Code Analysis++

Anastasia Kazakova
JetBrains
@anastasiak2512

NDC { TechTown }

About me

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

C++ Tools PMM

@anastasiak2512

Why do we need Code Analysis?

–

Software Quality

What is Software Quality?

Readability

Maintainability

Tools

Testing technologies

Repeatable tests

Security

Less UB

Robustness

Good API

Solves business tasks

Reliability

Size

Efficiency (resources)

Expressive code

Simplicity

Work as intended

Documented

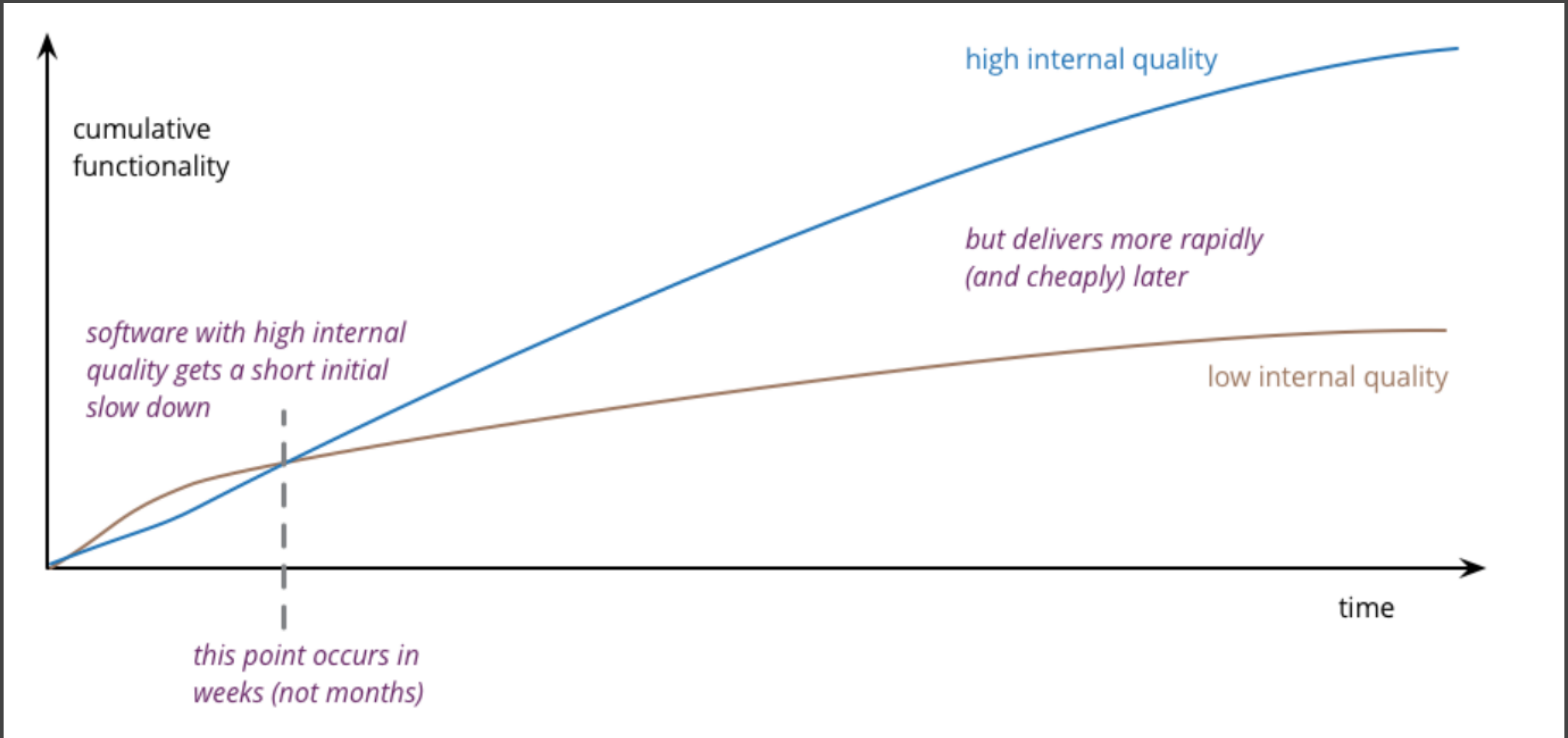
Reviews

What is Software Quality?

- Readability
- Maintainability
- Tools
- Testing technologies
- Repeatable tests
- Security
- Less UB
- Robustness
- Good API
- Solves business tasks
- Reliability
- Size
- Efficiency (resources)
- Expressive code
- Simplicity
- Work as intended
- Documented
- Reviews

A trade-off between quality and cost of development

-



C++ frustration

-

2022 Annual C++ Developers Survey “Lite”

(the results were nearly the same in 2021 and 2020)

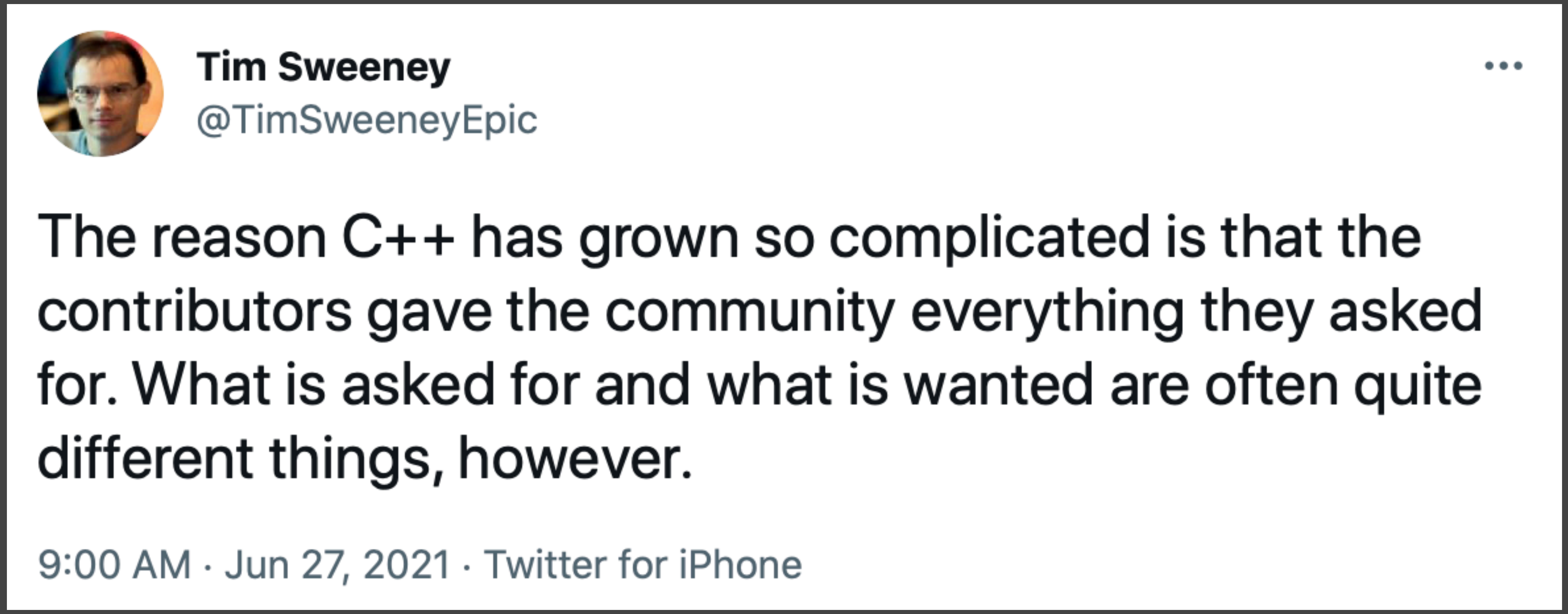
Frustration Points	Major %
Managing libraries my application depends on	48 %
Build times	44 %
Setting up a CI pipeline from scratch	34 %
Managing CMake projects	29 %
Setting up a dev env from scratch	28 %
Concurrency safety: Races, deadlocks, performance bottlenecks	25 %
Parallelism support	21 %
Managing Makefiles	20 %
Managing MSBuild projects	18 %
Debugging issues in my code	18 %
Memory safety: out-of-bounds safety issues	15 %
Memory safety: use-after-delete/free safety issues	14 %
Security issues: disclosure, vulnerabilities, exploits	10 %
Type safety: using an object as the wrong type	9 %
Memory safety: memory leaks	9 %
Moving existing code to the latest language standard	7 %

The answer to life, the universe, and everything

```cpp
template<class T, int ... X>
T pi(T(X...));
int main() {
    return pi<int, 42>;
}
```

```cpp
int main() {
    return 42;
}
```

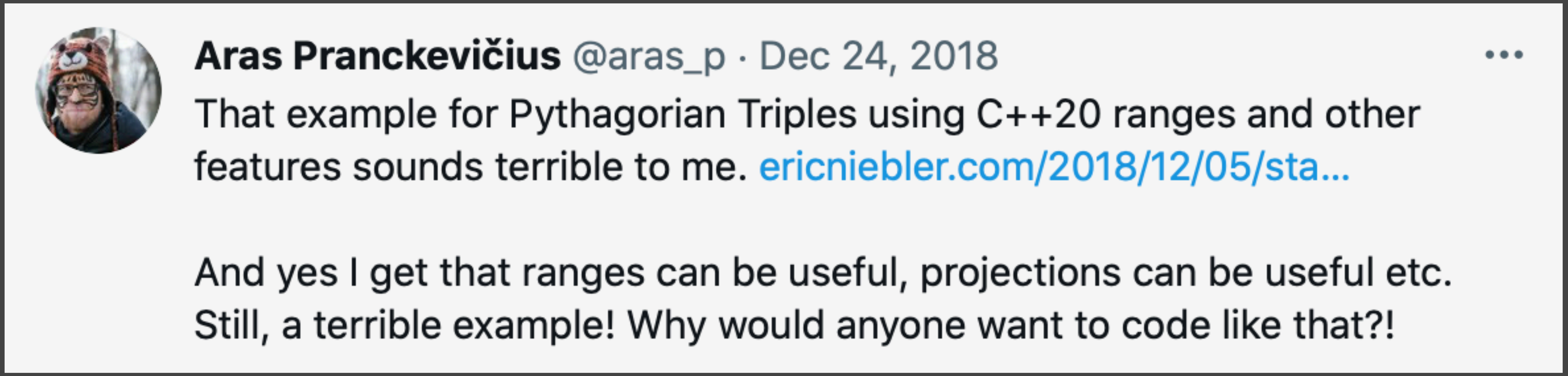
10 ways to do 1 thing



Tim Sweeney
@TimSweeneyEpic

The reason C++ has grown so complicated is that the contributors gave the community everything they asked for. What is asked for and what is wanted are often quite different things, however.

9:00 AM · Jun 27, 2021 · Twitter for iPhone



Aras Pranckevičius @aras_p · Dec 24, 2018

That example for Pythagorian Triples using C++20 ranges and other features sounds terrible to me. ericniebler.com/2018/12/05/sta...

And yes I get that ranges can be useful, projections can be useful etc. Still, a terrible example! Why would anyone want to code like that?!

10 ways to do 1 thing

—

“With a sufficient number of uses of an API, it does not matter what you promise in the contract:
all observable behaviours of your system will be depended on by somebody.”

(Hyrums Law, Software Engineering at Google,
by Titus Winter, Tom Manshrek, Hyrum Wright)

Undefined Behavior

The story of one vulnerability, which affected the 2.6.30 kernel

```c
static unsigned int tun_chr_poll(struct file *file,
poll_table * wait) {
    struct tun_file *tfile = file->private_data;
    struct tun_struct *tun = __tun_get(tfile);
    struct sock *sk = tun->sk;
    unsigned int mask = 0;

    if (!tun)
        return POLLERR;
```

Why do we need Code Analysis?

-

1. Improve software quality
2. Decrease developer frustration
3. Catch C++ pain-points. Avoid UB

W(all) W(extra) W(error)

```
if (MSVC)
    add_compile_options(/W4 /WX)
else()
   add_compile_options(-Wall -Wextra -Werror)
endif()
```

```
# not CPPFLAGS
CXXFLAGS += -Wall -Wextra -Werror
```

W(all) W(extra) W(error)

DevEco 2020
DevEco 2021
isocpp 2021
90 %
80 %
70 %
60 %
50 %
40 %
30 %
20 %
10 %
0 %
53 %
55 %
78 %
35 %
31 %
39 %
29 %
36 %
41 %
7 %
9 %
9 %
9 %
33 %
7 %
6 %
5 %
6 %
13 %
3 %
2 %
9 %
10 %
7 %
CMake
msbuild
Makefiles
Xcode
Ninja
Custom
Qmake
Autotools
Gradle

Can C++ do better?

std::source_location since C++20

to avoid macro-styled logging and tracing

https://en.cppreference.com/w/cpp/utility/source_location

```
void log(std::string_view message, std::source_location loc = std::source_location::current());

std::ostream trace(std::string_view msg, std::source_location location = std::source_location::current()) {
    return std::cout << location.file_name() <<':' << location.line() <<' '<< msg;
}
```

Can C++ do better?

-

Contracts P2521, P2388, P2461

Can C++ do better?

Lifetime safety: P1179

- Owner & Pointer concepts
- Built-in compiler checks
- GSL library annotations

```
void dangling_iterator()
{
    std::vector<int> v = { 1, 2, 3 };
    auto it = v.begin();
    *it = 0;
    v.push_back(4);
    *it = 0;        realloc => dangling iterator
}
```

Can C++ do better?

—

Lifetime safety: P1179

- Clang -Wlifetime, LLVM implementation gives 5% overhead
- C++ Core Check Lifetime Rules in Visual Studio 2019+
- Data Flow Analysis in CLion 2021.2+

Data Flow Analysis

DFA takes into account:

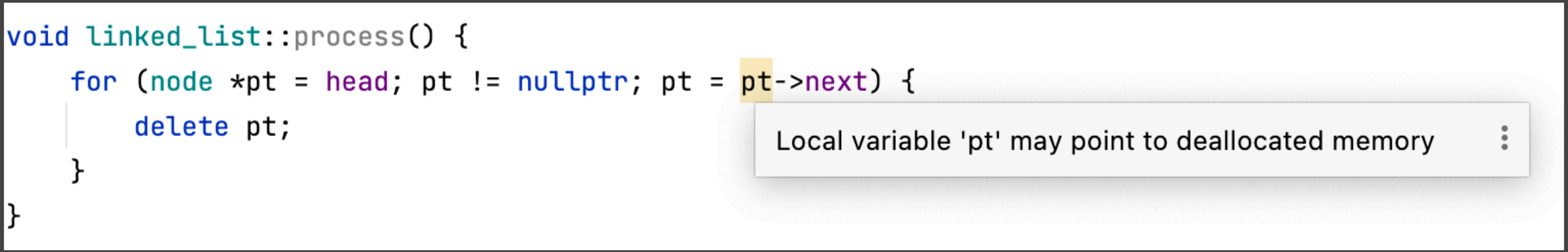
- Function parameters/arguments
- Function return value
- Fields and global variables

DFA’s output:

- Value ranges for variables

Data Flow Analysis
-

Local, scope = function bodies



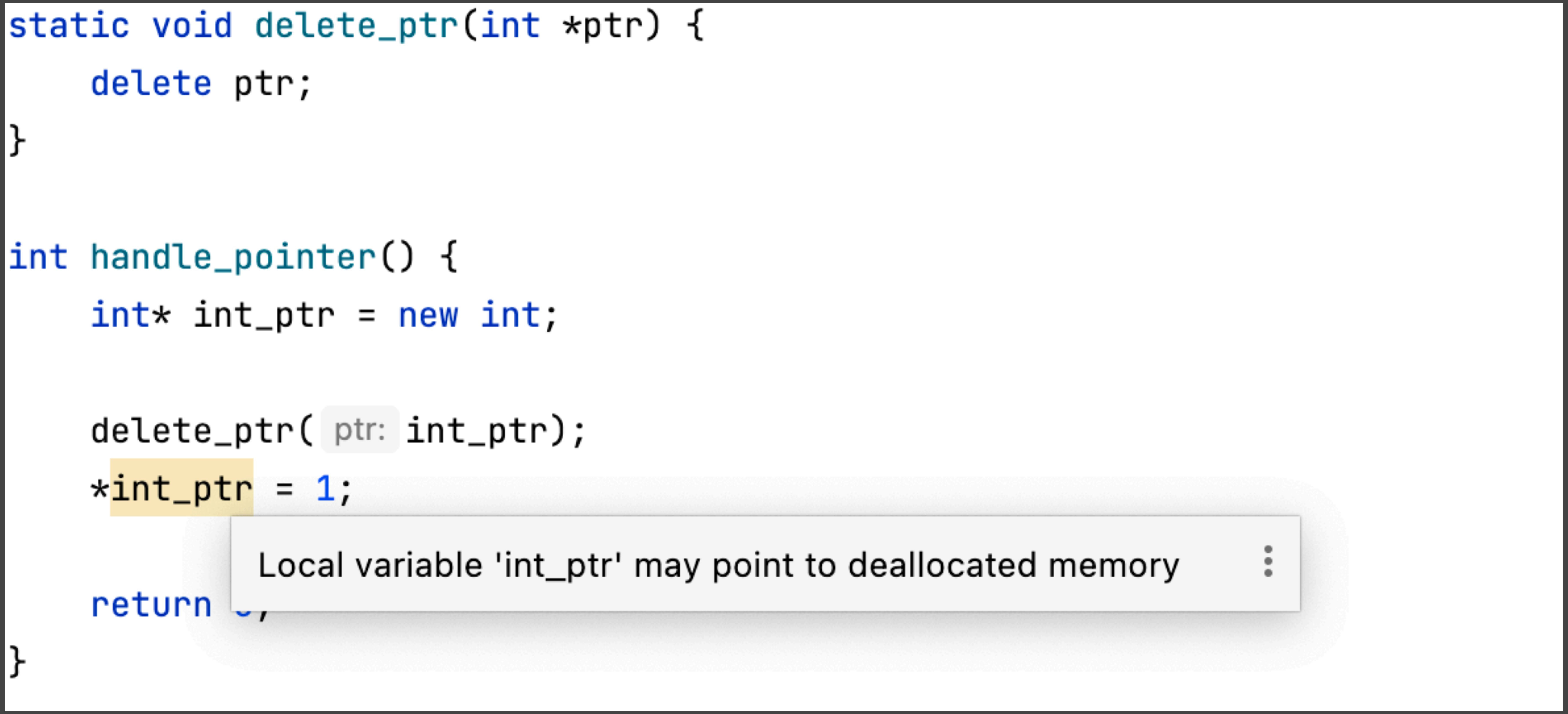
```
void linked_list::process() {
    for (node *pt = head; pt != nullptr; pt = pt->next) {
        delete pt;
    }
}
```

Local variable 'pt' may point to deallocated memory

Data Flow Analysis

-

Global, scope = translation unit



```
static void delete_ptr(int *ptr) {
    delete ptr;
}

int handle_pointer() {
    int* int_ptr = new int;

    delete_ptr( ptr: int_ptr);
    *int_ptr = 1;

    return 0;
}
```

Local variable 'int_ptr' may point to deallocated memory

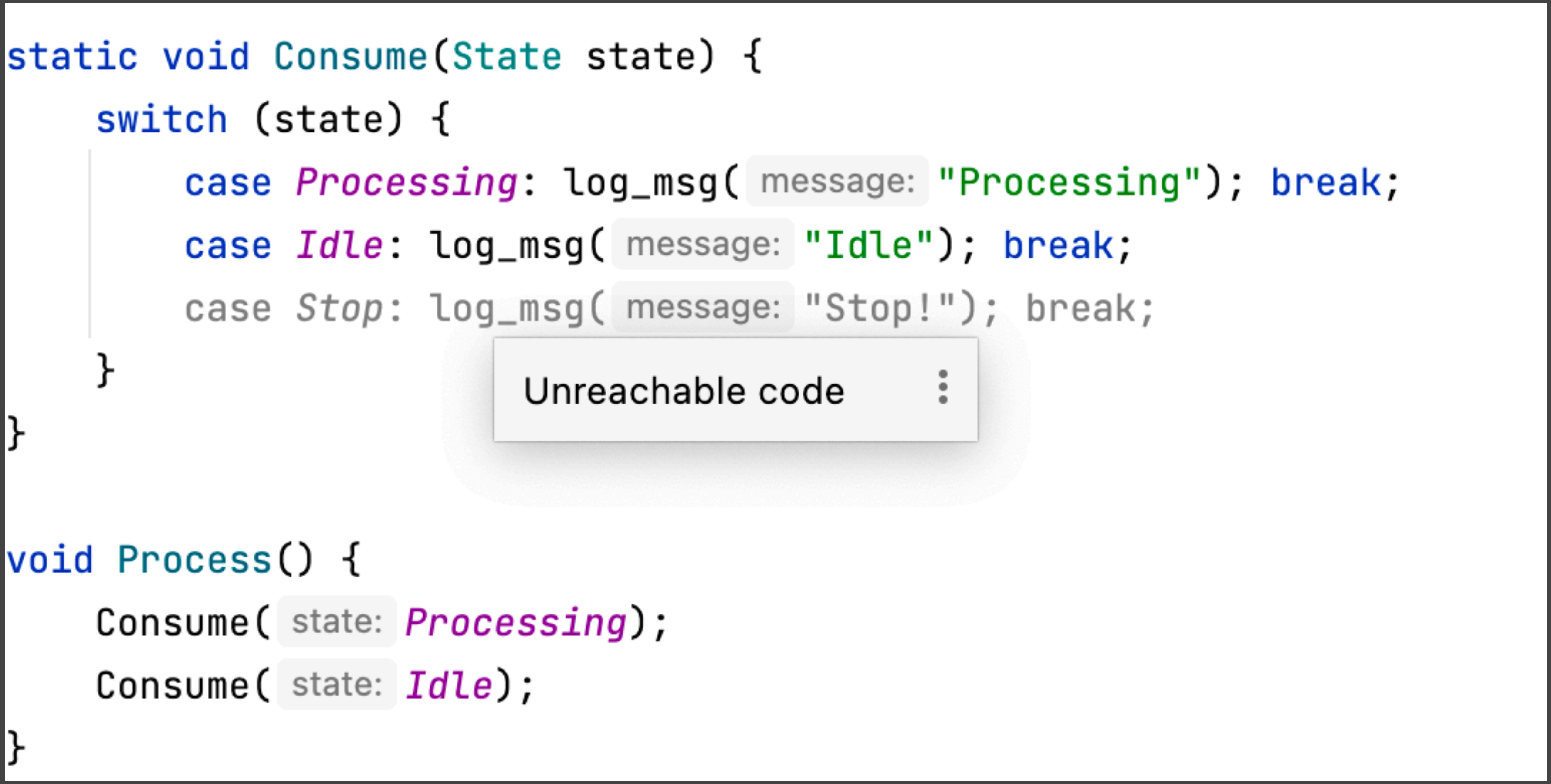
Data Flow Analysis

-

- Constant conditions
- Dead code
- Endless loops
- Infinite recursion
- Unused values

- Null dereference
- Escape analysis
- Dangling pointers
- Index out of bounds

Data Flow Analysis



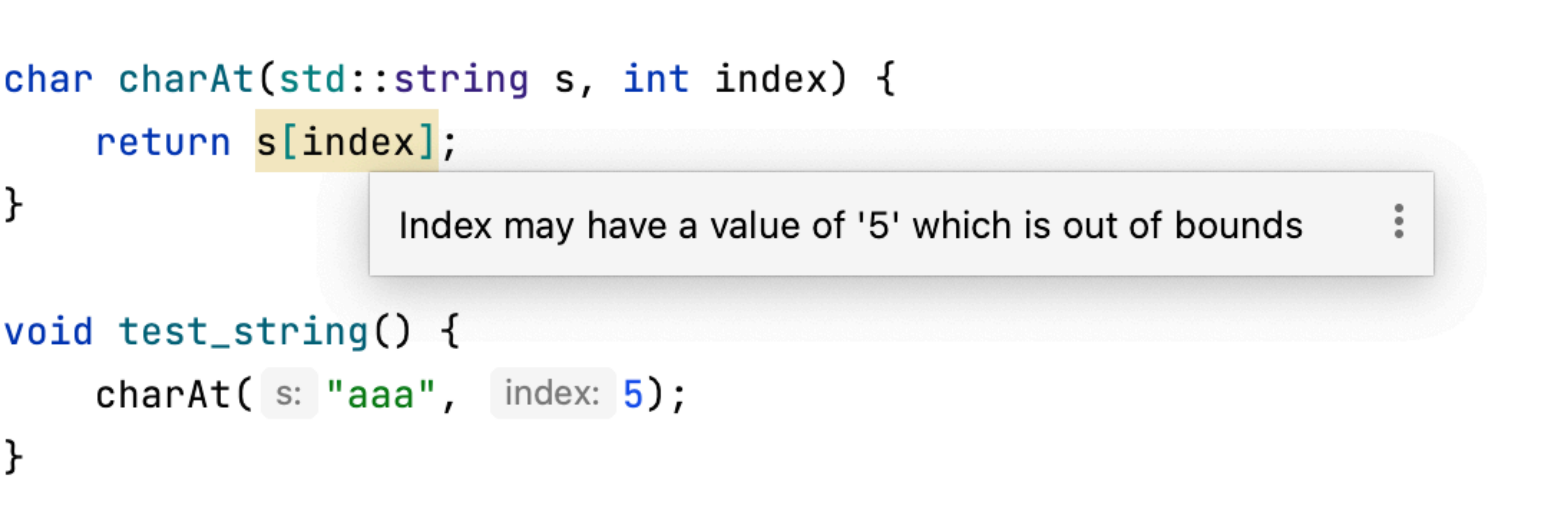
```
static void Consume(State state) {
    switch (state) {
        case Processing: log_msg( message: "Processing"); break;
        case Idle: log_msg( message: "Idle"); break;
        case Stop: log_msg( message: "Stop!"); break;
    }
}

void Process() {
    Consume( state: Processing);
    Consume( state: Idle);
}
```

Unreachable code

Data Flow Analysis

–



```
char charAt(std::string s, int index) {
    return s[index];
}

void test_string() {
    charAt( s: "aaa",  index: 5);
}
```

Index may have a value of '5' which is out of bounds

Data Flow Analysis

–

CLion:

- Local DFA since 1.x
- Local DFA on Clang since 2020.1
- Global (TU) DFA since 2021.1
- Index out-of-bound check since 2022.2
- …

Data Flow Analysis

–

Cross Translation Unit (CTU) Analysis

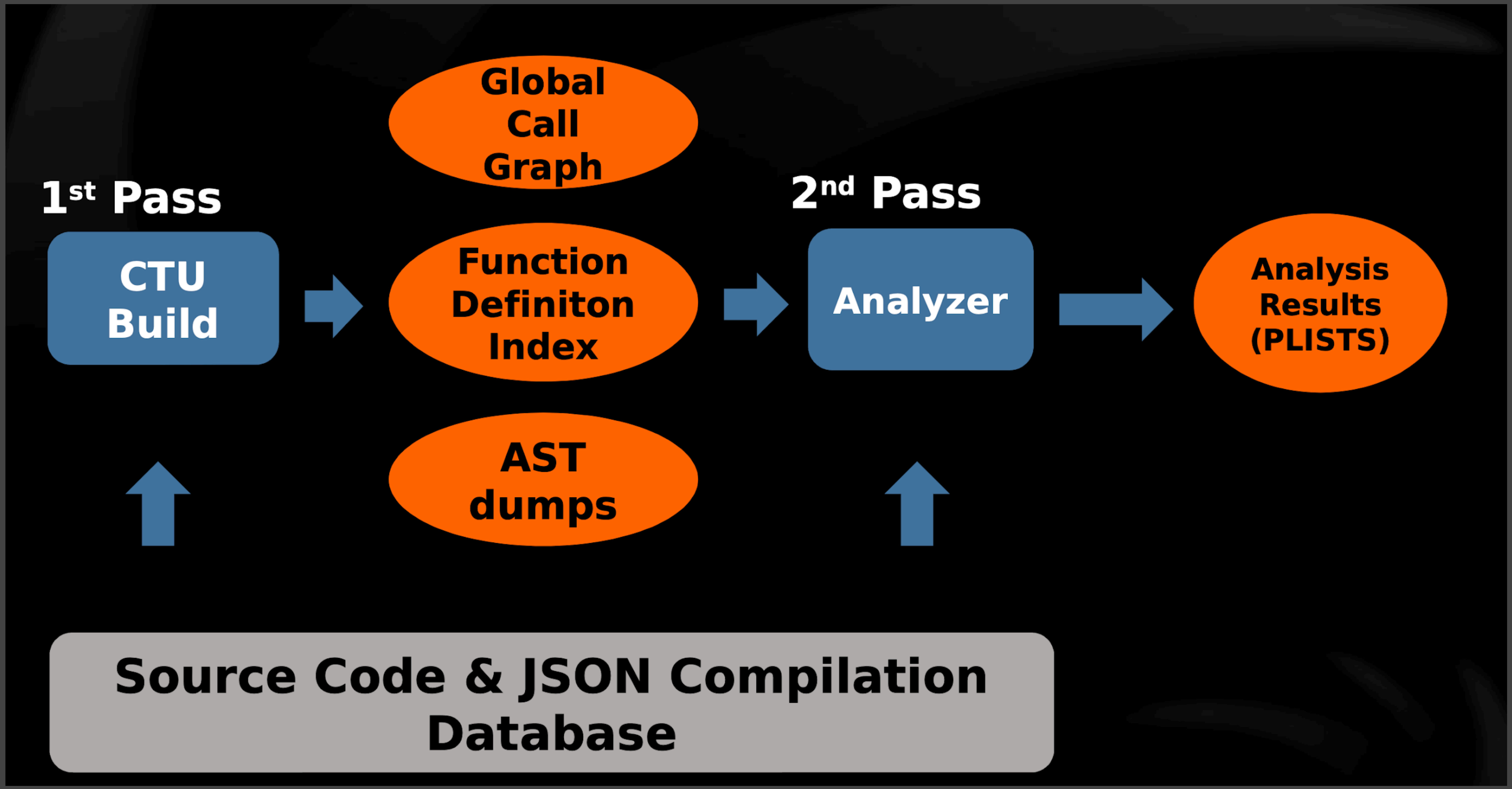
- Cross Translation Unit (CTU) Analysis (LLVM doc)
- CodeChecker https://github.com/Ericsson/codechecker

Data Flow Analysis

–

How to do it?

- Pre-dumped PCH
- Generate AST on-demand



C++ Core Guidelines
-

"Within C++ is a smaller, simpler, safer language struggling to get out."

(c) Bjarne Stroustrup

https://github.com/isocpp/CppCoreGuidelines

C++ Core Guidelines: Toolable

-

- *F.16: For “in” parameters, pass cheaply-copied types by value and others by reference to const*
  - E1: Parameter being passed by value has a size > 2 * sizeof(void*) ⇒ suggest reference to const
  - E2. Parameter passed by reference to const has a size < 2 * sizeof(void*) ⇒ suggest passing by value
  - E3. Warn when a parameter passed by reference to const is moved

C++ Core Guidelines: Abstract

—

- *F.1: “Package” meaningful operations as carefully named functions*
  - Detect identical and similar lambdas used in different places

- *F.2: A function should perform a single logical operation*
  - >1 “out” parameter - suspicious, >6 parameters - suspicious ⇒ heuristic
  - Rule of one screen: 60 lines by 140 characters ⇒ heuristic

- *F.3: Keep functions short and simple*
  - Rule of one screen ⇒ heuristic
  - Cyclomatic complexity “more than 10 logical path through” ⇒ heuristic

C++ Core Guidelines: Abstract

- Requires search for duplicates
- Takes into account syntax variations
- Requires heuristics

```
template<class T, int ... X>
T pi(T(X...));
int main() {
    return pi<int, 42>;
}
```

C++ Core Guidelines: Interfaces

-

- *F.4: If a function might have to be evaluated at compile time, declare it constexpr*
- *F.5: If a function is very small and time-critical, declare it inline*
- *F.6: If your function may not throw, declare it noexcept*

C++ Core Guidelines: Tools

-

- Guidelines Support Library
- Visual Studio C++ Core Guidelines checkers
- Clang-Tidy: cppcoreguidelines-*
- Sonar (Qube, Lint, Cloud)
- CLion
- ReSharper C++

No single tool to check them all!

DSL analysers

-

- MISRA / AUTOSAR for embedded
- Clazy for Qt
- UnrealHeaderTool for Unreal Engine code
- and more

MISRA

Certification stage	Development stage
Must have	Good to have
Change: high cost	Change: low cost
Certified lists of checks	Optional / customised lists
Fail / Pass	Details & Quick-Fixes

MISRA

-

- CLion MISRA
  - MISRA C 2012 (63 / 166)
  - MISRA C++ 2008 (65 / 211)
- SonarLint MISRA:
  - MISRA C 2004 (15 / 142)
  - MISRA C 2012 (11 / 166)
  - MISRA C++ 2008 (51 / 211)
- PVS-Studio

So many rules!

- C++ Core Guidelines
  - F.55: Don't use va_arg arguments
  - ES.34: Don't define a (C-style) variadic function
- MISRA
  - MISRA C:2004, 16.1 - Functions shall not be defined with a variable number of arguments.
  - MISRA C++:2008, 8-4-1 - Functions shall not be defined using the ellipsis notation.
- CERT
  - DCL50-CPP. - Do not define a C-style variadic function

Formatting rules

-

- ClangFormat
  - Standard in C++ nowadays
  - Breaking compatibility between versions
  - Fuzzy parsing

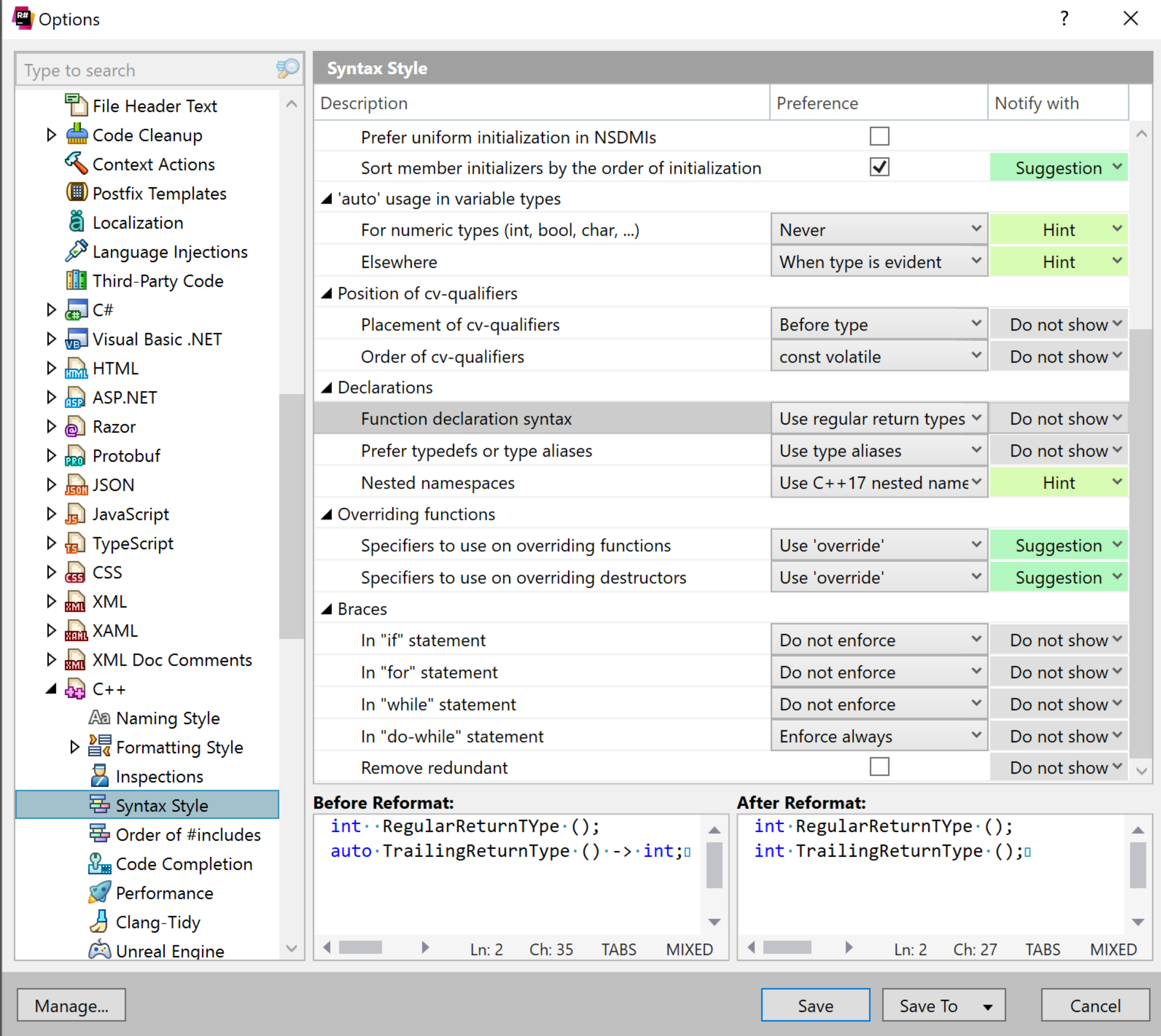
Naming rules

-

- Styles
  - camelCase, PascalCase, SCREAMING_SNAKE_CASE
  - Google style, LLVM, LLDB, Unreal Engine conversions
- Tooling
  - Checks & warnings
  - Rename refactoring
  - Integration into code generation

Syntax style rules –

- Auto: “Almost Always Auto”, “When Evident”, …
- “East const” vs. “West const”
- Typedefs vs. Type Aliases
- Trailing return types vs. regular
- Override, final, virtual



Options

Type to search

- File Header Text
- Code Cleanup
- Context Actions
- Postfix Templates
- Localization
- Language Injections
- Third-Party Code
- C#
- Visual Basic .NET
- HTML
- ASP.NET
- Razor
- Protobuf
- JSON
- JavaScript
- TypeScript
- CSS
- XML
- XAML
- XML Doc Comments
- C++
  - Naming Style
  - Formatting Style
  - Inspections
  - Syntax Style
  - Order of #includes
  - Code Completion
  - Performance
  - Clang-Tidy
  - Unreal Engine

Syntax Style

Description	Preference	Notify with
Prefer uniform initialization in NSDMIs	☐	
Sort member initializers by the order of initialization	☑	Suggestion
◢ 'auto' usage in variable types		
For numeric types (int, bool, char, ...)	Never	Hint
Elsewhere	When type is evident	Hint
◢ Position of cv-qualifiers		
Placement of cv-qualifiers	Before type	Do not show
Order of cv-qualifiers	const volatile	Do not show
◢ Declarations		
Function declaration syntax	Use regular return types	Do not show
Prefer typedefs or type aliases	Use type aliases	Do not show
Nested namespaces	Use C++17 nested name	Hint
◢ Overriding functions		
Specifiers to use on overriding functions	Use 'override'	Suggestion
Specifiers to use on overriding destructors	Use 'override'	Suggestion
◢ Braces		
In "if" statement	Do not enforce	Do not show
In "for" statement	Do not enforce	Do not show
In "while" statement	Do not enforce	Do not show
In "do-while" statement	Enforce always	Do not show
Remove redundant	☐	Do not show

Before Reformat:

```
int··RegularReturnTYpe·();
auto·TrailingReturnType·()·->·int;
```

Ln: 2 Ch: 35 TABS MIXED

After Reformat:

```
int·RegularReturnTYpe·();
int·TrailingReturnType·();
```

Ln: 2 Ch: 27 TABS MIXED

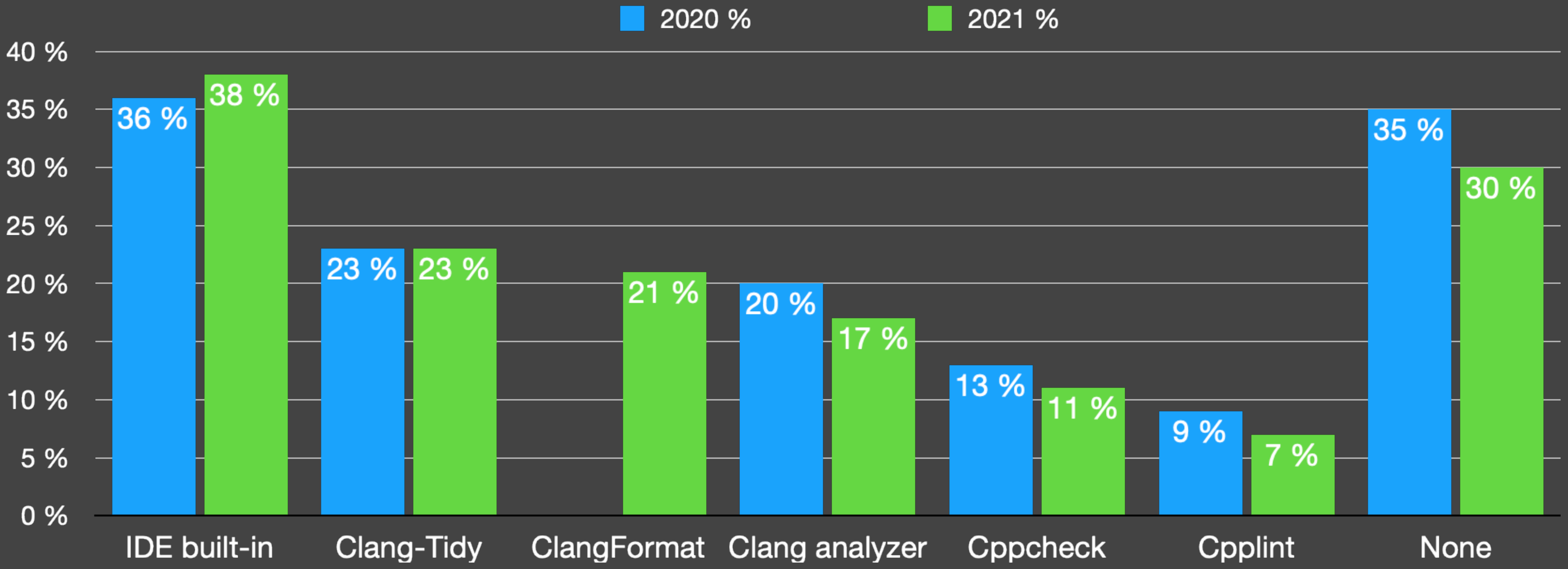
TL;DR: Code analysis++
-

1. Compiler errors & warnings
2. Language evolution
3. Data Flow Analysis
4. C++ Core Guidelines
5. DLS analysers
6. Style: Formatting, Naming, Syntax

How often do you rely on tools for code analysis?

What do you use for guideline enforcement or other code quality/analysis?

-

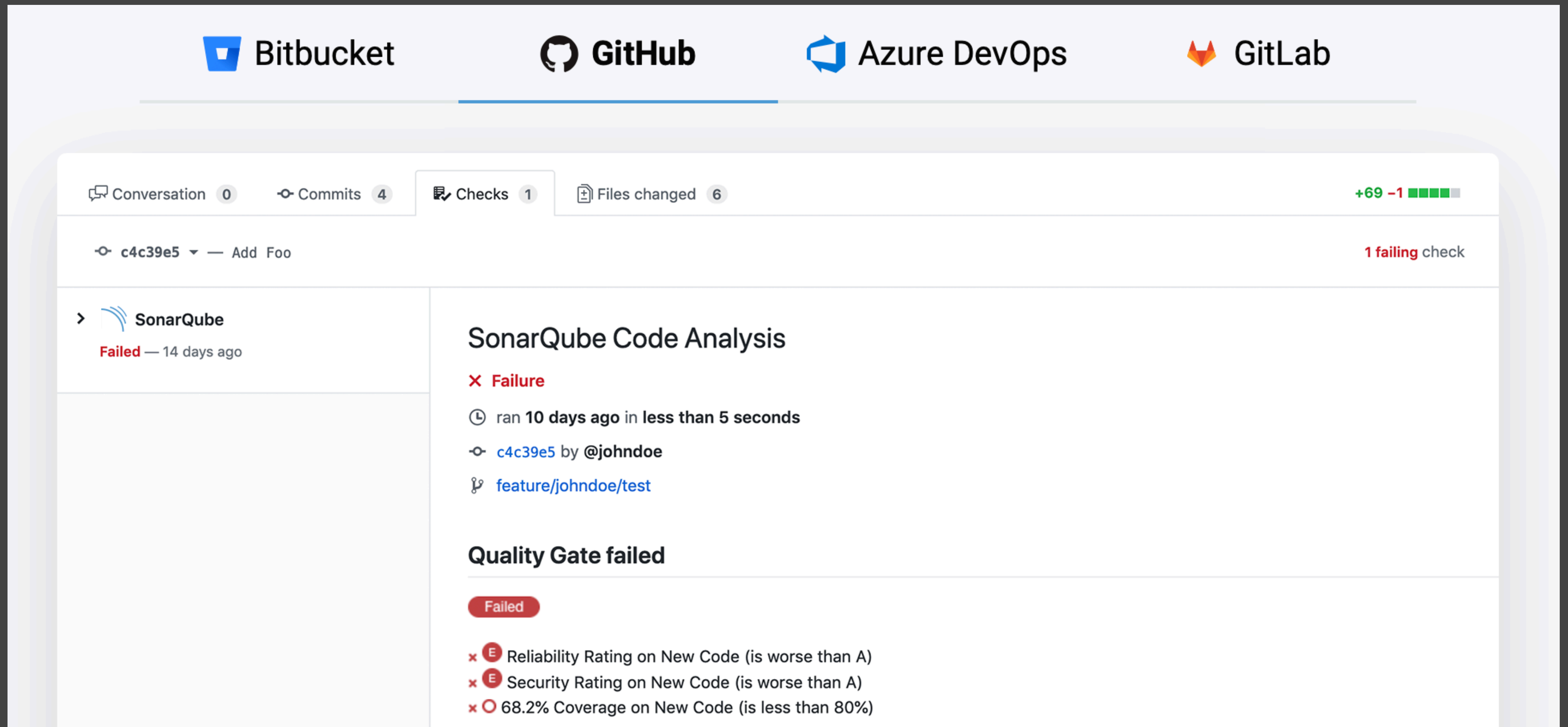


Code analysis on CI –

https://www.sonarsource.com

https://rules.sonarsource.com/cpp

- Linter 549 rules
- CI/CD integration
- Code reviews
- PR decorations



c4c39e5 — Add Foo

1 failing check

SonarQube
Failed — 14 days ago

SonarQube Code Analysis

✕ Failure

ran 10 days ago in less than 5 seconds

c4c39e5 by @johndoe

feature/johndoe/test

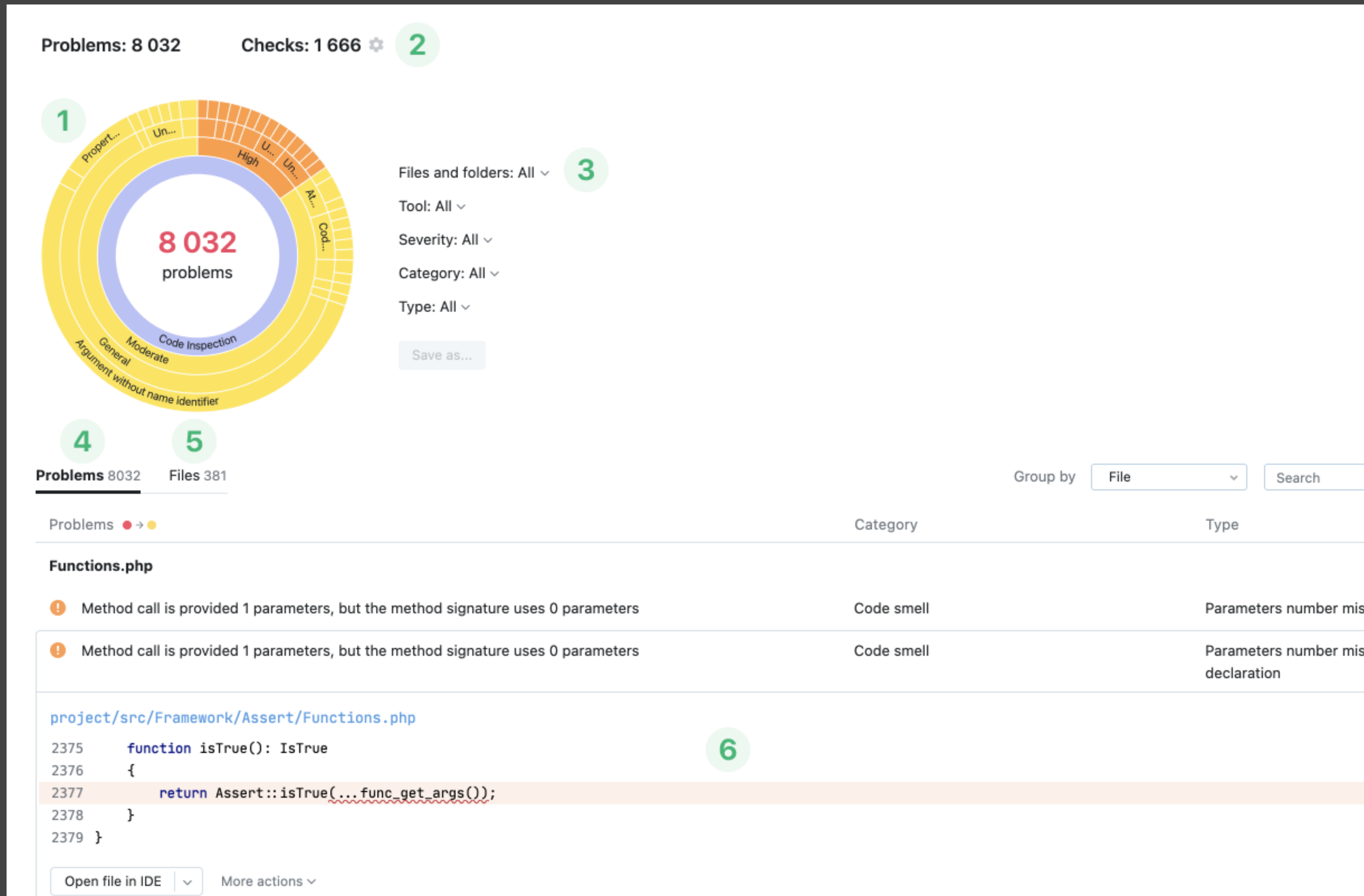
Quality Gate failed

Failed

- ✕ E Reliability Rating on New Code (is worse than A)
- ✕ E Security Rating on New Code (is worse than A)
- ✕ ○ 68.2% Coverage on New Code (is less than 80%)

https://www.jetbrains.com/qodana/

- Linters from JetBrains IDEs
- CI/CD integrations
- Java (released), Php/Python/JS (EAP), C++ (coming soon)



Problems: 8 032   Checks: 1 666

8 032 problems

Files and folders: All
Tool: All
Severity: All
Category: All
Type: All

Problems 8032   Files 381   Group by: File   Search

Problems	Category	Type
Functions.php		
Method call is provided 1 parameters, but the method signature uses 0 parameters	Code smell	Parameters number mis
Method call is provided 1 parameters, but the method signature uses 0 parameters	Code smell	Parameters number mis declaration

project/src/Framework/Assert/Functions.php

```
    function isTrue(): IsTrue
    {
        return Assert::isTrue(...func_get_args());
    }
}
```

Open file in IDE   More actions

References

1. Is High Quality Software Worth the Cost? By Martin Fowler https://martinfowler.com/articles/is-quality-worth-cost.html
2. 2022 Annual C++ Developer Survey “Lite" https://isocpp.org/files/papers/CppDevSurvey-2022-summary.pdf
3. The State of Developer Ecosystem 2021 by JetBrains https://www.jetbrains.com/lp/devecosystem-2021/cpp/ (2022 coming soon…)
4. Cross Translation Unit (CTU) Analysis in LLVM https://clang.llvm.org/docs/analyzer/user-docs/CrossTranslationUnit.html)
5. CodeChecker by Ericsson https://github.com/Ericsson/codechecker