

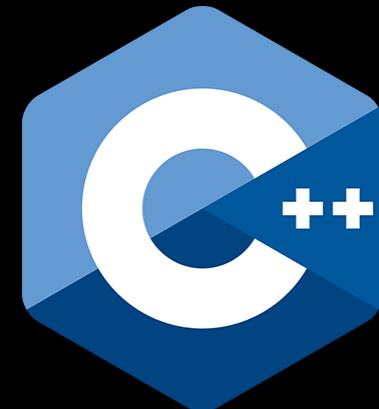
C++ now

C++ Painkillers

Anastasia Kazakova

2024
1

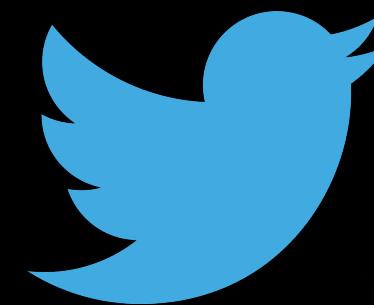
About me



C++: Embedded,
Telecom, 4G/LTE



Head of Marketing
& BizDev



@anastasiak2512



The Dutch C++ Group



cppunderthesea.nl

C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI

C++ Painkillers stories

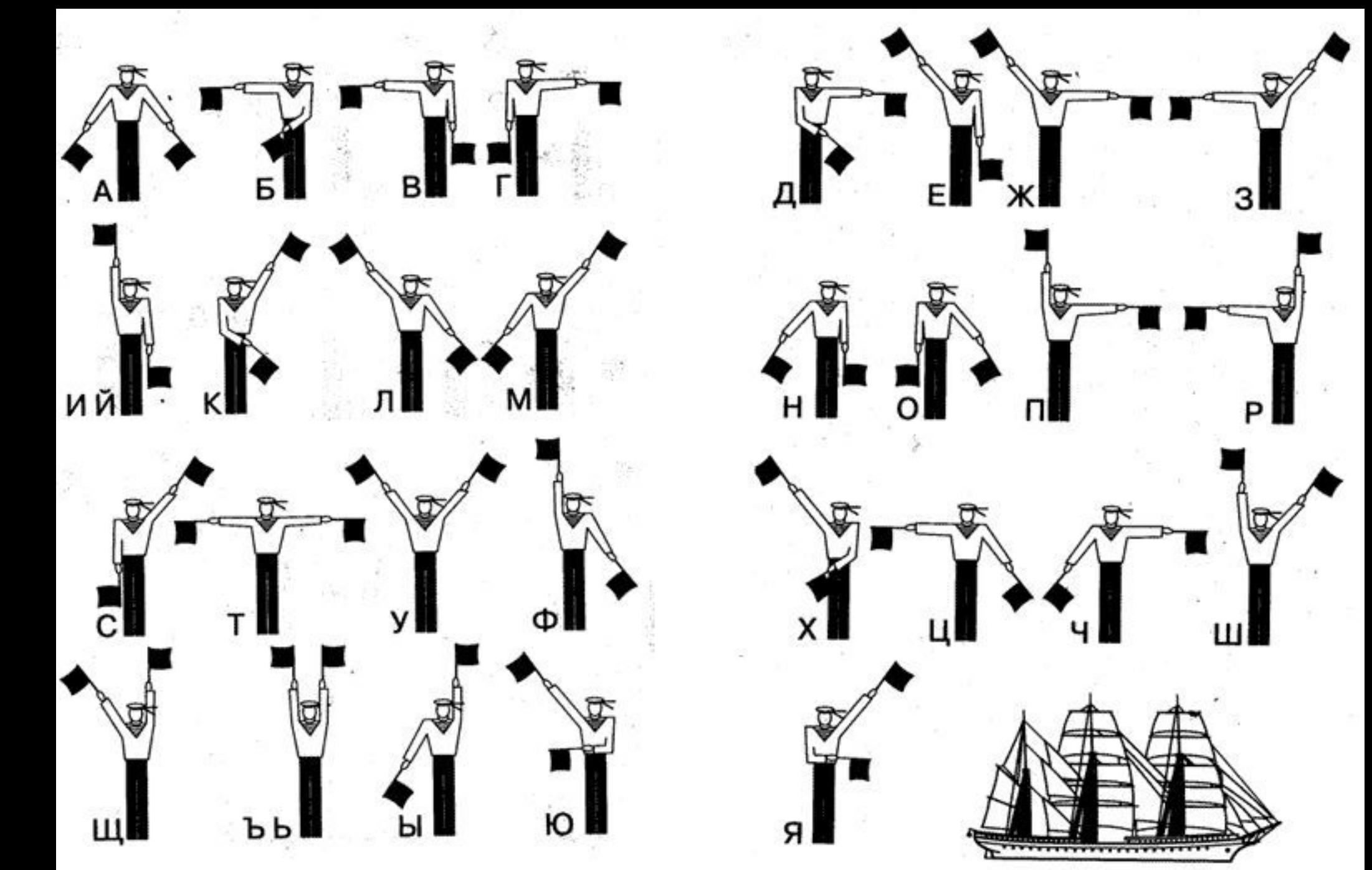
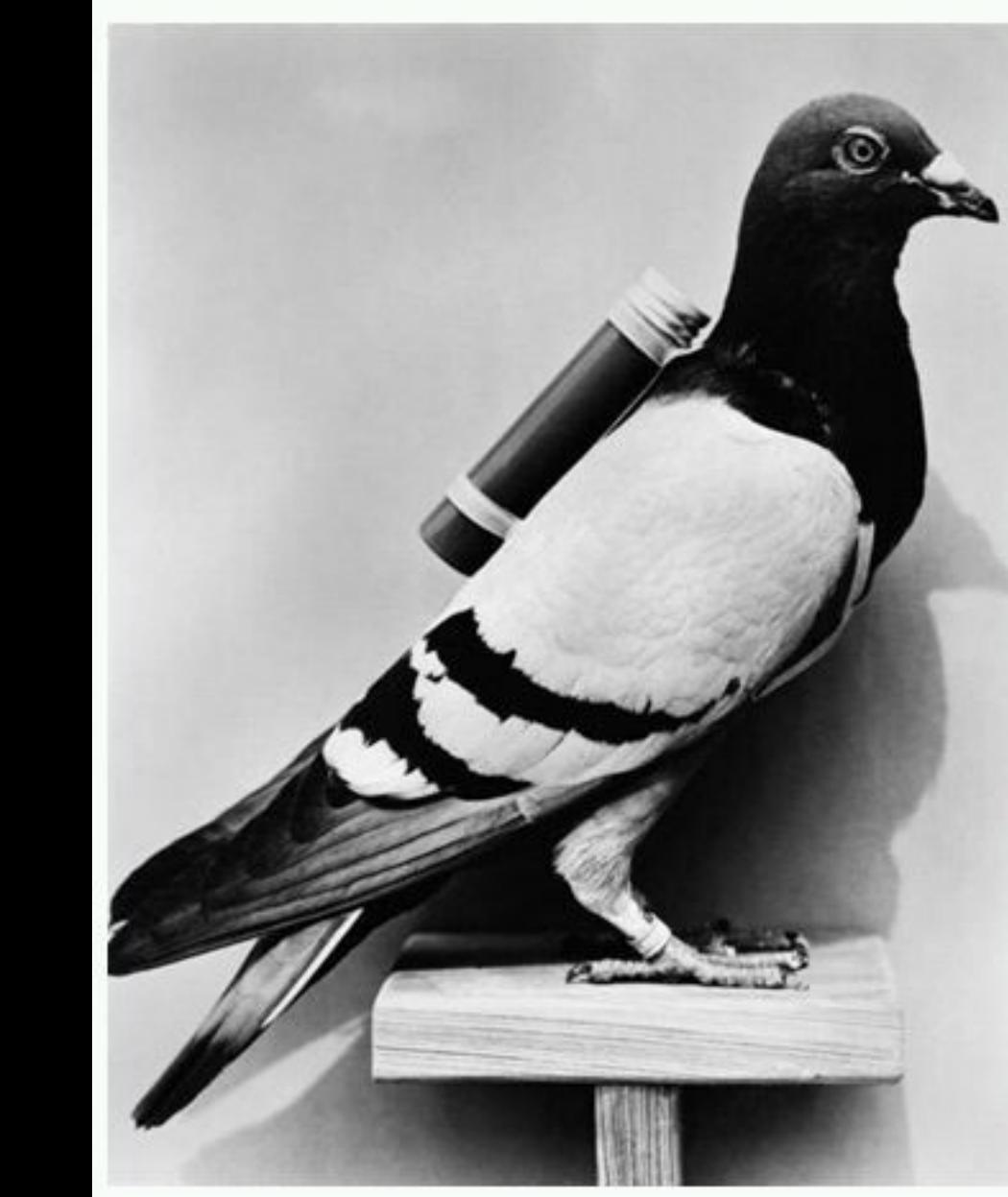
Story #0 "Popov" 

Popov 📻



[en.wikipedia.org/wiki/Aleksandr_Popov_\(physicist\)](https://en.wikipedia.org/wiki/Aleksandr_Popov_(physicist))

Popov 📻



Popov 



C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI

Do we have a problem?

Language

Tooling

Other

Tools are on
top for years!

<i>Which of these do you find frustrating about C++ dev?</i>	<i>Major %</i>
Managing libraries my application depends on	45 %
Build times	43 %
Setting up a CI pipeline from scratch	30 %
Managing CMake projects	30 %
Concurrency safety: Races, deadlocks, performance bottlenecks	27 %
Setting up a dev env from scratch	26 %
Parallelism support	23 %
Managing Makefiles	20 %
Memory safety: Bounds safety issues	20 %
Memory safety: Use-after-delete/free	20 %
Debugging issues in my code	18 %
Managing MSBuild projects	16 %
Unicode, internationalization, and localization	16 %
Security issues: disclosure, vulnerabilities, exploits	12 %
Type safety: Using an object as the wrong type	12 %
Memory safety: Memory leaks	12 %
Moving existing code to the latest language standard	9 %

Do we have a problem?

["How To Address 7 Major C++ Pain Points with CLion"](#) © CLion blog

["C++ MythBusters"](#) and ["C++ MythBusters Strike 2"](#) © Victor Ciura

C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI

Vitamin, Painkiller, or Cure?

“We throw away the candy.

We look at vitamins.

We really like painkillers.”

© Kevin Fong, venture capitalist

Vitamin, Painkiller, or Cure?

Painkiller = a quick-fix, a tool

Vitamin = guidelines, code styles, best practices

Cure = a language feature

C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI

Tools help: with macros #1

```
#define MAGIC 100
#define CALL_DEF(val, class_name) int call_##class_name() \
{ return val; }
```

```
#define CLASS_DEF(class_name) class class_##class_name { \
public: \
    int count_##class_name; \
    CALL_DEF(MAGIC, class_name) \
};
```

```
CLASS_DEF(A)
CLASS_DEF(B)
CLASS_DEF(C)
```

Declared in: MacroDebug.cpp

Definition:

```
#define CLASS_DEF(class_name) class class_##class_name { \
public: \
    int count_##class_name; \
    CALL_DEF(MAGIC, class_name) \
};
```

Replacement:

```
class class_A {
public:
    int count_A;
    int call_A() { return 100; }
};
```



Declared in: MacroDebug.cpp

Definition:

```
#define CLASS_DEF(class_name) class class_##class_name { \
public: \
    int count_##class_name; \
    CALL_DEF(MAGIC, class_name) \
};
```

Replacement:

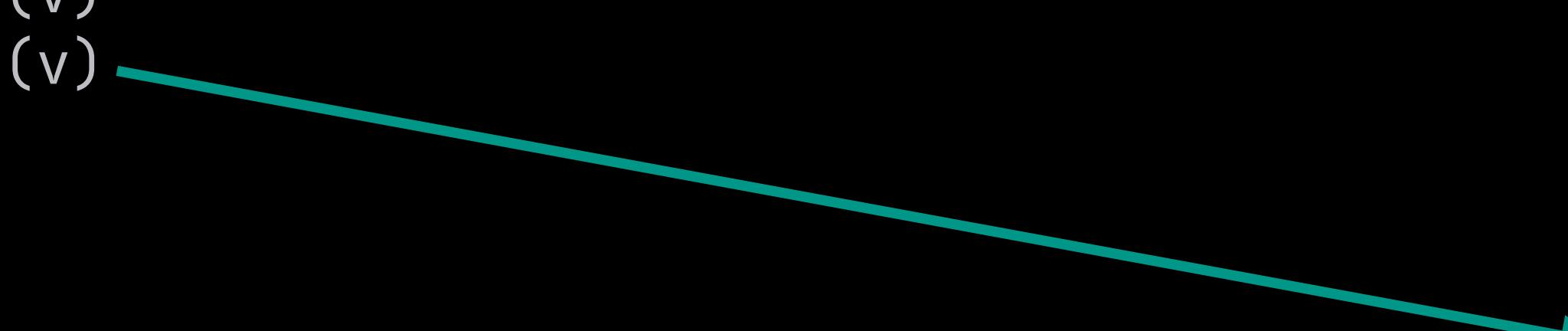
```
class class_C {
public:
    int count_C;
    int call_C() { return 100; }
};
```



Tools help: with macros #2

```
static int v;

#define __NEW_VAR(name, num) static void *__v_##num = (void *)&name;
#define _NEW_VAR(name, num) __NEW_VAR(name, num)
#define NEW_VAR(name) _NEW_VAR(name, __COUNTER__)

void counter_macro_sample() {
    NEW_VAR(v) 
    NEW_VAR(v)
    NEW_VAR(v) 
}
```

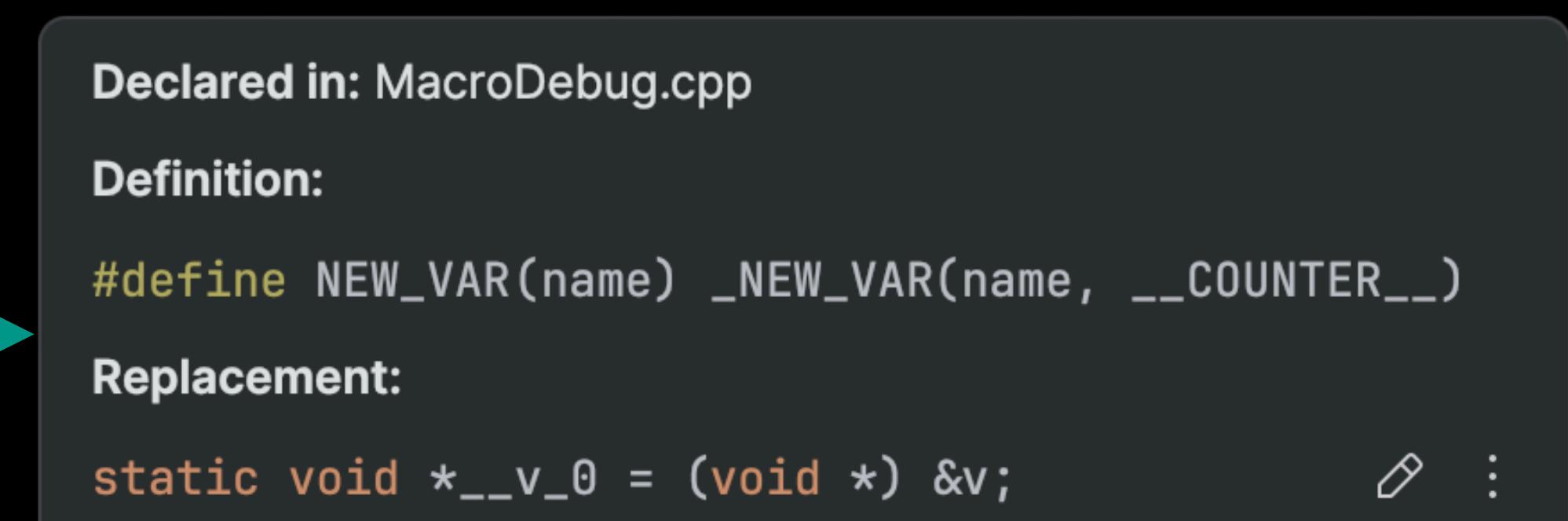
Declared in: MacroDebug.cpp

Definition:

```
#define NEW_VAR(name) _NEW_VAR(name, __COUNTER__)
```

Replacement:

```
static void *__v_0 = (void *) &v;
```



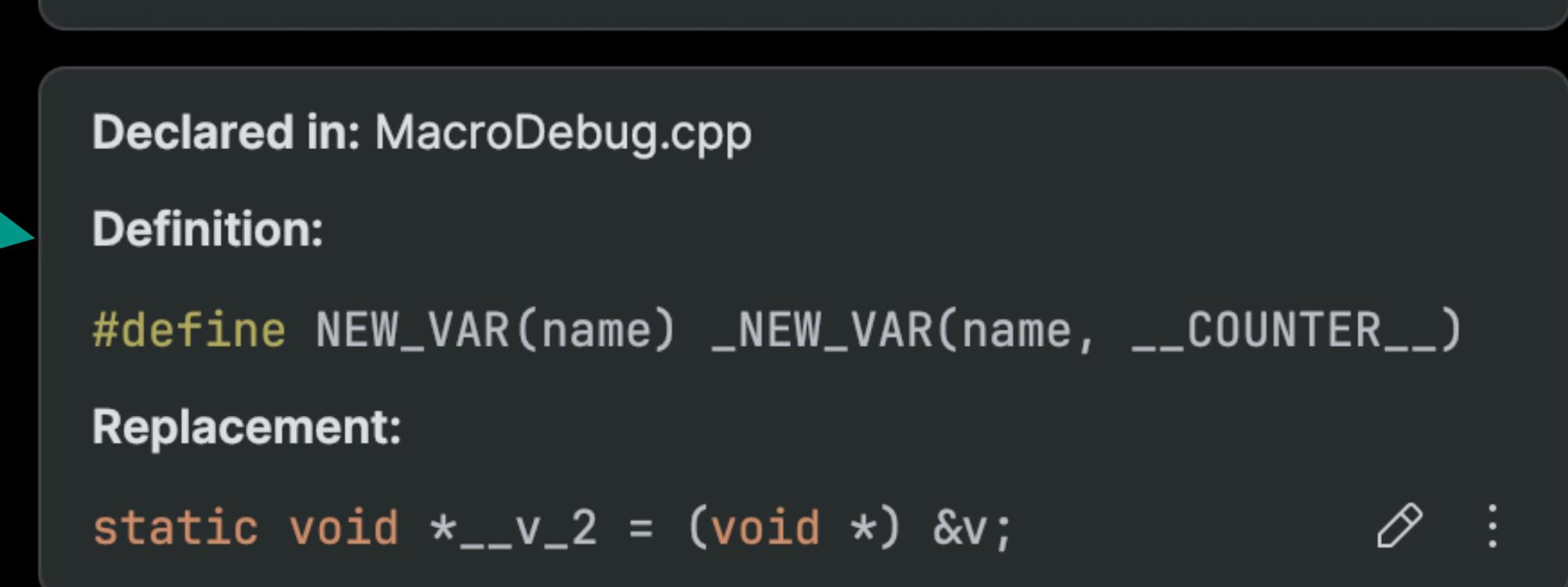
Declared in: MacroDebug.cpp

Definition:

```
#define NEW_VAR(name) _NEW_VAR(name, __COUNTER__)
```

Replacement:

```
static void *__v_2 = (void *) &v;
```



Tools help: with macros #3

```
BOOST_AUTO_TEST_SUITE(demo_suite)

BOOST_AUTO_TEST_CASE(test_equal_1
{
    BOOST_CHECK_EQUAL(MY_ONE, 1);
}

BOOST_AUTO_TEST_CASE(test_equal_wo
{
    char *name = new char[4];
    name[1] = 't';
    name[2] = 'e';
    name[3] = 's';
    name[4] = 't';

    BOOST_CHECK_EQUAL(name, "test"
}

BOOST_AUTO_TEST_SUITE_END()
```

Declared in: unit_test_suite.hpp

Definition:

```
#define BOOST_AUTO_TEST_SUITE(...) \
    BOOST_TEST_INVOKE_IF_N_ARGS( 1, \
        BOOST_AUTO_TEST_SUITE_NO_DECOR, \
        BOOST_AUTO_TEST_SUITE_WITH_DECOR, \
        __VA_ARGS__)
```

Replacement:

```
struct test_equal_word : public BOOST_AUTO_TEST_SUITE_FIXTURE {
    void test_method();
};

static void test_equal_word_invoker() {
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" fixture ctor").str());
    test_equal_word t;
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" fixture setup").str());
    boost::unit_test::setup_conditional(t);
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" test entry").str());
    t.test_method();
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" fixture teardown").str());
    boost::unit_test::teardown_conditional(t);
    ::boost::unit_test::unit_test_log.set_checkpoint(
        ::boost::unit_test::const_string("_file_name_", sizeof("_file_name_") - 1), static_cast<std::size_t>(16),
        (::boost::wrap_stringstream().ref() << "" << "test_equal_word" << "\" fixture dtor").str());
}

struct test_equal_word_id {
};

static boost::unit_test::ut_detail::auto_test_unit_registrar test_equal_word_registrar162 __attribute__((__unused__))( \
    boost::unit_test::make_test_case(&test_equal_word_invoker, "test_equal_word", "_file_name_", 16), \
    boost::unit_test::decorator::collector_t::instance());

void test_equal_word::test_method()
```

Tools help: with macros #4

```
#define DECL(z, n, text) text ## n = n;
BOOST_PP_REPEAT(5, DECL, int x)
```

Declared in: repeat.hpp

Definition:

```
# define BOOST_PP_REPEAT BOOST_PP_CAT(BOOST_PP_REPEAT_,
BOOST_PP_AUTO_REC(BOOST_PP_REPEAT_P, 4))
```

Replacement:

```
int x0 = 0;
int x1 = 1;
int x2 = 2;
int x3 = 3;
int x4 = 4;
```



Tools help: with overloads

```
class Fraction {...};

bool operator==(const Fraction& lhs, const Fraction& rhs) {...};

bool operator!=(const Fraction& lhs, const Fraction& rhs) {...};

Fraction operator*(Fraction lhs, const Fraction& rhs) {...};

std::ostream& operator<<(std::ostream& out, const Fraction& f)
{
    return out << f.num() << '/' << f.den();
}

void fraction_sample() {
    Fraction f1(3, 8), f2(1, 2);

    std::cout << f1 << " * " << f2 << " = " << f1 * f2 << '\n';
}
```

Tools help: with overloads

```
std::cout << f1 << " * " << f2 << " = " << f1 * f2 << '\n';
```

Declared in: operator_usages_highlight.cpp

std::basic_ostream<char> &operator<<(std::basic_ostream<char> &out, const Fraction & :)

Declared in: ostream

namespace std

template<_Traits>

basic_ostream<char, _Traits> &operator<<(basic_ostream<char, _Traits> &__os, const char *__str)

"operator<<" on cppreference.com ↗



```
std::cout << f1 << " * " << f2 << " = " << f1 * f2 << '\n';
```

Declared in: ostream

namespace std

public:

basic_ostream &basic_ostream::operator<<(int __n)

Tools help: named parameters

Named arguments

by Ehsan Akhgari and Botond Ballo, [N4172](#)

Self-explanatory Function Arguments

by Axel Naumann, [P0671](#)

Boost.Parameter

by David Abrahams and Daniel Wallin, [docs](#)

Tools help: named parameters

The screenshot shows the Compiler Explorer interface. On the left, the C++ source code is displayed:

```
38 constexpr auto adapt = [](auto& f, auto... input_arg_spec) {
39     auto arg_spec = hana::make_tuple(input_arg_spec...);
40     auto arg_names = get_names(arg_spec);
41     return [&f, arg_names](auto... input_arg_spec) {
42         auto args = hana::make_map(input_arg_spec...);
43         auto arg_pack = extract(arg_names, args);
44         return hana::unpack(arg_pack, f);
45     };
46 };
47 }

48 // Want to write
49 // foo(c = "hello world", b = 0.5, a = 10);
50
51 int main() {
52     auto my_foo = adapt(foo,
53                         "a"_arg,
54                         "b"_arg,
55                         "c"_arg
56     );
57 }
```

The code uses the `hana` library to handle named arguments. The variable `my_foo` is defined as an adapter function that takes a regular `foo` function and wraps it with named argument handling.

On the right, the assembly output for the `main` function is shown:

```
1 main:
2     mov    rax, 42
3     ret
```

The assembly output shows a simple `main` function that moves the value 42 into the `rax` register and returns.

Richard Powell

Named Arguments from Scratch

Tools help: named parameters

#1 Parameters of the same type, like weight, height, positions, etc.

```
4
5 → GameState::GameState(int fieldWidth,
6                               int fieldHeight)
7                               : field_(aleft: 0, atop: 0, awidth: fieldWidth, aheight: fieldHeight),
8                               ball_(pos: QPointF(xpos: field_.width() / 2, ypos: field_.height() - 30),
9                                     speed: QPointF(xpos: 200, ypos: -200)),
10                              paddle_(pos: QPointF(xpos: field_.width() / 2, ypos: field_.height() - 10),
11                                     width: 60, height: 20),
12                              score_(0) {
```

Tools help: named parameters

#2 Pass-by-value and pass-by-reference

```
→ void GameState::processCollisions() {
    if (applyCollision( & ball_, type: getCollisionWithWalls( b: ball_, bounding: getField()))

        for (auto iter : iterator<Brick *> = bricks_.begin(); iter != bricks_.end(); ++iter) {
            if (applyCollision( & ball_, type: getCollisionWithBrick( b: ball_, brick: iter->aabb))
                bricks_.erase( position: iter);
                score_ += 1;
                return;
            }

        if (applyCollision( & ball_, type: getCollisionWithBrick( b: ball_, brick: paddle_.aabb()))
    }
```

Tools help: type hints

Types:

- variables
- lambda
- deduced return types
- structured bindings

```
template<typename T, typename U>
auto doOperation(T t, U u) -> decltype(t + u) {
    return t + u;
}

void fun_type() {
    auto op :int = doOperation( 3, 0);
    auto op1 :long = doOperation( 3L, 0);
    auto op2 :double = doOperation( 3.0, 0);

    std::cout << op << " " << op1 << " " << op2;
}

void type_hints_bind() {
    int a = 1, b = 2;
    const auto&[x :int &, y :int &] = std::tie( &a, &b);
    const auto tuple :const tuple<int, char, double> = std::tuple(1, 'a', 2.3);
    const auto& [i :const int , c :const char , d :const double ] = tuple;
}
```

Tools help: with DSLs

Unreal Engine
reflection
macros-based dialect

The screenshot shows a code editor with two floating tool tips. The top tip is for the `UPROPERTY` macro, explaining its purpose in defining property metadata and variable specifiers, and providing examples of usage. The bottom tip is for the `EditDefaultsOnly` specifier, explaining it allows editing on archetypes but not instances, and noting its incompatibility with visible specifiers.

```
405 // Current health of the Pawn
406 UPROPERTY(EditAnywhere, BlueprintReadWrite, Replicated, Category = Health)
407 float
408
409
410 /** Ta
411 ^o
412
413 /** Pa
414 <-
415
416 /** Ki
417 <-
418
419 /** Re
420 <-
421
422 /**
423 * Kill
424 * @par
```

UPROPERTY(EditAnywhere, BlueprintReadWrite, Replicated, Category = Health)

Unreal Engine reflection macro

Properties are declared using standard C++ variable syntax, preceded by the `UPROPERTY` macro which defines *property metadata and variable specifiers*.

Remarks: The `UPROPERTY` macro enables a member variable of a *UCLASS* or a *USTRUCT* to be used as a *UPROPERTY*. A *UPROPERTY* has many uses. It can allow the variable to be *replicated, serialized, and accessed from Blueprints*. They are also used by the *garbage collector* to keep track of how many references there are to a *UObject*. Learn more: [Properties ↗](#), [Property Specifiers ↗](#).

Example:

```
UPROPERTY([specifier, specifier, ...], [meta(key=value, key=value, ...)])
Type VariableName;
```

```
285
286     /** default inventory list */
287     UPROPERTY(EDO, Category = Inventory)
288     TArray<# EditDefaultsOnly
289         # EditInstanceOnly
290         /** we
291             Press ⌘ to insert, ⌘ to replace
292             UPROPERTY(Transient, Replicated)
293             TArray<class AShooterWeapon*> Inventory;
```

EditDefaultsOnly

(Unreal Engine reflection specifier)

Indicates that this property can be edited by property windows, but only on archetypes. This Specifier is incompatible with any of the "Visible" Specifiers.

Tools help: with guidelines

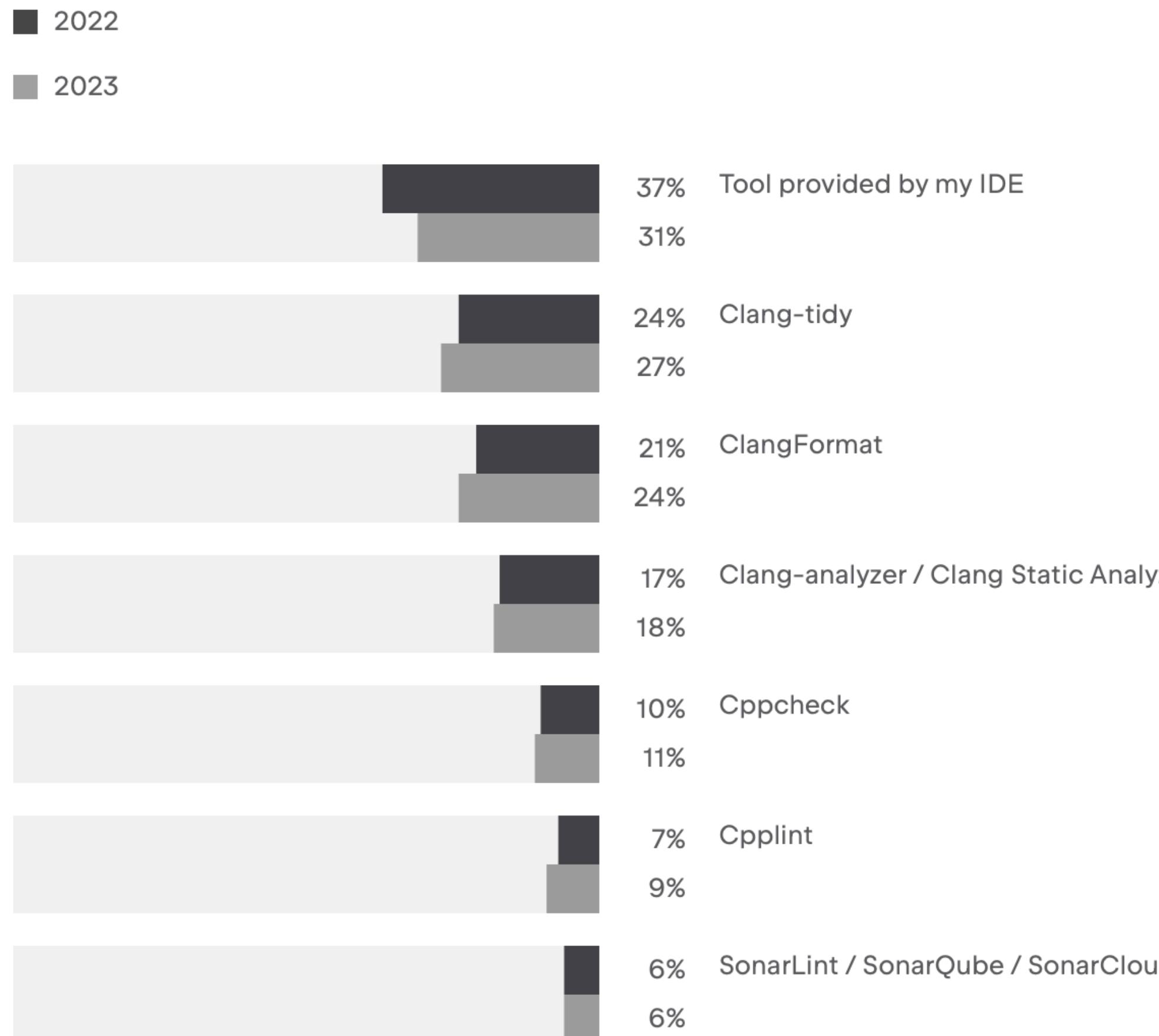
Checks and fixes in ClangTidy

cppcoreguidelines-avoid-capturing-lambda-coroutines	
cppcoreguidelines-avoid-const-or-ref-data-members	
cppcoreguidelines-avoid-do-while	
cppcoreguidelines-avoid-goto	
cppcoreguidelines-avoid-non-const-global-variables	
cppcoreguidelines-avoid-reference-coroutine-parameters	
cppcoreguidelines-init-variables	Yes
cppcoreguidelines-interfaces-global-init	
cppcoreguidelines-macro-usage	
cppcoreguidelines-misleading-capture-default-by-value	Yes
cppcoreguidelines-missing-std-forward	
cppcoreguidelines-narrowing-conversions	
cppcoreguidelines-no-malloc	
cppcoreguidelines-no-suspend-with-lock	
cppcoreguidelines-owning-memory	
cppcoreguidelines-prefer-member-initializer	Yes
cppcoreguidelines-pro-bounds-array-to-pointer-decay	
cppcoreguidelines-pro-bounds-constant-array-index	Yes
cppcoreguidelines-pro-bounds-pointer-arithmetic	
cppcoreguidelines-pro-type-const-cast	
cppcoreguidelines-pro-type-cstyle-cast	Yes
cppcoreguidelines-pro-type-member-init	Yes
cppcoreguidelines-pro-type-reinterpret-cast	
cppcoreguidelines-pro-type-static-cast-downcast	Yes
cppcoreguidelines-pro-type-union-access	
cppcoreguidelines-pro-type-vararg	
cppcoreguidelines-rvalue-reference-param-not-moved	
cppcoreguidelines-slicing	
cppcoreguidelines-special-member-functions	
cppcoreguidelines-virtual-class-destructor	Yes

modernize-avoid-bind	Yes
modernize-avoid-c-arrays	Yes
modernize-concat-nested-namespaces	Yes
modernize-deprecated-headers	Yes
modernize-deprecated-ios-base-aliases	Yes
modernize-loop-convert	Yes
modernize-macro-to-enum	Yes
modernize-make-shared	Yes
modernize-make-unique	Yes
modernize-min-max-use-initializer-list	Yes
modernize-pass-by-value	Yes
modernize-raw-string-literal	Yes
modernize-redundant-void-arg	Yes
modernize-replace-auto_ptr	Yes
modernize-replace-disallow-copy-and-assign-macro	Yes
modernize-replace-random-shuffle	Yes
modernize-return-braced-init-list	Yes
modernize-shrink-to-fit	Yes
modernize-type-trait	Yes
modernize-unary-static-assert	Yes
modernize-use-auto	Yes
modernize-use-bool-literals	Yes
modernize-use-constraints	Yes
modernize-use-default-member-init	Yes
modernize-use-designated-initializers	Yes
modernize-use-emplace	Yes
modernize-use-equals-default	Yes
modernize-use-equals-delete	Yes
modernize-use-nodiscard	Yes
modernize-use-noexcept	Yes
modernize-use-nullptr	Yes
modernize-use-override	Yes
modernize-use-starts-ends-with	Yes
modernize-use-std-numbers	Yes
modernize-use-std-print	Yes
modernize-use-trailing-return-type	Yes
modernize-use-transparent-functors	Yes
modernize-use-ungaught-exceptions	Yes
modernize-use-using	Yes

Tools help: with guidelines

Which of the following tools do you or your team use for guideline enforcement or other code quality/analysis?



Bryce Adelstein Lelbach

Principal Architect at NVIDIA

I think the decrease in IDE-provided analysis tools here is indicative of people incorporating static analysis into their CI, like running the Clang-tidy/ClangFormat/Clang static analyzer in GitHub Actions.

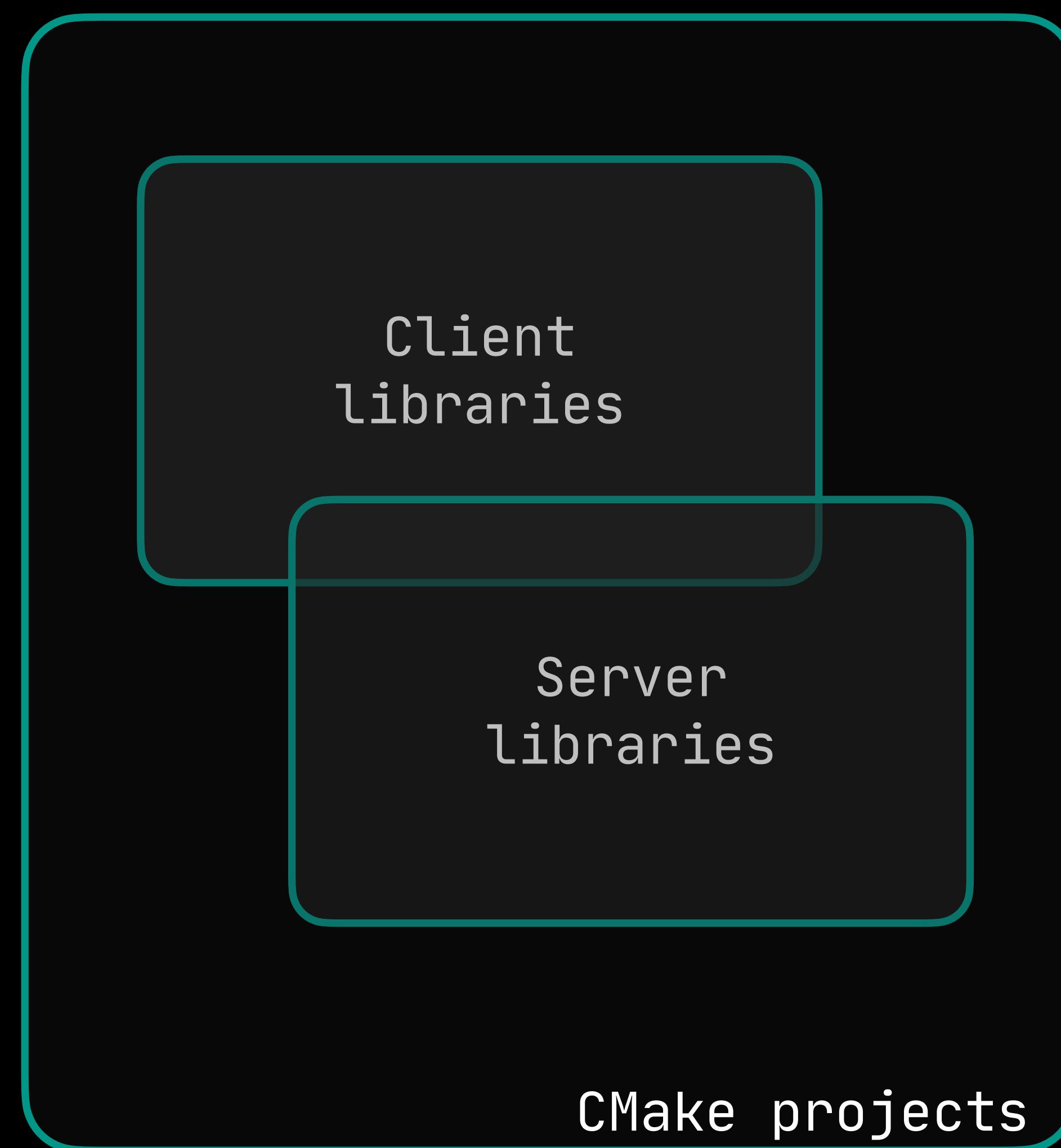
X (formerly Twitter)

C++ Painkillers stories

Story #1

“Dependency validation with NetBeans”

Dependency validation with NetBeans



\$ cmake ..

CMake configures the
project

NetBeans

\$ cmake --build

CMake builds an already-
generated project binary
tree

C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI

Role of the C++ tooling

C++Now 2017

A look at C++ through the glasses of a language tool

CppCon 2017

New standards to the rescue: the view through an IDE's glasses

CppCon 2017

Tools from the C++ eco-system to save a leg

CppCon 2018

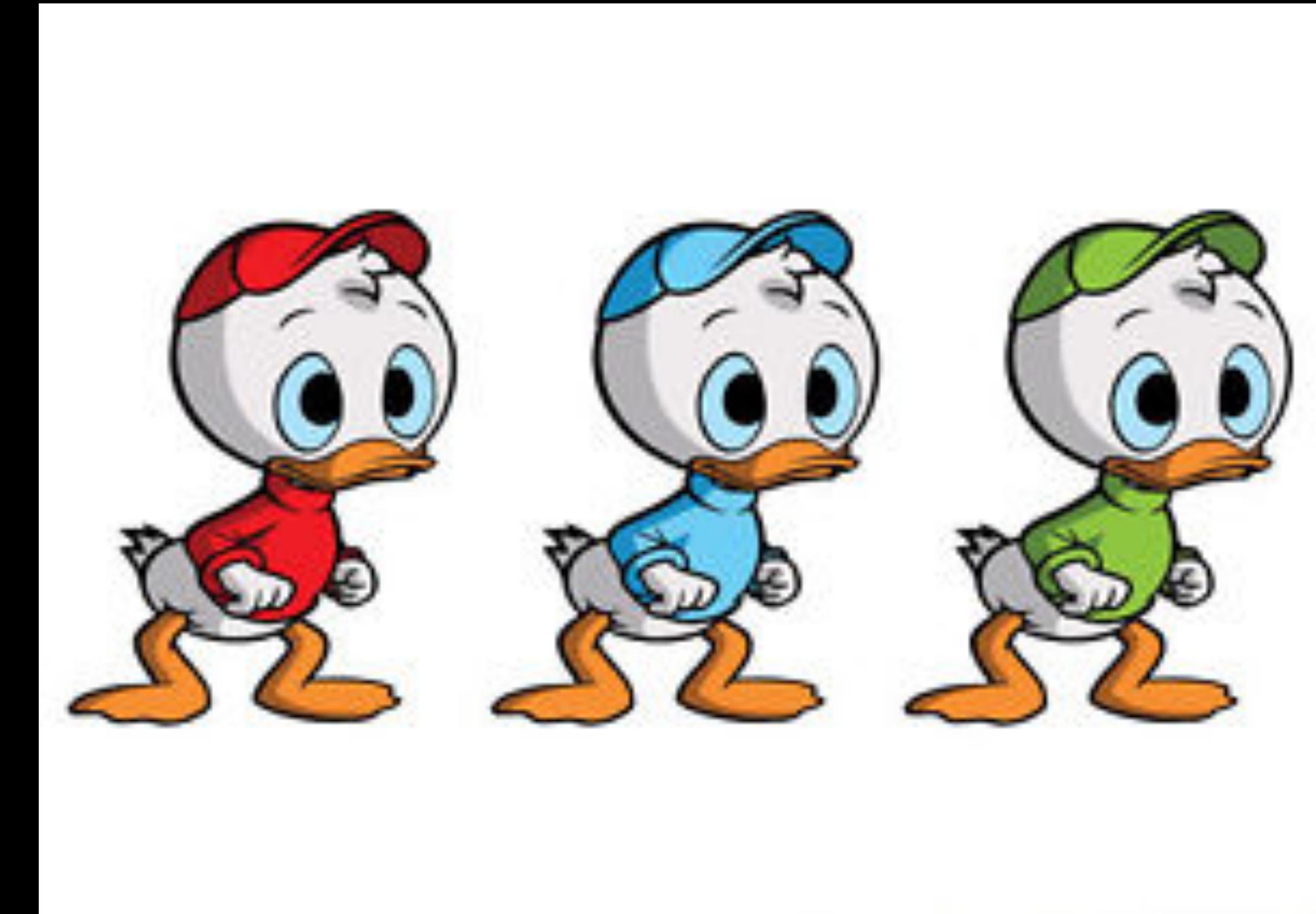
Debug C++ Without Running

C++ toolability: sameness

```
template<class T, int ... X>
T pi(T(X...));
```



```
int main() {
    return pi<int, 42>;
}
```



```
constexpr auto rexpr = ^int;
typename[:rexpr:] a = 42;
```

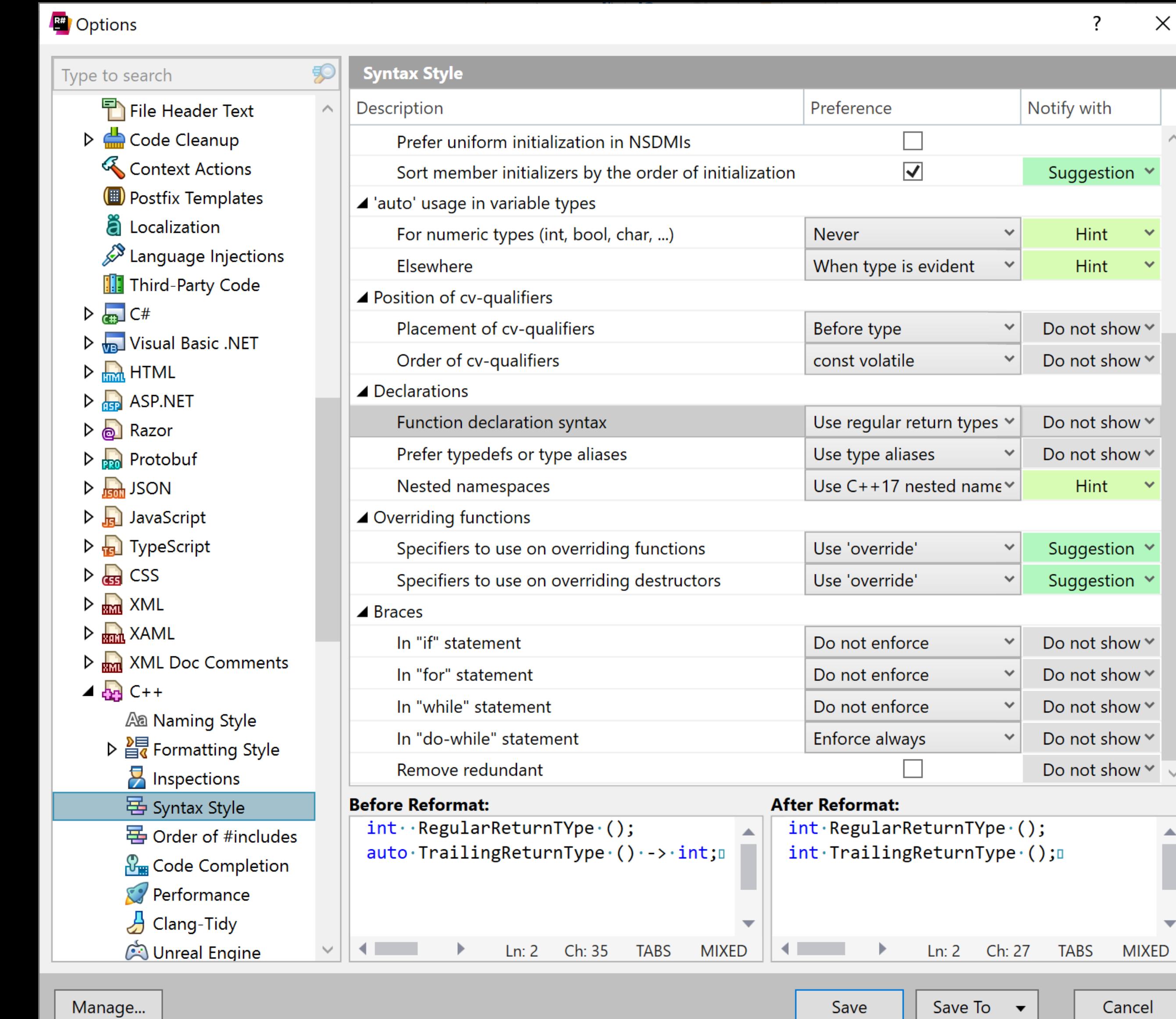
C++ toolability: sameness

Almost Always Auto or when type is evident or never for numeric types.

Const before or after the type it applies to.

Trailing return type for lambdas or always.

Virtual explicitly (UE) or override/ final and no virtual (C++ Core Guidelines).



C++ toolability

```
template<int>
struct x {
    int z;
    bool p;
    x(int i) { }
};
```

```
void test(int y) {
```

```
    const int a = 100;
```

```
x<a> k = x<a>(0)
```

Size = 8 bytes



⋮

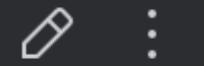


```
    auto k = x<a>(0);
    auto l = y<a>(0);
```

}

```
bool l = y<a>(0)
```

Size = 1 byte



C++ toolability

Declared in: main.cpp

```
y::y(x)
```

```
void test() {
    struct x {
    };

    struct y {
        y(x) {};
        x(z);
    };
}
```

Declared in: main.cpp

```
x y::z
```

Size = 1 byte

C++ toolability

```
void test() {  
    struct x {  
        x(int) {};  
};
```

```
x a = (x)-5  


---

Size = 1 byte
```

```
int y = 100;  
  
auto a = (x)-5;  
auto b = (y)-5;  
}
```

```
int b = (y)-5  


---

Size = 4 bytes
```

C++ toolability

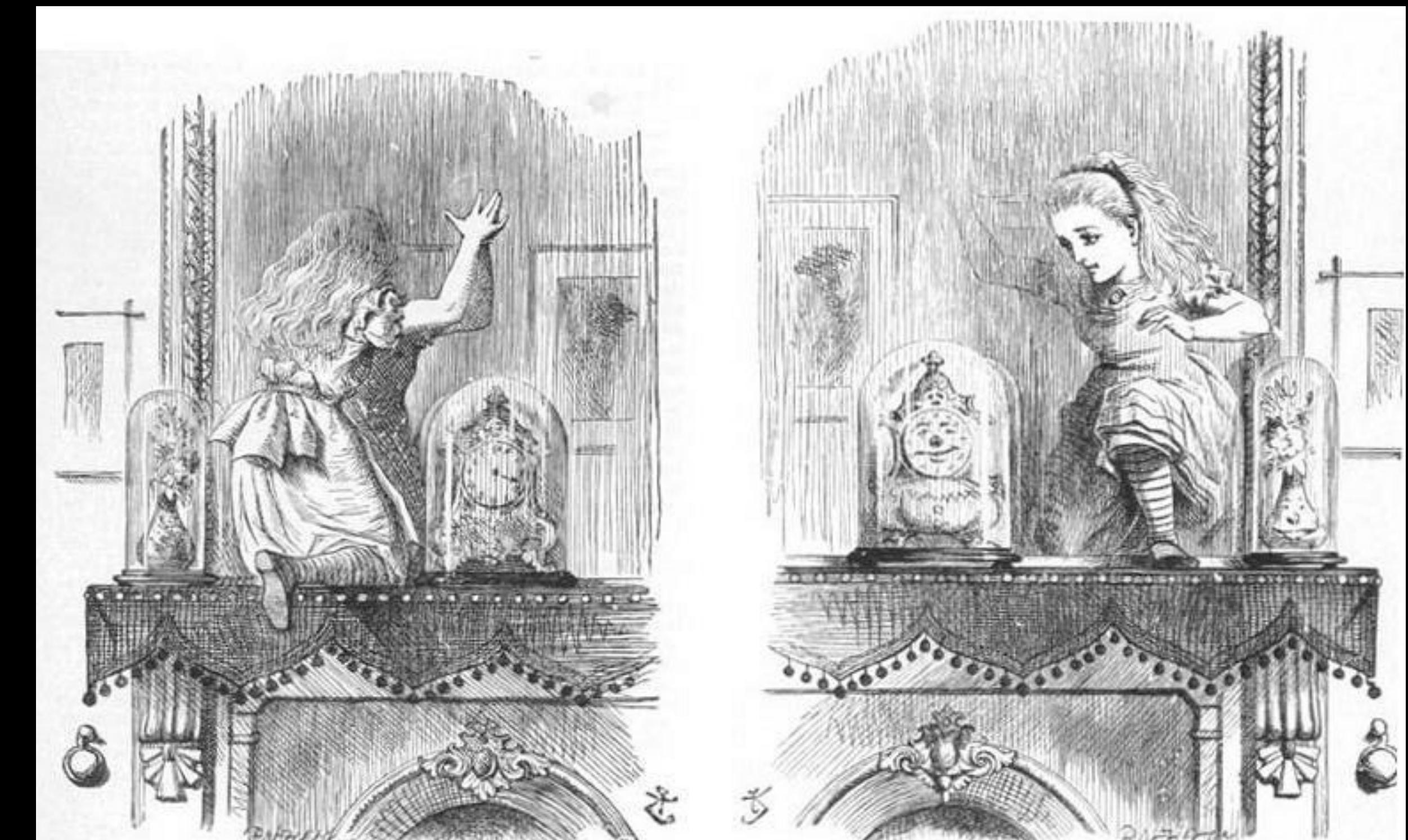
```
int(x), y, a, b, c, d, e, f, *z;
```

```
int(x), y, a, b, c, d, e, f, new int;
```



C++ toolability

To parse C++
we need to distinguish
types from **non-types**



C++ toolability

Fuzzy parsing is kinda possible but...

```
46 template<int N>
47 struct A {
48     A(int p) {}
49     operator int() { return N; }
50 };
51
52 ▷ int main() {
53     // Formatting setting for templates and
54     // integer expressions are different
55     int A = 10;
56     if (A < 3 + 6 > (2 + 8) + 1 > 5) {
57         std::cout << "print A" << std::endl;
58     }
59
60     int B = 10;
61     if (B < 20 > 5 + 5 > 5 + 10) {
62         std::cout << "print B" << std::endl;
63     }
64     return 0;
65 }
```

f main

56:39 (39 chars, 1 line break) LF UTF-8 .clang-tidy ClangFormat C++: cf_parser | Debug

ClangFormat

C++ toolability

Resolve depends on:

- order of the definitions
- default arguments
- overload resolution
- project model context

```
//foo.h
#ifndef MAGIC
template<int>
struct x {
    x(int i) { }
};
#else
int x = 100;
#endif

//foo.cpp
#include "foo.h"
void test(int y) {
    const int a = 100;
    auto k = x<a>(0);
}
```

x<100> k = x<a>(0)
Size = 1 byte

bool k = x<a>(0)
Size = 1 byte

C++ toolability

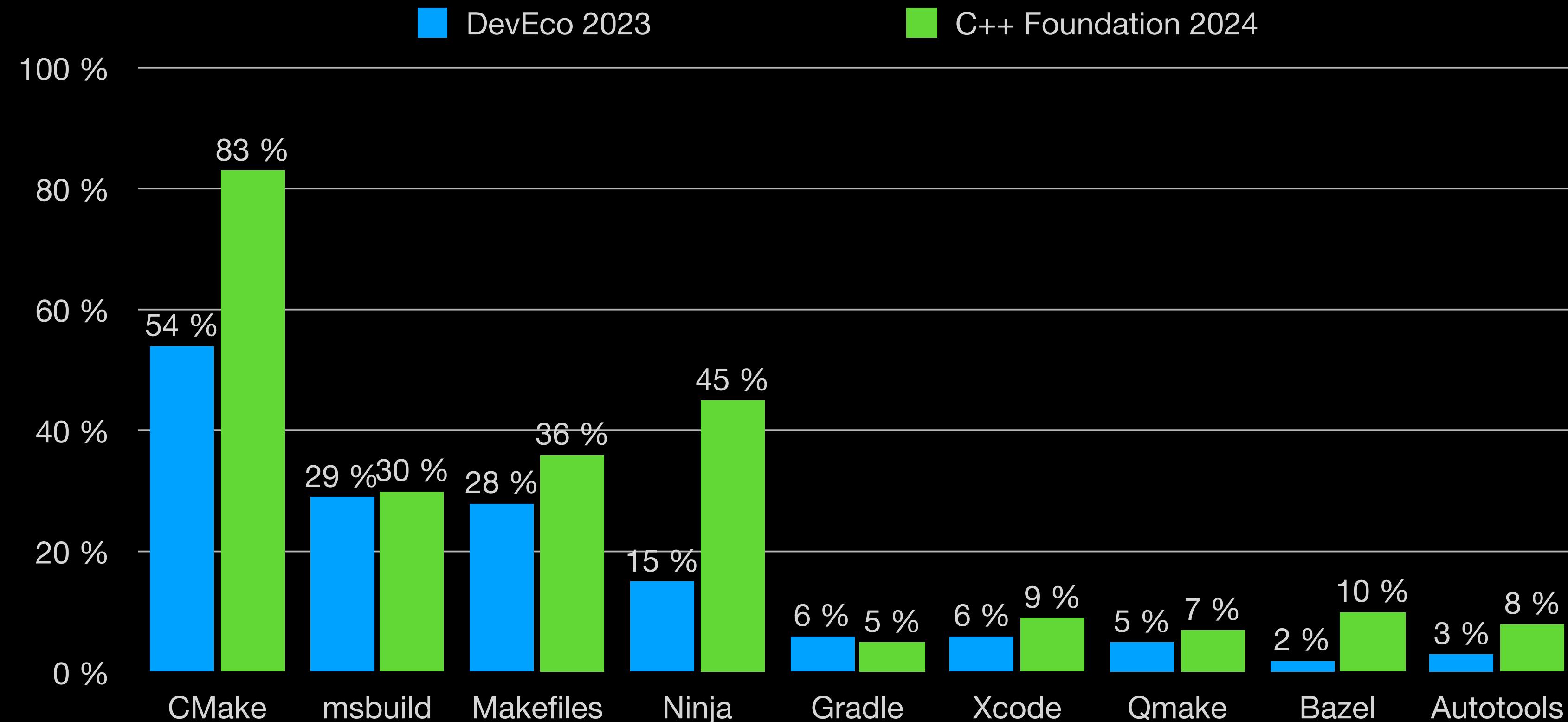
You wanna standard C++
Build System?

You got One!

It's called CMake!

© Bryce Adelstein

Lelbach, CppNow 2021



C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI

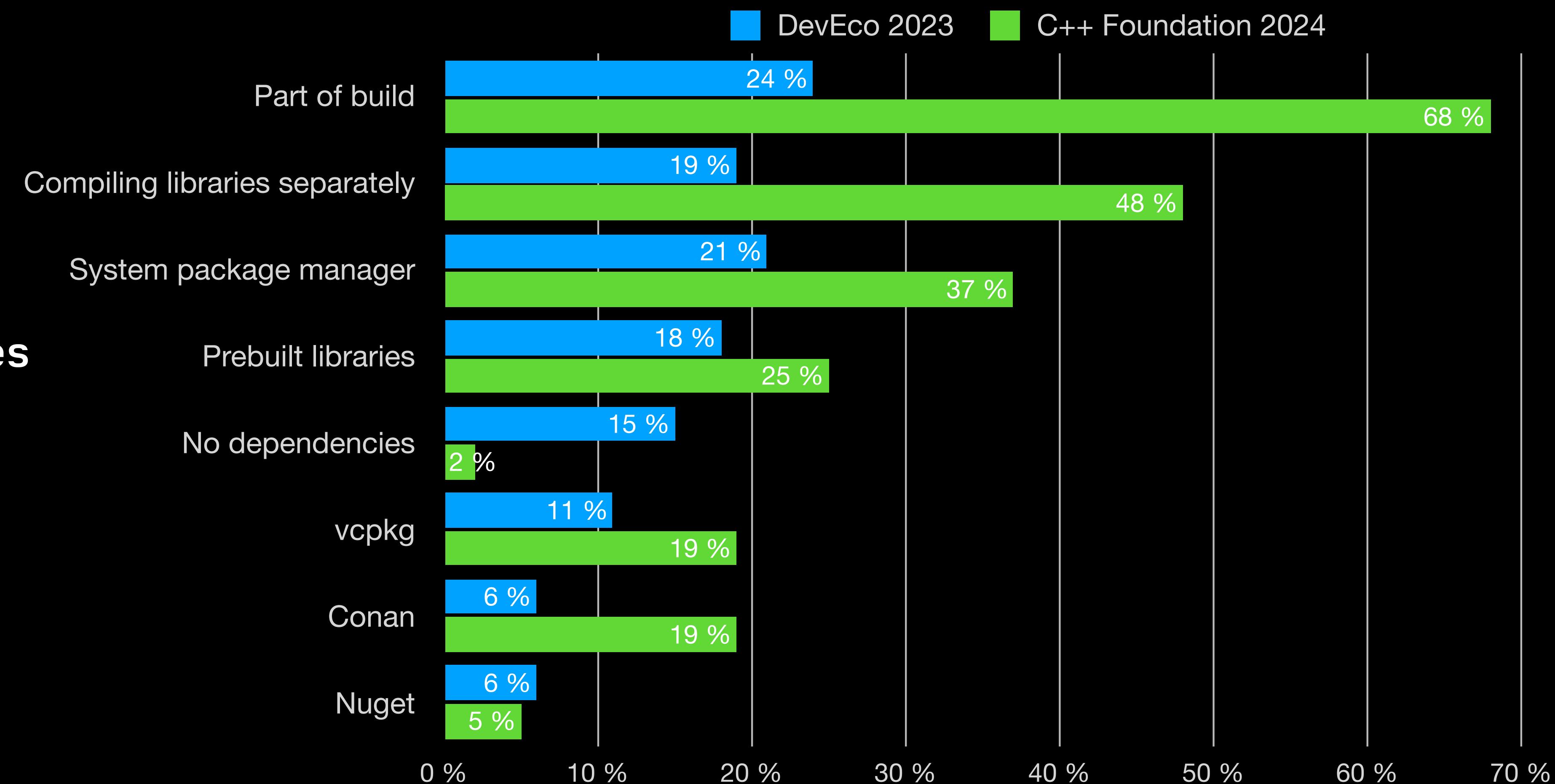
Tools for C++ painpoints

Which tools to use?

<i>Which of these do you find frustrating about C++ dev?</i>	<i>Major %</i>
Managing libraries my application depends on	45 %
Build times	43 %
Setting up a CI pipeline from scratch	30 %
Managing CMake projects	30 %
Concurrency safety: Races, deadlocks, performance bottlenecks	27 %
Setting up a dev env from scratch	26 %
Parallelism support	23 %
Managing Makefiles	20 %
Memory safety: Bounds safety issues	20 %
Memory safety: Use-after-delete/free	20 %
Debugging issues in my code	18 %
Managing MSBuild projects	16 %
Unicode, internationalization, and localization	16 %
Security issues: disclosure, vulnerabilities, exploits	12 %
Type safety: Using an object as the wrong type	12 %
Memory safety: Memory leaks	12 %
Moving existing code to the latest language standard	9 %

Tools for C++ painpoints

#1 Managing libraries



Tools for C++ painpoints

#1 Managing libraries
40% repos with
`CMakeLists.txt` inside

Query summary

Query type: **Count**

✓ Forks dropped
✓ Archived dropped

Primary language of repositories: `C` `C++`

Licenses: Any

Updated after: **01/01/2020, 00:00:00**

Stars: **[10 .. Any]**

File's language: `Text` `CMake`

File's path **REGEX: CMakeLists.txt**

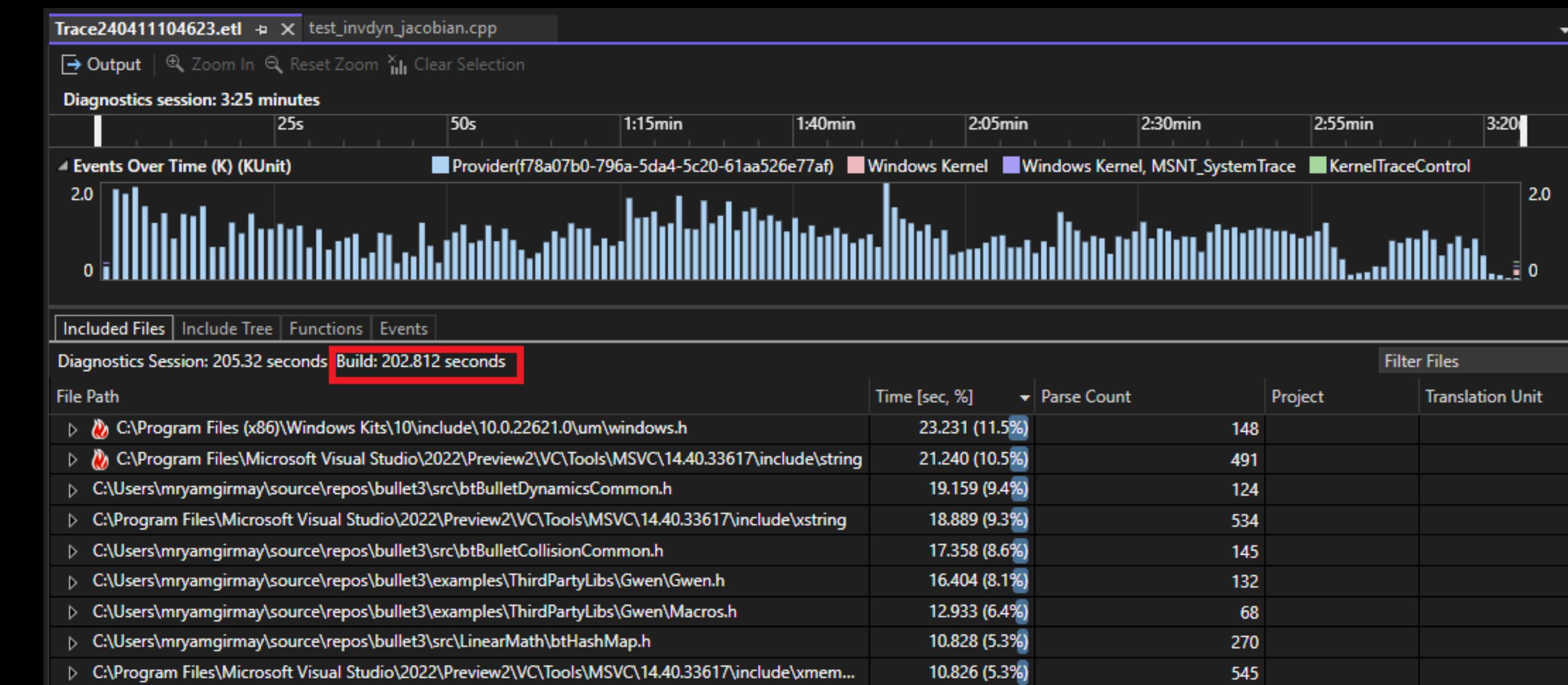
Percentage of repositories to analyze, %: **100**

Tools for C++ painpoints

-

#2 Build times

- Build insights
- Include cleanup
- Include Diagnostics



The screenshot shows a code editor with a diagnostic report overlaid. The report highlights a specific line of code in green, indicating it is a potential painpoint:

```
// Test of kinematic consistency: check if finite differences of ve
| // match positions
▶
✓#include <gtest/gtest.h>
#include "Bullet3Common/b3Random.h"

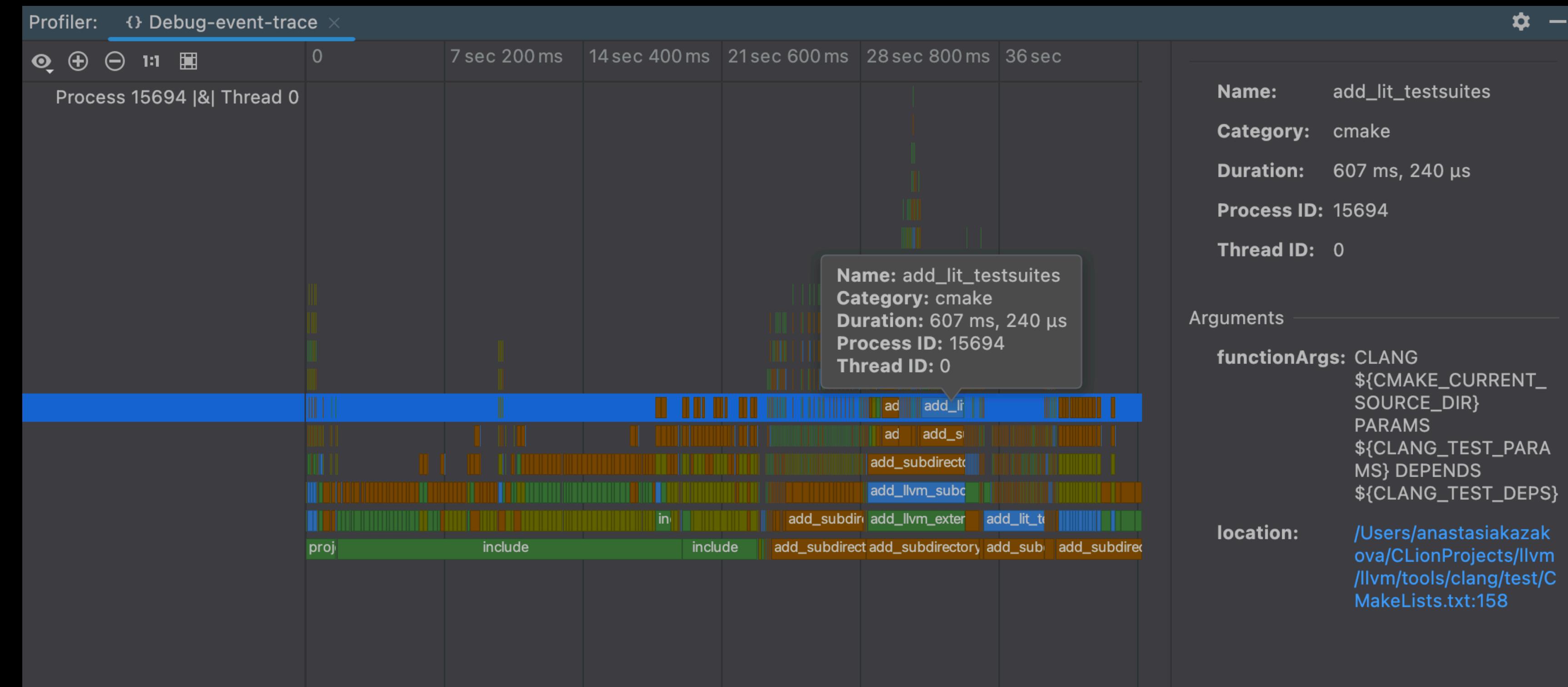
#include "CloneTreeCreator.hpp"
#include "CoilCreator.hpp"
#include "DillCreator.hpp"
#include "RandomTreeCreator.hpp"
#include "BulletInverseDynamics/MultiBodyTree.hpp"
#include <MultiBodyTreeCreator.hpp>
#include <Bullet3Common/b3Scalar.h>
#include <BulletInverseDynamics/details/IDLinearMathInterface.hpp>
```

Tools for C++ painpoints

#2 Build times

← CMake configuration

profiling



Tools for C++ painpoints

#3 CI pipelines

- test frameworks
- CI integrations
- variety of toolchains
- visual tools
- IDEs integrations

The screenshot shows a TeamCity Pipelines interface for a project named "Open Source Project". The current run is labeled "#15" and is shown as "Running" with a progress bar indicating it has been running for 20m (20 minutes) and is currently at 1m 23s. The pipeline consists of several steps:

- Frontend - Build_1: 1m 3s, Success, Tests passed: 12.
- Backend - Build: 2m, Success.
- Tests: 1m 45s, Success, Tests passed: 3.
- Frontend - Build_2: 1m 45s, Success.
- Publish artifacts: Success.
- Deployment: Step 1/2, currently in progress.

A detailed log for the "Backend - Deploy" step is visible on the right, showing the following entries:

```
10:19:05 Finalize build settings
... 10:19:05 > ⚡ Collecting changes in 1 VCS root 1s
... 10:19:05 > ⚡ Parallel tests: freeze 1s
10:19:05 The build is removed from the queue to be prepared for the start
... 10:19:05 ✘ ⚡ Step 1/1: simpleRunner (Command Line) 10s
10:19:05 Will use agent side checkout
10:19:05 + Clearing temporary directory: /mnt/agent/temp/buildTmp
10:19:05 - Container wrapper: prepare reusable container
10:19:05 Process exited with code 1
10:19:05 - Container wrapper: prepare reusable container
10:19:05 Process exited with code 1
10:19:05 Process exited with code 1
10:19:05 Process exited with code 1
10:19:05 The build is removed from the queue to be prepared for the start
```

Tools for C++ painpoints

#4 Managing CMake

The slide has a dark green background with a circular 'C++ now' logo in the top left. In the top right, it says 'CMake + Conan: 3 Years Later'. Below that, there's a portrait of Mateusz Pusz. The title 'A Motivating Example (Poll #2)' is at the top. A blue callout box contains the question 'What will be printed during the configuration phase?'. A black code block shows the following CMake script:

```
set(foo ON)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

At the bottom, there are logos for JET BRAINS, Bloomberg Engineering, and epam, along with the text 'C++Now 2021 - CMake + Conan: 3 years later'. On the far right, the number '8' is visible. In the bottom right corner of the slide area, the text 'CppNow.org' is displayed.

[CMake + Conan: 3 Years Later - Mateusz Pusz - \[CppNow 2021\]](#)

Tools for C++ painpoints

#4 Managing CMake

- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI

The screenshot shows a code editor with CMakeLists.txt open. The cursor is at the end of the line `target_link_libraries(ark`. A completion dropdown is open, showing suggestions for `arkanoid`, `arkanoidLib`, and `arkanoidTest`. Below the dropdown, the CMake documentation for `add_executable` is displayed, including its contents and a detailed description.

```
add_executable(arkanoid ${SOURCE_FILES})
target_link_libraries(ark| arkanoidLib)

macro(copy_dll tar
      add_custom_com
TARGET
COMMAND
$<TARGET_FILE:Qt${QT_VERSION}::${comp}>
$<TARGET_FILE_DIR:${target}>
```

13 | 13 add_ex
14 | 14 add_executable()
15 | 15 cuda_add_executable()
16 | 16 qt_add_executable()
17 | 17 \$<TARGET_FILE:Qt\${QT
18 | 18 \$<TARGET_FILE_DIR:\${
19 | 19)
20 | 20 endmacro()
21 |
22 | 22 if (WIN32)
23 |
24 | 24 copy_dll(arkanoid Core)
25 |
26 | 26 copy_dll(arkanoid Gui)
27 | 27 copy_dll(arkanoid Widgets)

add_executable

Contents

- add_executable
 - Normal Executables
 - Imported Executables
 - Alias Executables
 - See Also

Add an executable to the project using the specified source files.

Tools for C++ painpoints

-

#4 Managing CMake

- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI

The screenshot shows the CLion IDE interface. The top bar displays the project name "demoMore" and branch "master". The main editor window shows the file "CMakeLists.txt" with the following content:

```
22
23 if (WIN32)
24     copy_dll(arkanoid Core)
25     copy_dll(arkanoid Gui)
26     copy_dll(arkanoid Widgets)
27
28     if (MINGW) # hacky way to make things work; proper way would be building Qt with correct
29         add_custom_command(
30             TARGET arkanoid POST_BUILD
31             COMMAND ${CMAKE_COMMAND} -E copy_if_different
32             ${TARGET_FILE_DIR}:Qt${QT_VERSION}:Widgets>/libstdc++-6.dll
33             ${TARGET_FILE_DIR}:arkanoid>
34     )
35 endif()
36
37 add_subdirectory(3rdparty/googletest)
38 add_executable(arkanoidTest test.cpp)
39 target_link_libraries(arkanoidTest gtest_main arkanoidLib)
```

The line "endif()" at line 34 is highlighted in green. The line "add_subdirectory(3rdparty/googletest)" at line 37 is highlighted in blue. The line "target_link_libraries(arkanoidTest gtest_main arkanoidLib)" at line 39 is also highlighted in blue.

Below the editor, the CMake tool window is open, showing the following details:

- CMake
- add_subdirectory Debug_demo/CMakeLists.txt
- CLionFullDemo/CMakeLists.txt:36
- ENV
- CMakeCache.txt
- targets
- QT_VERSION = "6"
- SOURCE_FILES = "main.cpp;GameWidget.cpp;GameWidget.h"

The bottom status bar indicates the current frame: CLionFullDemo > Debug_demo > CMakeLists.txt.

Tools for C++ painpoints

#4 Managing CMake

- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI

The screenshot shows a dark-themed AI assistant interface. At the top, there are tabs for "AI Assistant", "All Chats", "New Chat", and a menu icon. In the main area, a user message "Me" asks "How to link openmp to CMake". The AI Assistant responds with a code snippet:

```
CMake
find_package(OpenMP)

if(OpenMP_CXX_FOUND)
    add_executable(your_executable your_source_files.cpp)
    target_link_libraries(your_executable PUBLIC
        ${OpenMP_CXX_LIBRARIES})
```

Below the code, a note says "7 more line(s)".

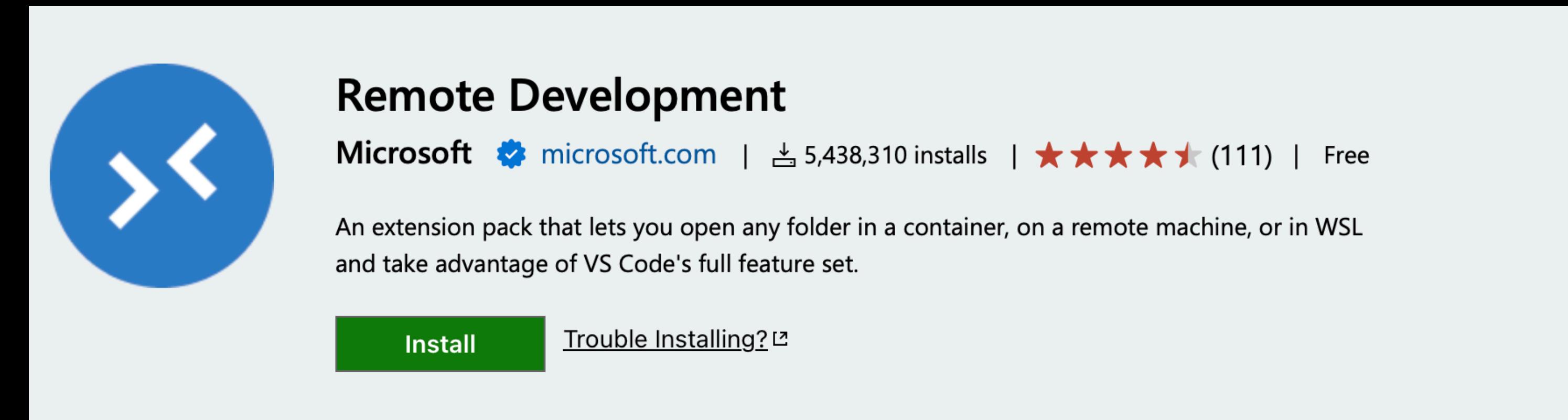
In the above example:

- `find_package(OpenMP)` will look for the OpenMP in your system. If CMake is able to find OpenMP, the boolean variable `OpenMP_CXX_FOUND` is set to true, else false.

Tools for C++ painpoints

#6 Setting up dev env

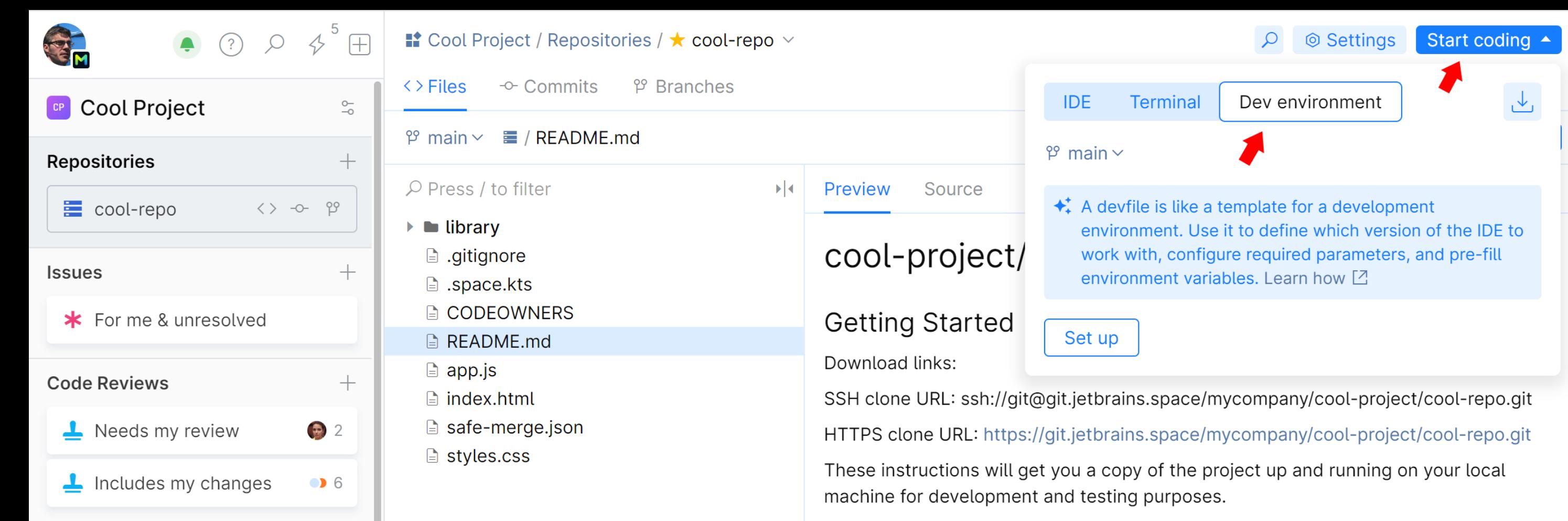
- start instantly
- onboard easily
- zero configuration time
- local & remote
- reproducible dev artifacts



Remote Development
Microsoft  microsoft.com | ↴ 5,438,310 installs | ★★★★★ (111) | Free

An extension pack that lets you open any folder in a container, on a remote machine, or in WSL and take advantage of VS Code's full feature set.

[Install](#) [Trouble Installing?](#)



The image shows the VS Code interface with the "Dev environment" tab selected in the top right corner. A red arrow points to this tab. The main view displays a GitHub repository named "cool-project" with a "cool-repo" branch. The "Files" tab is active, showing files like README.md, app.js, index.html, safe-merge.json, and styles.css. To the right, there's a "Getting Started" section with instructions for cloning the project using SSH or HTTPS, and a "Set up" button. The status bar at the bottom indicates "cool-project" and "main".

Tools for C++ painpoints

#8 Thread, memory, and type safety

Painkillers

- Data Flow Static Analysis
- Sanitizers
- Valgrind

Tools for C++ painpoints

DFA: lifetime safety

```
void sample() {  
    int *p = nullptr;  
    {  
        int x = 0;  
        p = &x;  
        std::cout << *p;  
    }  
    std::cout << *p;  
}
```

Local variable 'p' may point to memory which is out of scope :

int *p = nullptr



Tools for C++ painpoints

DFA: lifetime safety

```
const char *sample() {
    auto string = std::string("text");
    auto view = std::string_view(string);
    auto ptr = view.begin();
    return ptr;
}
```

The address of the local variable 'view' may escape the function :

const char *ptr = view.begin()  :

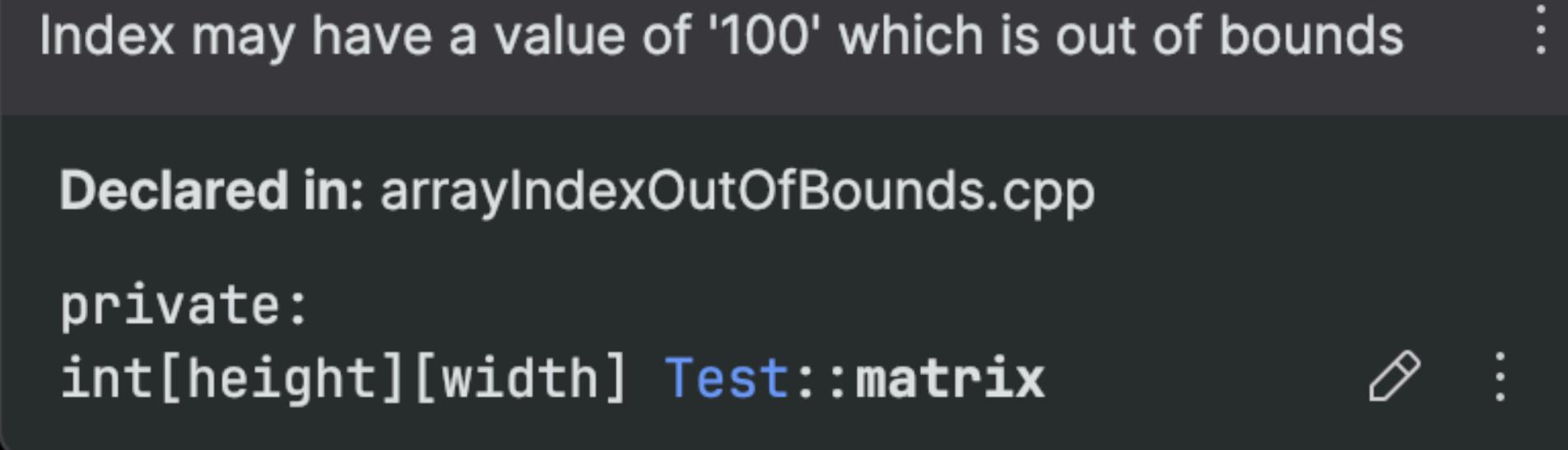
Tools for C++ painpoints

DFA: bounds safety

```
class Test {
    static const int width = 200;
    static const int height = 100;

    int matrix[height][width];

public:
    void test() {
        for (int i = 0; i < width; i++)
            for (int j = 0; j < height; j++)
                matrix[i][j] = 0;
    }
};
```



Tools for C++ painpoints

DFA: Use-after-delete/free

```
static void delete_ptr(int *ptr) {  
    delete ptr;  
}  
  
int handle_pointer() {  
    int* int_ptr = new int;  
  
    delete_ptr(int_ptr);  
    *int_ptr = 1;  
                              
    return 0;  
}
```

Local variable 'int_ptr' may point to deallocated memory :

int *int_ptr = new int



Tools for C++ painpoints

DFA: NULL dereferencing

```
class Deref {
    int* foo() {
        return nullptr;
    }

public:
    void bar() {
        int* buffer = foo();
        buffer[0] = 0;
    }
};
```

Pointer may be null

int *Deref::buffer = foo()

Tools for C++ painpoints

+

Makefiles

MSBuild

Debug

C++ modernisation

<i>Which of these do you find frustrating about C++ dev?</i>	<i>Major %</i>
Managing libraries my application depends on	45 %
Build times	43 %
Setting up a CI pipeline from scratch	30 %
Managing CMake projects	30 %
Concurrency safety: Races, deadlocks, performance bottlenecks	27 %
Setting up a dev env from scratch	26 %
Parallelism support	23 %
Managing Makefiles	20 %
Memory safety: Bounds safety issues	20 %
Memory safety: Use-after-delete/free	20 %
Debugging issues in my code	18 %
Managing MSBuild projects	16 %
Unicode, internationalization, and localization	16 %
Security issues: disclosure, vulnerabilities, exploits	12 %
Type safety: Using an object as the wrong type	12 %
Memory safety: Memory leaks	12 %
Moving existing code to the latest language standard	9 %

Tools for C++ painpoints

Tools build a **safe environment** for C++ developers
where they can simply focus on **ideas** and **code**
them.

C++ Painkillers stories

Story #2

“Your data w/o your tools”

Your data w/o your tools

```
ide1: BM-DMA at 0xc008-0xc00f, BIOS settings: hdc:pio, hdd:pio
ne2k-pci.c:v1.03 9/22/2003 D. Becker/P. Gortmaker
    http://www.scyld.com/network/ne2k-pci.html
hda: QEMU HARDDISK, ATA DISK drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
hdc: QEMU CD-ROM, ATAPI CD/DVD-ROM drive
ide1 at 0x170-0x177,0x376 on irq 15
ACPI: PCI Interrupt Link [LNKC] enabled at IRQ 10
ACPI: PCI Interrupt 0000:00:03.0[A] -> Link [LNKC] -> GSI 10 (level, low) -> IRQ
10
eth0: RealTek RTL-8029 found at 0xc100, IRQ 10, 52:54:00:12:34:56.
hda: max request size: 512KiB
hda: 180224 sectors (92 MB) w/256KiB Cache, CHS=178/255/63, (U)DMA
hda: set_multmode: status=0x41 { DriveReady Error }
hda: set_multmode: error=0x04 { DriveStatusError }
ide: failed opcode was: 0xef
hda: cache flushes supported
    hda1
hdc: ATAPI 4X CD-ROM drive, 512kB Cache, (U)DMA
Uniform CD-ROM driver Revision: 3.20
Done.
Begin: Mounting root file system... .
/init: /init: 151: Syntax error: 0xforce=panic
Kernel panic - not syncing: Attempted to kill init!
```

C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI

Tools and language evolution

SG15, Tooling

Topics related to creation of developer tools for standard C++, including but not limited to modules and package management.

Tools and language evolution

Many new C++ language features now
start with **Clang-based implementation**

Tools and language evolution

C++20

Modules

Modules (since C++20)

Most C++ projects use multiple translation units, and so they need to share [declarations](#) and [definitions](#) across those units. The usage of [headers](#) is prominent for this purpose, an example being the [standard library](#) whose declarations can be provided by [including the corresponding header](#).

Modules are a language feature to share declarations and definitions across translation units. They are an alternative to some use cases of headers.

Modules are orthogonal to [namespaces](#).

```
// helloworld.cpp
export module helloworld; // module declaration

import <iostream>; // import declaration

export void hello() // export declaration
{
    std::cout << "Hello world!\n";
}
```

```
// main.cpp
import helloworld; // import declaration

int main()
{
    hello();
}
```

Tools and language evolution: Modules

CMake + VS generator

```
add_executable(ModuleSample a.ixx main.cpp)
```

Tools and language evolution: Modules

CMake +

GCC/Clang compiler flags

```
function(add_module name)
    file(MAKE_DIRECTORY ${PREBUILT_MODULE_PATH})
    add_custom_target(${name}.pcm
        COMMAND
            ${CMAKE_CXX_COMPILER}
            -std=gnu++20
            -x c++
            -fmodules
            -c
            ${CMAKE_CURRENT_SOURCE_DIR}/${ARGN}
            -Xclang -emit-module-interface
            -o ${PREBUILT_MODULE_PATH}/${name}.pcm
    )
endfunction()
```

Tools and language evolution: Modules

Natively in CMake 3.25
[\(CMake examples\)](#)

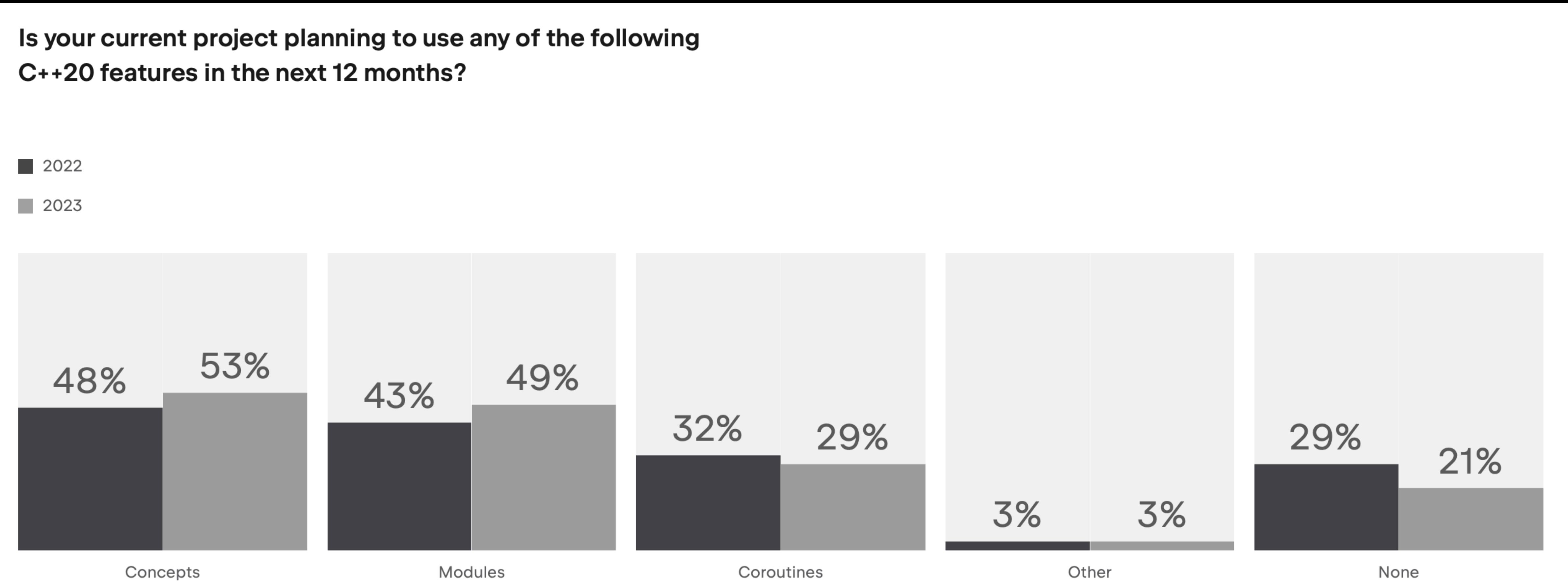
```
add_executable(simple)
target_sources(simple
    PRIVATE
        main.cxx
    PRIVATE
        FILE_SET CXX_MODULES
        BASE_DIRS
            "${CMAKE_CURRENT_SOURCE_DIR}"
        FILES
            importable.cxx)
target_compile_features(simple PUBLIC cxx_std_20)

add_test(NAME simple COMMAND simple)
```

Tools and language evolution: Modules

CMake 3.26: support for modules was added
into file-api

Tools and language evolution: Modules



Tools and language evolution: Contracts

C++26 ?

Contracts

```
#include <cassert>

void f(int p) {
    assert(p > 7);

    if (p == 6) {
        //d Condition is always false ...
    } else
        //d Simplify 'if' statement ⌂ ↵ More actions... ⌂ ↵
    }
}
```

Tools and language evolution: Reflection

C++26 ?

Reflection



Tools and language evolution: Reflection

C++26 ?

Reflection

```
struct TU_Ticket {  
    static consteval int next() {  
        // ...  
    }  
};  
  
void a() {  
    TU_Ticket::next();  
}  
  
std::integral_constant<int, TU_Ticket::next()> b();
```

<https://lists.isocpp.org/sg7/2024/02/0480.php>

Tools and language evolution

P2656

C++ Ecosystem International Standard

Users should be able to **mix and match their preferred build systems, compilers, linkers, package managers, static analyzers, runtime analyzers, debuggers, profilers, etc.** without needing the tools to have vendor specific knowledge of each other. Vendors should be able to **focus on direct tool improvements**, rather than figuring out how to interact with yet another proprietary interface.

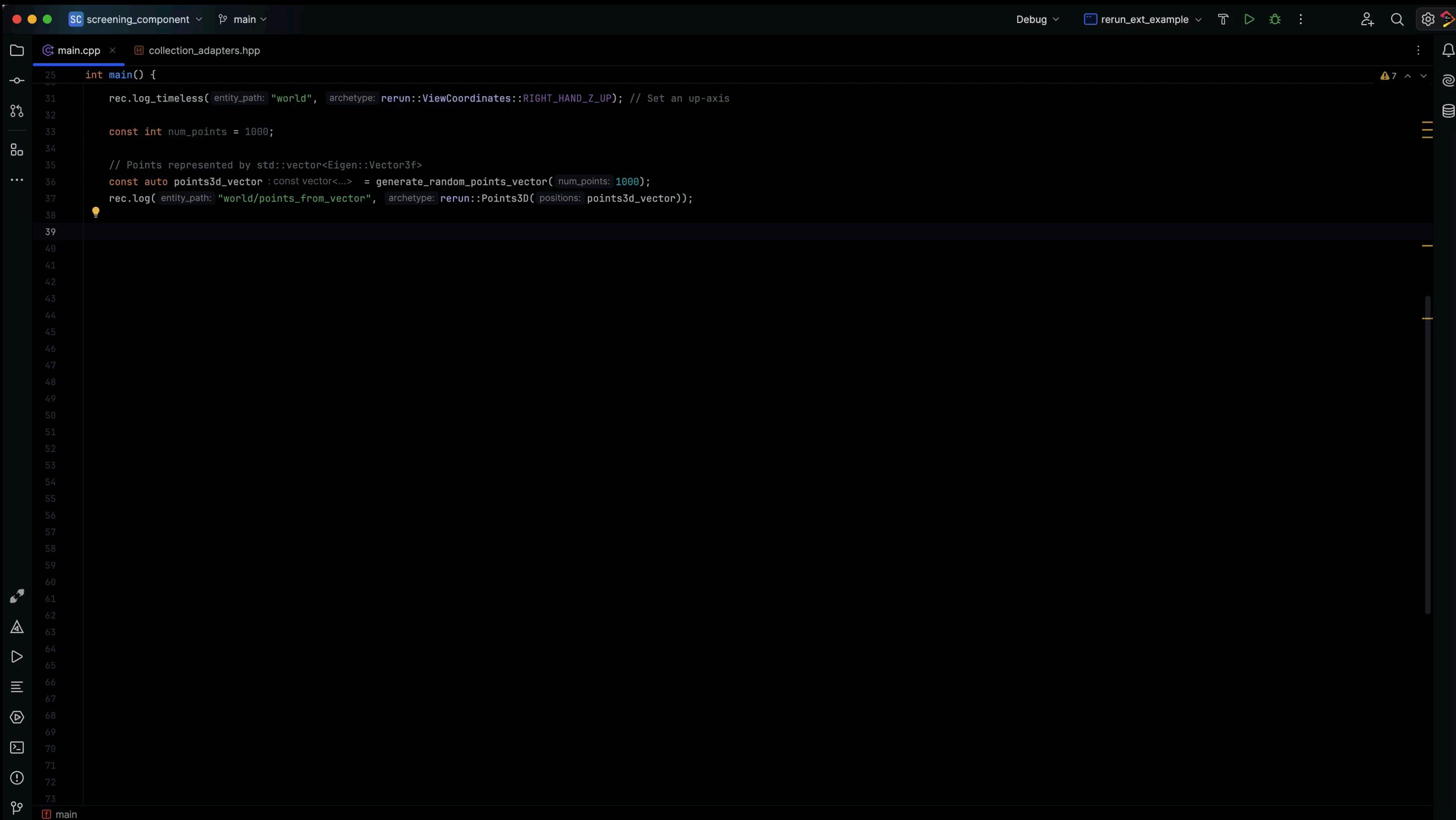
1. Definitions.
2. Build System \Leftrightarrow Package Manager Interoperation.
3. Minimum set of recognized file extensions.
4. Tool introspection.
5. Portable diagnostics format via SARIF.
6. Command line portability.

C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI

Era of AI

Writing
context-aware
code



The screenshot shows a dark-themed code editor interface. At the top, there's a toolbar with icons for file operations, search, and settings. The title bar indicates the project is "screening_component" and the file is "main". The main area displays the "main.cpp" file. The code is as follows:

```
int main() {
    rec.log_timeless(entity_path: "world", archetype: rerun::ViewCoordinates::RIGHT_HAND_Z_UP); // Set an up-axis

    const int num_points = 1000;

    // Points represented by std::vector<Eigen::Vector3f>
    const auto points3d_vector :const vector<...> = generate_random_points_vector( num_points: 1000);
    rec.log( entity_path: "world/points_from_vector", archetype: rerun::Points3D( positions: points3d_vector));
}
```

The code uses the Rerun library for logging data. The sidebar on the left shows the file structure and other files like "collection_adapters.hpp". The bottom navigation bar includes icons for file, edit, run, and terminal.

Era of AI

Improving the code



```
main.cpp x : ✓
1 #include <array>
2 #include <iostream>
3
4 ▷ int main() {
5     constexpr std::array Numbers{1, 3, 5, 6, 7, 9, 10, 11, 13};
6
7     size_t oddNumberCount = 0;
8     for (auto &number : const int & : Numbers) {
9         if (number % 2) {
10             oddNumberCount++;
11         }
12     }
13
14     std::cout << oddNumberCount;
15
16     return 0;
17 }
```

Era of AI

Summarizing for
VCS or documentation

```
Collisions.cpp ×

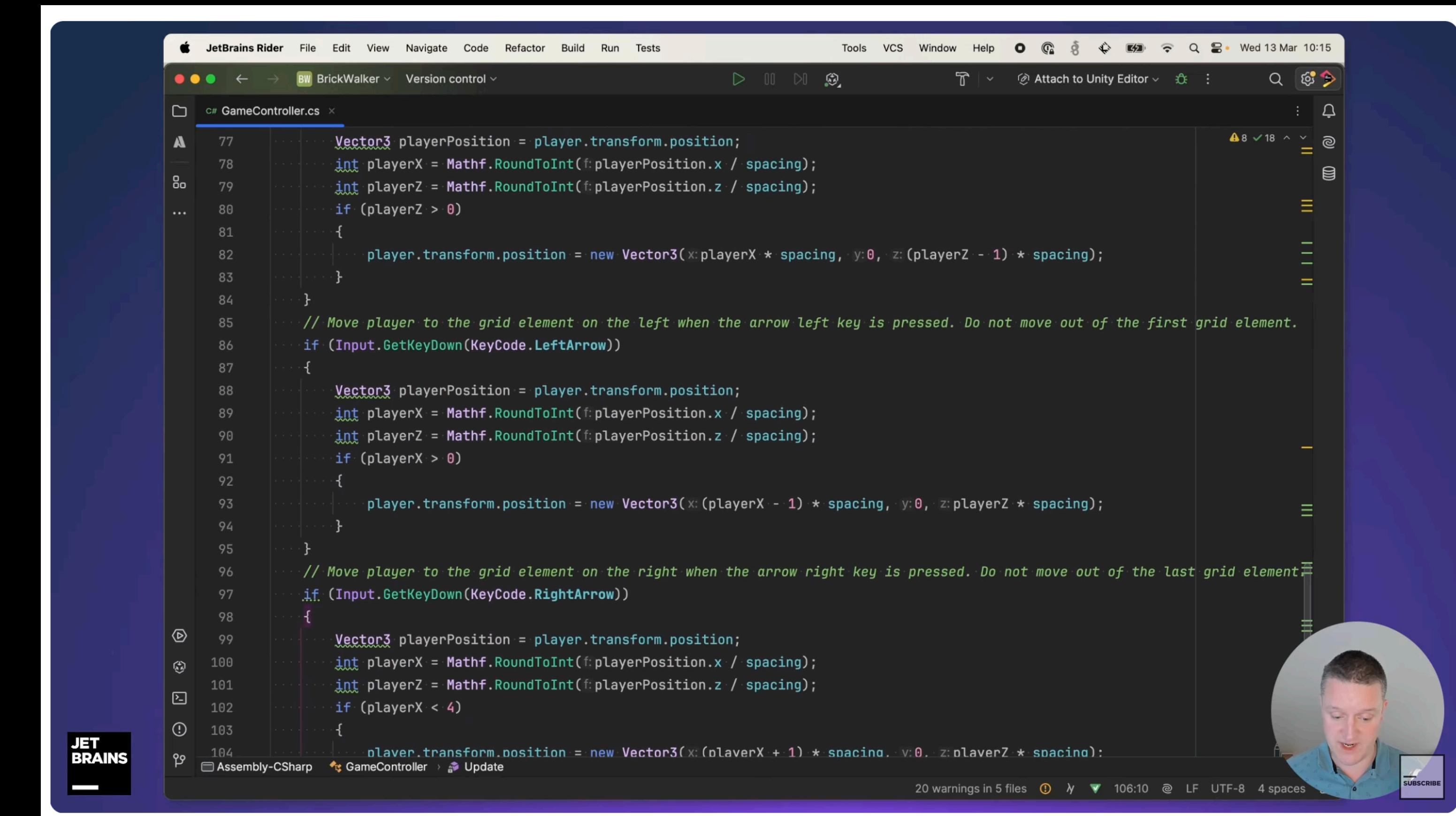
7     if (ballBB.top() < bounding.top()) return Top;
8     return None;
9 }
10
11 → CollisionType getCollisionWithBrick(const Ball& b, const QRectF& bri
12     auto ballBB:QRectF = b.aabb();
13     auto inters:QRectF = ballBB.intersected(brick);
14     if (inters.isEmpty()) {
15         return None;
16     }
17     else {
18         if (inters.width() > inters.height()){
19             if(inters.center().y()
```

Era of AI: real-world examples

How Copilot is being used by the Time Travel Debugging team for repetitive C++ coding © Microsoft

Can I script a Unity Game with AI Assistant in under 15 minutes? ©

Maarten Balliauw, JetBrains



The screenshot shows the JetBrains Rider IDE interface with the file 'GameController.cs' open. The code is written in C# and handles player movement logic. It uses Vector3 and Mathf.RoundToInt to calculate player position based on transform.position and spacing. It includes logic for moving the player left and right within a grid, ensuring they stay within bounds. The IDE has a dark theme, and the code editor shows syntax highlighting for C# keywords and variables.

```
77     Vector3 playerPosition = player.transform.position;
78     int playerX = Mathf.RoundToInt(playerPosition.x / spacing);
79     int playerZ = Mathf.RoundToInt(playerPosition.z / spacing);
80     if (playerZ > 0)
81     {
82         player.transform.position = new Vector3(playerX * spacing, 0, (playerZ - 1) * spacing);
83     }
84 }
85 // Move player to the grid element on the left when the arrow left key is pressed. Do not move out of the first grid element.
86 if (Input.GetKeyDown(KeyCode.LeftArrow))
87 {
88     Vector3 playerPosition = player.transform.position;
89     int playerX = Mathf.RoundToInt(playerPosition.x / spacing);
90     int playerZ = Mathf.RoundToInt(playerPosition.z / spacing);
91     if (playerX > 0)
92     {
93         player.transform.position = new Vector3((playerX - 1) * spacing, 0, playerZ * spacing);
94     }
95 }
96 // Move player to the grid element on the right when the arrow right key is pressed. Do not move out of the last grid element.
97 if (Input.GetKeyDown(KeyCode.RightArrow))
98 {
99     Vector3 playerPosition = player.transform.position;
100    int playerX = Mathf.RoundToInt(playerPosition.x / spacing);
101    int playerZ = Mathf.RoundToInt(playerPosition.z / spacing);
102    if (playerX < 4)
103    {
104        player.transform.position = new Vector3((playerX + 1) * spacing, 0, playerZ * spacing);
105    }
106 }
```

Era of AI: real-world examples

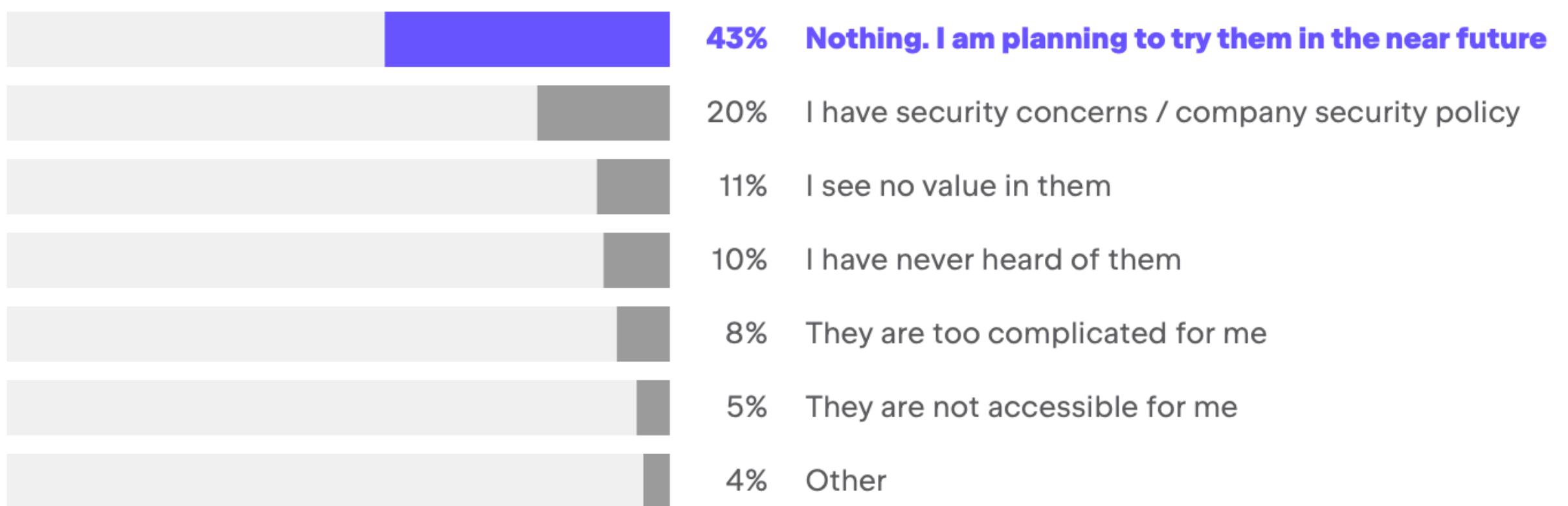
GDC: Are generative AI tools being used at your company or department?

49% Generative AI tools are currently being used in their company

31% personally use Generative AI tools

21% of AAA developers said their companies have banned the use of such tools. Compared to **9%** indie.

What stops you from learning about generative AI tools?



*This question was shown only to the developers who chose “None” in the previous question.

C++ Painkillers

1. Stories about the tools
2. Why this talk
3. Vitamin, Painkiller, or Cure?
4. Where and when do tools help
5. C++ toolability
6. Tools for C++ painpoints
7. Start thinking about tools earlier
8. Era of AI
9. Questions