How To Address 7 Major C++ Pain Points with Tools 🛠️

Anastasia Kazakova, JetBrains

NDC TechTown 2024

About me

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

The Dutch C++ Group

cppunderthesea.nl
11 October 2024
Breda, Netherlands

Anastasia Kazakova
Head of Marketing and BizDev

JETBRAINS

Topics to cover
-

1. Painpoints and painkillers
2. Tools DO help
3. C++ toolability is hard
4. Tools for C++ painpoints
5. Era of AI

Tell us a joke about C++ and its tooling!

Why do C++ developers make great philosophers?

Because after wrestling with the lack of tooling for hours,
they start questioning the meaning of life,
memory leaks,
and whether their compiler even exists!

Do we have a problem?

–

Language

Tooling

Other

Tools are on top for years!

Which of these do you find frustrating about C++ dev?	Major %
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%

Candy, Vitamin, Painkiller

—

“We throw away the candy.
We look at vitamins.
We really like painkillers.”
© Kevin Fong, venture capitalist

Cure, Vitamin, Painkiller

–

Painkiller = a quick-fix, a tool

Vitamin = guidelines, code styles, best practices

Cure = a language feature

Simple enough to learn as a **first-time** programmer

–

…It even looks way more complicated than using C++…

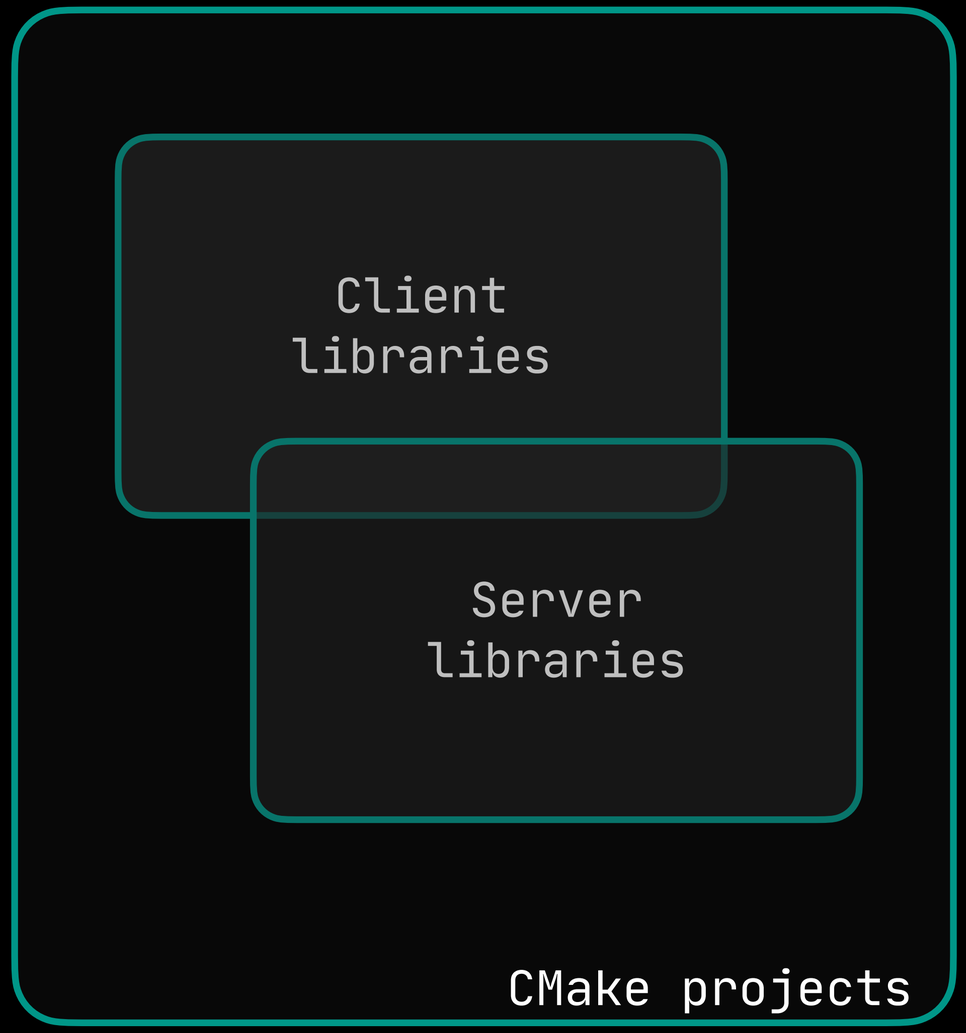
Verse

```
race:
    # Move back to its starting position.
    set NavResultGoToNext = Navigatable.NavigateTo(NavTargetEnd)

    # Sets NPC to look at its previous position which will make it walk backwards.
    # This is meant to show the utility of the focus interface.
    Focus.MaintainFocus(GoToPoint)

    if(ShowAIDebug?):
        DrawDebugLookAt(Character, GoToPoint)
    else:
        Sleep(Inf)
```

Story time:

“Dependency validation with NetBeans”

Dependency validation with NetBeans




```
$ cmake ..
```

CMake configures the project

NetBeans

```
$ cmake --build
```

CMake builds an already-generated project binary tree

Tools DO help.

Practical examples for C++

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

—

```
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_CASE(...)                                       \
    BOOST_TEST_INVOKE_IF_N_ARGS( 1,                                      \
        BOOST_AUTO_TEST_CASE_NO_DECOR,                                   \
        BOOST_AUTO_TEST_CASE_WITH_DECOR,                                 \
         __VA_ARGS__)                                                    \
```

Replacement:

```
struct test_equal_word : public BOOST_AUTO_TEST_CASE_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 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

1

```
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
```

2

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

Declared in: ostream

```
namespace std

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

"operator<<" on cppreference.com ↗

3

```
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

CppCon 2018: Richard Powell “Named Arguments from Scratch”

Tools help: named parameters

#1 Distinguish parameters of the same type

```
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) {
13
```

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

#1 auto variables

#2 lambda types

#3 deduced return types

#4 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( t: 3,  u: 0);
    auto op1 : long  = doOperation( t: 3L,  u: 0);
    auto op2 : double  = doOperation( t: 3.0,  u: 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;
}
```

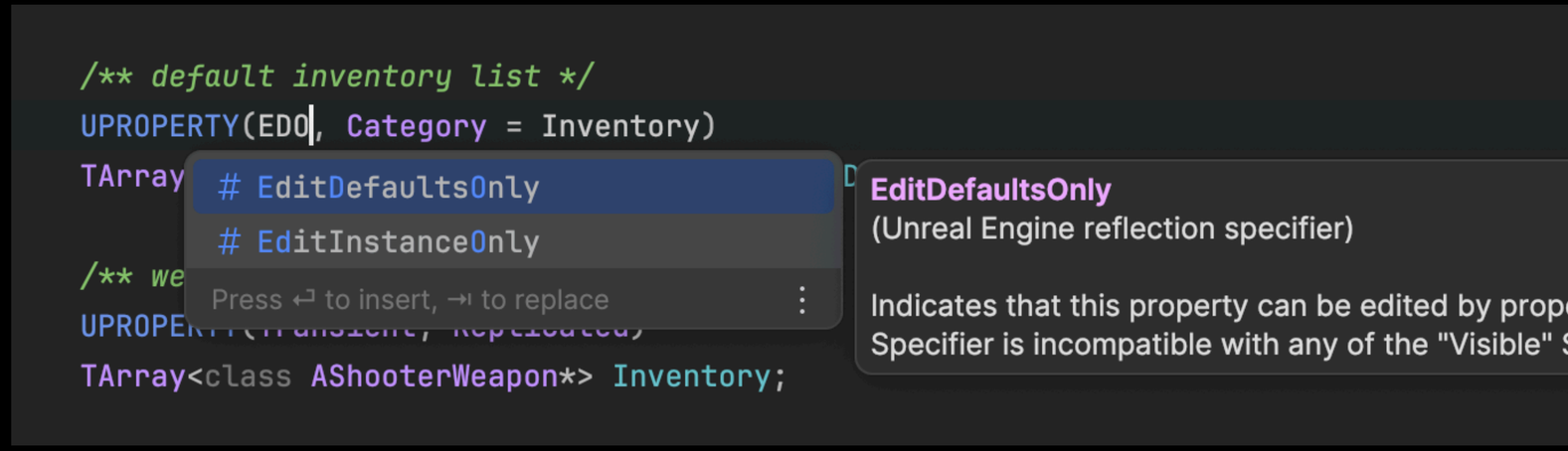
Tools help: with DSLs

—

#1 Unreal Engine

reflection

macros-based dialect



#2 Qt signals

Tools help: with guidelines

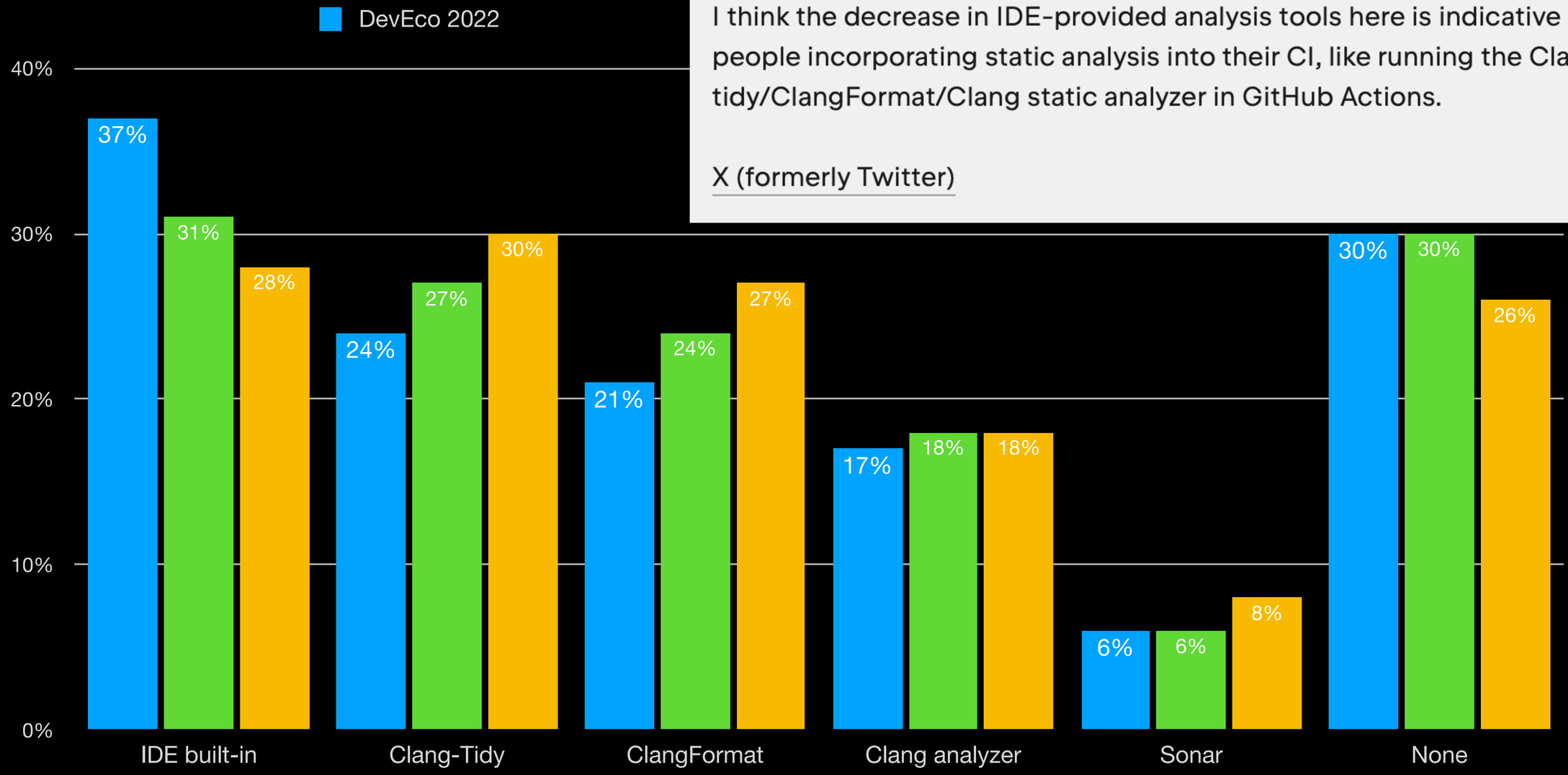
—

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	
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-traits	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-uncaught-exceptions	Yes
modernize-use-using	Yes

Tools help: with guideline
–



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.

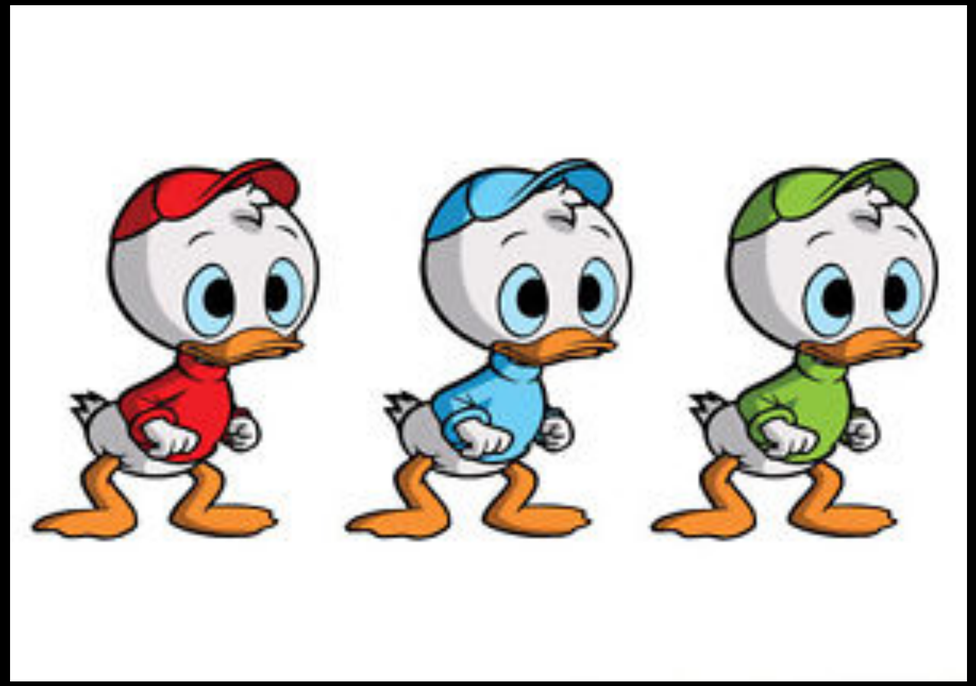
X (formerly Twitter)

C++ and Toolability:

A story about friendship?

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

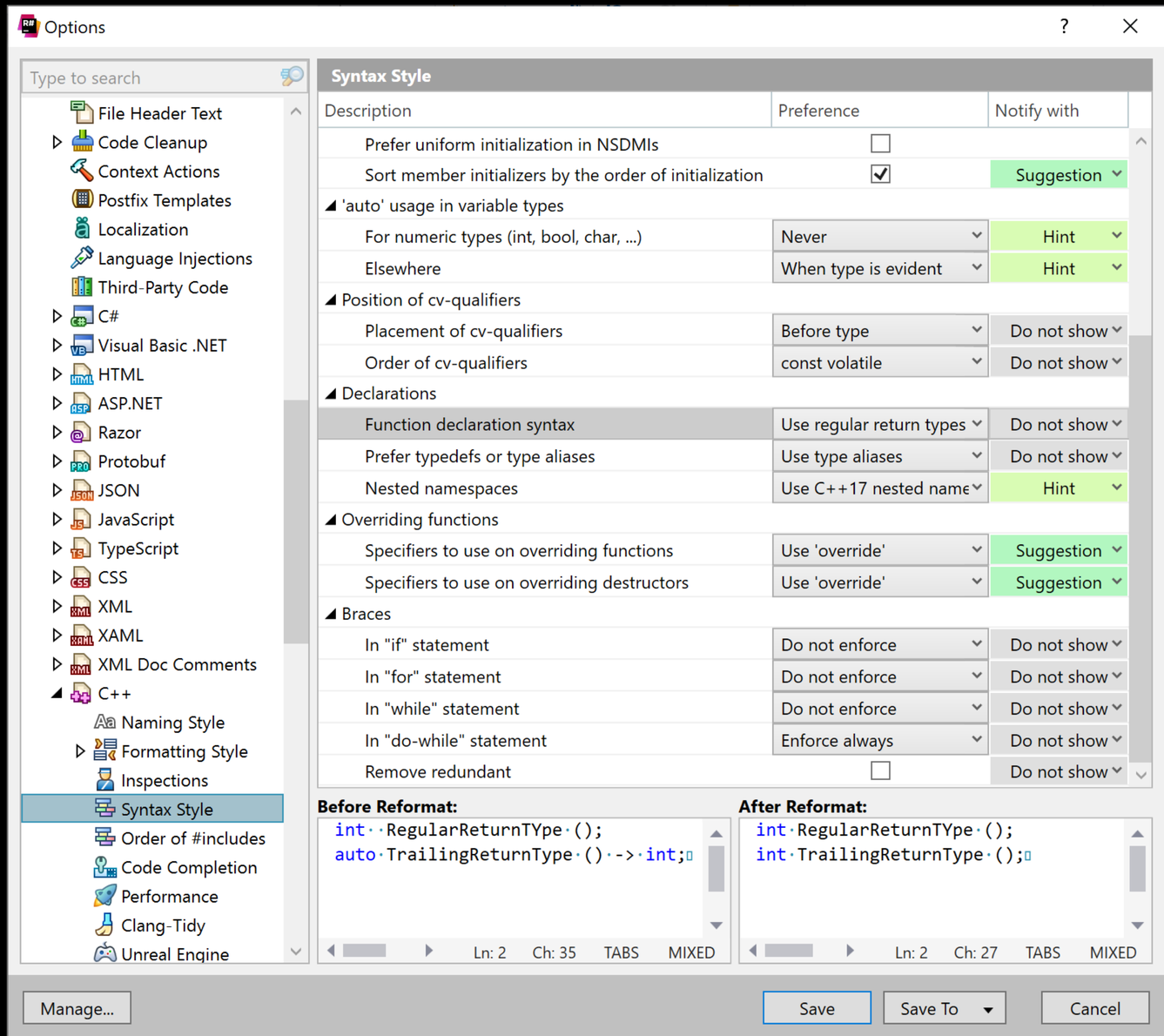
—

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

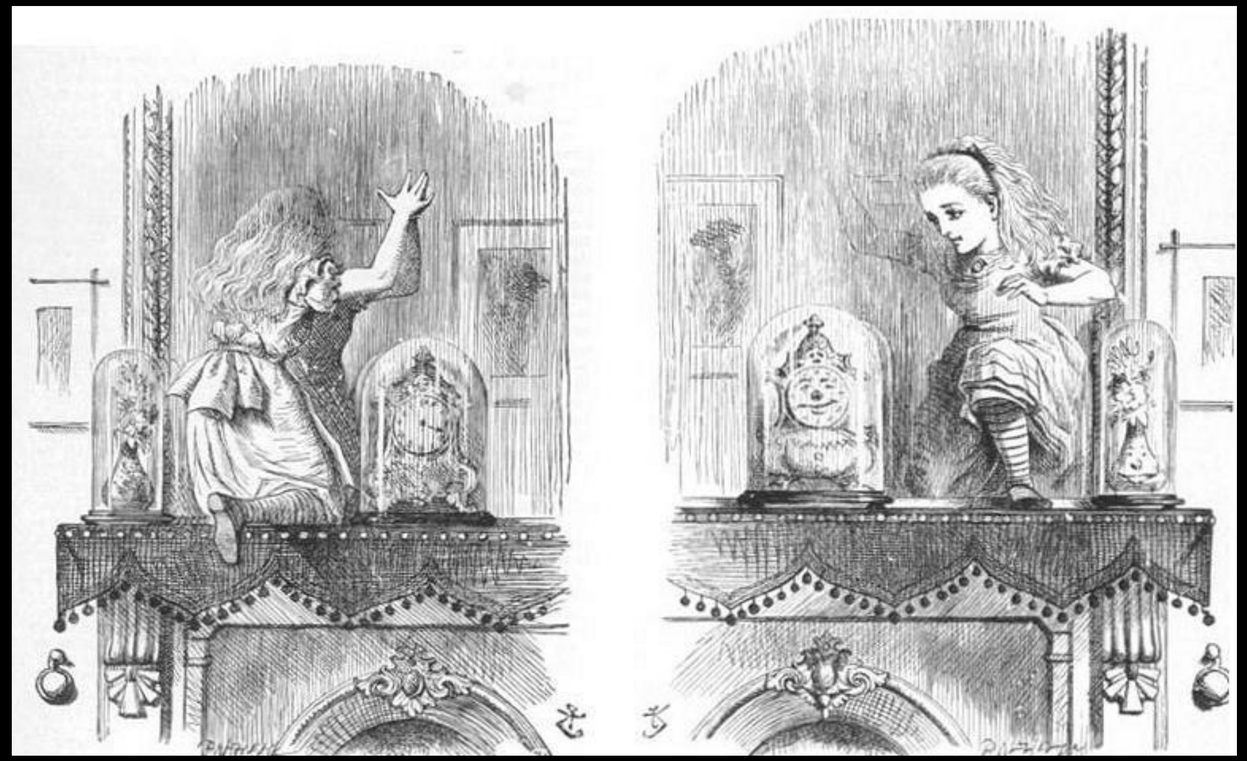
```
int(x), y, a, b, c, d, e, f, *z;

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

C++ toolability

To parse C++

we need to distinguish

types from non-types

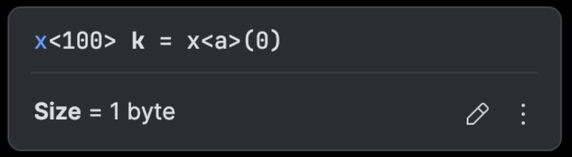


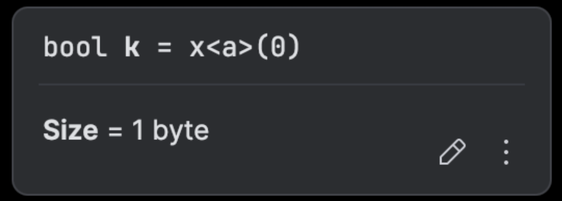
C++ toolability
–

Resolve depends on:

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

```cpp
//foo.h
#ifdef 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);
}
```

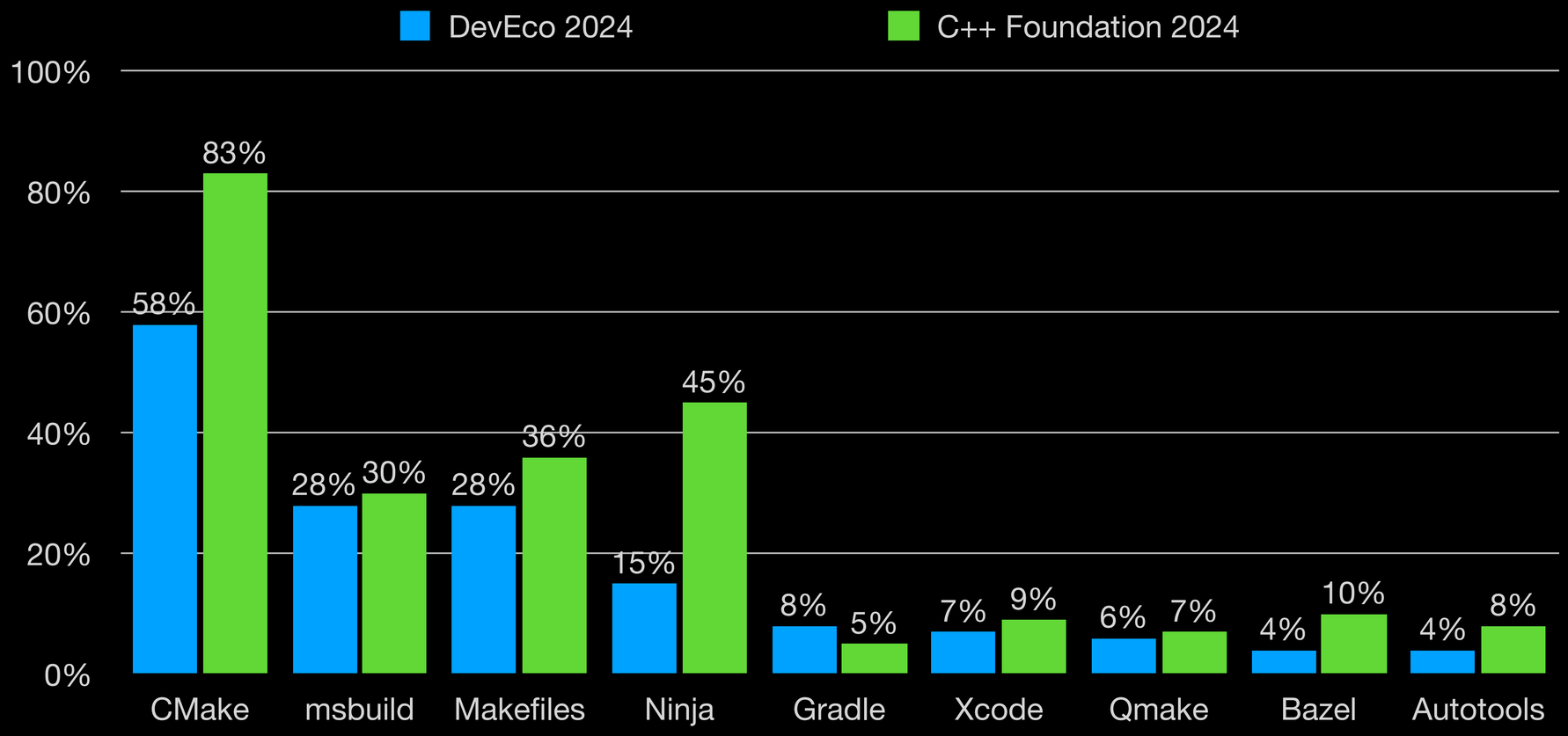




C++ toolability

-

- CMake!
- msbuild for GameDev
- Makefiles for Embedded
- Bazel for monorepos



C++ and Tools:

Assign a painkiller

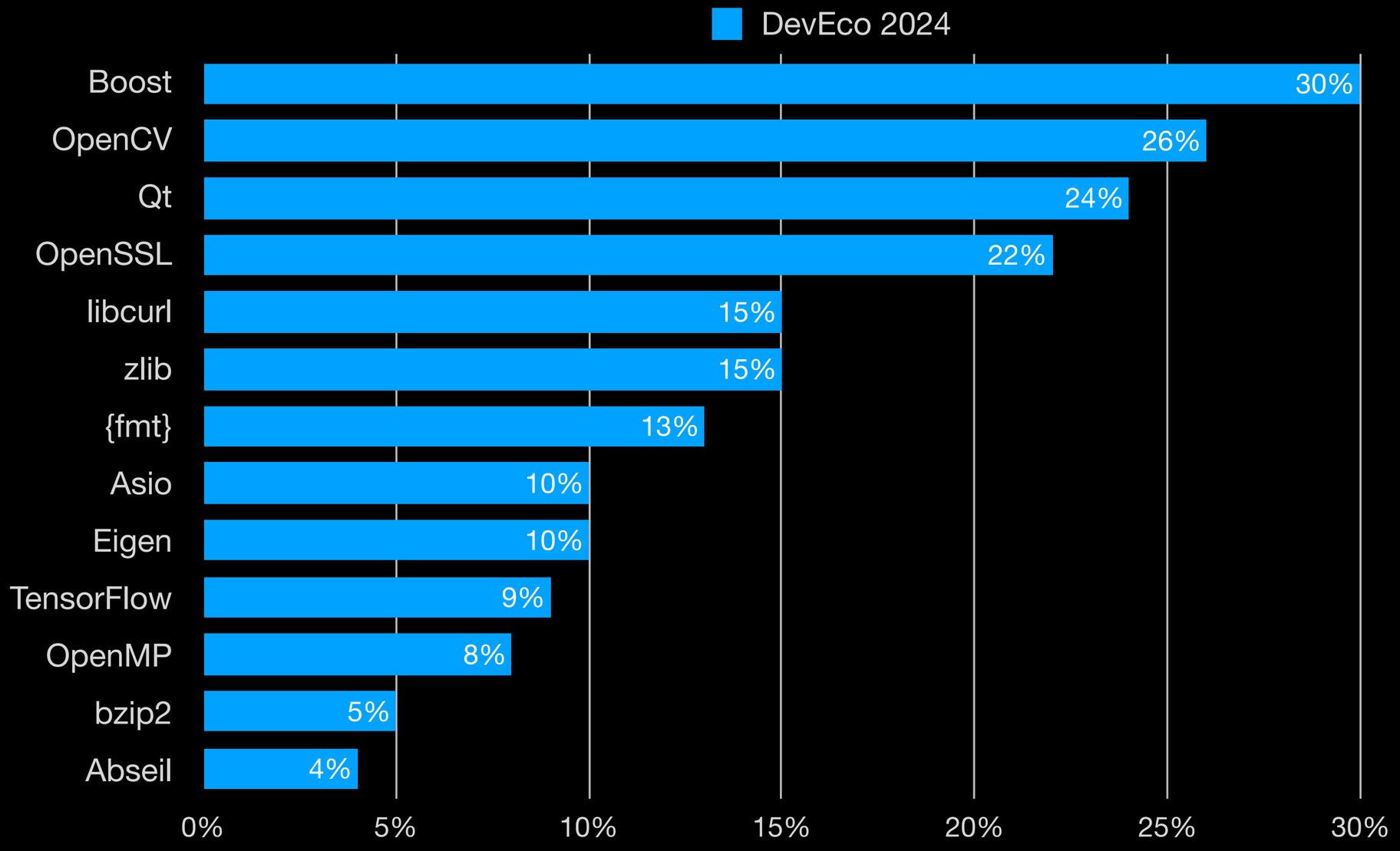
Tools for C++ painpoints

–

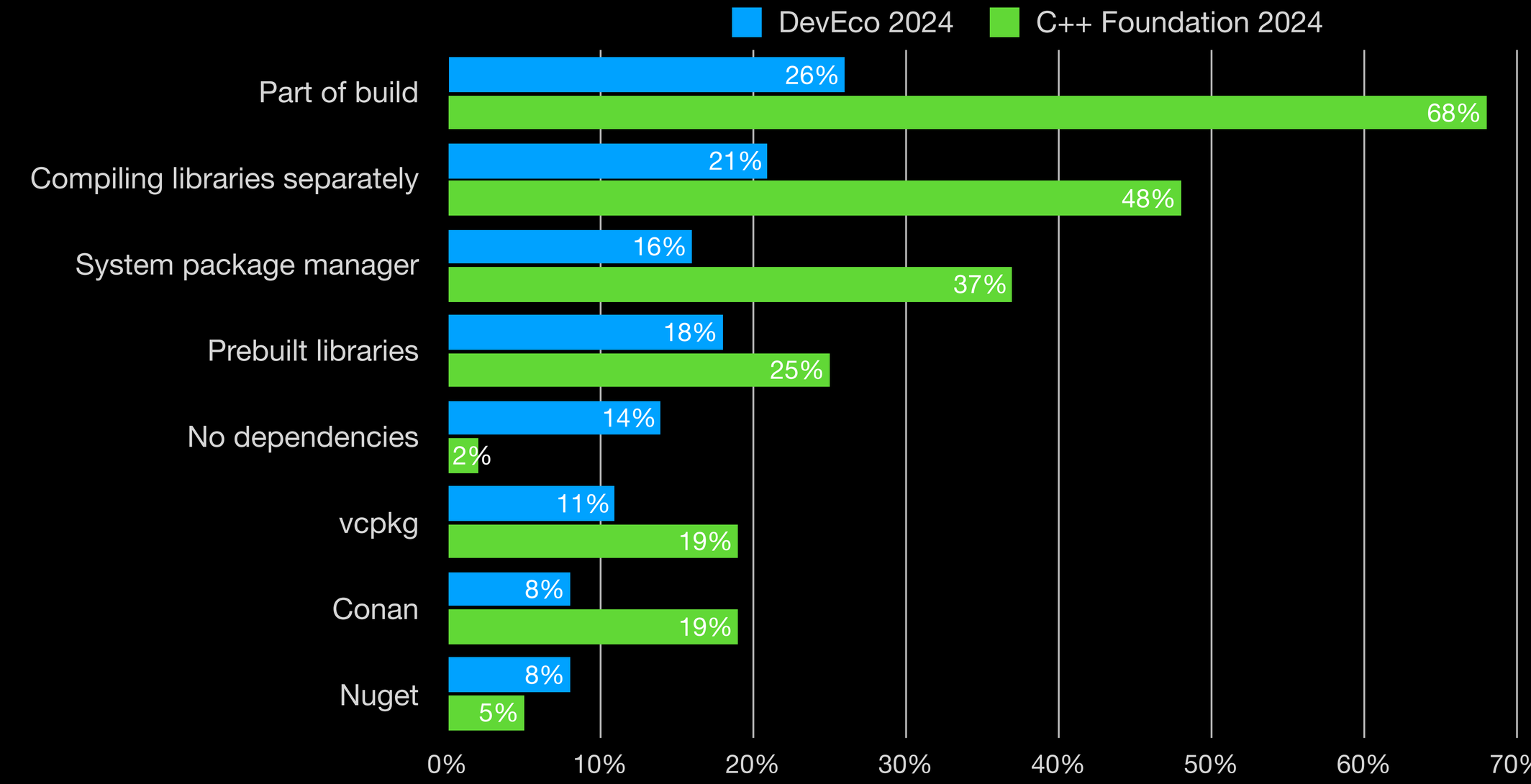
Which tools to use?

Which of these do you find frustrating about C++ dev?	*Major %*
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%

Painpoints: #1 Managing libraries



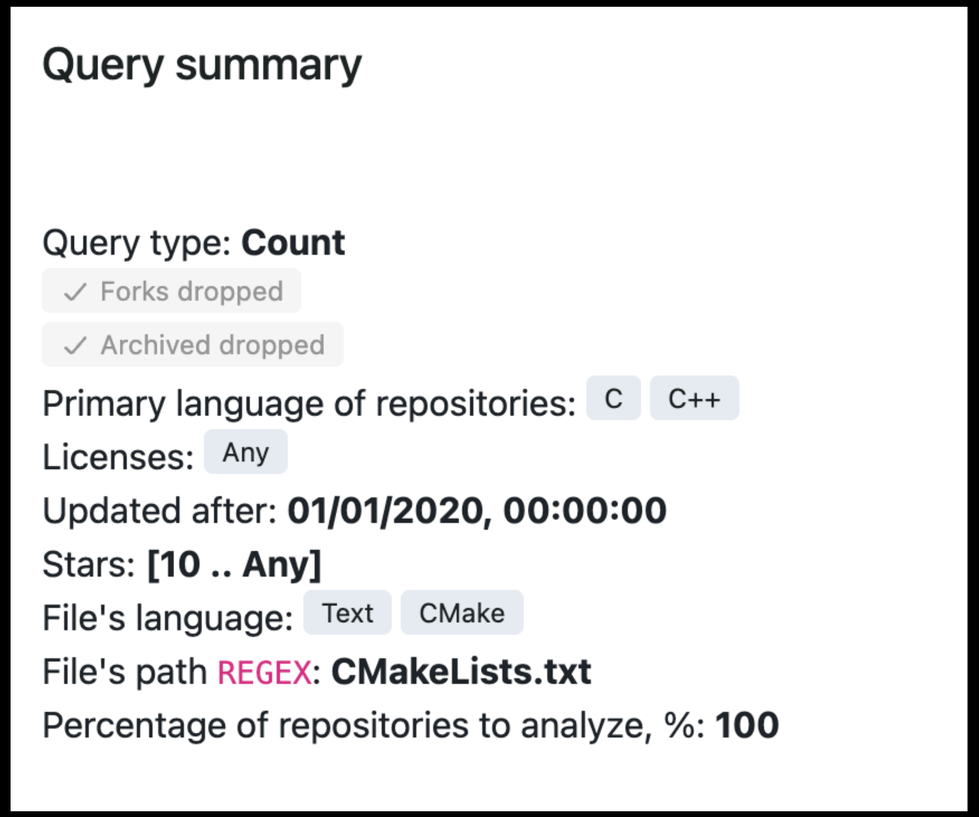
Painpoints: #1 Managing libraries



Painpoints: #1 Managing libraries
–

40% repos with

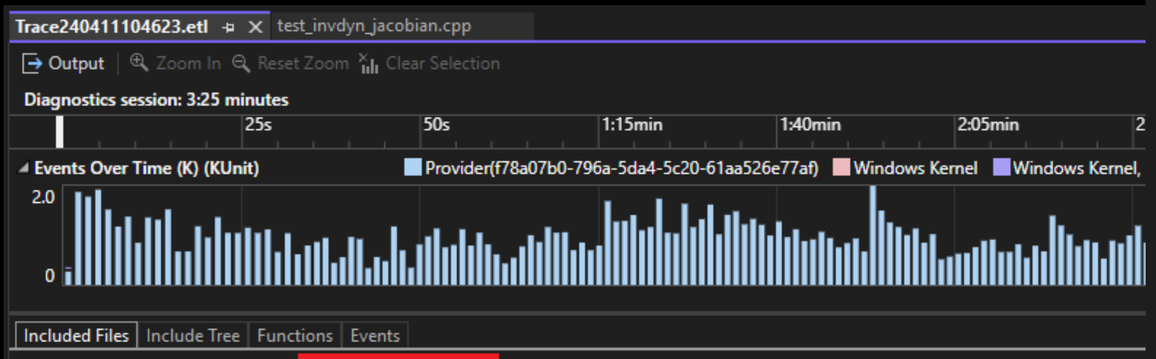
CMakeLists.txt inside



Painpoints: #2 Build times
—

- Build insights
- Include cleanup
- Include Diagnostics

```
// Test of kinematic consistency: check if finite differences of
// 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
#include <BulletInverseDynamics/IDConfig.hpp>
```

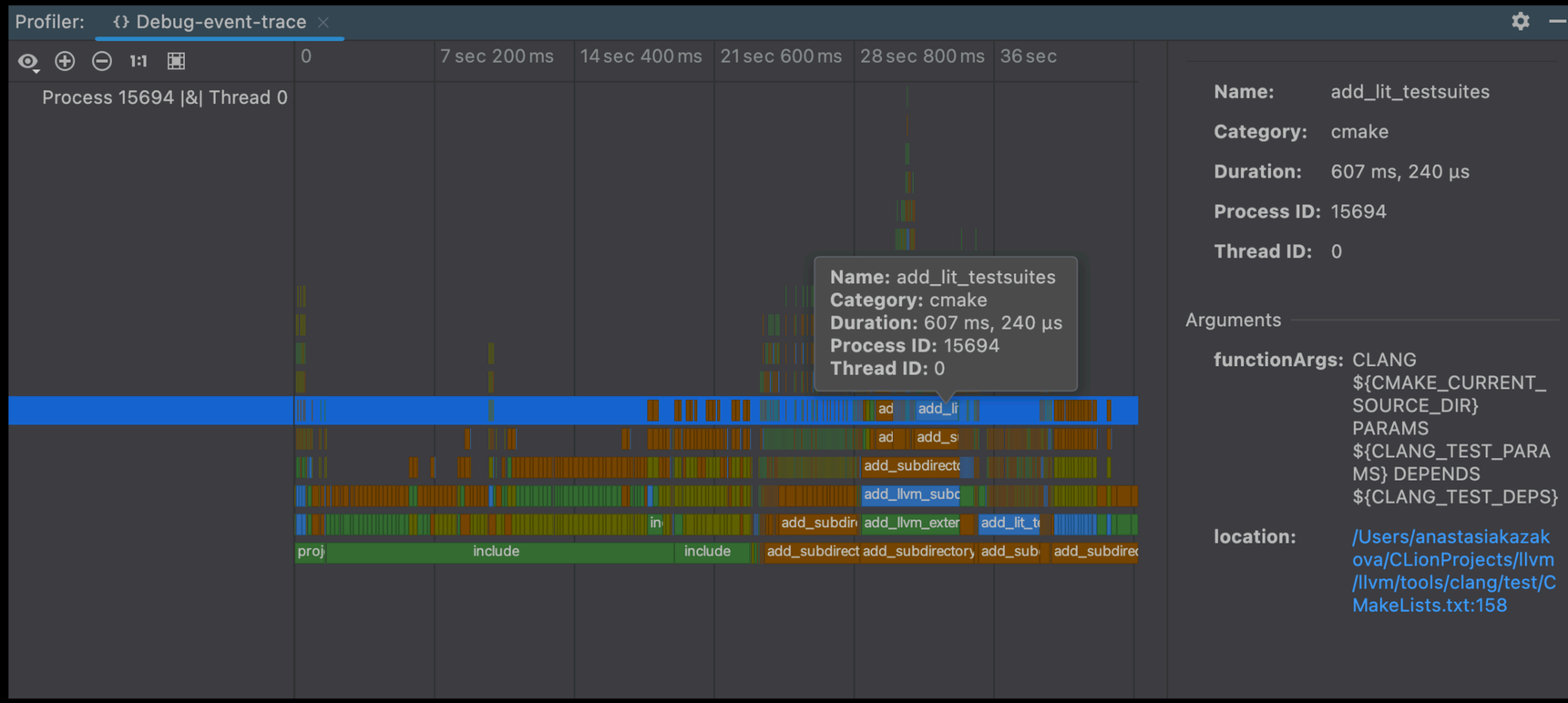


Diagnostics Session: 205.32 seconds Build: 202.812 seconds

File Path	Time [sec, %]	Parse Count	Project	Translation Unit
C:\Program Files (x86)\Windows Kits\10\include\10.0.22621.0\um\windows.h	23.231 (11.5%)	148		
C:\Program Files\Microsoft Visual Studio\2022\Preview2\VC\Tools\MSVC\14.40.33617\include\string	21.240 (10.5%)	491		
C:\Users\mryamgirmay\source\repos\bullet3\src\btBulletDynamicsCommon.h	19.159 (9.4%)	124		
C:\Program Files\Microsoft Visual Studio\2022\Preview2\VC\Tools\MSVC\14.40.33617\include\xstring	18.889 (9.3%)	534		
C:\Users\mryamgirmay\source\repos\bullet3\src\btBulletCollisionCommon.h	17.358 (8.6%)	145		
C:\Users\mryamgirmay\source\repos\bullet3\examples\ThirdPartyLibs\Gwen\Gwen.h	16.404 (8.1%)	132		
C:\Users\mryamgirmay\source\repos\bullet3\examples\ThirdPartyLibs\Gwen\Macros.h	12.933 (6.4%)	68		
C:\Users\mryamgirmay\source\repos\bullet3\src\LinearMath\btHashMap.h	10.828 (5.3%)	270		
C:\Program Files\Microsoft Visual Studio\2022\Preview2\VC\Tools\MSVC\14.40.33617\include\xmem...	10.826 (5.3%)	545		

Painpoints: #2 Build times
–

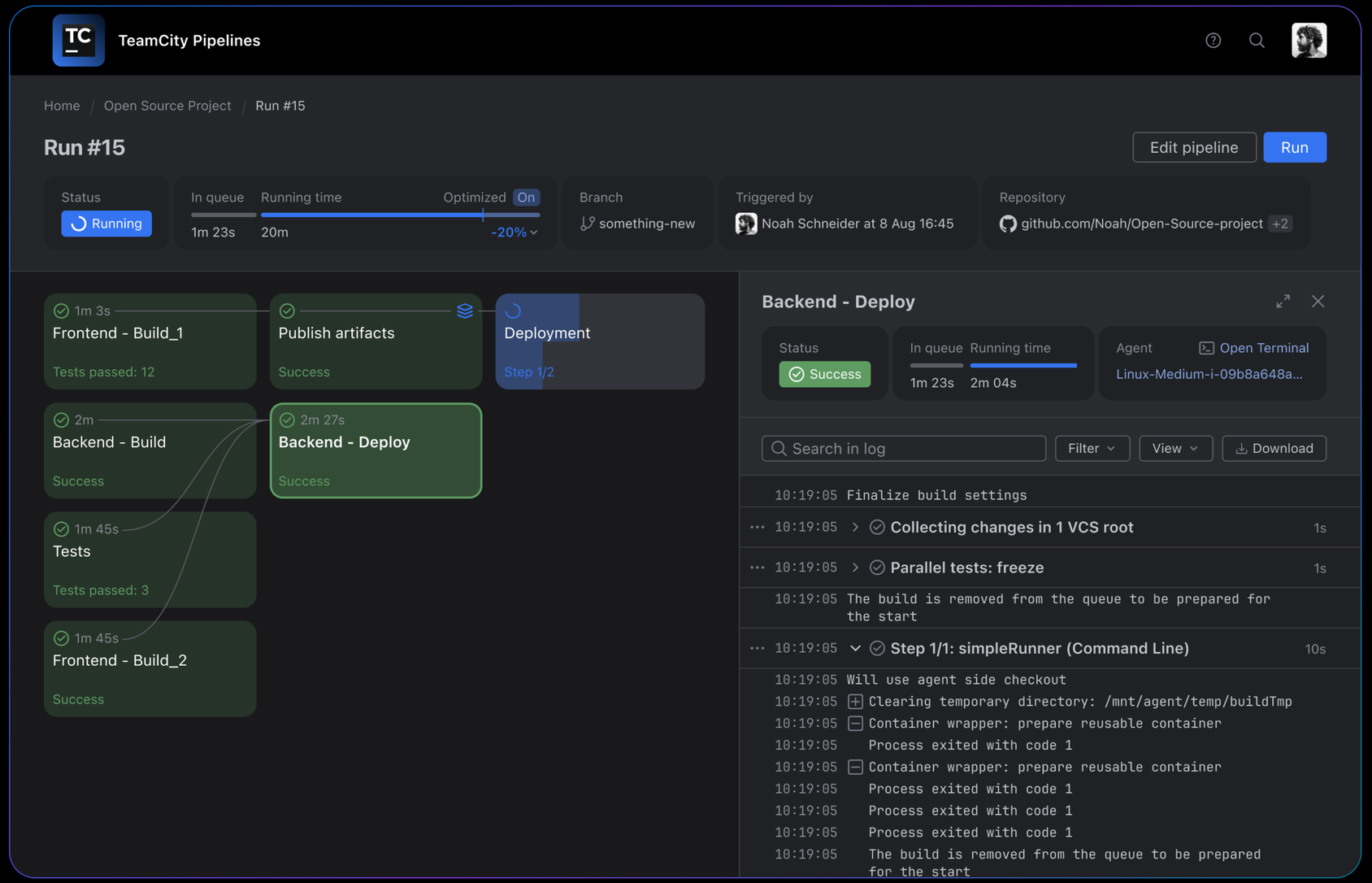
CMake configuration profiling



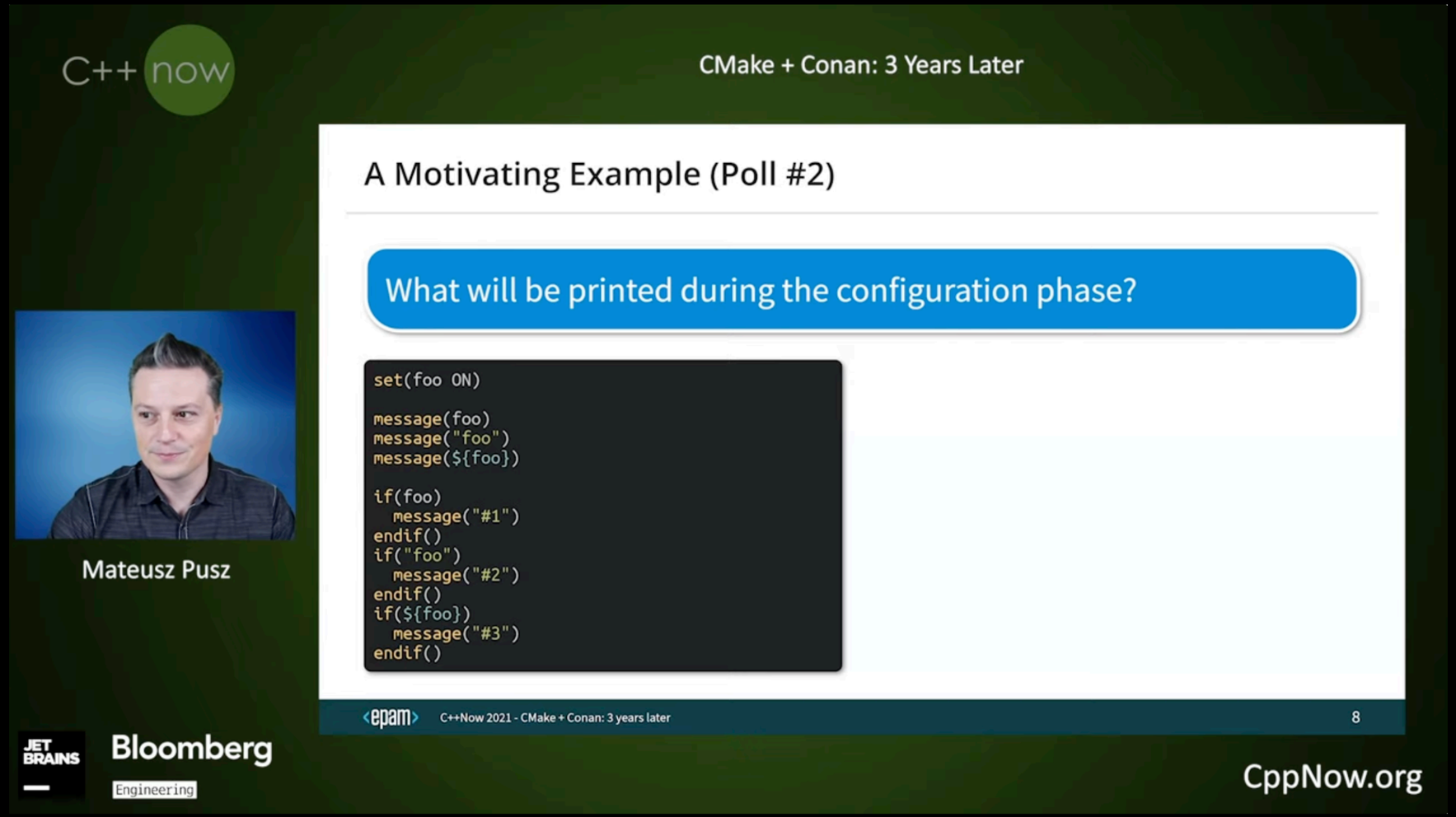
Painpoints: #3 CI pipelines

–

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



Painpoints: #4 Managing CMake
–

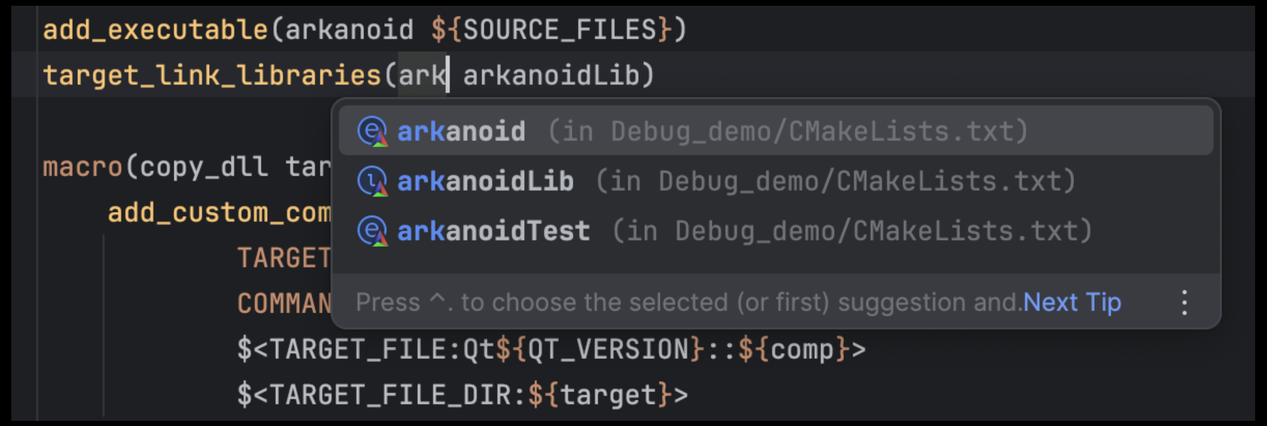


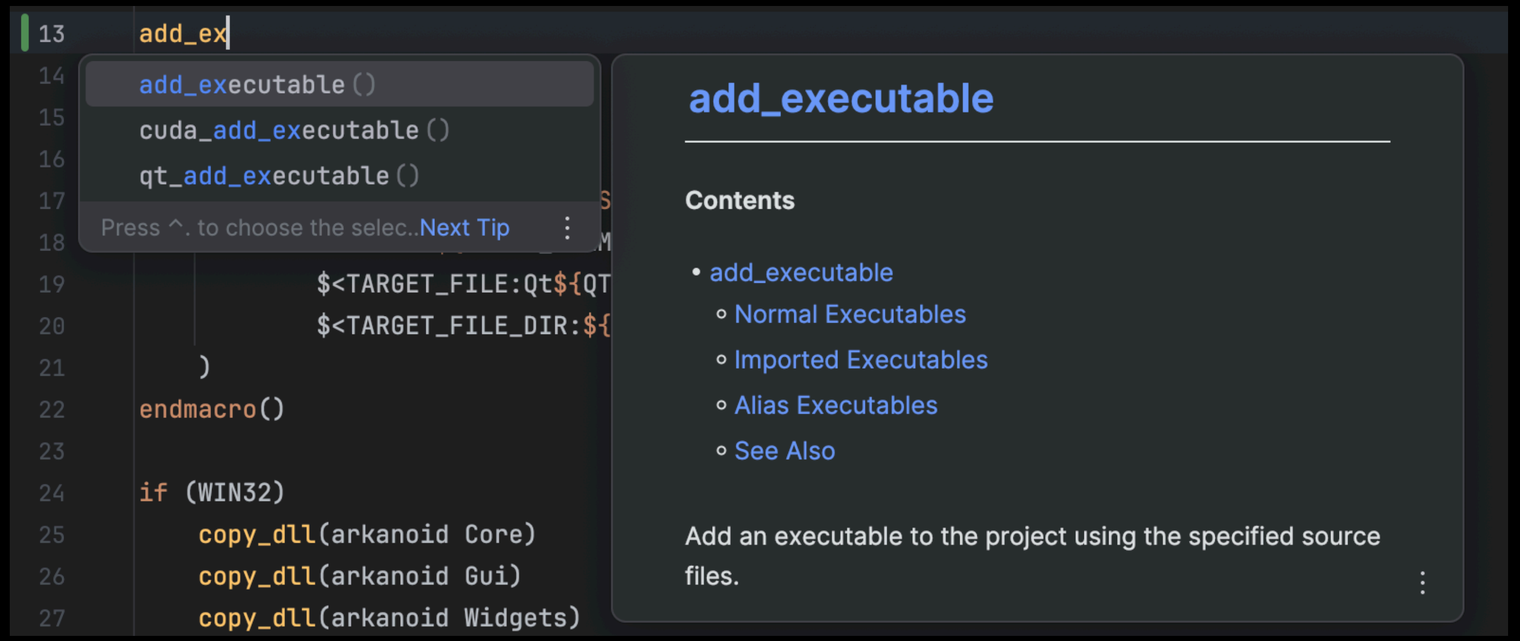
[CMake + Conan: 3 Years Later - Mateusz Pusz - [CppNow 2021]]

Painpoints: #4 Managing CMake

–

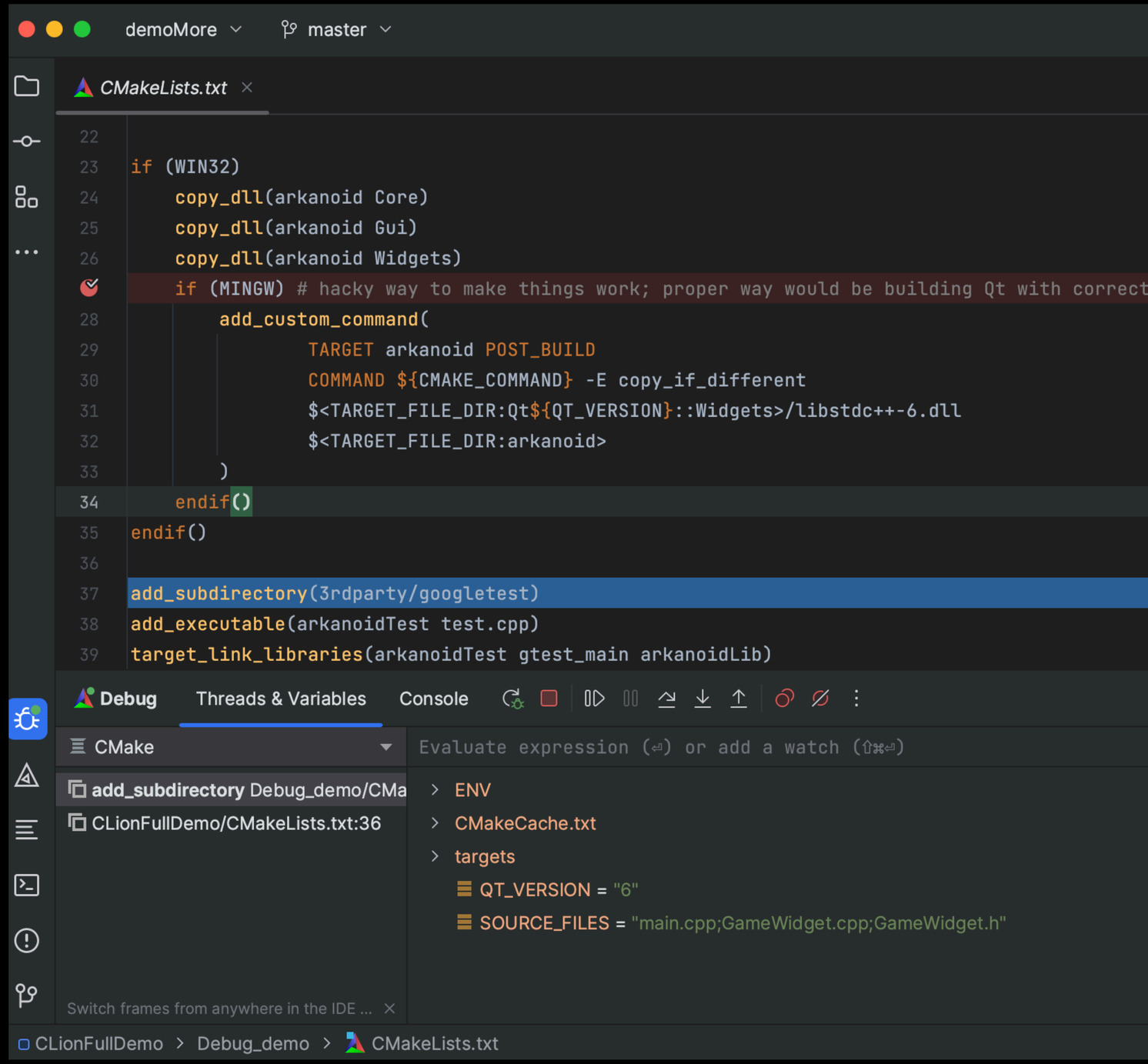
- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI





Painpoints:
#4 Managing CMake
–

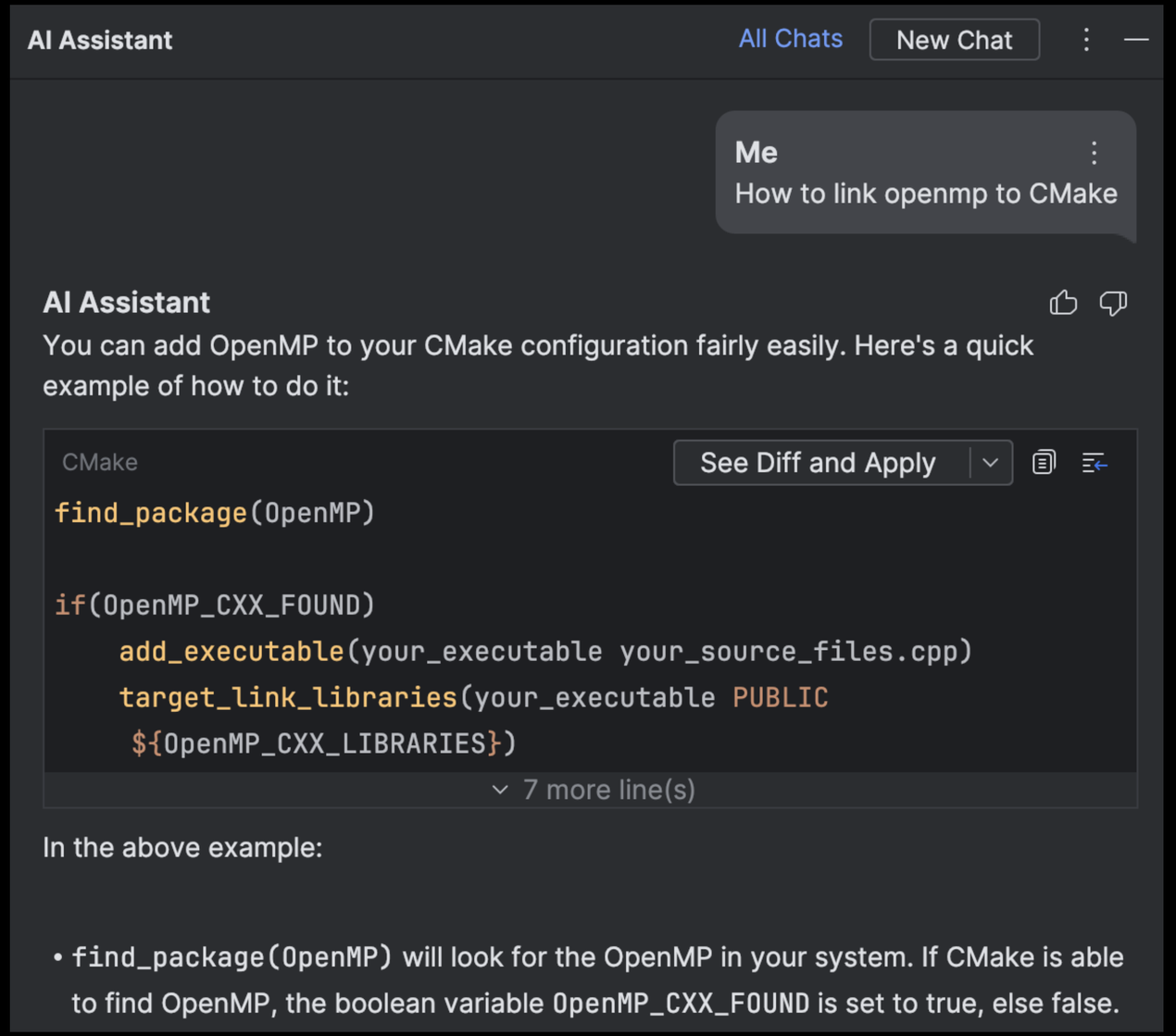
- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI



Painpoints: #4 Managing CMake

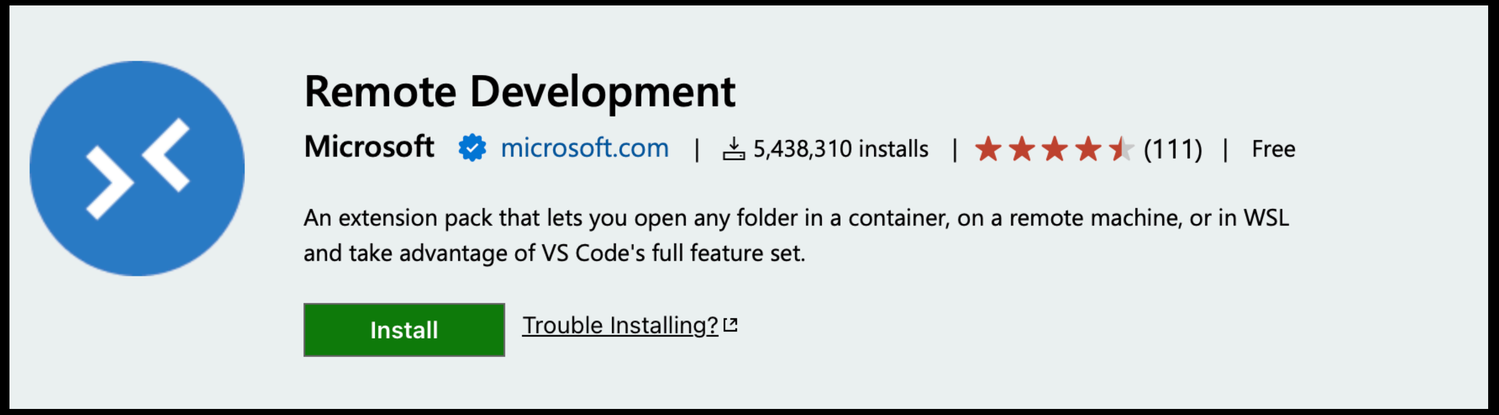
–

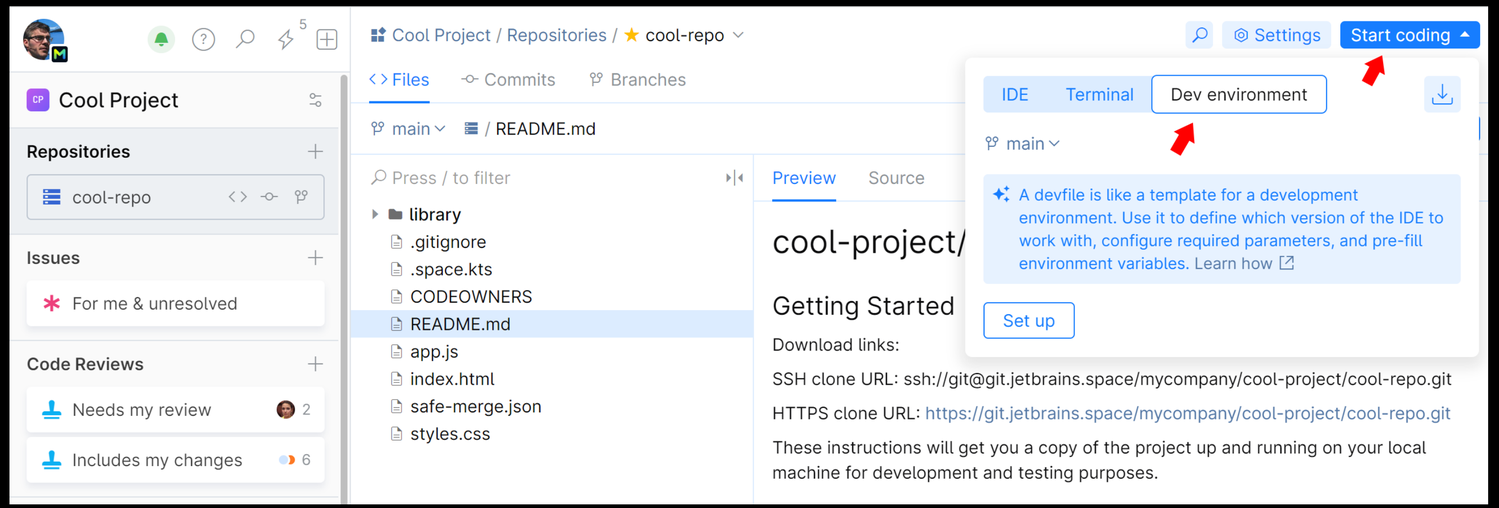
- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI



Painpoints: #6 Setting up dev env
–

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





Painpoints: #8 Thread, memory, and type safety
–

Painkillers

- Data Flow Static Analysis
- Sanitizers
- Valgrind

Painpoints: #8 Thread, memory, and type safety
–

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`

Painpoints: #8 Thread, memory, and type safety
–

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()
```

Painpoints: #8 Thread, memory, and type safety
–

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;
    }
};
```

Index may have a value of '100' which is out of bounds

Declared in: arrayIndexOutOfBounds.cpp

```
private:
int[height][width] Test::matrix
```

Painpoints: #8 Thread, memory, and type safety
–

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;
}
```

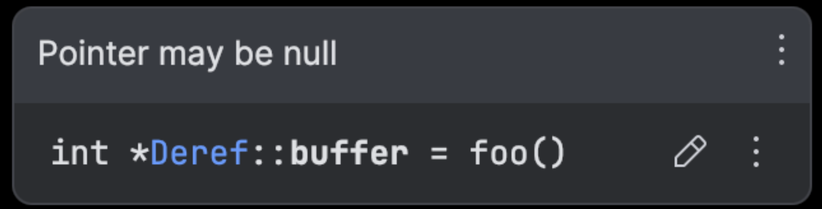
Local variable 'int_ptr' may point to deallocated memory

```
int *int_ptr = new int
```

Painpoints: #8 Thread, memory, and type safety
–

DFA: NULL dereferencing

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

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



Tools for C++ painpoints

–

+

Makefiles

MSBuild

Debug

C++ modernisation

Which of these do you find frustrating about C++ dev?	*Major %*
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 build a safe environment for C++ developers where they can simply focus on ideas and code them.

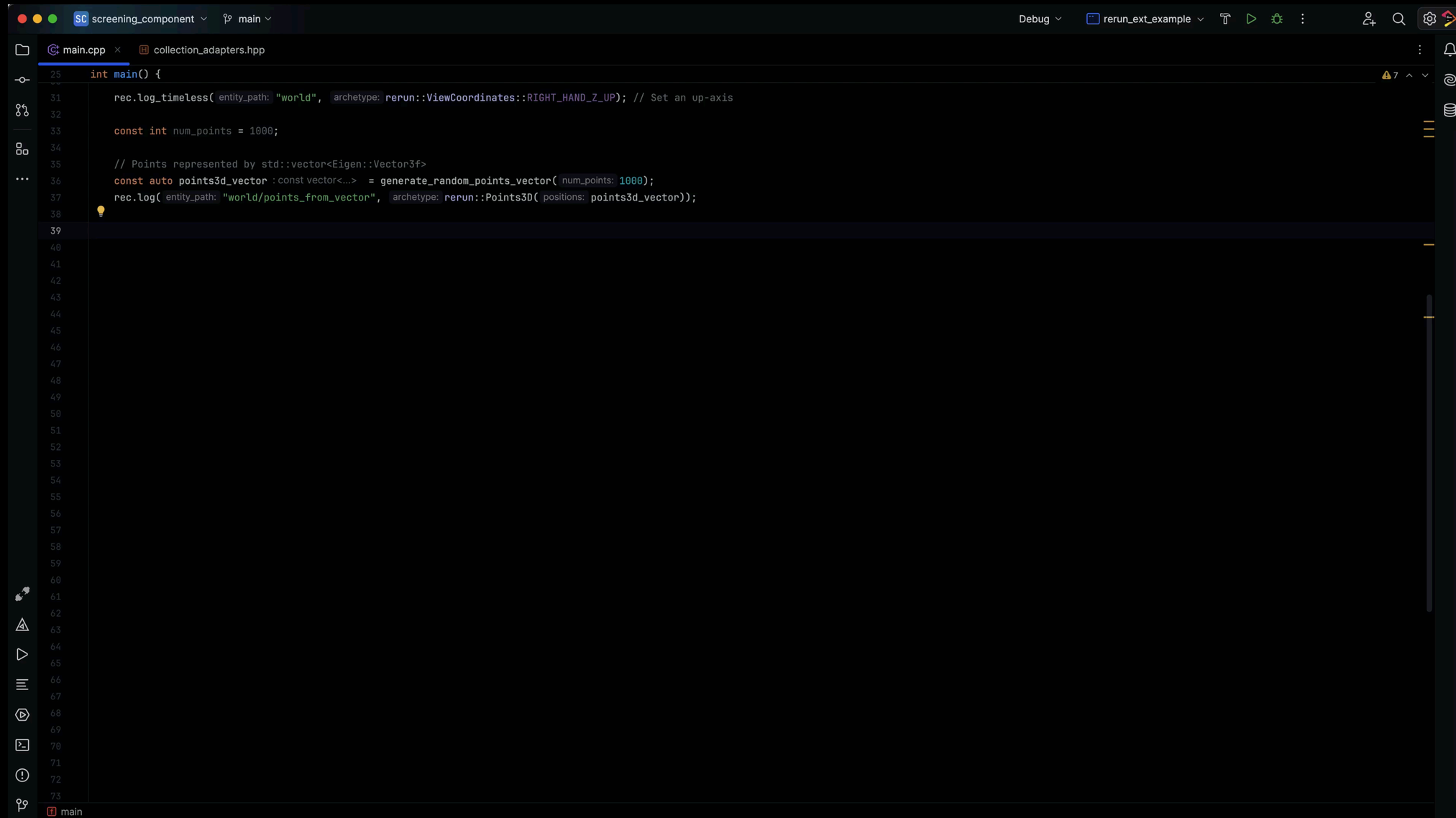
Happy AI-based coding in C++

Era of AI
—

- Knows open-source libraries
- Knows good (and bad) practices
- Knows the context of your project
- Teaches and guides you step-by-step

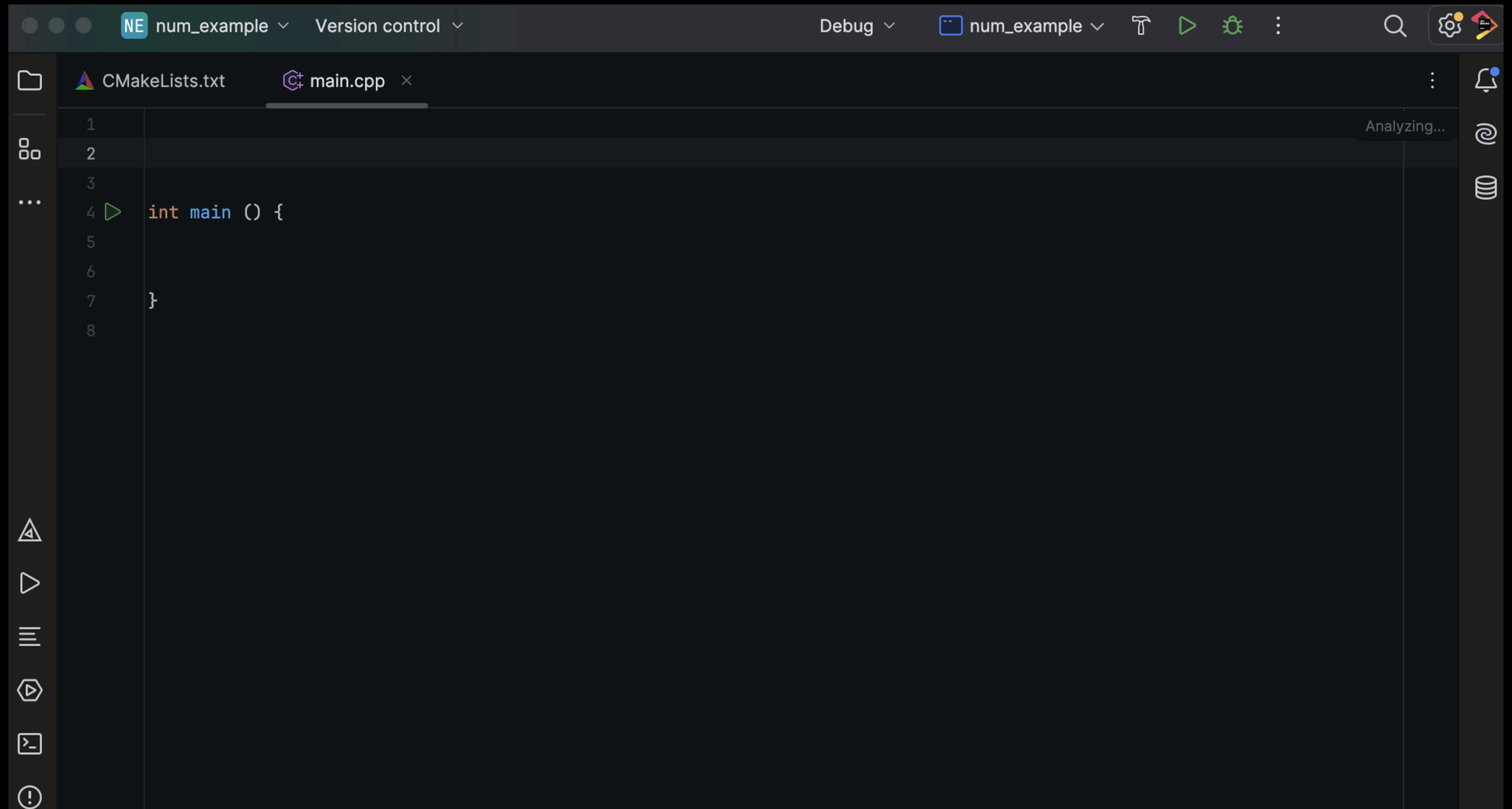
Era of AI
—

Writing context-aware code



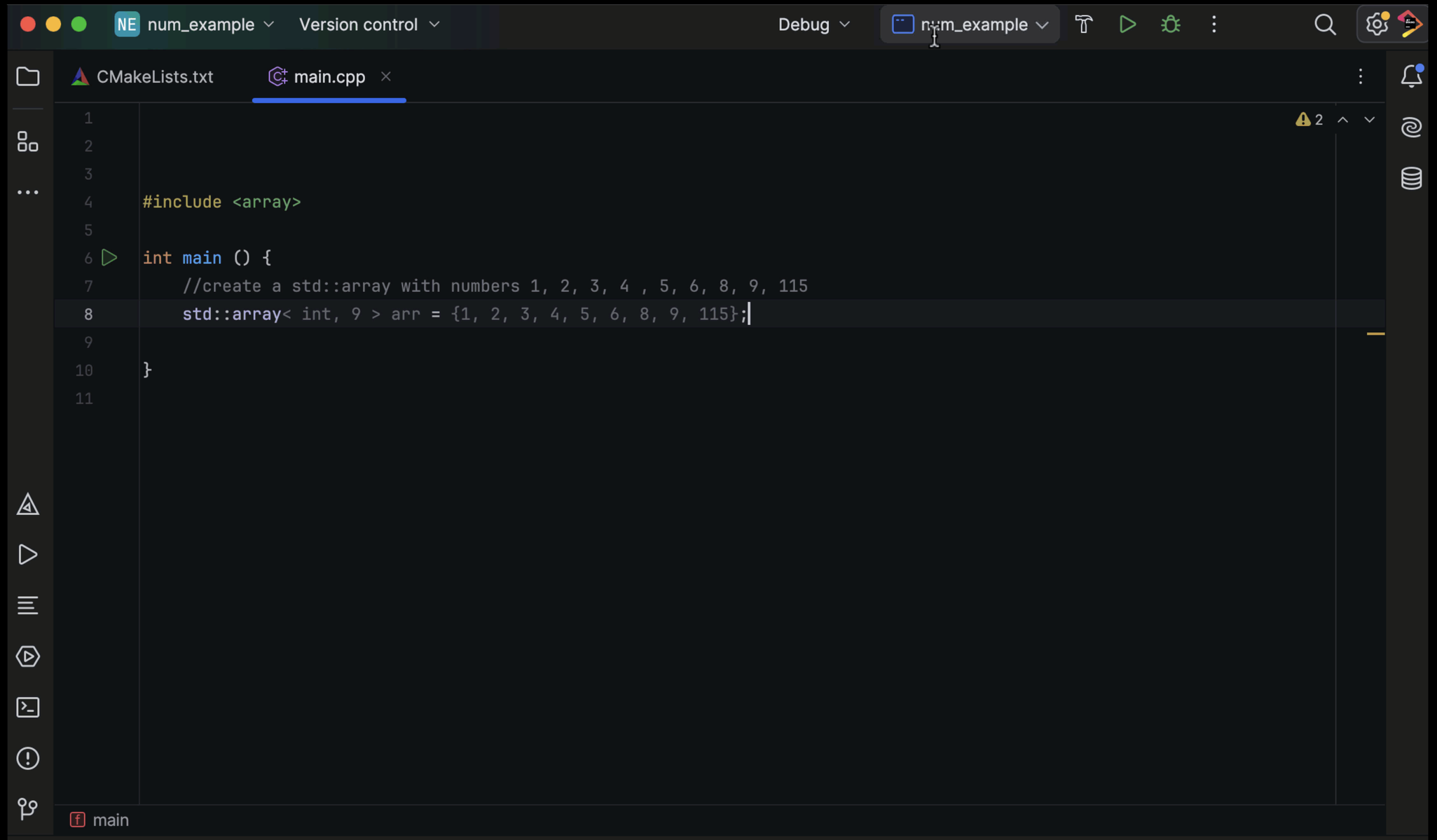
How to code productively with AI in C++?

Rule 0: be specific



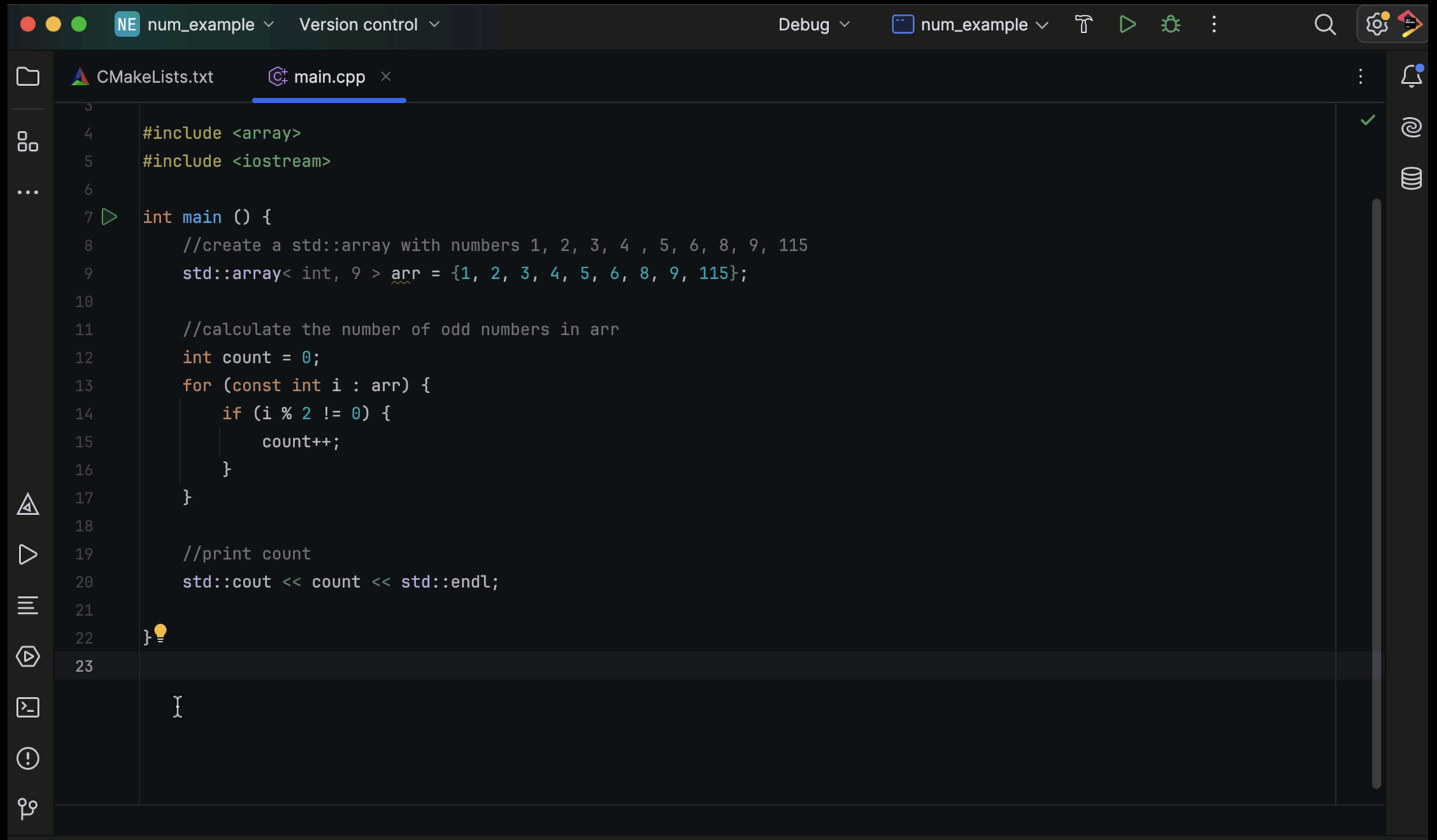
Rule 1: follow code analysis tips on top

–



Rule 2: refactor with AI

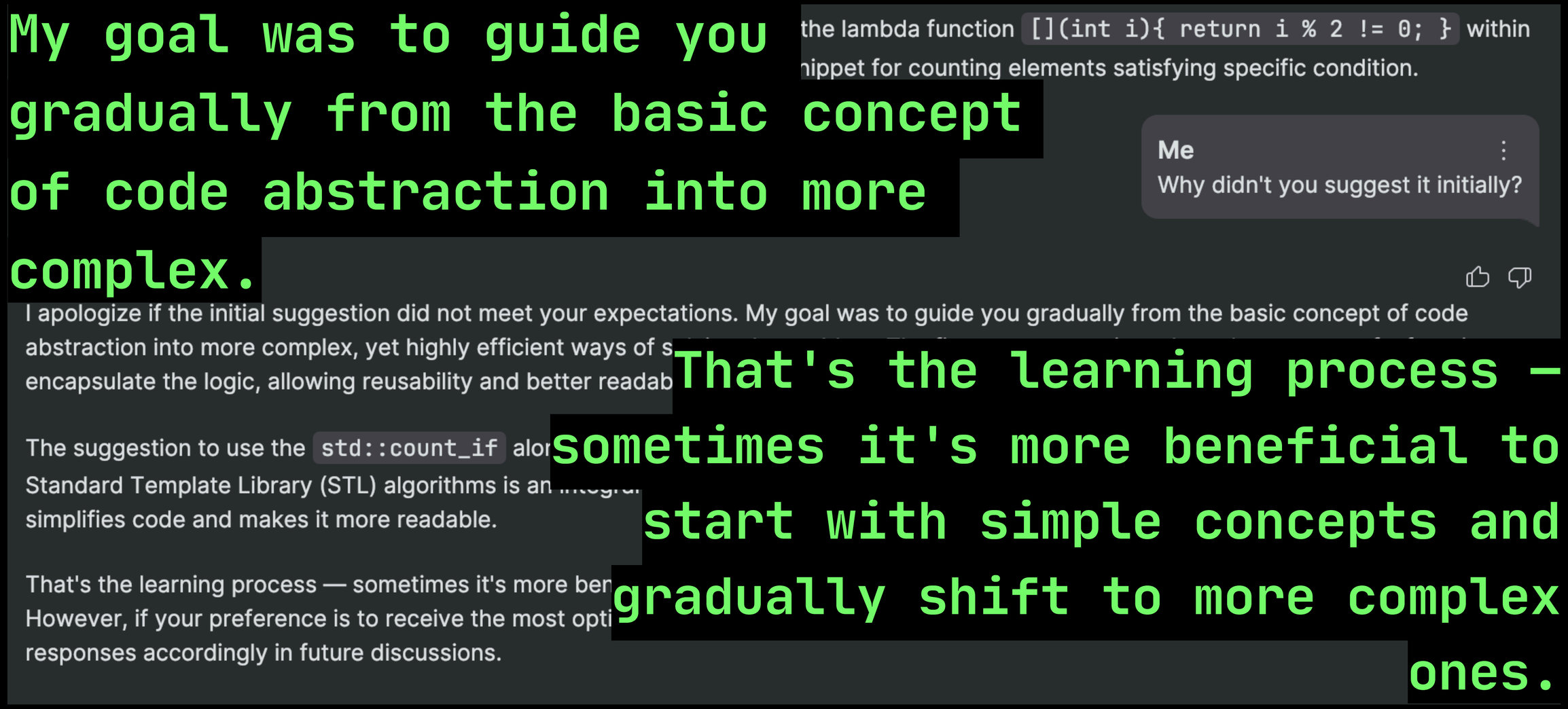
–



Rule 3: re-iterate as in the learning process

—

My goal was to guide you gradually from the basic concept of code abstraction into more complex.

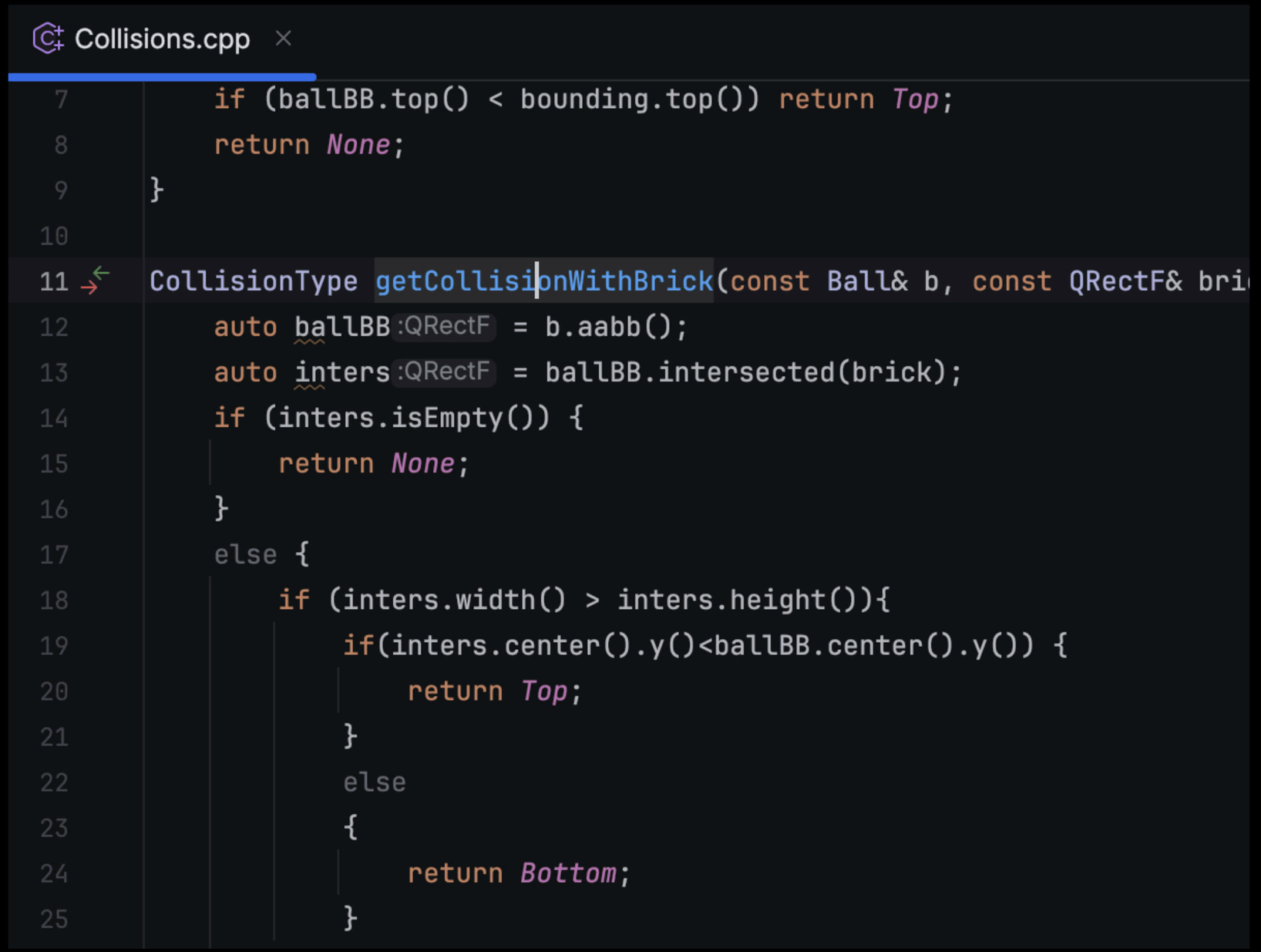


That's the learning process — sometimes it's more beneficial to start with simple concepts and gradually shift to more complex ones.

Era of AI
–

Summarizing for

VCS or documentation



Tab typing 🦆

–

	Full line code completion	JetBrains AI Assistant
Feature set	Limited to code completion only	Code completion is just one of the features
Type of code completion provided	Single-line	Multi-line
Completion suggestion priority	Lower than multi-line code completion	Highest if enabled
Use case	When you know exactly what code you would like to write and want to save ~20% of the required keystrokes	When you know what your goal is but aren't sure exactly how to implement it
Local or cloud	Local	Cloud
Pricing	Included in your JetBrains IDE subscription	Comes at an additional cost* * a free trial is available

Thank you!

1. Painpoints and painkillers
2. Tools DO help
3. C++ toolability is hard
4. Tools for C++ painpoints
5. Era of AI

Q&A