C++ now

C++ Painkillers

Anastasia Kazakova

2024

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  ”

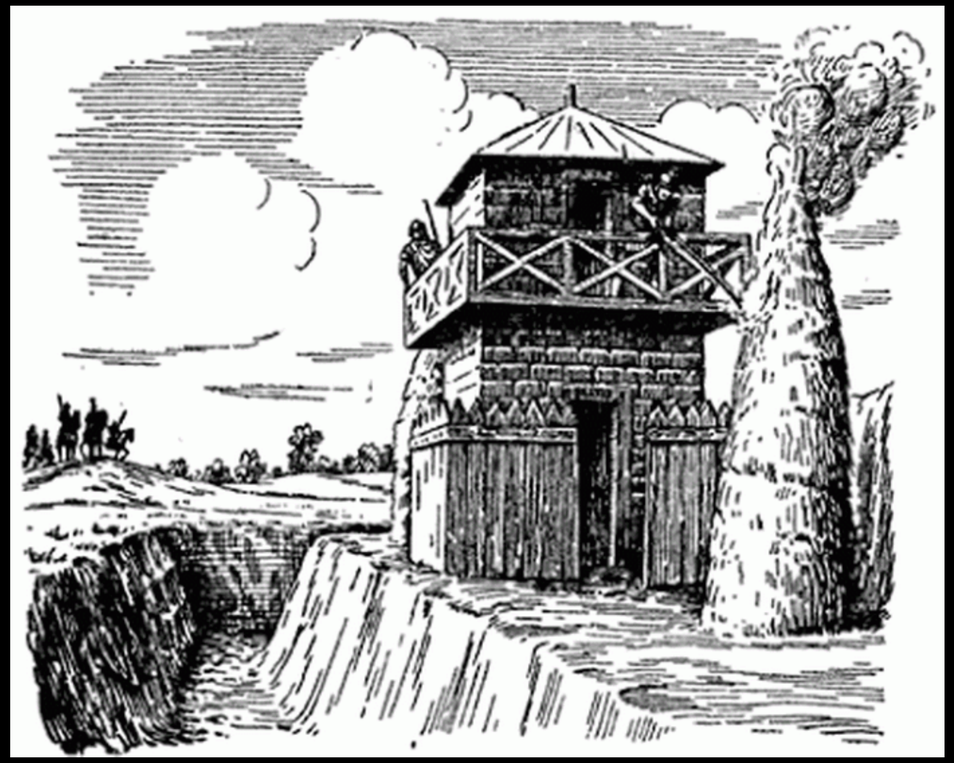
Popov 📻

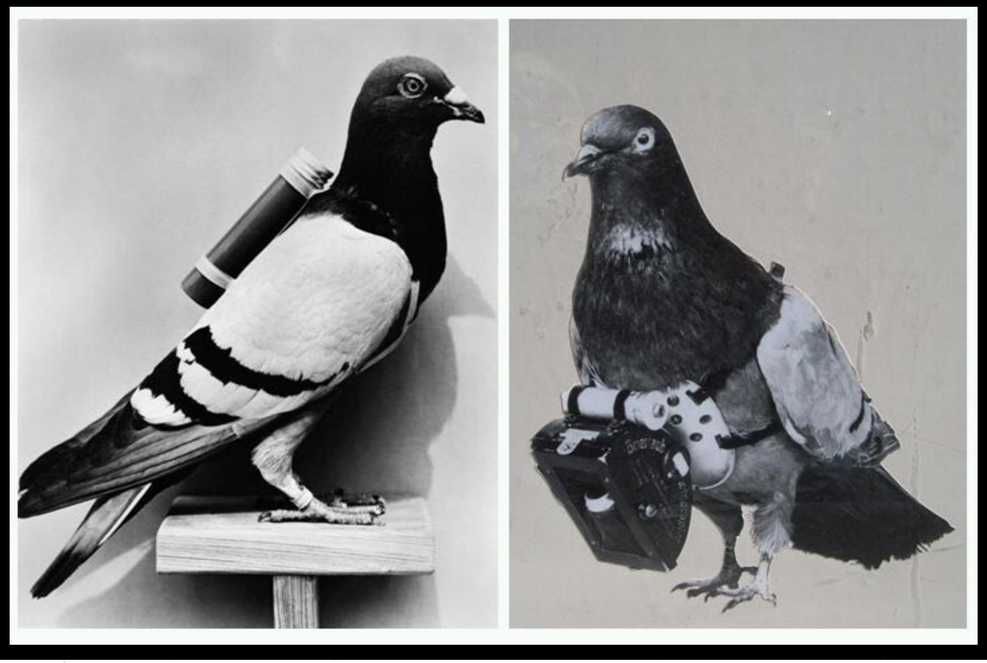
–

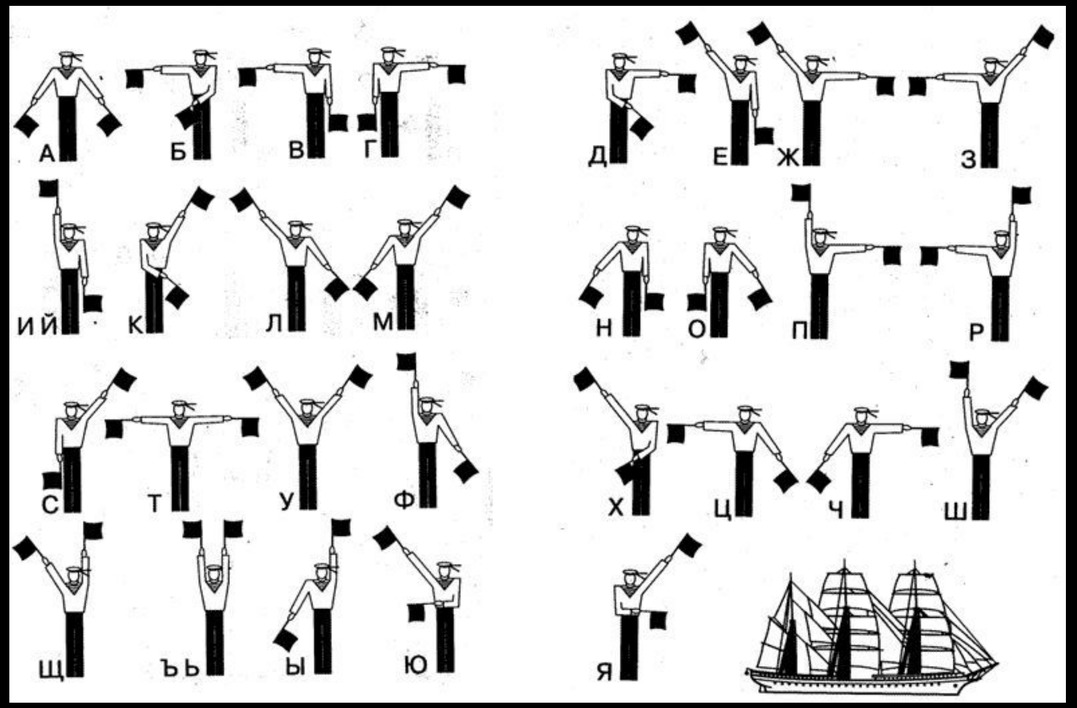
en.wikipedia.org/wiki/Aleksandr_Popov_(physicist)

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!

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 %

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_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 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:

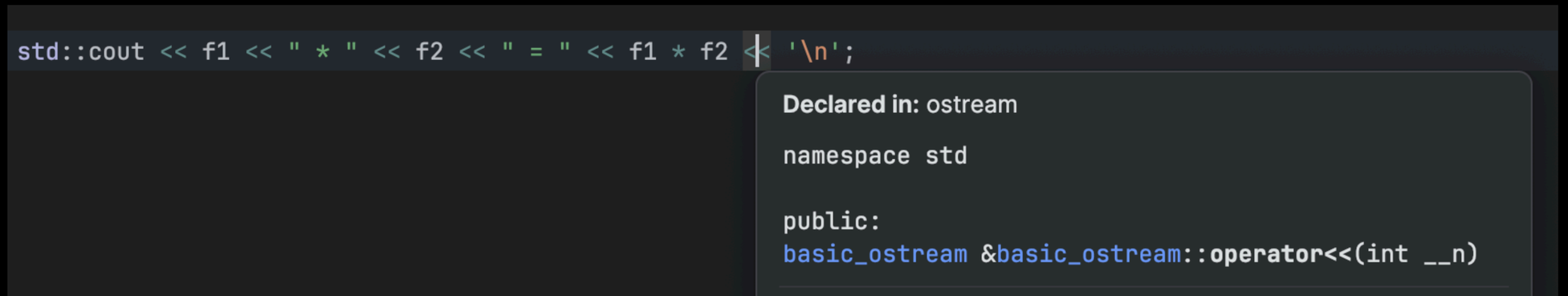
```
int x0 = 0;
int x1 = 1;
int x2 = 2;
int x3 = 3;
int x4 = 4;
```

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



Tools help: named parameters

–

Named arguments

by Ehsan Akhgari and Botond Ballo, N4172

Self-explanatory Function Arguments

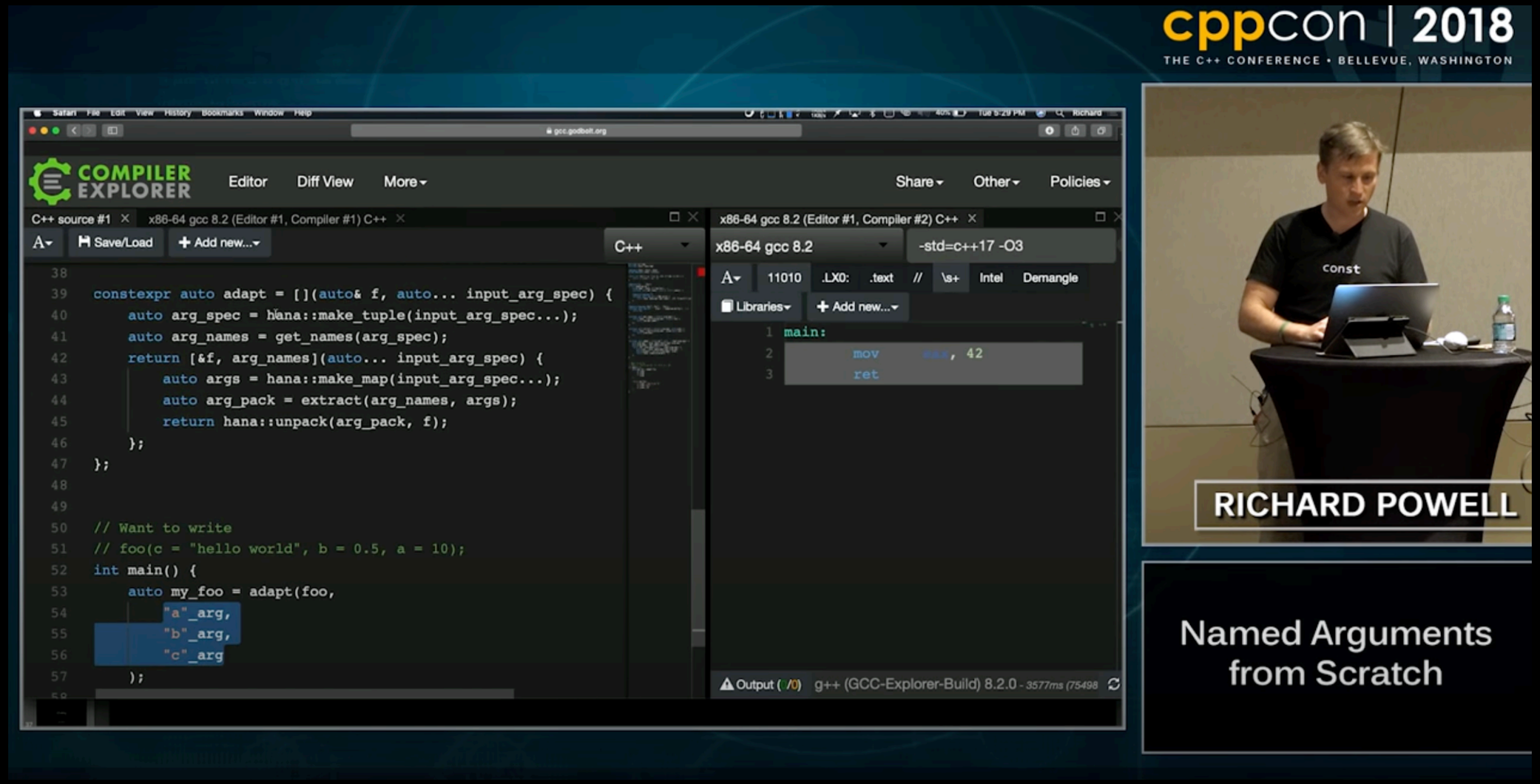
by Axel Naumann, P0671

Boost.Parameter

by David Abrahams and Daniel Wallin, docs

Tools help: named parameters

–



[CppCon 2018: 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) {
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

—

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( 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;
}
```

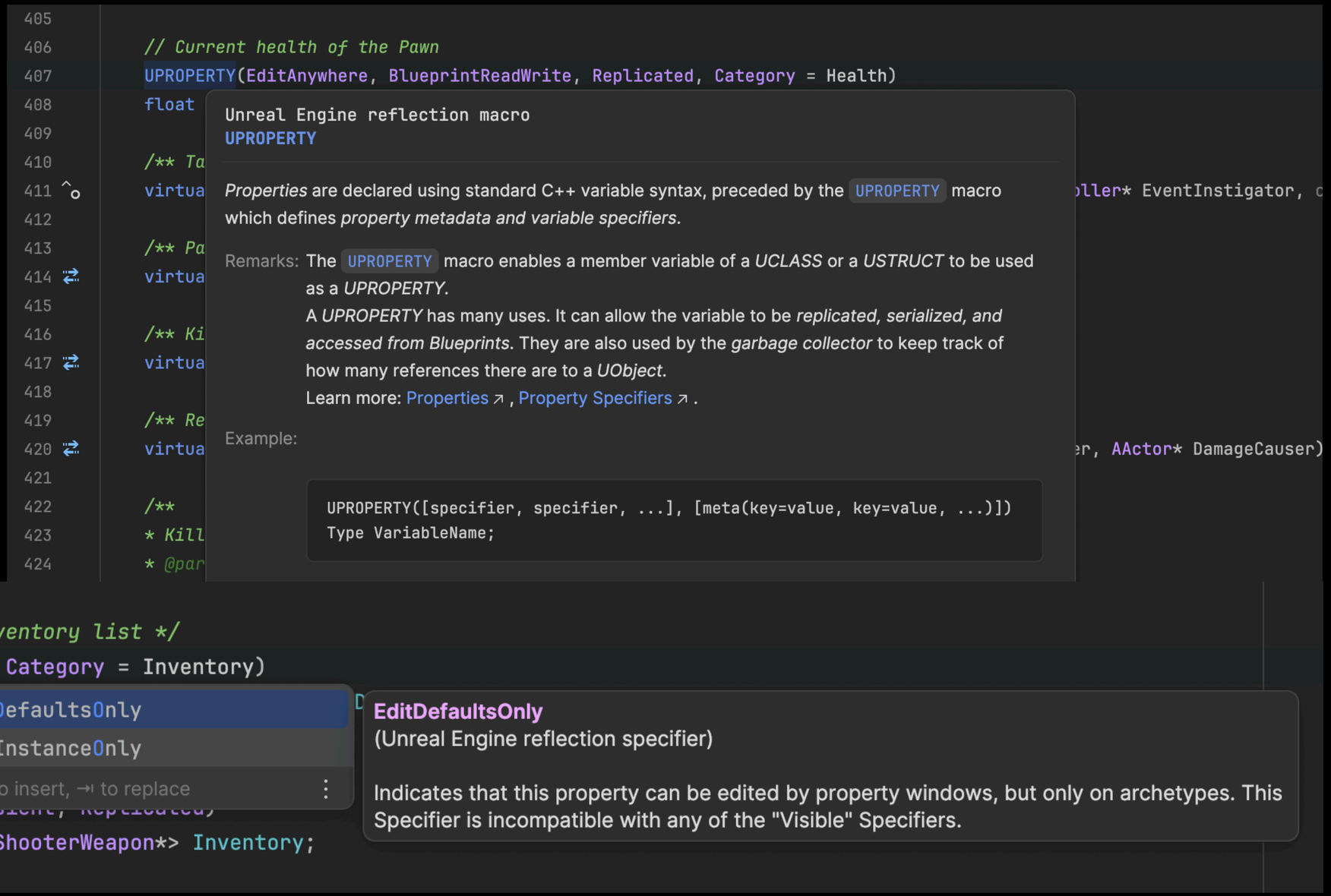
Tools help: with DSLs

—

Unreal Engine

reflection

macros-based dialect



Tools help: with guidelines

—

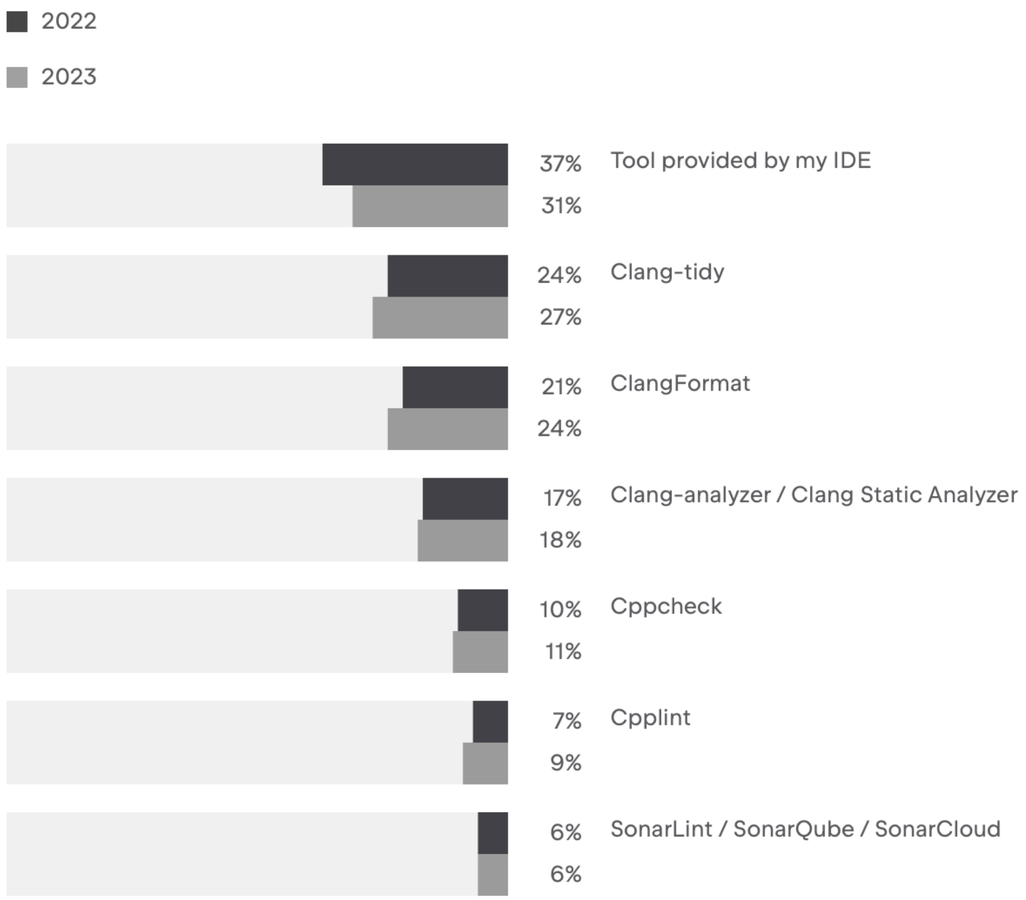
Checks and fixes in ClangTidy

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 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.

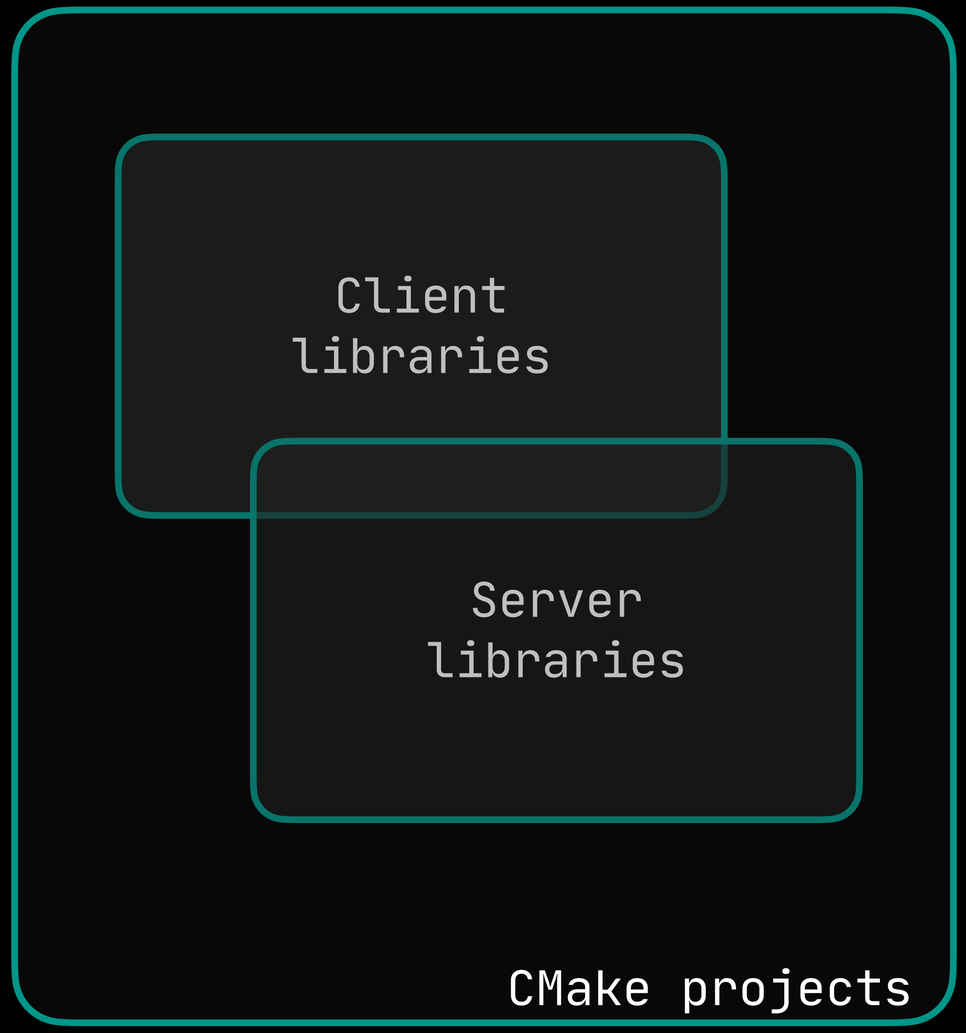
X (formerly Twitter)

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

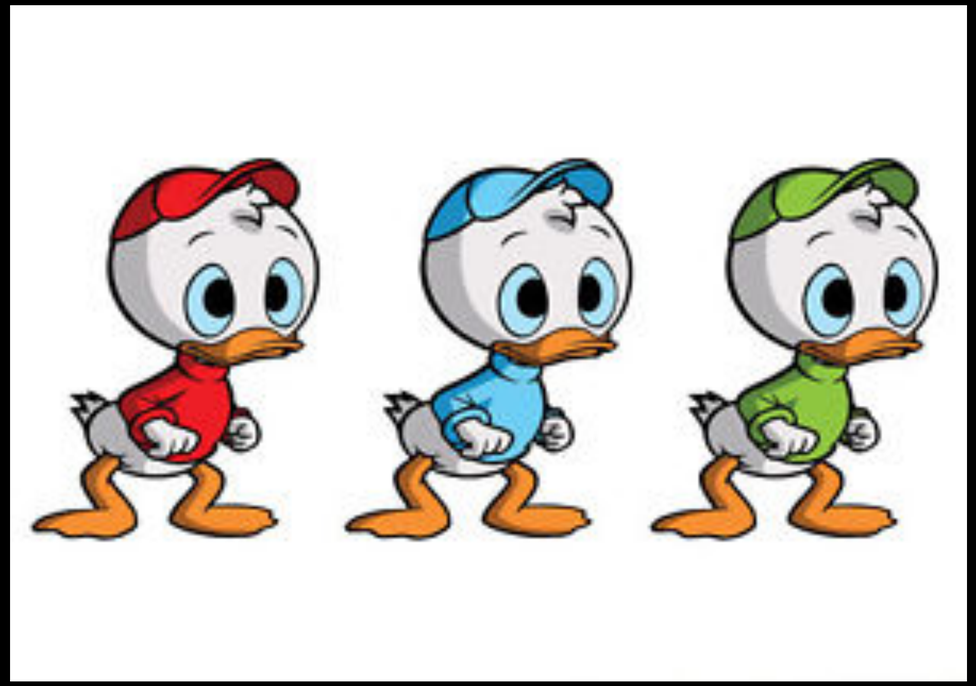
CppCon 2017
New standards to the rescue: the view through an IDE’s glasses

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

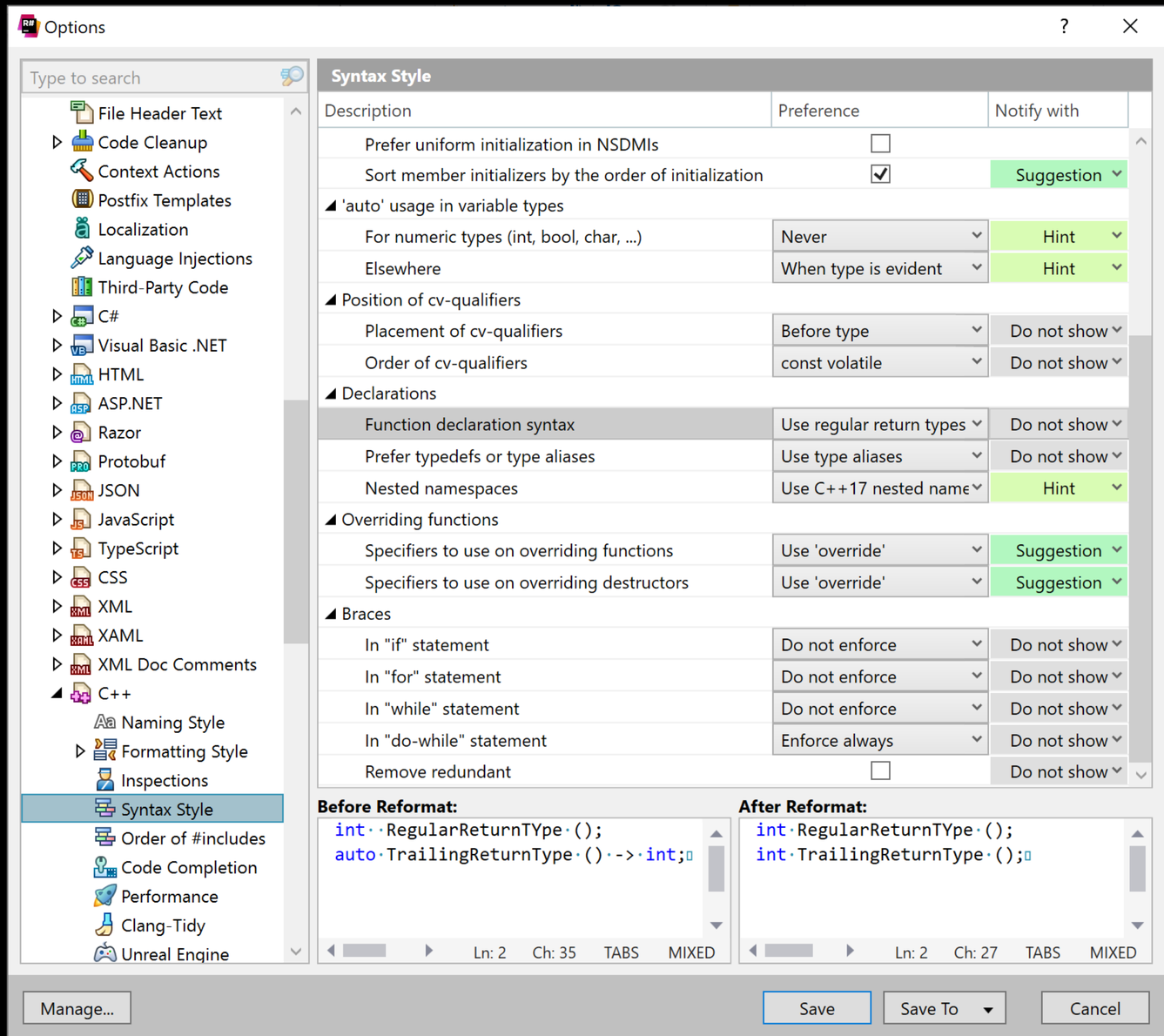
—

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

Const before or after the type it applies to.

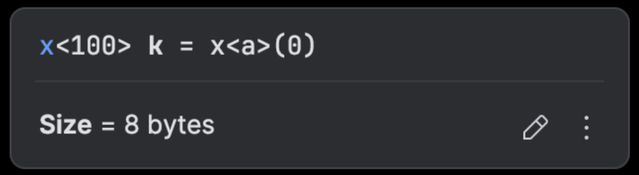
Trailing return type for lambdas or always.

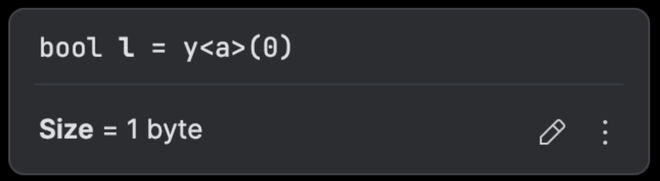
Virtual explicitly (UE) or override/ final and no virtual (C++ Core Guidelines).



C++ toolability

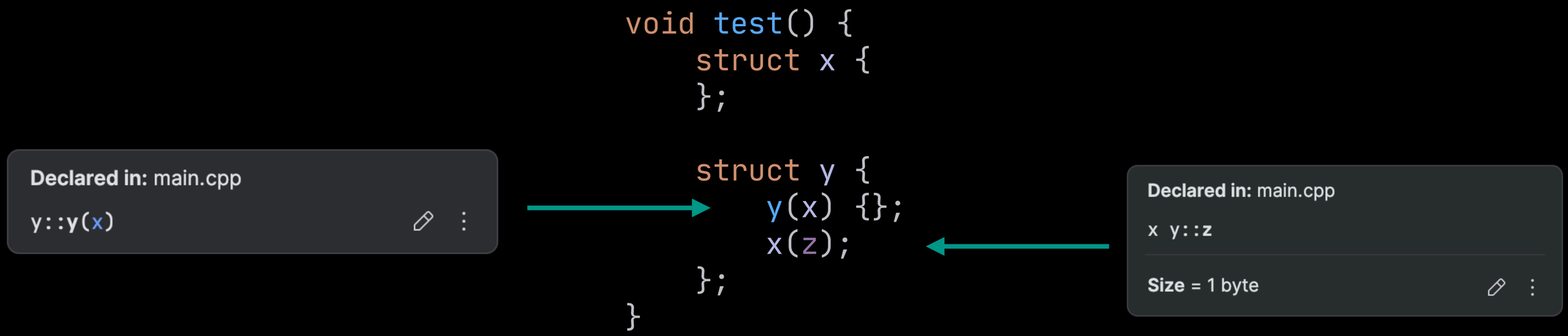
```
template<int>
struct x {
    int z;
    bool p;
    x(int i) { }
};

void test(int y) {

    const int a = 100;

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





C++ toolability

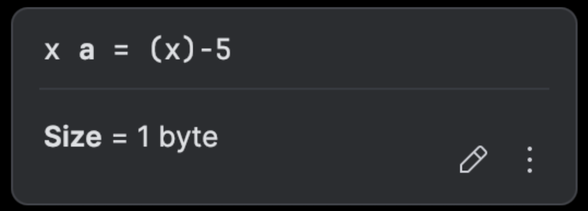
-

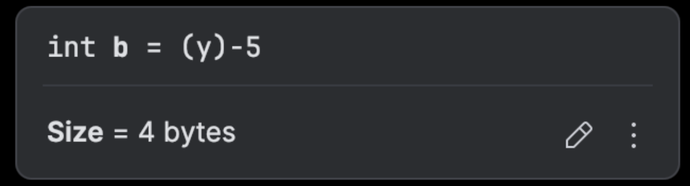


C++ toolability

—

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

    int y = 100;

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





C++ toolability

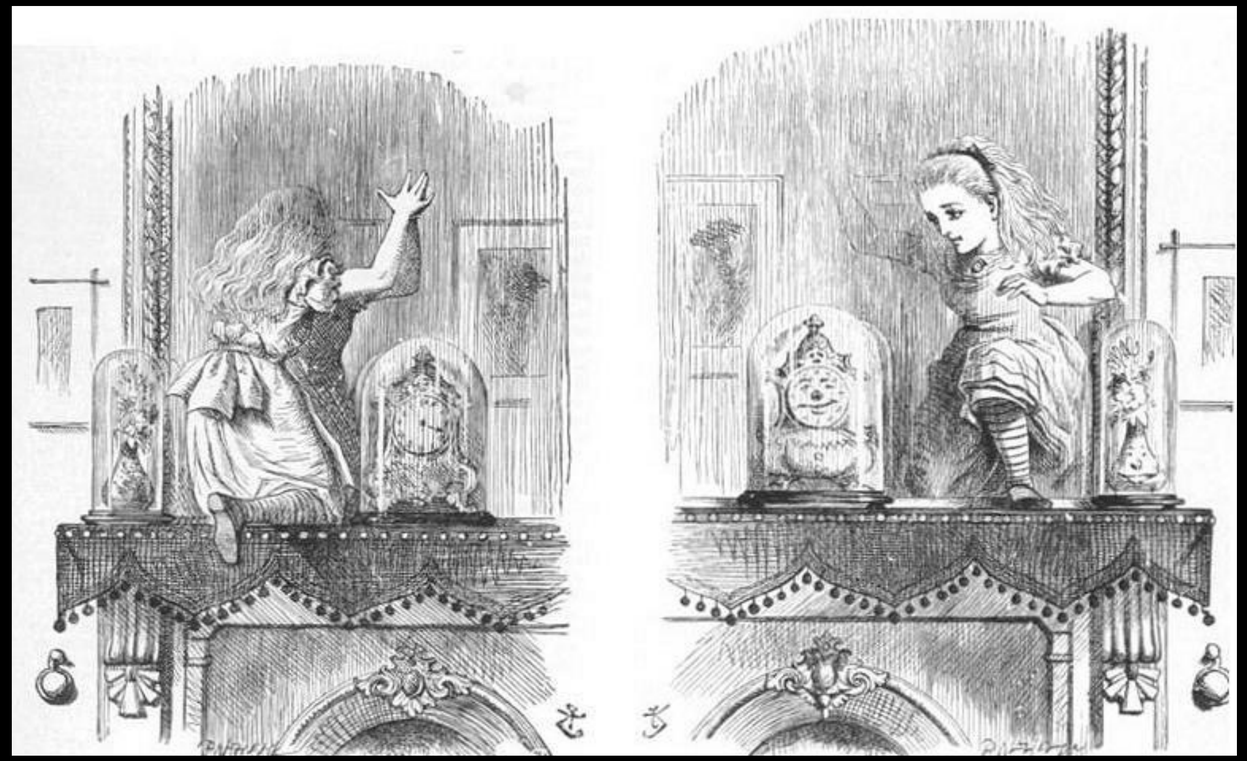
```
int(x), y, a, b, c, d, e, f, *z;

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…

```cpp
template<int N>
struct A {
    A(int p) {}
    operator int() { return N; }
};

int main() {
    // Formatting setting for templates and
    // integer expressions are different
    int A = 10;
    if (A < 3 + 6 > (2 + 8) + 1 > 5) {
        std::cout << "print A" << std::endl;
    }

    int B = 10;
    if (B < 20 > 5 + 5 > 5 + 10) {
        std::cout << "print B" << std::endl;
    }
    return 0;
}
```

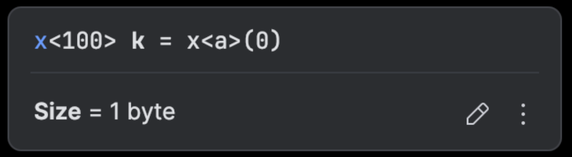
ClangFormat

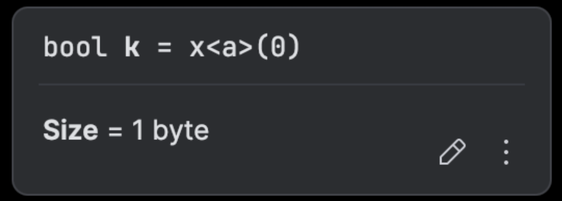
C++ toolability
–

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);
}
```





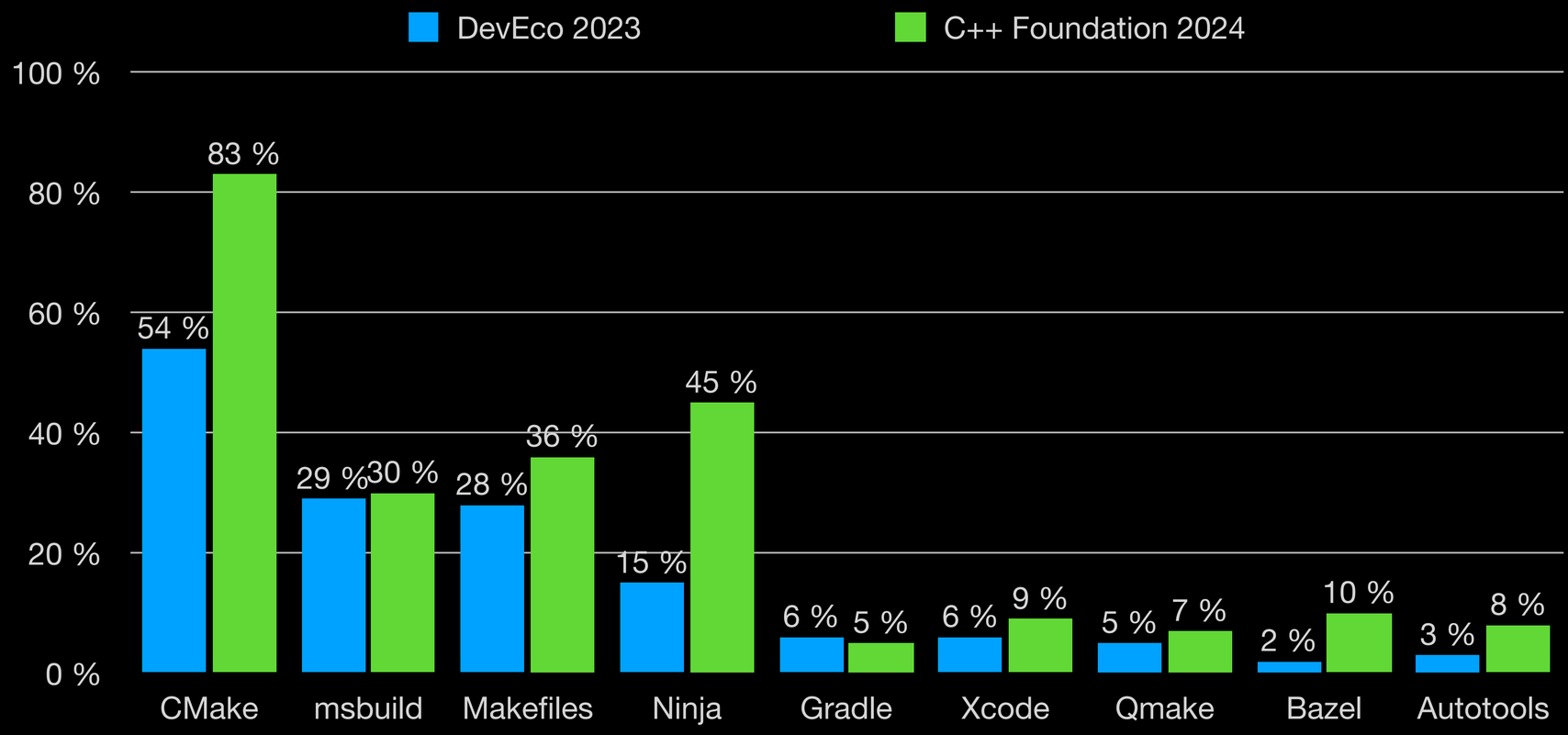
C++ toolability

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

–

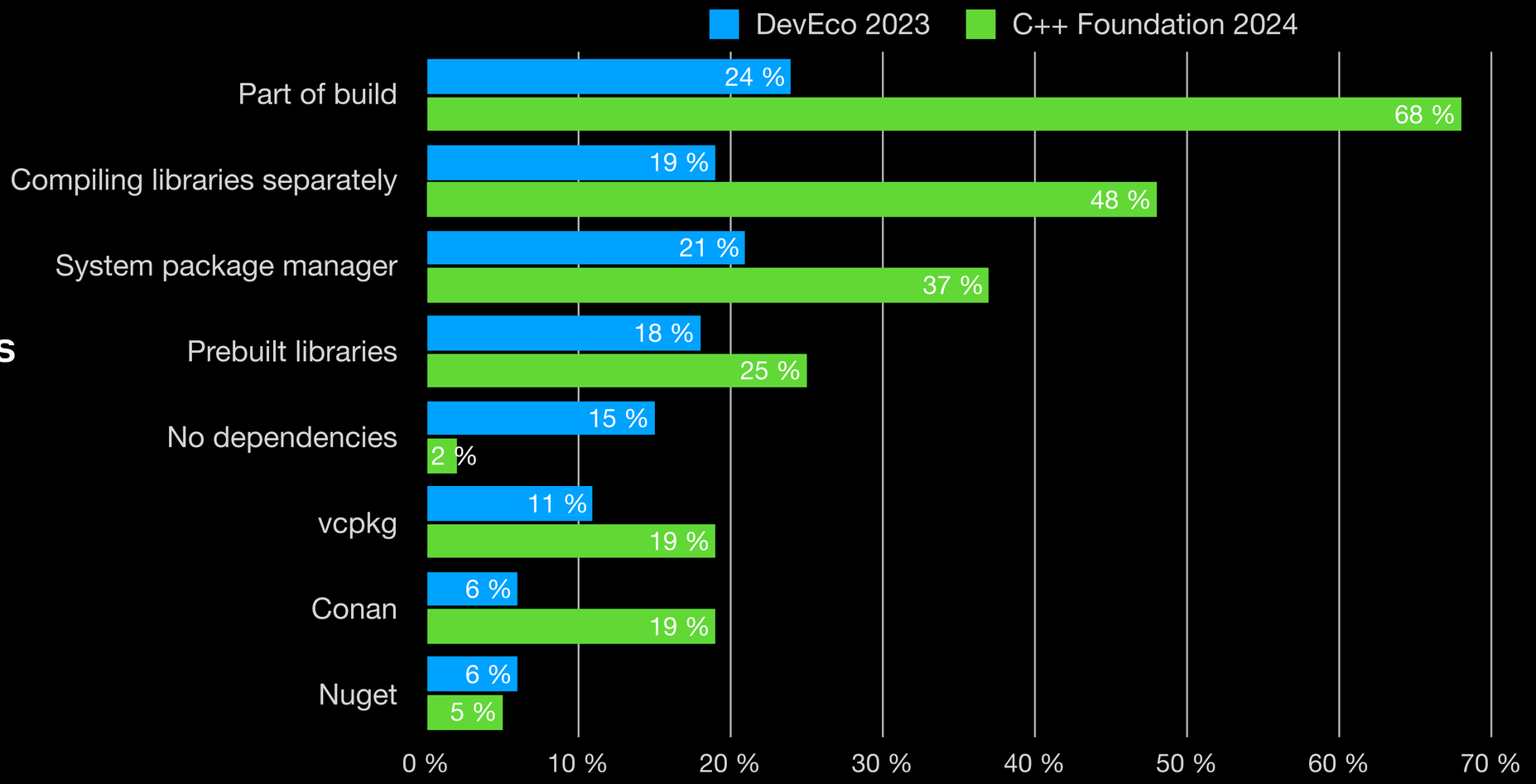
Which tools to use?

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 for C++ painpoints

–

#1 Managing libraries

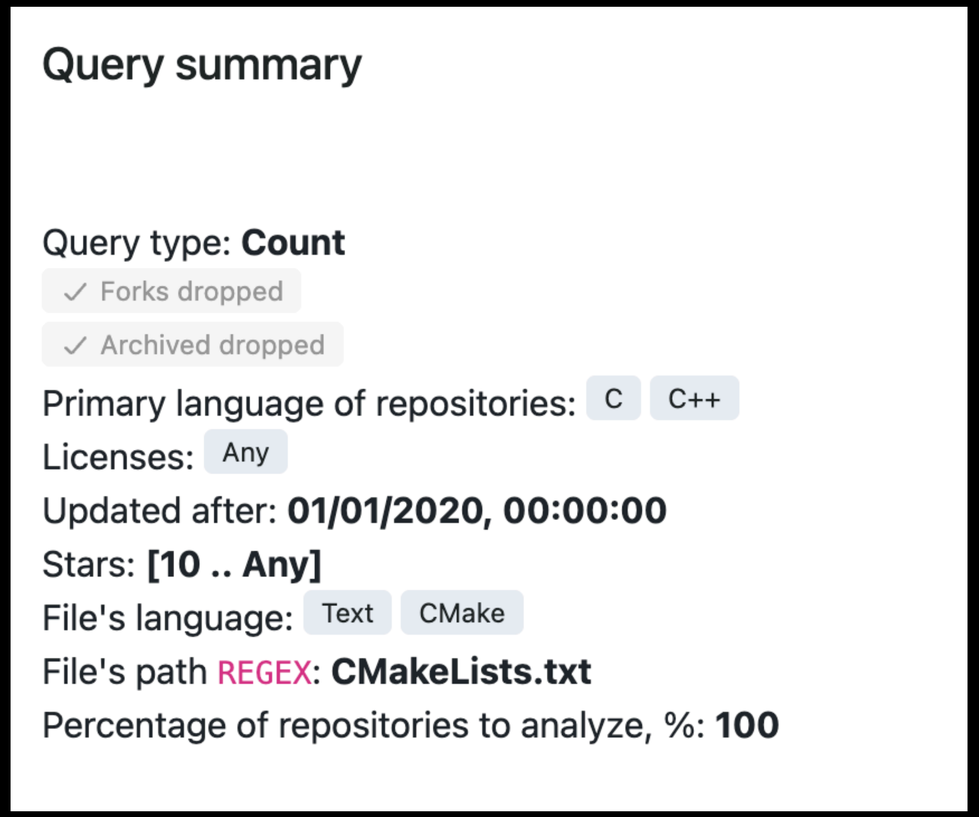


Tools for C++ painpoints
–

#1 Managing libraries

40% repos with

CMakeLists.txt inside

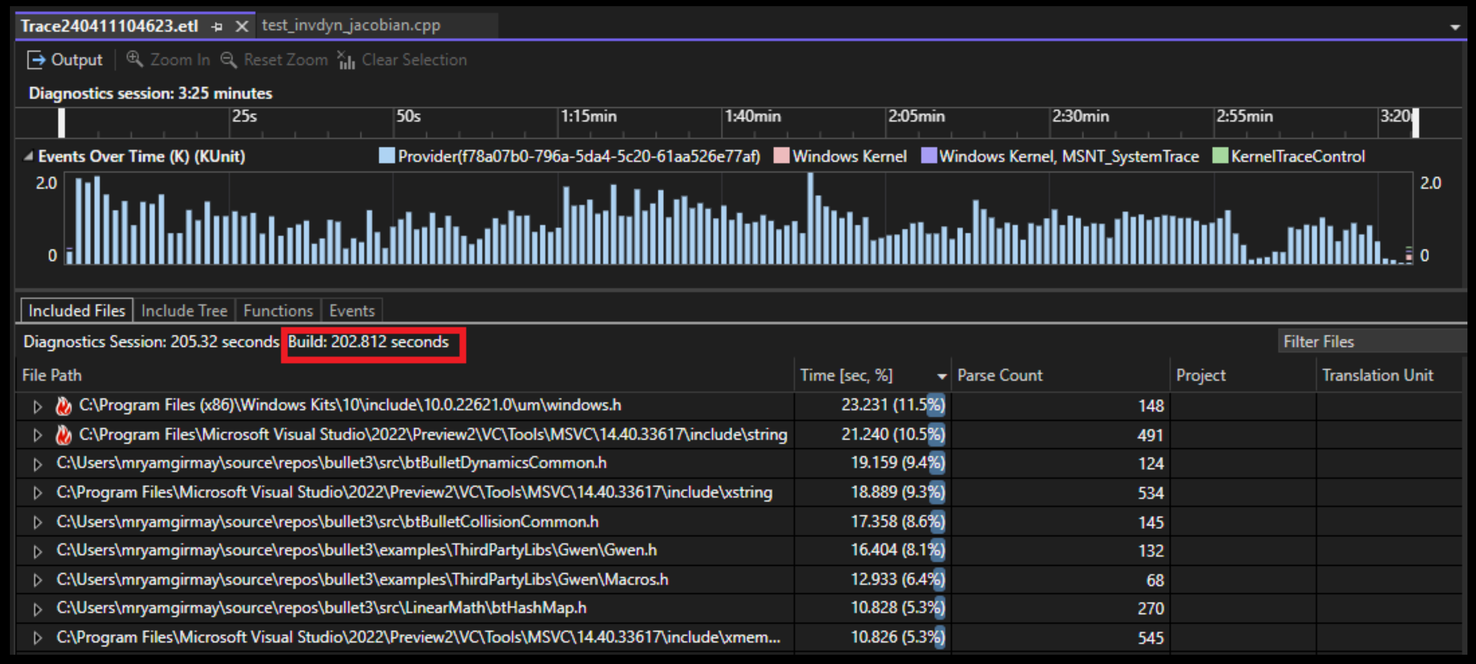


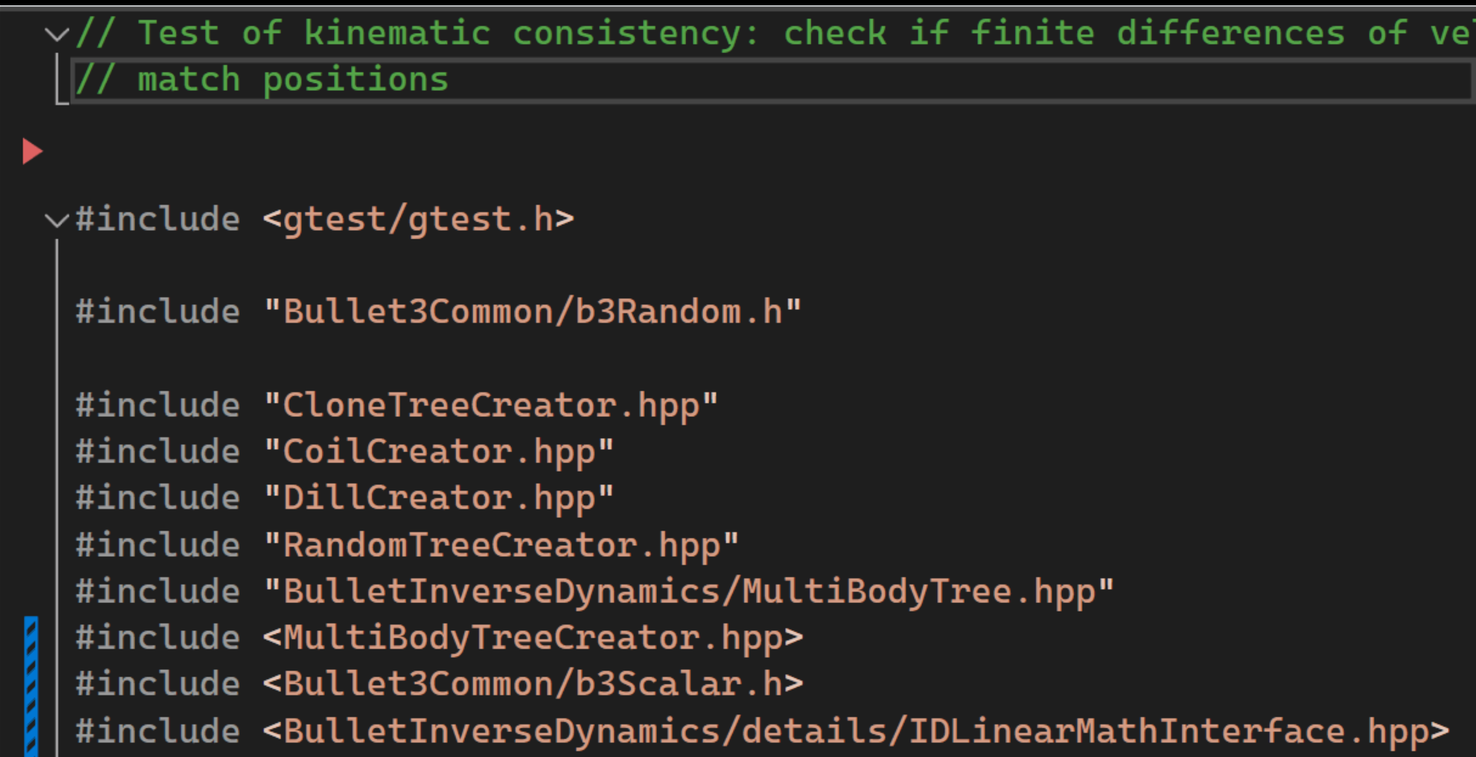
Tools for C++ painpoints

—

#2 Build times

- Build insights
- Include cleanup
- Include Diagnostics

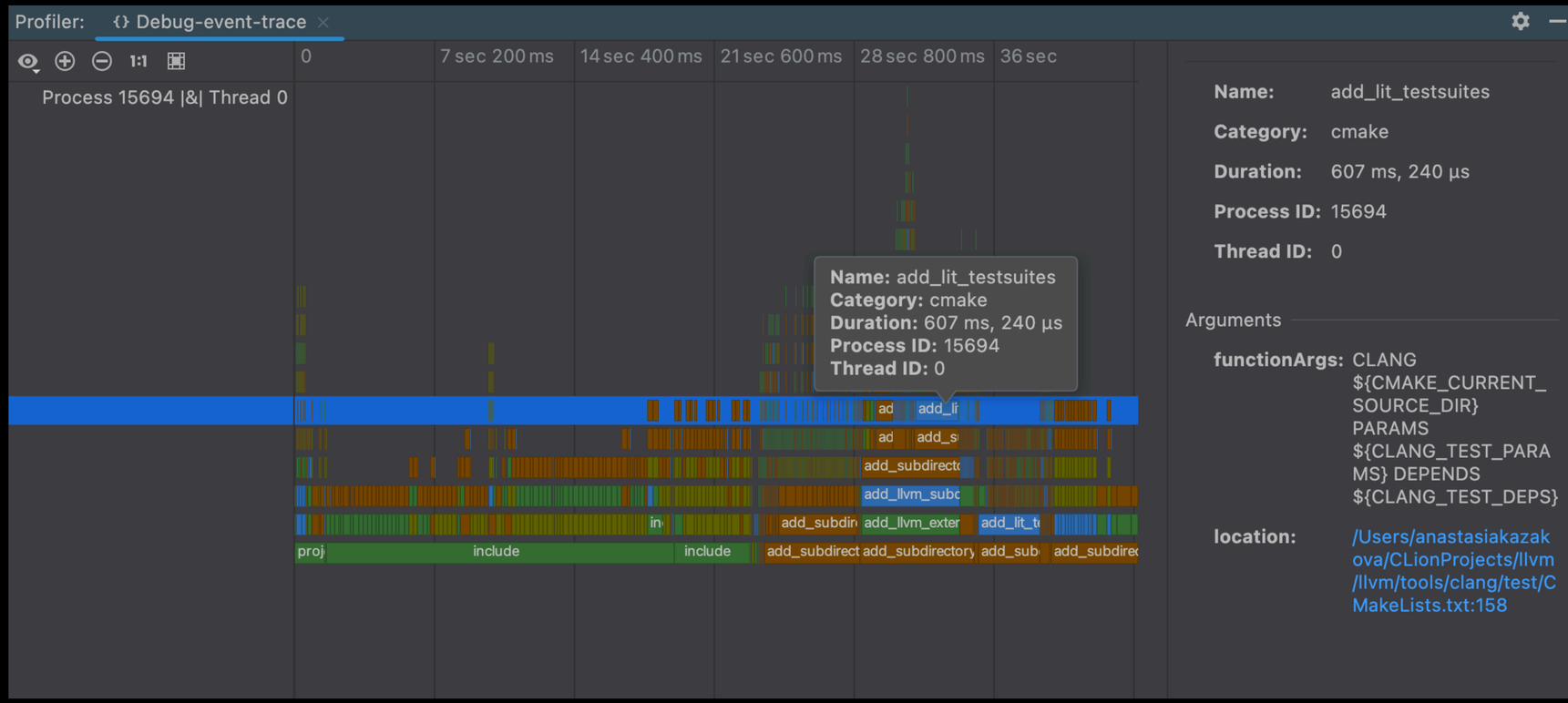




Tools for C++ painpoints
–

#2 Build times

← CMake configuration profiling

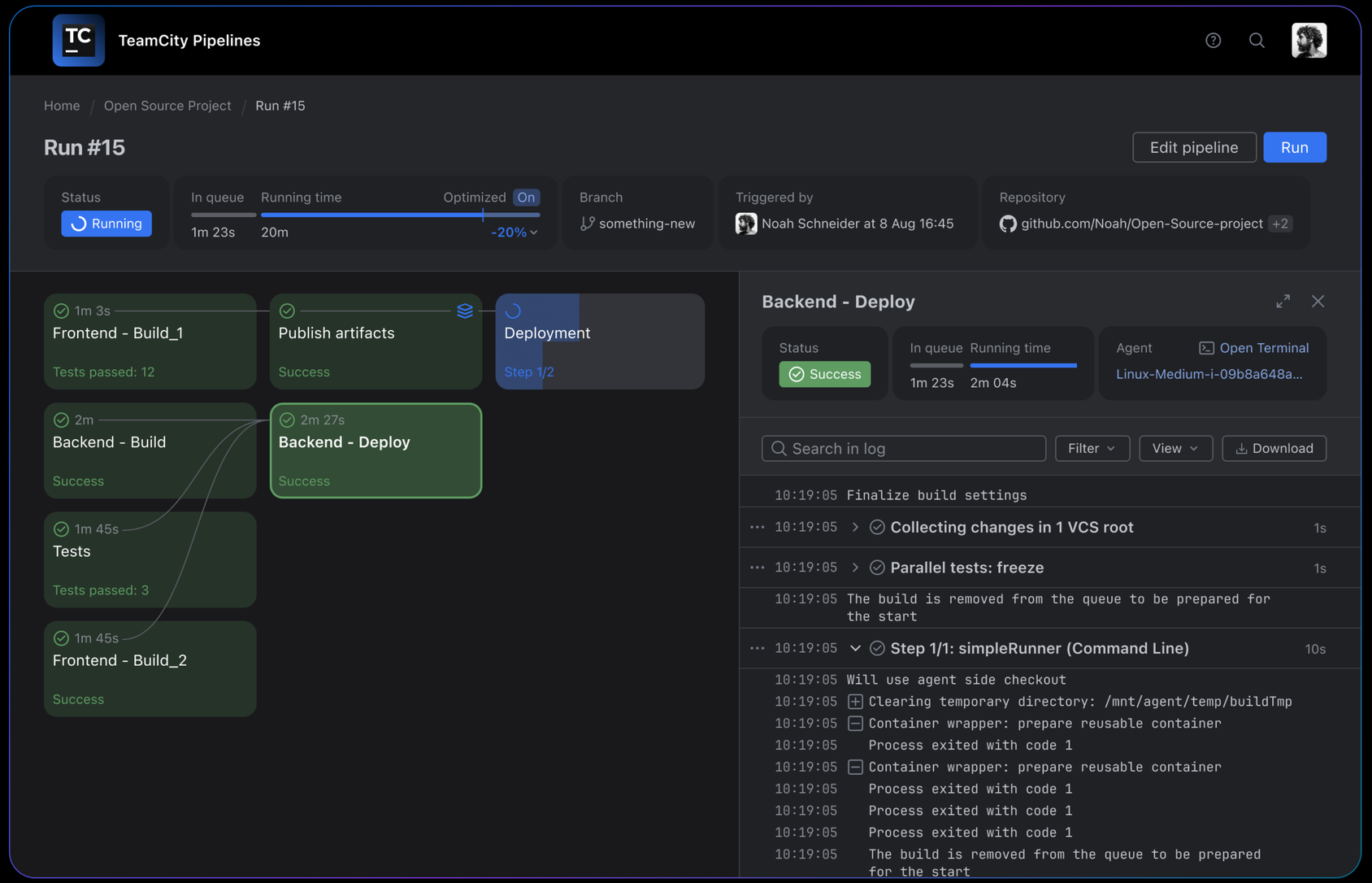


Tools for C++ painpoints

–

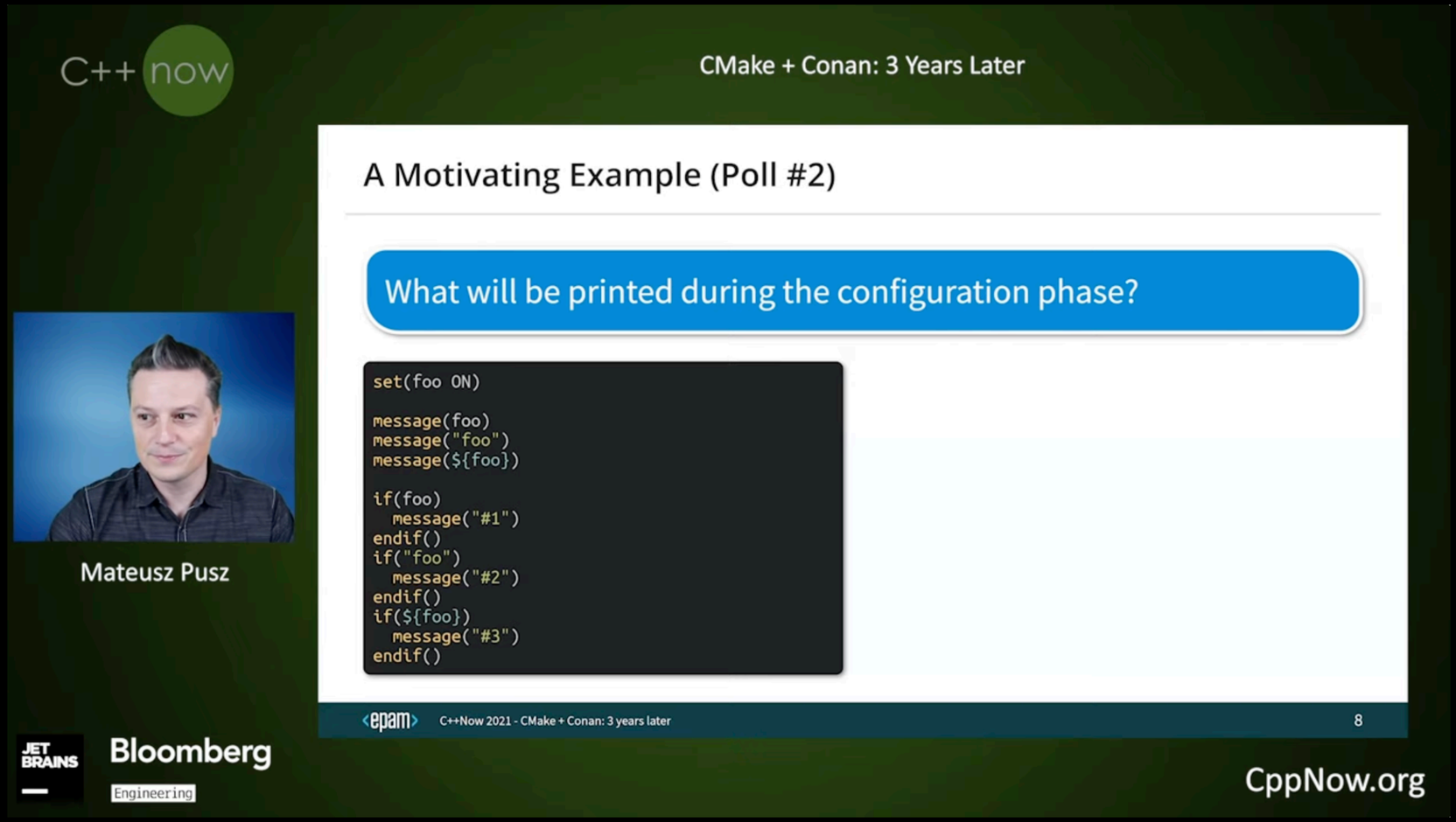
#3 CI pipelines

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



Tools for C++ painpoints

#4 Managing CMake

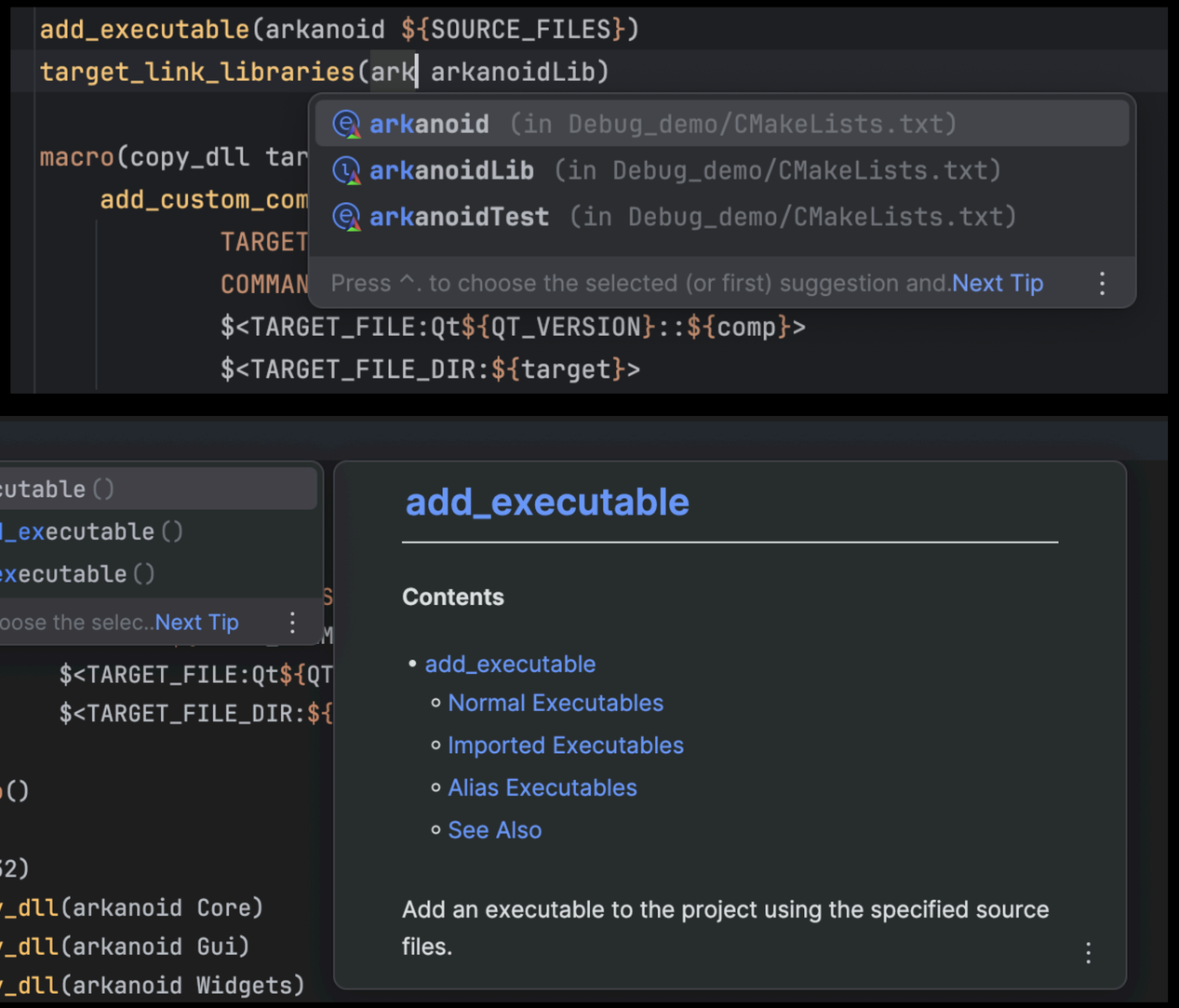


[CMake + Conan: 3 Years Later - Mateusz Pusz - [CppNow 2021]]

Tools for C++ painpoints

#4 Managing CMake

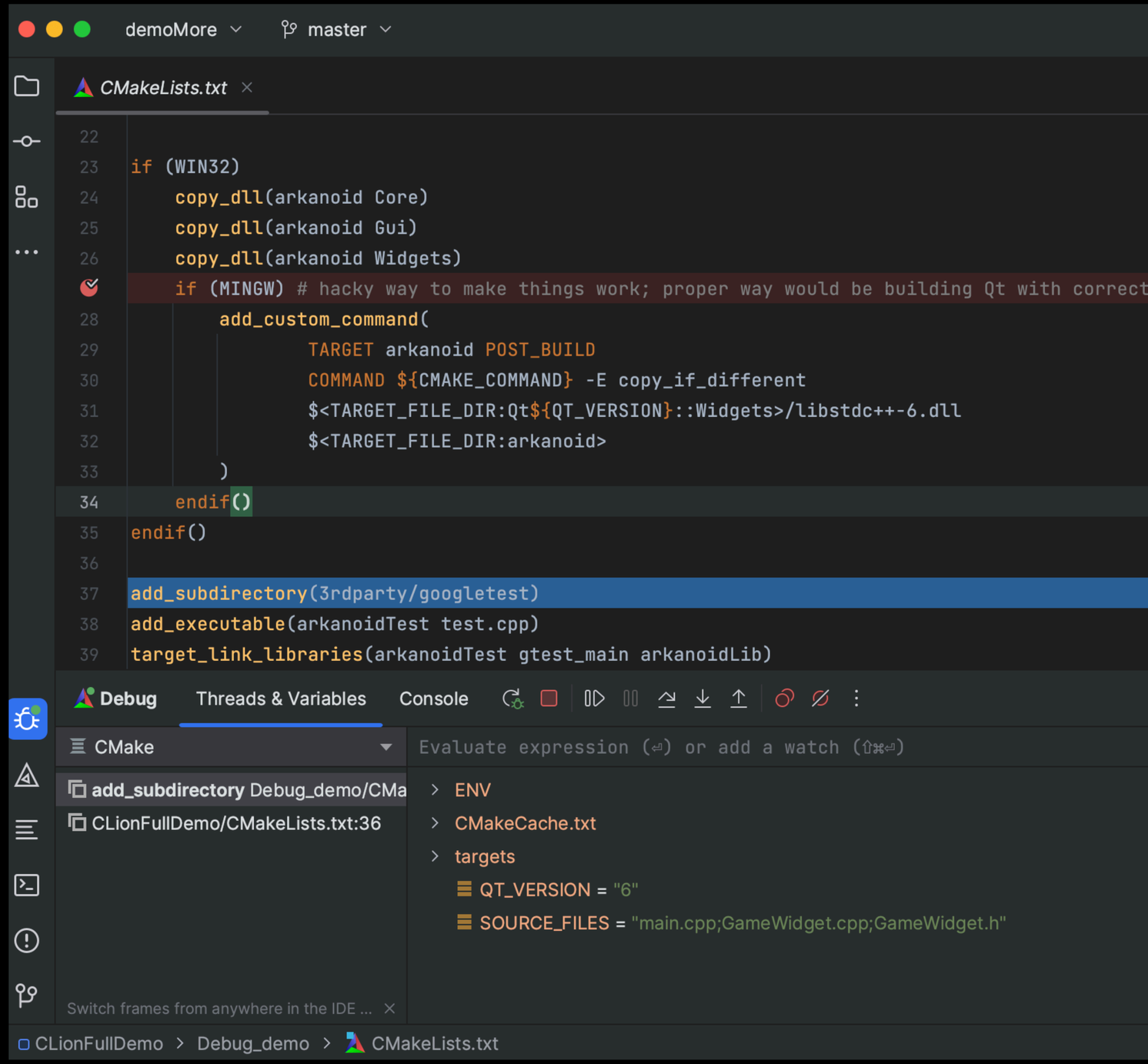
- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI



Tools for C++ painpoints

#4 Managing CMake

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

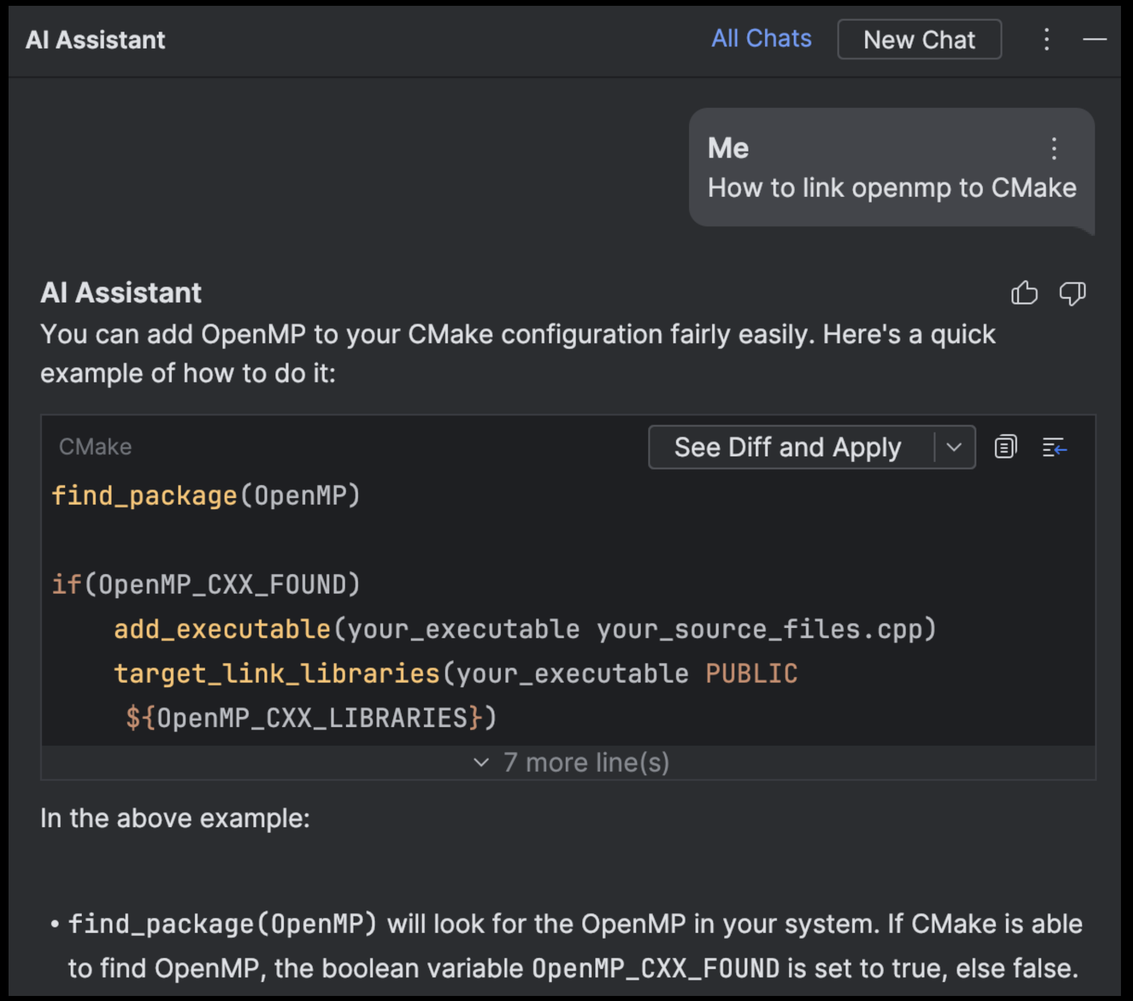


Tools for C++ painpoints

—

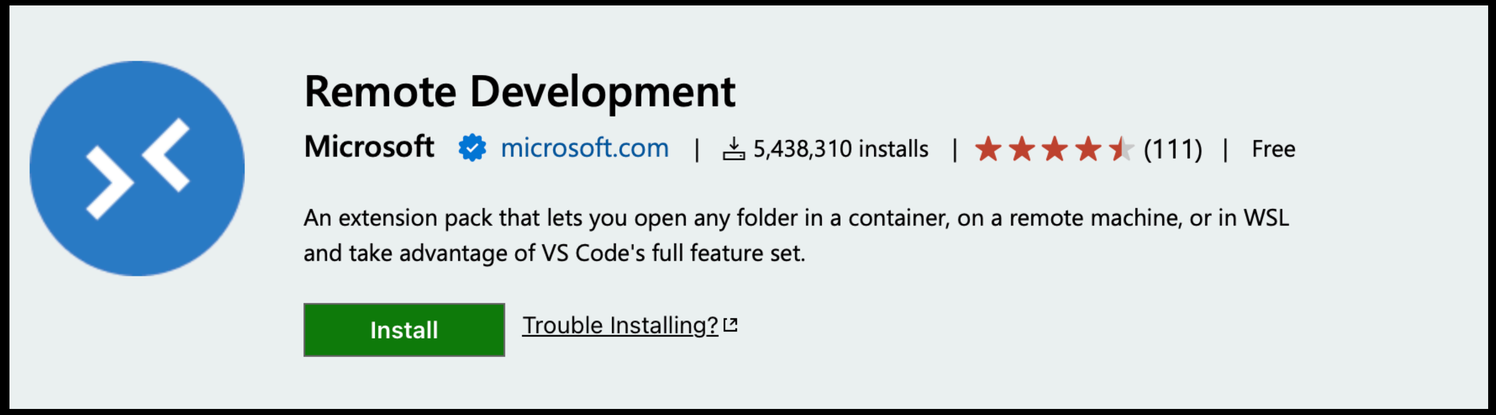
#4 Managing CMake

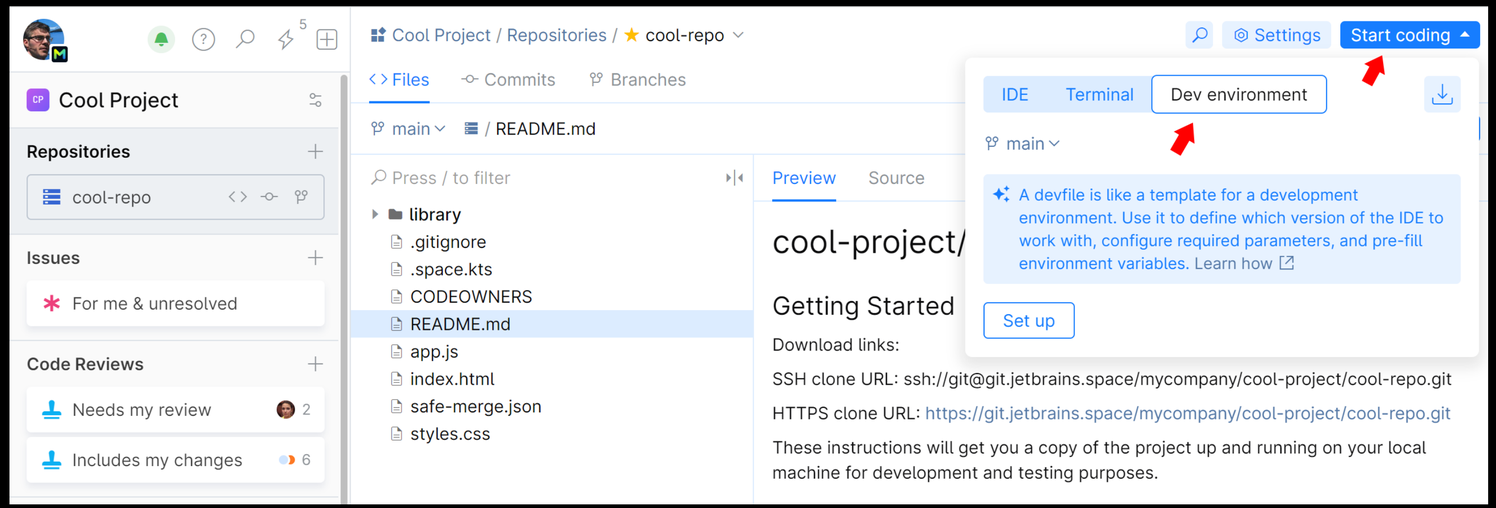
- CMake as a language
- Debugging CMake
- Profiling CMake
- CMake Presets
- CMake File API
- Help from AI



Tools for C++ painpoints
—

#6 Setting up dev env
- start instantly
- onboard easily
- zero configuration time
- local & remote
- reproducible dev artefacts





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;
    }
};
```

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

Declared in: arrayIndexOutOfBounds.cpp

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

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;
}
```

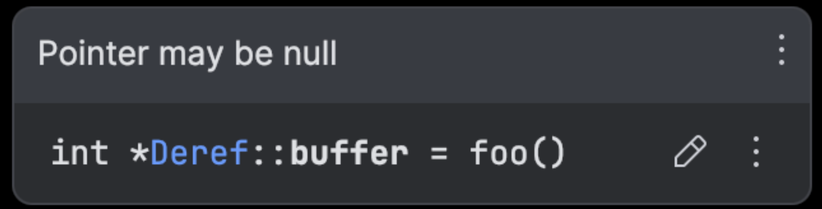
Local variable 'int_ptr' may point to deallocated memory

```
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;
    }
};
```



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 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
 hda: 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)
```

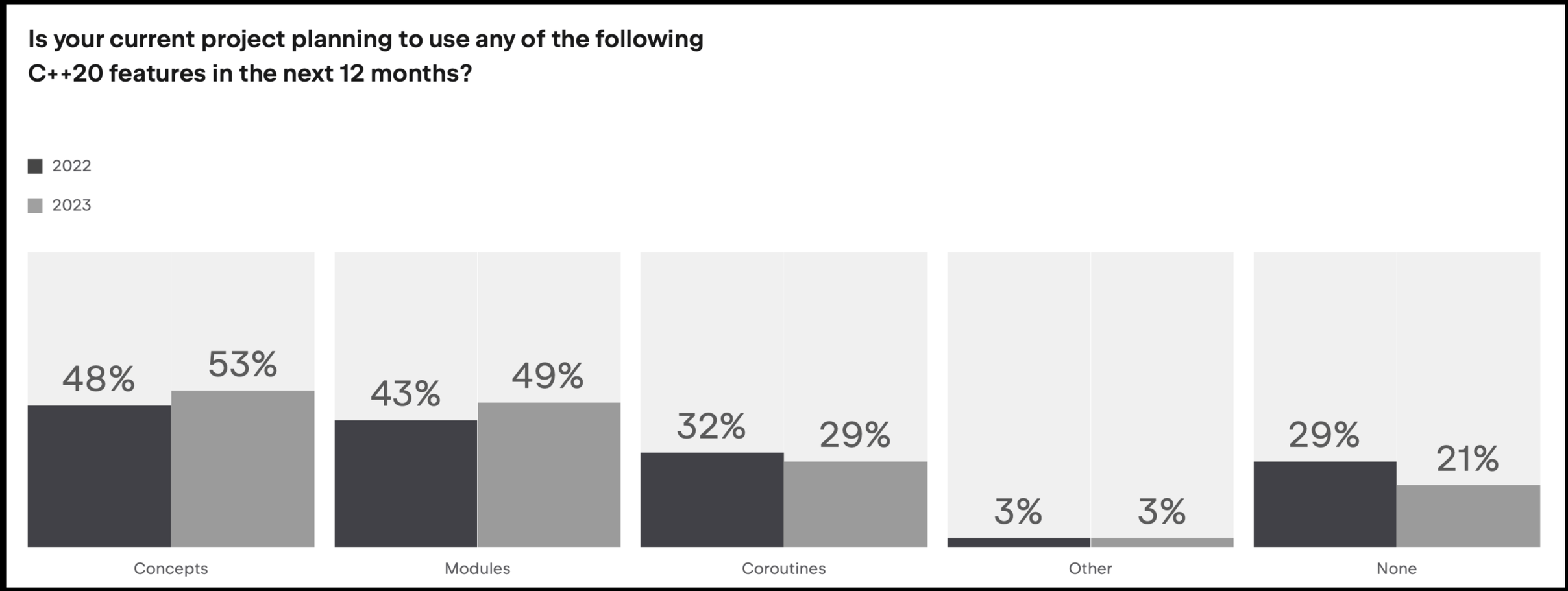
Tools and language evolution: Modules

–

CMake 3.26: support for modules was added

into file-api

Tools and language evolution: Modules

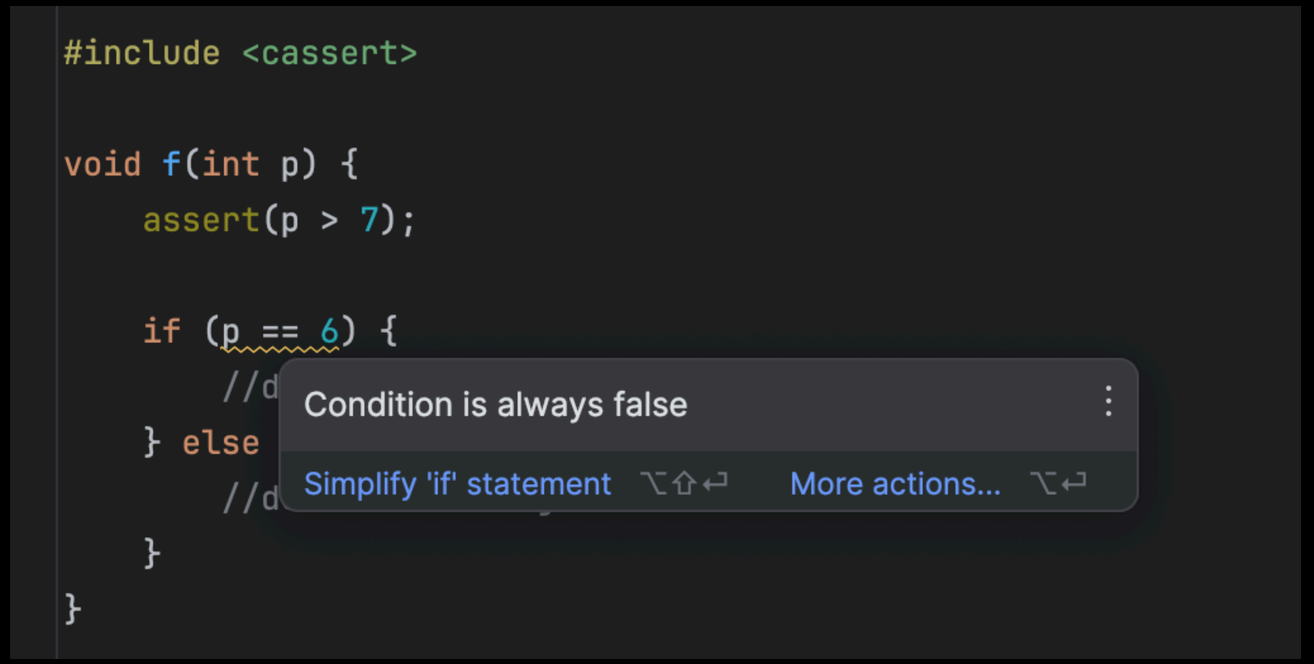


Tools and language evolution: Contracts

–

C++26 ?

Contracts



Tools and language evolution: Reflection

C++26 ?

Reflection

Tools and language evolution: Reflection
—

C++26 ?

Reflection

```cpp
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 ⟺ Package Manager Interoperation.
3. Minimum set of recognized file extensions.
4. Tool introspection.
5. Portable diagnostics format via SARIF.
6. Command line portability.

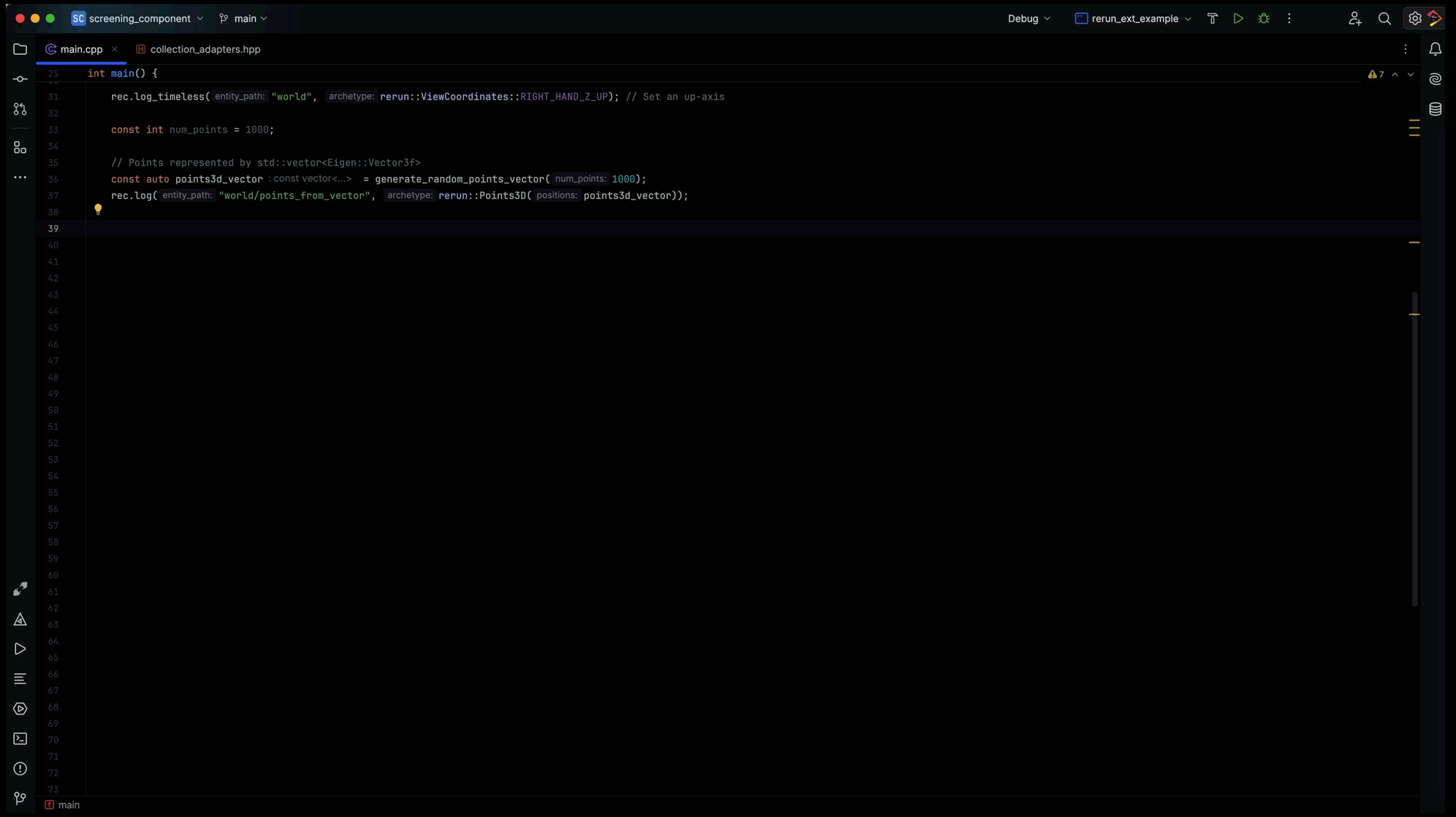
C++ Painkillers

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



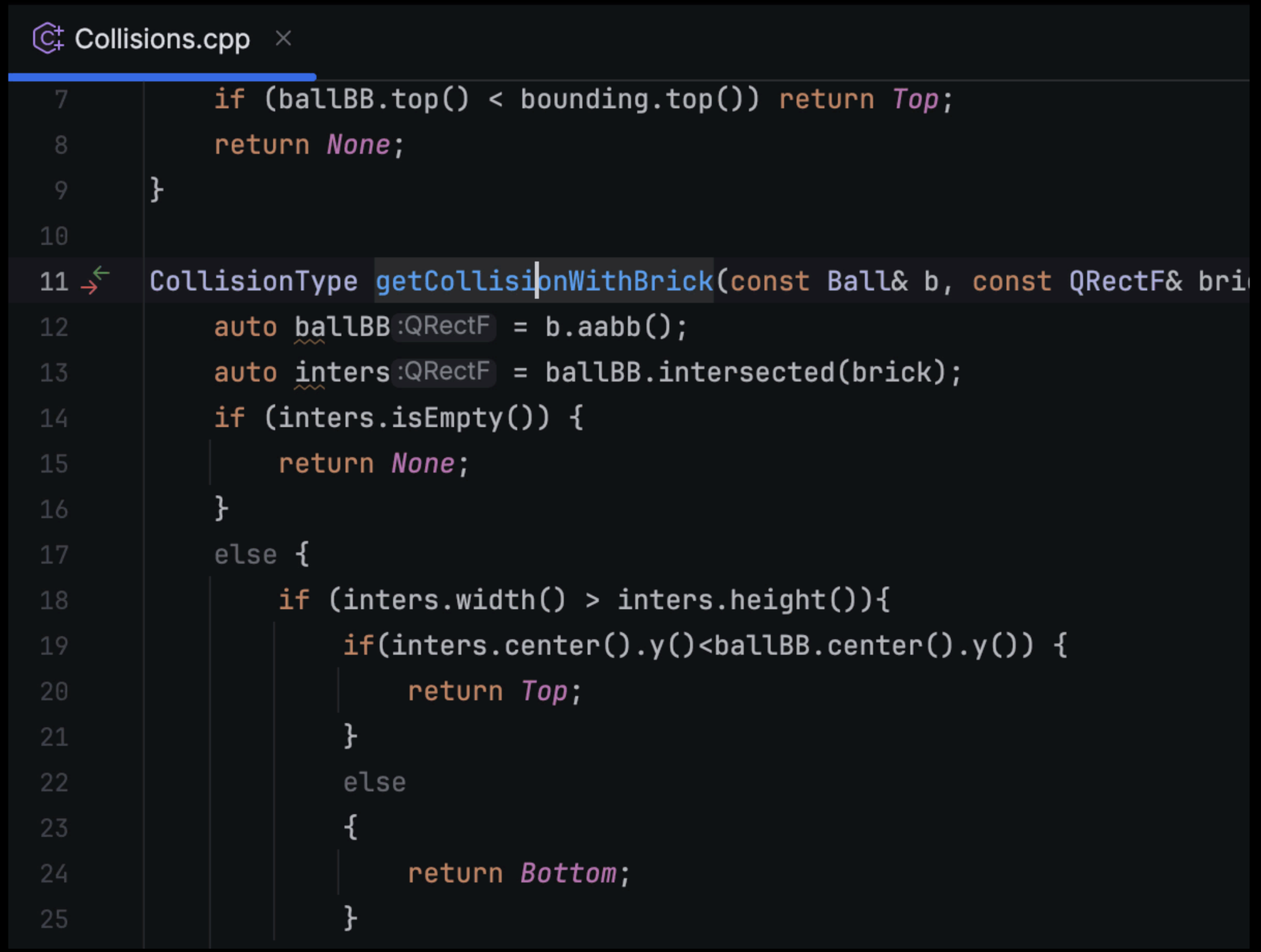
Era of AI
–

Improving the code

Era of AI
–

Summarizing for

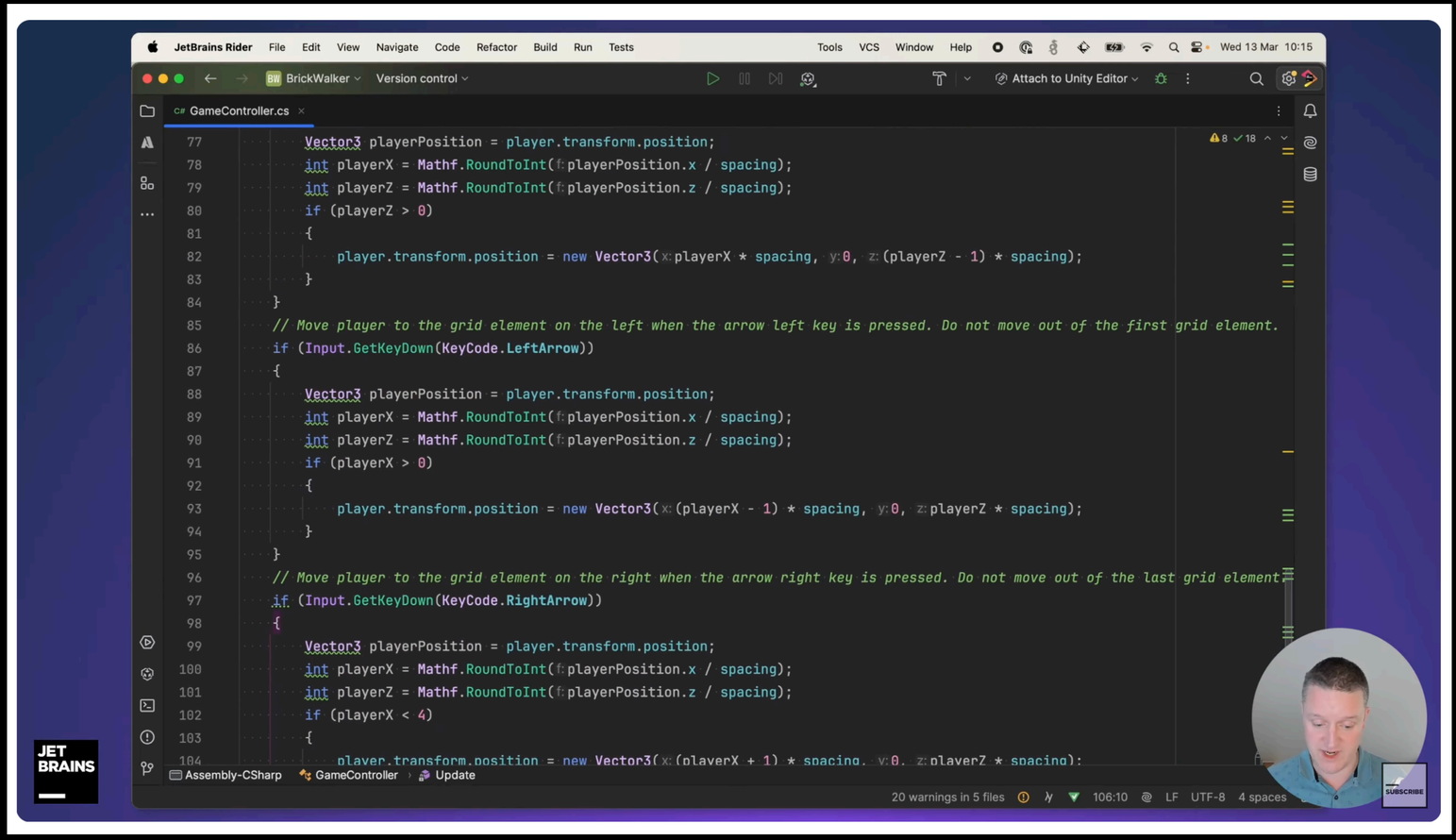
VCS or documentation



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

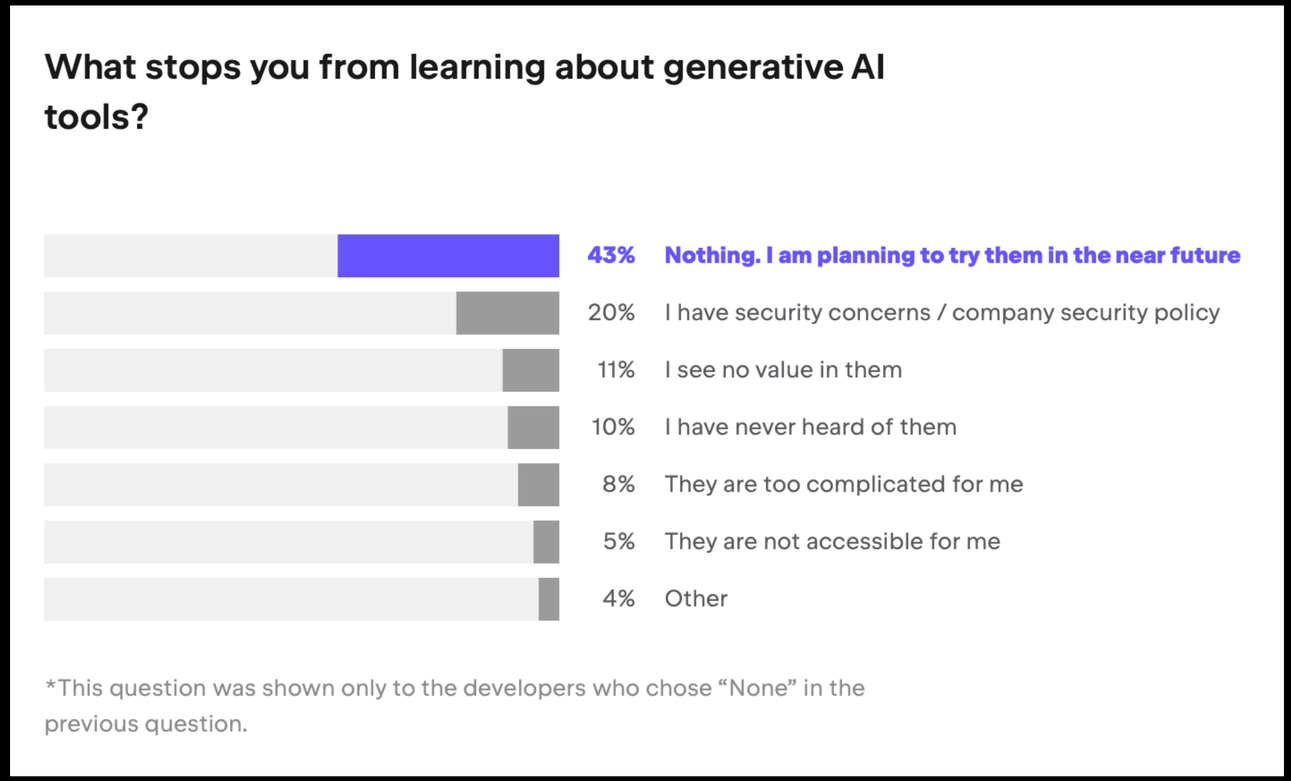


Era of AI: real-world examples

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.



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