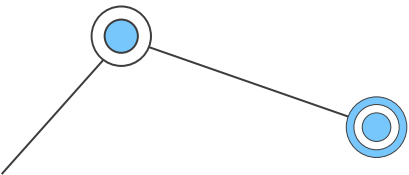


Clang-Tidy - suppressions

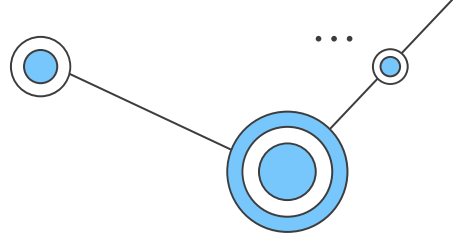


Clang-Tidy nie jest w tym najlepszy, ale mamy kilka opcji:

- wyłączenie bloku kodu - **NOLINTBEGIN ... NOLINTEND**
- wykluczenie plików z analizy - **list(FILTER ALL_SOURCES EXCLUDE REGEX \${CMAKE_BINARY_DIR})**
- stworzenie dummy configa - **file(WRITE \${CMAKE_CURRENT_BINARY_DIR}/.clang-tidy
"---\nChecks: ' -*,bugprone-no-escape' \n...\n")**



Clang-Tidy - automatyczny refactor



To w czym Clang-Tidy jest świetny to automatyczne aplikowanie poprawek. Służą do tego opcje:
-fix, -fix-errors, -fix-notes.

```
$ clang-tidy app/app.cpp -checks=* -fix-errors -- -I libmath ...
```

```
C:\devstuff\cppcheck-and-clang-tidy\app\app.cpp:9:5: warning: use a trailing return  
type for this function [modernize-use-trailing-return-type]
```

```
int main()
```

```
~~~ ^
```

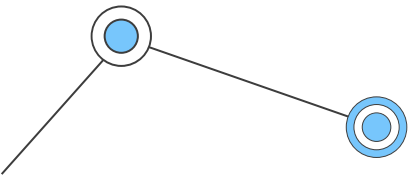
```
auto      -> int
```

```
C:\devstuff\cppcheck-and-clang-tidy\app\app.cpp:9:1: note: FIX-IT applied suggested  
code changes
```

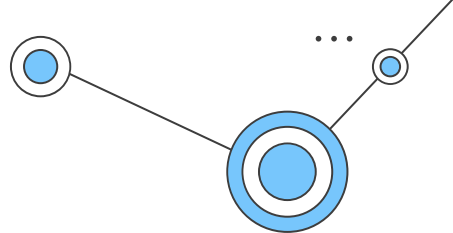
```
int main()
```

```
^
```

```
...
```



Clang-Tidy - CI



Gdy chcemy Clang-Tidy włączyć w pipeline to skorzystamy z opcji `-warnings-as-errors=*`.

```
$ clang-tidy app/app.cpp -checks=* -warnings-as-errors=* -- ...
```

```
22 warnings generated.
```

```
C:\devstuff\cppcheck-and-clang-tidy\app\app.cpp:9:1: error: a trailing return type  
is disallowed for this function declaration  
[fuchsia-trailing-return,-warnings-as-errors]
```

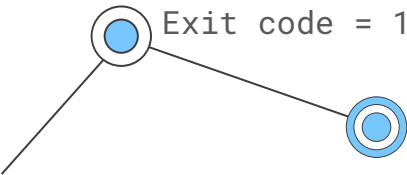
```
auto main() -> int
```

```
^
```

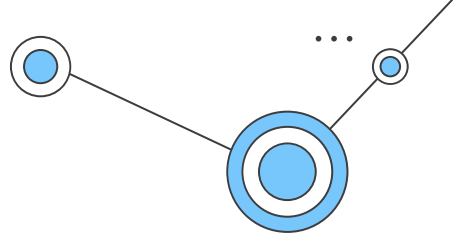
```
...
```

```
$ echo Exit code = %errorlevel%
```

```
Exit code = 1
```



Run-Clang-Tidy

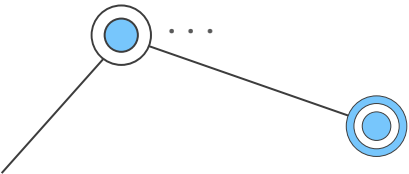


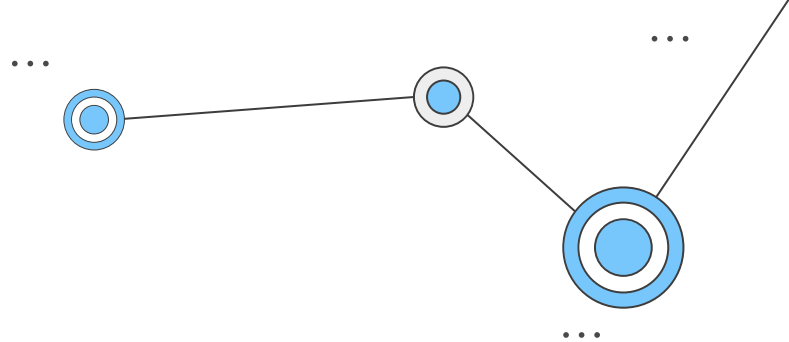
Clang-Tidy analizuje pliki jeden po drugim, `run-clang-tidy` potrafi wykorzystać 100% zasobów maszyny. Dodatkowo przed startem drukuje listę aktywnych checków.

```
$ run-clang-tidy -p=build -use-color
```

Enabled checks:

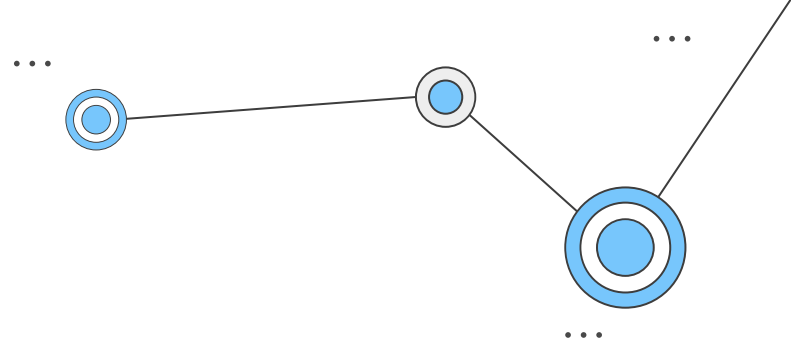
```
modernize-avoid-bind  
modernize-avoid-c-arrays  
modernize-concat-nested-namespaces  
modernize-deprecated-headers  
modernize-deprecated-ios-base-aliases  
modernize-loop-convert  
modernize-macro-to-enum  
modernize-make-shared  
modernize-make-unique
```



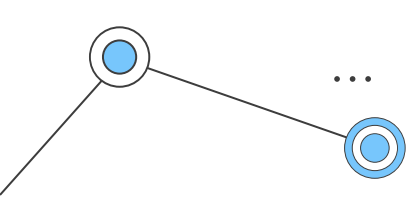


Case study

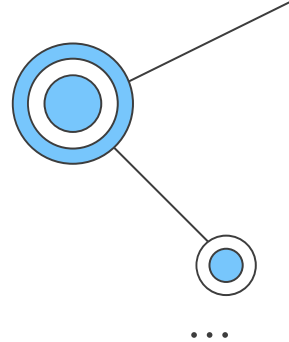
🔗 sprawdź ostatnie commity w repo: <https://github.com/the-risk-taker/gree-remote> aby prześledzić proces integracji narzędzi



Podsumowanie



Podsumowanie



- Używaj wszystkich dostępnych narzędzi
- Stosowanie tych narzędzi uczy nowych rzeczy
- Narzędzia najlepiej wprowadzać stopniowo do projektu
- Narzędzia powinny być wpięte w pipeline CI/CD
- **Uwaga: Używanie tych narzędzi uzależnia!**



A na zakończenie...

clang-format
clang
dynamic-analyzers
compilers
fuzz-testing
cppdepend
unit-testing
compiler-flags
ninja
linters
pvs-studio
sanitizers
clang-tidy
cppcheck
cppclean
coverity
static-analyzers
source-control
gcc
valgrind
formatters
cmake
msvc
best-practices
mutation-testing