Code Analysis++

Anastasia Kazakova

About me

- Anastasia Kazakova, @anastasiak2512
- C++ Dev: Embedded, Networking
- C++ Tools PMM and .NET Tools Marketing Lead, JetBrains
- St. Petersburg C++ UG: https://www.meetup.com/St-Petersburg-CPP-User-Group/
- C++ Russia: https://cppconf.ru/en/

Intro

- Intro
- Part I: Software quality, Developer frustration, UB. Language helps
- Part II: Static Analysis: Basics
- Part III: Static Analysis: Advanced
- Part IV: Gamification Idea & QA

Tools

- Visual Studio: https://visualstudio.microsoft.com
- CLion: https://www.jetbrains.com/clion/
- ReSharper C++ for VS: https://www.jetbrains.com/resharper-cpp/
- Clang-Tidy: https://clang.llvm.org/extra/clang-tidy/
- SonarLint: https://www.sonarlint.org

Software Quality

What is SW quality for you?

Anastasia Kazakova
@anastasiak2512

While preparing for my workshop at #CppOnSea, I want to ask you: reply with the very first thing that comes to your mind when you think about software quality.

Readability

Maintainability

tools

fuzzer

battery life

Repeatable tests

less UB

Robustness

Orthogonality

SW helps solving problems

The Last Spike

Languages

Memory management

Expressive code

Simplicity

Work as intended

Documented

Reviews

Reliability

Efficiency

Security

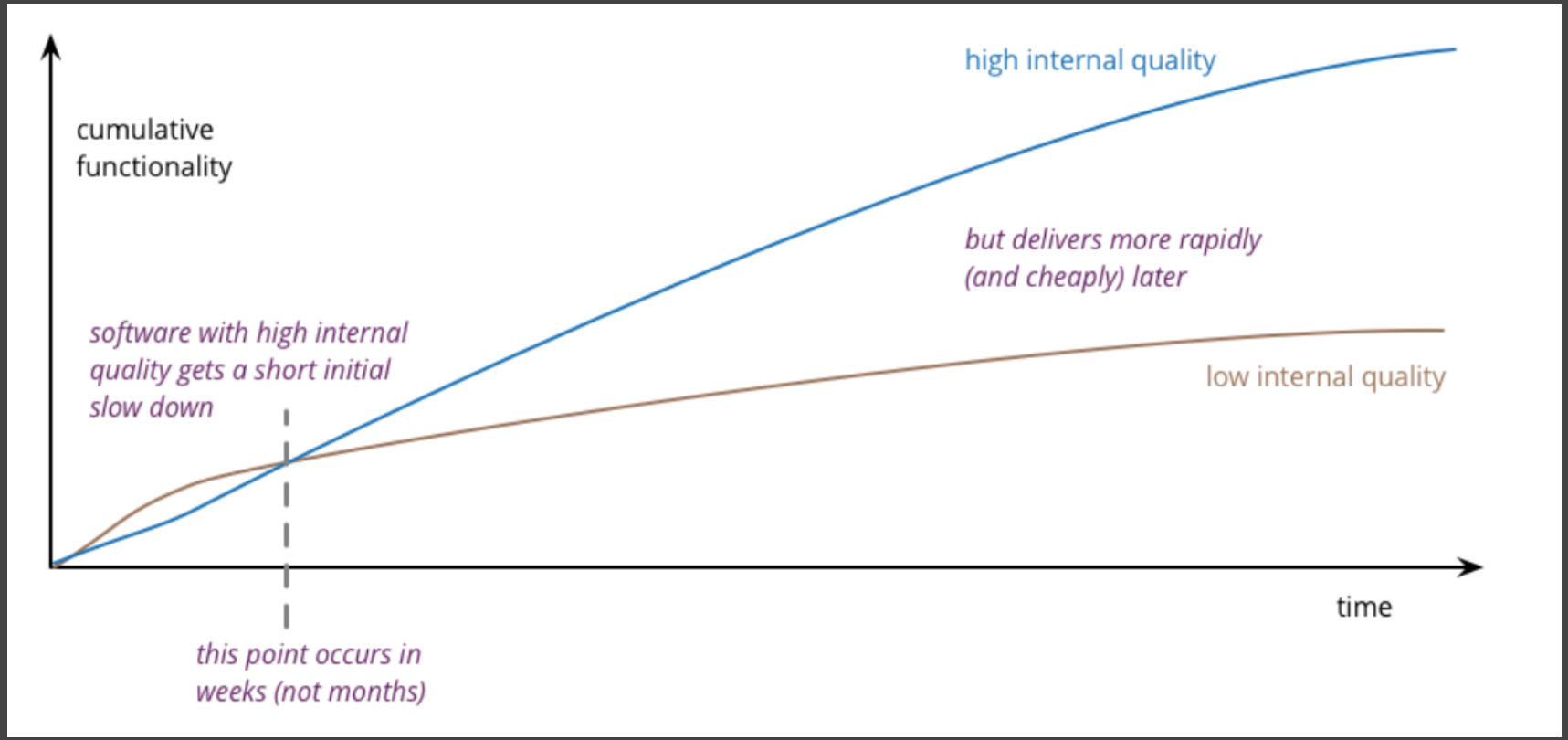
Maintainability

Size

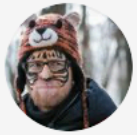
Software Quality

- Ok, so we defined SW quality
- Find a trade-off between quality and cost of development (spoiler: no!)
- External vs internal quality
- Martin Fowler: Is High Quality Software Worth the Cost? 29 May 2019

High quality software is cheaper to produce!



Developer Frustration



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?!

The way c++ is changing is scary.

So this is an example of how badly C++ can copy #dlang and make it worse.

The language's becoming too elite, it can loose type safety and compatibility, not every new is good.

Problem is, just because the "features" are there, some people will use them. If you're coding alone, all is peachy. But working in a team? 10 ways of doing 1 thing != good language.

“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)

C++ development frustration: style
–

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

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

“10 ways of doing 1 thing != good language.”
Twitter, @ArenMook, 24 Dec 2018

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

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

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

C++ worst enemy is its backwards compatibility going back to C.

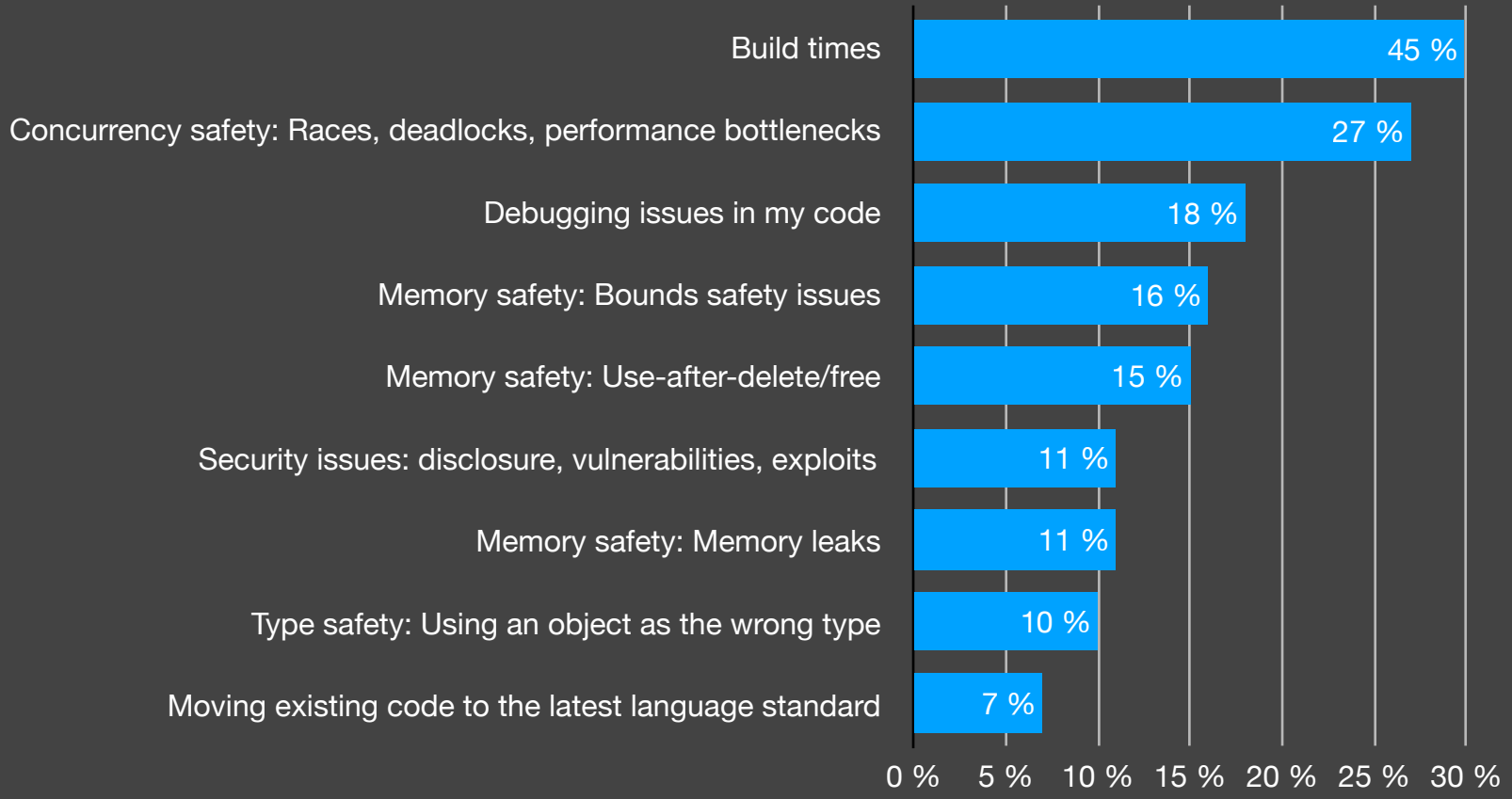
Any simplifying feature is only added to the language, complicated features are almost never removed.

The big problem is that a small feature added may change the way you think or work. Which leads to obsolete code very fast.

Which of these do you find frustrating about C++ development? (C++ Foundation Lite Survey 2021) –

Major Frustration Points	major %
Managing libraries my application depends on	48 %
Build times	45 %
Managing CMake projects	32 %
Setting up a CI pipeline from scratch	31 %
Concurrency safety: Races, deadlocks, performance bottlenecks	27 %
Setting up a dev env from scratch	26 %
Managing Makefiles	23 %
Parallelism support	22 %
Managing MSBuild projects	18 %
Debugging issues in my code	18 %
Memory safety: Bounds safety issues	16 %
Memory safety: Use-after-delete/free	15 %
Security issues: disclosure, vulnerabilities, exploits	11 %
Memory safety: Memory leaks	11 %
Type safety: Using an object as the wrong type	10 %
Moving existing code to the latest language standard	7 %

Which of these do you find frustrating about C++ development? (C++ Foundation Lite Survey 2021)
–



Undefined Behavior

UB

- data races
- memory accesses outside of array bounds
- signed integer overflow
- null pointer dereference
- access to an object through a pointer of a different type
- etc.

Compilers are not required to diagnose undefined behavior!

UB story

Fun with NULL pointers, part 1: https://lwn.net/Articles/342330/

```
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 code analysis
–

- Improve software quality
- Lower developer frustration
- Avoid UB

Language

Language helps!

- Lifetime safety: http://wg21.link/p1179
  - Owner and Pointer types + ACFG analysis
  - Diagnostic w/o annotations implemented in compiler
  - Clangd-based implementation gives 5% overhead right now

Language helps!

```
void sample1() {
    int* p = nullptr;
    {
        int x = 0;
        p = &x;
        *p = 42; //OK
    }
    *p = 42;      //ERROR
}
```

Language helps!

- Lifetime safety: http://wg21.link/p1179
  - Owner and Pointer types + ACFG analysis
  - Diagnostic w/o annotations implemented in compiler
  - Clangd-based implementation gives 5% overhead right now
  - Annotations: *gsl::SharedOwner*, *gsl::Owner*, *gsl::Pointer*, etc. in libraries or user code

Language helps!

```
struct [[gsl::Owner(int)]] MyIntOwner {
    MyIntOwner();
    int &operator*();
};
struct [[gsl::Pointer(int)]] MyIntPointer {
    MyIntPointer(int *p = nullptr);
    MyIntPointer(const MyIntOwner &);
    int &operator*();
    MyIntOwner toOwner();
};

MyIntPointer test5() {
    const MyIntOwner owner = MyIntOwner();
    auto pointer = MyIntPointer(owner);
    return pointer;   //ERROR
}
```

Language helps!

- Lifetime safety: http://wg21.link/p1179
- Contracts: http://wg21.link/p2358
  - For programming errors and failures of the C++ abstract machine
  - Assertions, pre- & post- conditions
  - Still under discussion, no implementation so far

Language helps!

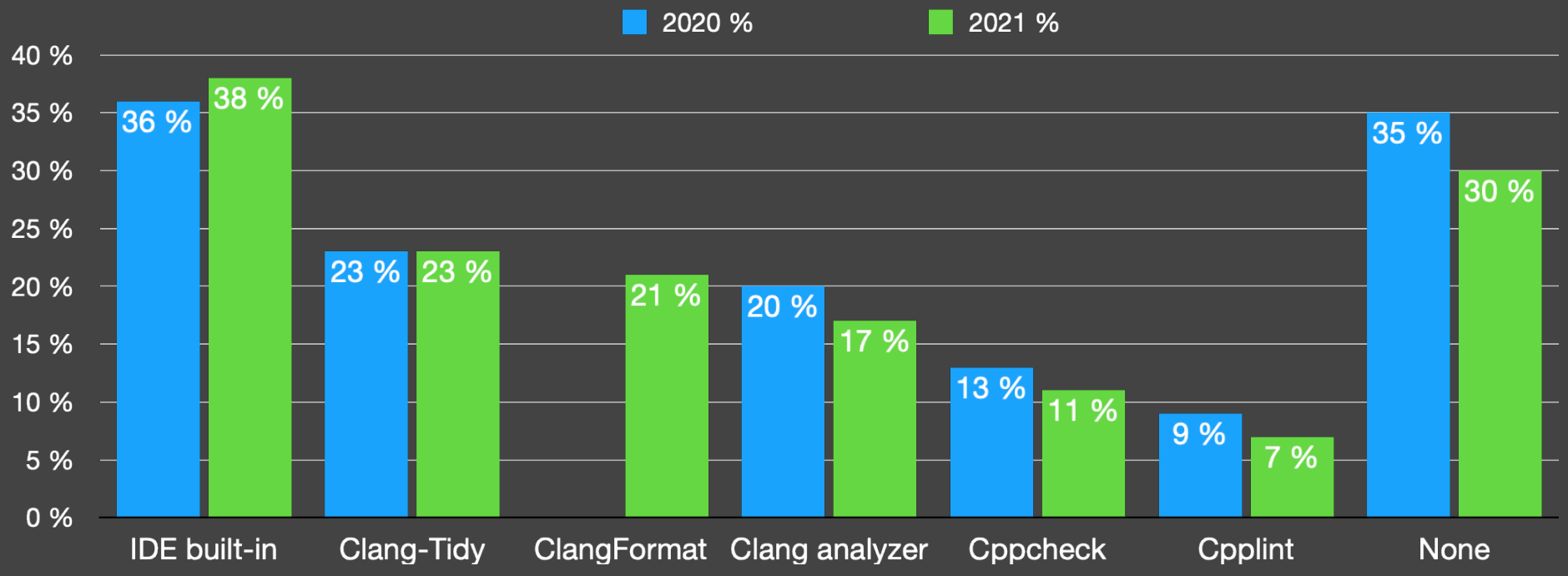
- Lifetime safety: http://wg21.link/p1179
- Contracts: http://wg21.link/p2358
- Parameter passing: https://github.com/hsutter/708/blob/main/708.pdf
  - in / inout / out / move / forward semantics
  - Still under discussion, no implementation so far

Tooling

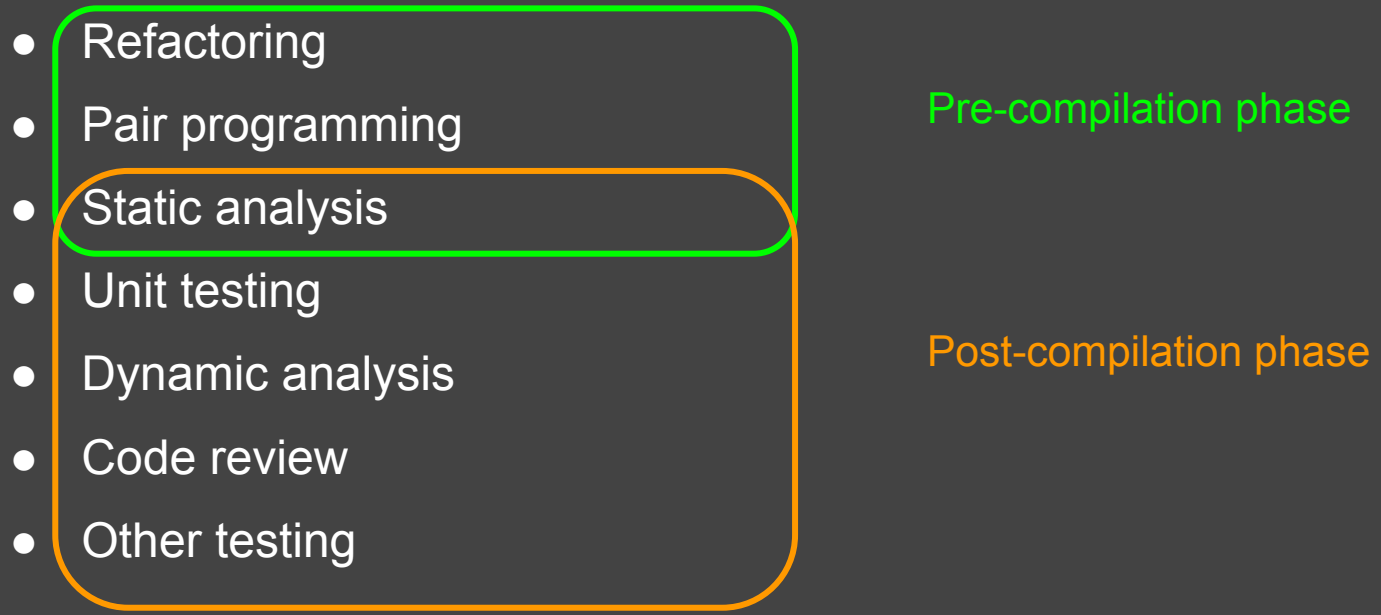
Compiler vs Analyzer
–

Language & Compiler	Stand-alone analyzer
Core tool – hard to update	Side tool – any adopted by the team is ok
Code base might requires specific compiler versions	No strong requirements for analyzer version
Set of checks is defined by compiler vendor	Custom checks are possible
Standard to everyone	Depends on the tool

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



Code Analysis: when?



Code Analysis: when?

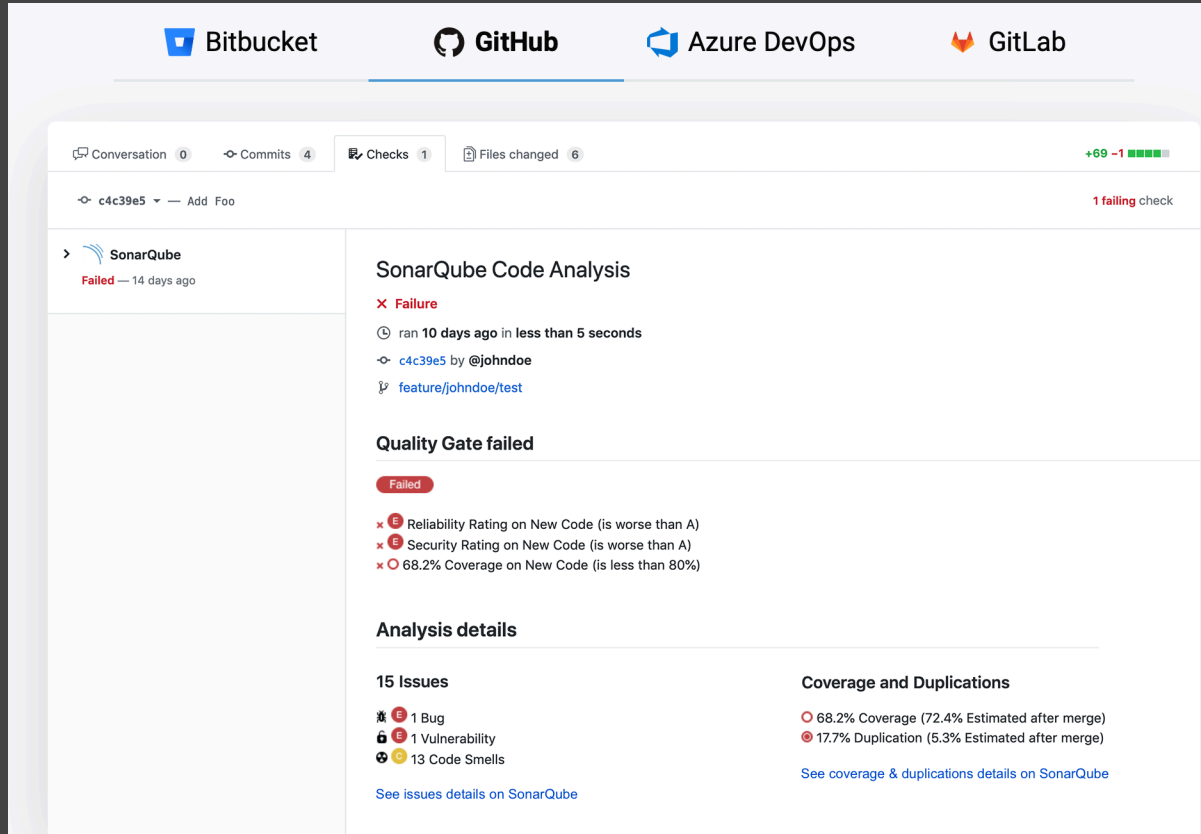
- Static analysis reports on CI
- Static analysis checks in Code Review
- Static analysis checks for Pull Requests

SonarLint, SonarCloud, SonarQube
–

https://www.sonarsource.com

https://rules.sonarsource.com/cpp

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

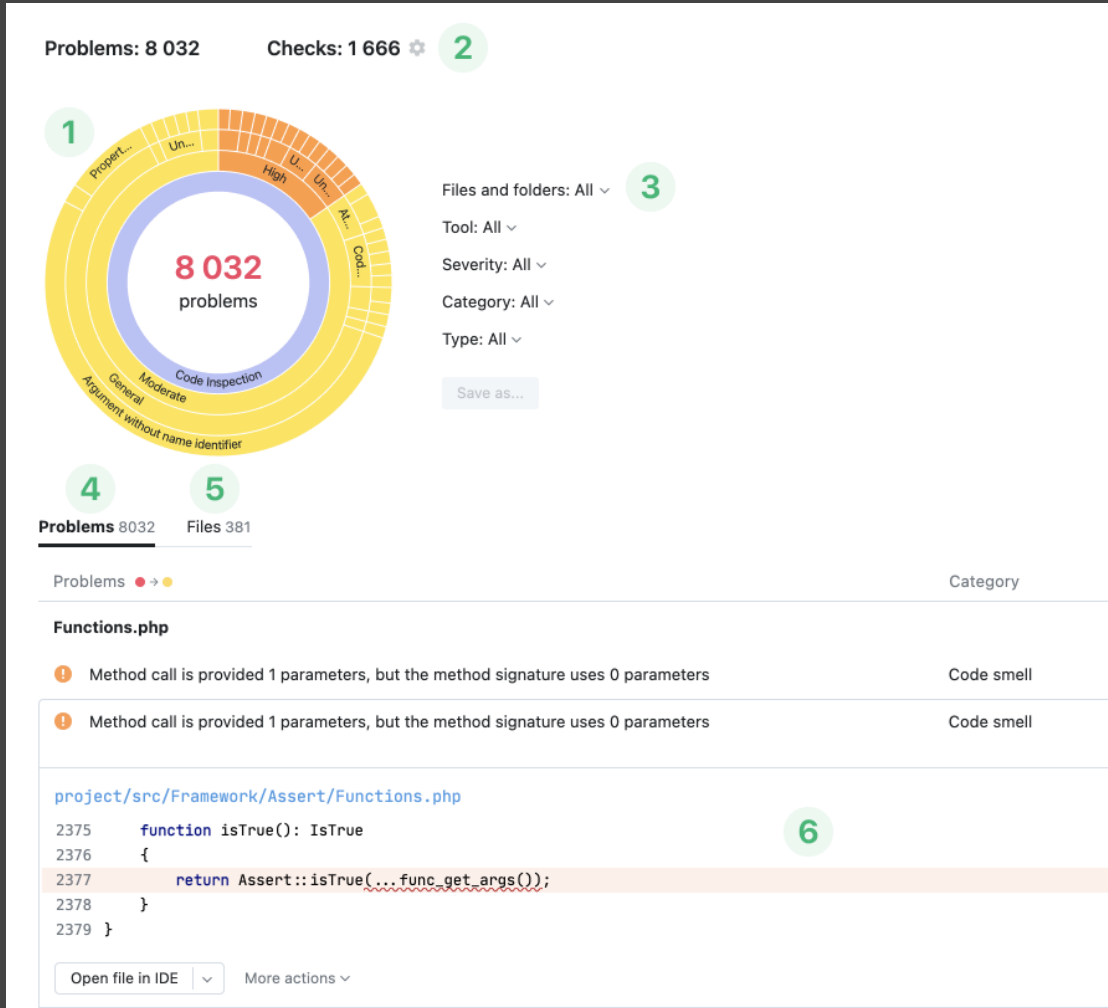


JetBrains Qodana

–

https://www.jetbrains.com/help/qodana/getting-started.html

- Linters from JetBrains IDEs
- CI/CD integrations
- Early Preview

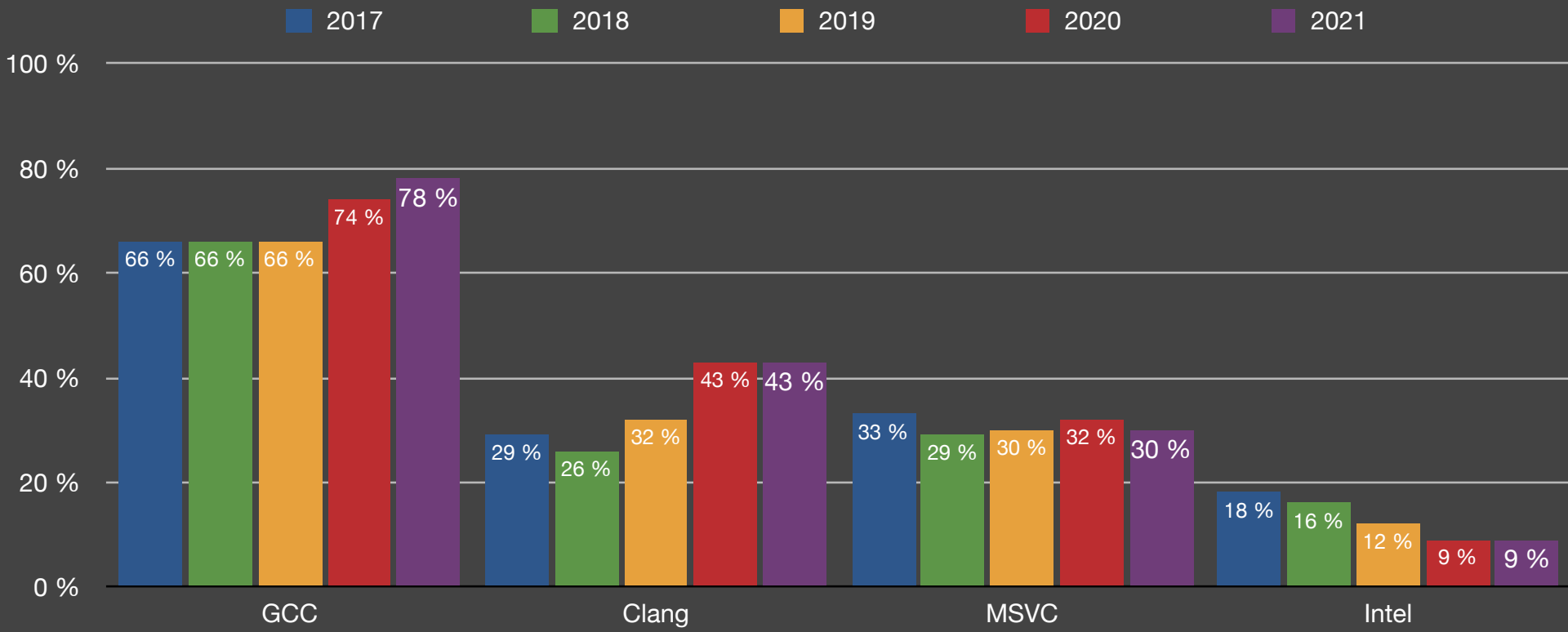


Static analysis tools

–

- Compiler errors and warnings

Compilers


2017
2018
2019
2020
2021
100 %
80 %
60 %
40 %
20 %
0 %
66 %
66 %
66 %
74 %
78 %
29 %
26 %
32 %
43 %
43 %
33 %
29 %
30 %
32 %
30 %
18 %
16 %
12 %
9 %
9 %
GCC
Clang
MSVC
Intel

How often do you compile with

-Wall -Wextra -Werror ?

Compilers: demo

[-Wsign-compare]

```
int a = -27;
unsigned b = 20U;
if (a > b)
    return 27;
return 42;
```

[-Wsizeof-pointer-memaccess]

```
int x = 100;
int *ptr = &x;
memset(ptr, 0, sizeof (ptr));
```

[-Wmisleading-indentation]

```
if (some_condition(cond))
    foo();
    bar();
```

Compilers: -Wlifetime

–

https://godbolt.org/z/ex37KeE8h

Compiler checks vs in-IDE analyzer

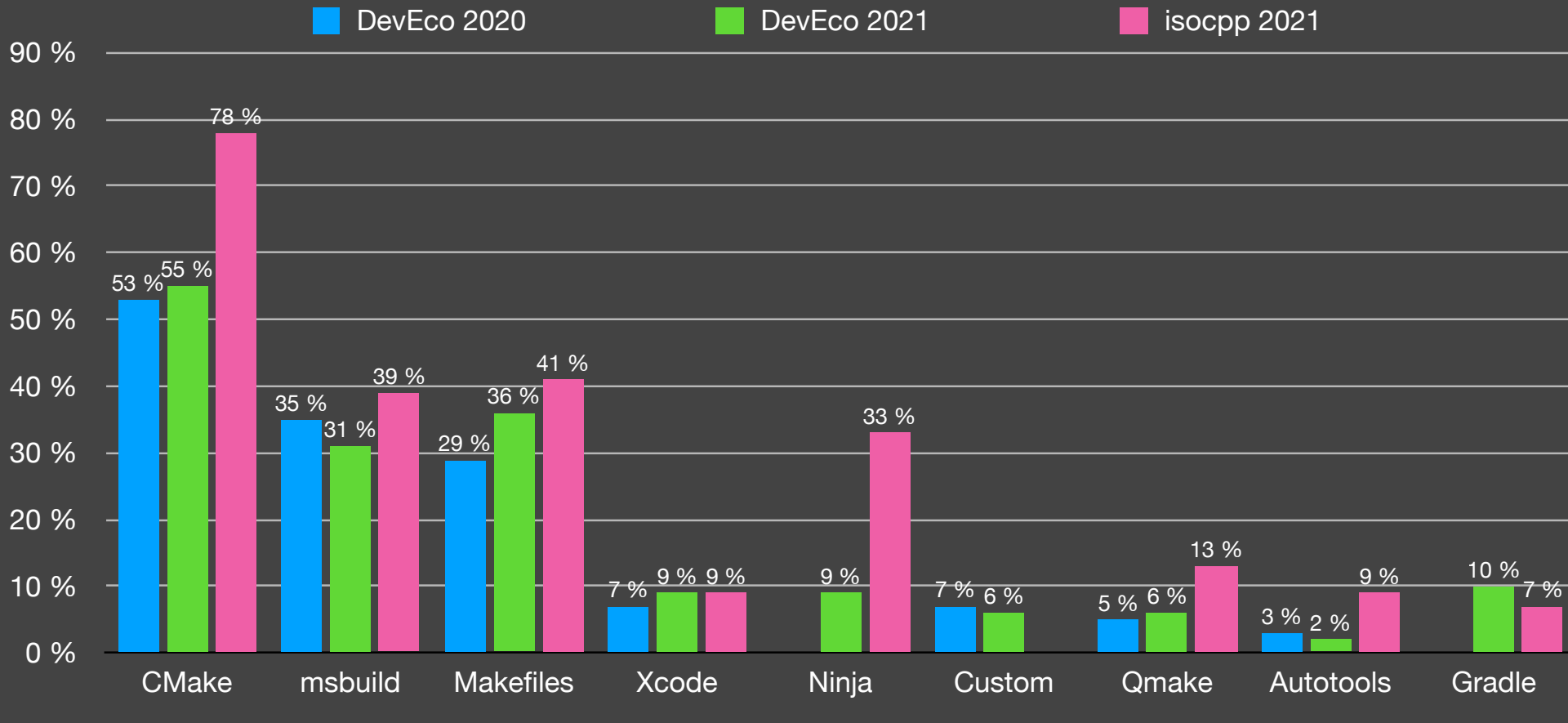
Checks during compilation	Checks in the IDE
Check code after it’s written	Check code while writing it
Analysing the code with proper flags / vars	Should use compilation flags & env
Using specific compiler	Can get checks from other compiler
Different compiler flags	Checks are independent from compiler

Compilers & Project Models

```
if (MSVC)
    # warning level 4 and all warnings as errors
    add_compile_options(/W4 /WX)
else()
    # lots of warnings and all warnings as errors
    add_compile_options(-Wall -Wextra -Werror)
endif()
```

```
CXXFLAGS += -Wall -Wextra -Werror
```

Project Models



Static analysis tools

–

- Compiler errors and warnings
- Lifetime safety

Lifetime safety

```
std::string get_string();
void dangling_string_view()
{
   std::string_view sv = get_string();
   auto c = sv.at(0);
}
```

Object backing the pointer will be destroyed at the end of the full-expression

Lifetime safety
–

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

Using invalid operator

Lifetime safety

–

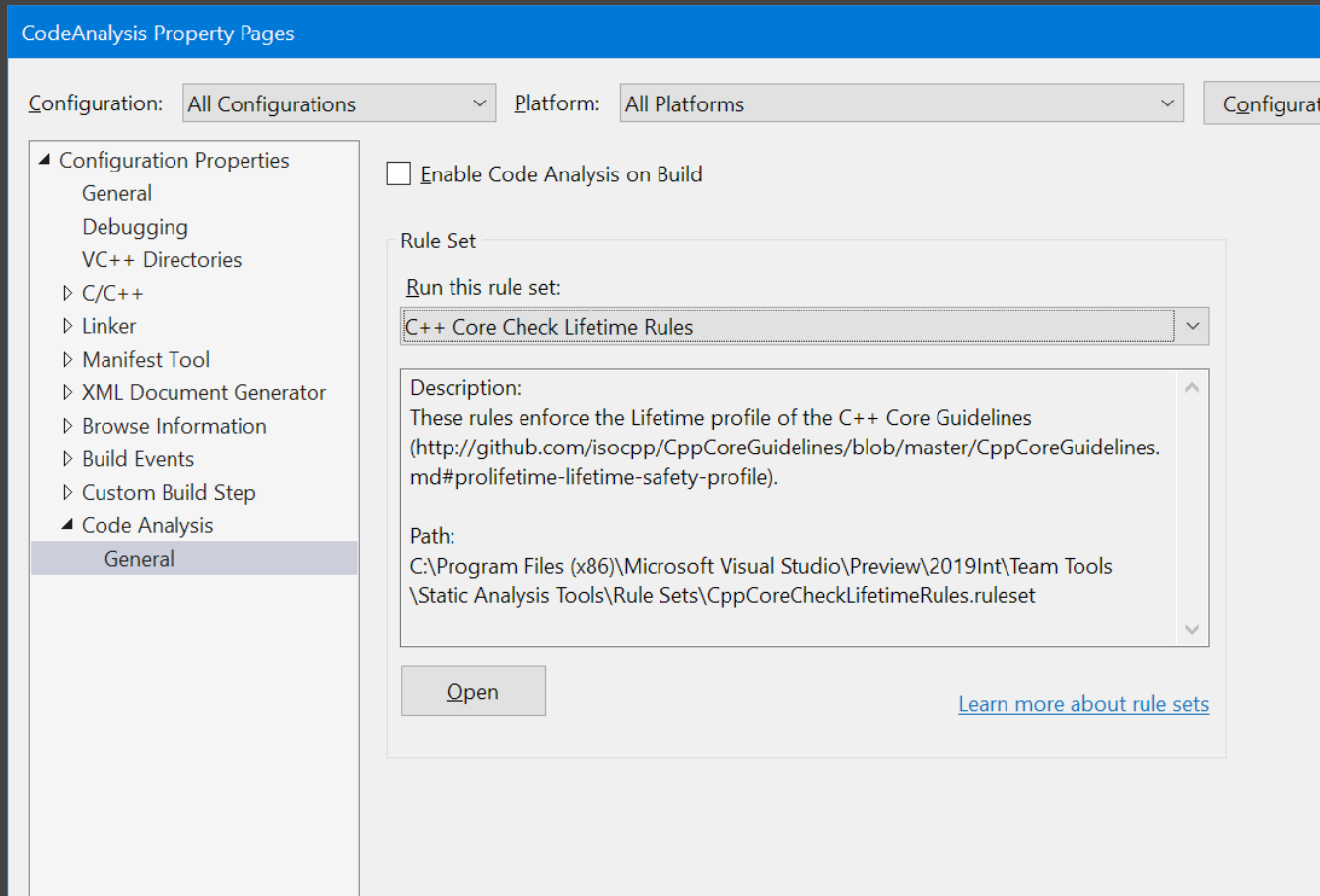
Lifetime analysis live demo with CLion 2021.2 EAP:

https://www.jetbrains.com/clion/nextversion/

https://github.com/anastasiak2512/code_analysis_pp

Lifetime safety
–

Visual Studio 2019



Static analysis tools

–

- Compiler errors and warnings
- Lifetime safety
- Data Flow Analysis

Data Flow Analysis

- DFA analyzes the data:
  - Function parameters/ arguments
  - Function return value
  - Fields and global variables

```
enum class States {
    Started, Stopped, Waiting
};

void Sample3(States current) {
    current = States::Started;

    if (current == States::Stopped) {
    }
}
```

Data Flow Analysis

DFA analysis live demo with CLion 2021.2 EAP:

https://www.jetbrains.com/clion/nextversion/

https://github.com/anastasiak2512/code_analysis_pp

Data Flow Analysis

- DFA local/global:
  - Constant conditions
  - Dead code
  - Endless loops
  - Infinite recursion
  - Unused values
  - Null dereference
  - Escape analysis
  - Dangling pointers
- DFA global-only:
  - Constant function result
  - Constant function parameter
  - Unreachable calls of function

Interprocedural Data Flow Analysis

- The complexity is non-linear, so TU-wide is proper trade-off
- Translation unit is chosen as a single “DFA unit”. The same approach is in clang static analyser
- Operates with Private and Unsafe entities

Interprocedural Data Flow Analysis: how-to

–

- Private Entities
- Unsafe Entities

```
//TU 1
class X {
    void foo(int p);
    virtual void bar();
public:
    void test();
};
void C::foo(int p) {
    if (p == 2) ;
}
void C::test() {
    foo(2);
}
//TU 2
void C::bar() {
    foo(3);
}
```

Interprocedural Data Flow Analysis

–

- No-go and switch to local-only mode
  - for uncompiled code
  - for unused code
  - file is included in another TU

Data Flow Analysis: where
–

CLion:

- Local DFA since 1.x
- Local DFA on Clang since 2020.1
- Global (TU) DFA since 2021.1
- Lifetimes in 2021.2

PVS-Studio:

- Value Range Analysis

Data Flow Analysis: global (CTU)

Cross Translation Unit (CTU) Analysis

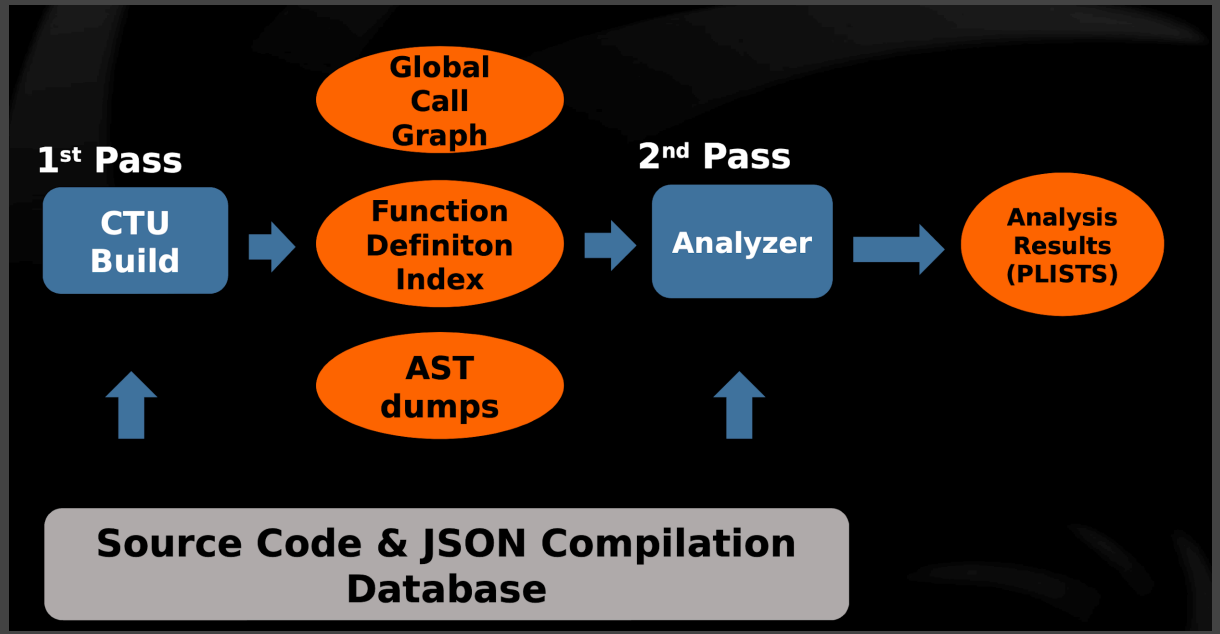
https://clang.llvm.org/docs/analyzer/user-docs/CrossTranslationUnit.html

1. Pre-dumped PCH
2. Generate AST on-demand

Data Flow Analysis: CTU – CodeChecker
–

CodeChecker

https://github.com/Ericsson/codechecker



Static analysis tools

–

- Compiler errors and warnings
- Lifetime safety
- Data Flow Analysis
- C++ Core Guidelines

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

C++ Core Guidelines via Clang-Tidy live demo with CLion 2021.2 EAP:

https://www.jetbrains.com/clion/nextversion/

https://github.com/anastasiak2512/code_analysis_pp

C++ Core Guidelines: tools

–

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

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
- *F.43: Never (directly or indirectly) return a pointer or a reference to a local object*

C++ Core Guidelines: not really

- *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 => action?
  - Rule of one screen: 60 lines by 140 characters => action?
- *F.3: Keep functions short and simple*
  - Rule of one screen => action?
  - Cyclomatic complexity “more than 10 logical path through” => action?

Finding code duplicates

https://stackoverflow.com/questions/191614/how-to-detect-code-duplication-during-development

- CCFinderX
- Duplo
- Simian
- ...others

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

C++ Core Guidelines: should we?

–

- *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*

Static analysis tools

–

- Compiler errors and warnings
- Lifetime safety
- Data Flow Analysis
- C++ Core Guidelines
- Clang-Tidy

Clang-Tidy

https://clang.llvm.org/extra/clang-tidy/checks/list.html

abseil-* (18), android-* (15), cert-* (35), Clang Static Analyzer, cppcoreguidelines-* (31), google-* (22), modernize-* (31), performance-* (15), …

Clang-Tidy live demo with CLion 2021.2 EAP:

https://www.jetbrains.com/clion/nextversion/

https://github.com/anastasiak2512/code_analysis_pp

Clang-Tidy: */ -*

*,<disabled-checks>

vs

-*,<enabled-checks>

Clang-Tidy: 'operator->' must resolve to a function declared within the '__llvm_libc' namespace

Static analysis tools
–

- Domain-specific analysis tools:
  - Clazy, MISRA/AUTOSAR, Unreal Header Tool, …

MISRA

- Automotive, self-driving experience
- Embedded in general, MISRA/AUTOSAR
- …

MISRA live demo with CLion 2021.2 EAP:

https://www.jetbrains.com/clion/nextversion/

https://github.com/anastasiak2512/code_analysis_pp

MISRA
–

Certification stage	Development stage
Must have	Good to have
High costs	Low costs
Defined checks and error messages	Flexible set of checks, detailed messages
Rule violations messages only	Checks + Quick-fixes

MISRA

- CLion MISRA: https://confluence.jetbrains.com/display/CLION/MISRA+checks+supported+in+CLion
  - MISRA C 2012 (62 / 166)
  - MISRA C++ 2008 (55 / 211)
- SonarLint MISRA:
  - https://rules.sonarsource.com/cpp/tag/misra-c++2008 (51 rules)
  - https://rules.sonarsource.com/cpp/tag/misra-c2004 (14 rules)
  - https://rules.sonarsource.com/cpp/tag/misra-c2012 (10 rules)

We all care about the same!

–

- 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

Clazy

- Qt projects
- https://github.com/KDE/clazy#list-of-checks

Clazy live demo with CLion 2021.2 EAP:

https://www.jetbrains.com/clion/nextversion/

https://github.com/anastasiak2512/code_analysis_pp

Static analysis tools
–

- Domain-specific analysis tools:
  - Clazy, MISRA/AUTOSAR, Unreal Header Tool, …
- Style
  - Formatting
  - Naming
  - Syntax style

Formatting

- ClangFormat
  - Formatting standard in C++ nowadays
  - Breaking compatibility
  - Fuzzy parsing

ClangFormat helpers live demo with CLion 2021.2 EAP:

https://www.jetbrains.com/clion/nextversion/

https://github.com/anastasiak2512/code_analysis_pp

Naming

- Naming
  - camelCase, PascalCase, SCREAMING_SNAKE_CASE
  - Google style, LLVM, Unreal Engine conversions
  - Requires Rename refactoring + support in code generation/refactorings

Naming live demo with CLion 2021.2 EAP:

https://www.jetbrains.com/clion/nextversion/

https://github.com/anastasiak2512/code_analysis_pp

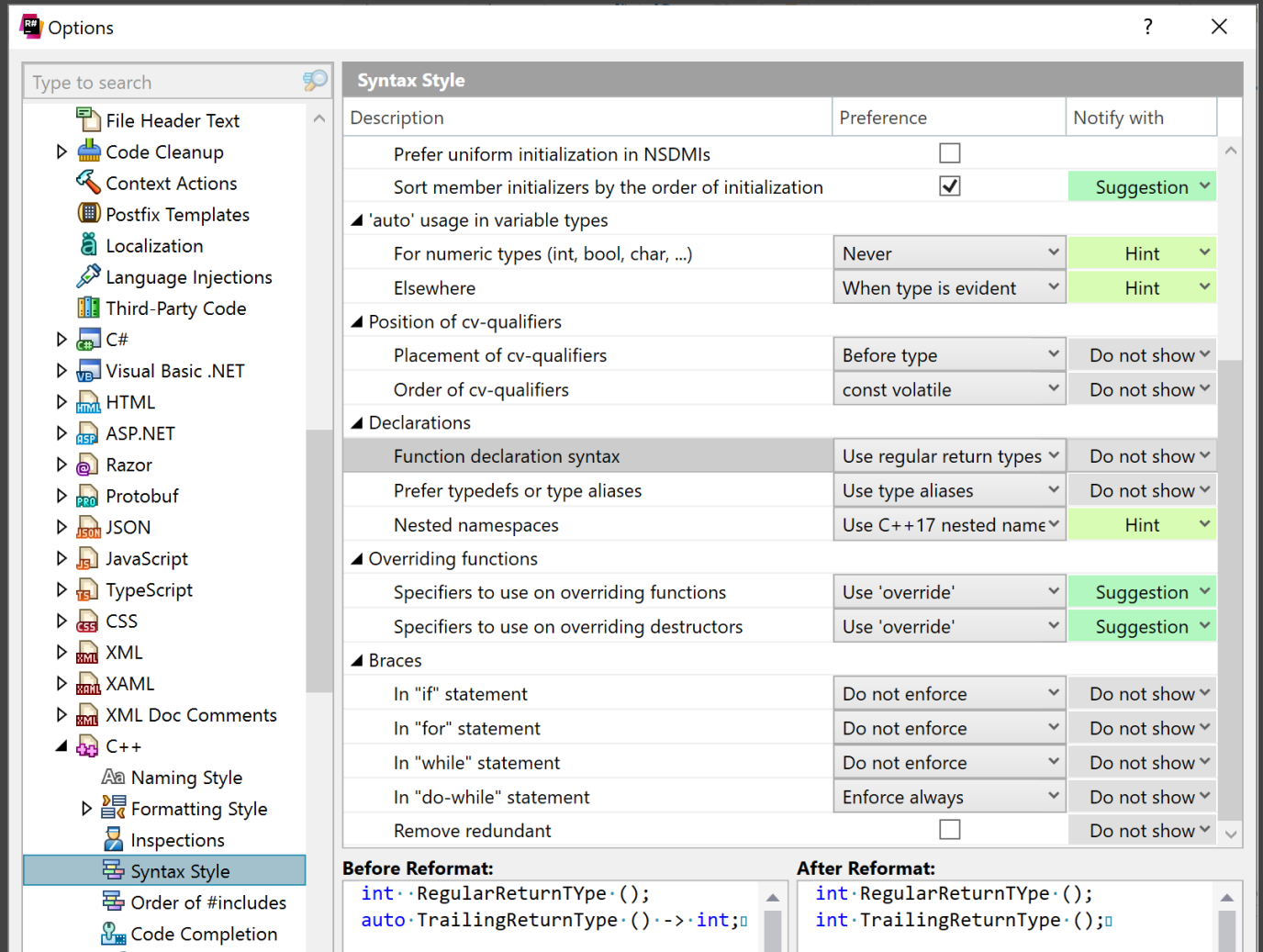
Syntax style

- Syntax style
  - “Almost Always Auto” or only “When Evident”
  - “East const” vs. “West const”
  - Typedefs vs. Type Aliases
  - Trailing or regular return types in function declarations
  - Overriding policy: override/final and no virtual
  - And many-many others…

In ReSharper C++ since 2021.1: https://blog.jetbrains.com/rscpp/2021/03/30/resharper-cpp-2021-1-syntax-style/

Syntax style

–



Static analysis tools

–

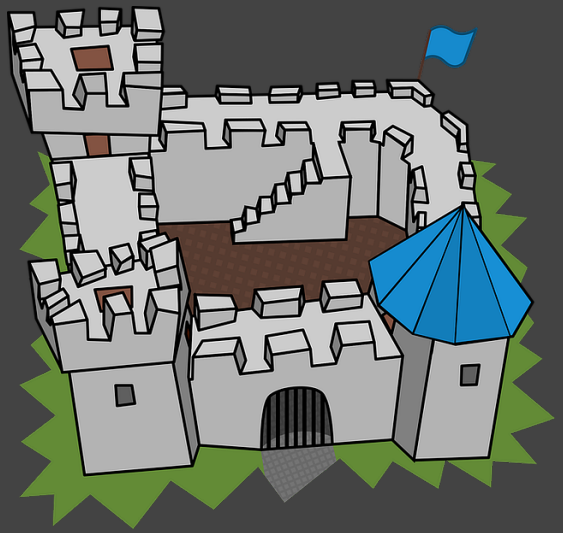
- Compiler errors and warnings
- Lifetime safety
- Data Flow Analysis
- C++ Core Guidelines
- Clang-Tidy
- Domain-specific analysis tools: Clazy, MISRA/AUTOSAR, etc.
- Style: Formatting, Naming, Syntax style

Static analysis tools

–

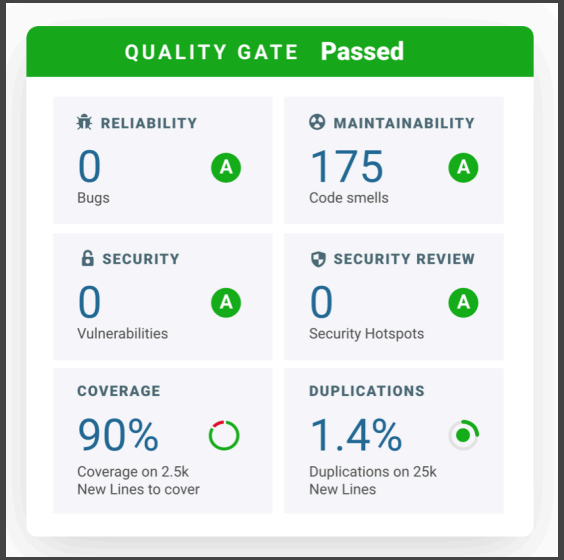
- Software quality & developers frustration
- Language helps!
- Tools help!
  - Compiler
  - Stand-alone analyzers, tools bundled into IDEs
  - C++ Core Guidelines
  - Clang-family tools
  - Coding guidelines

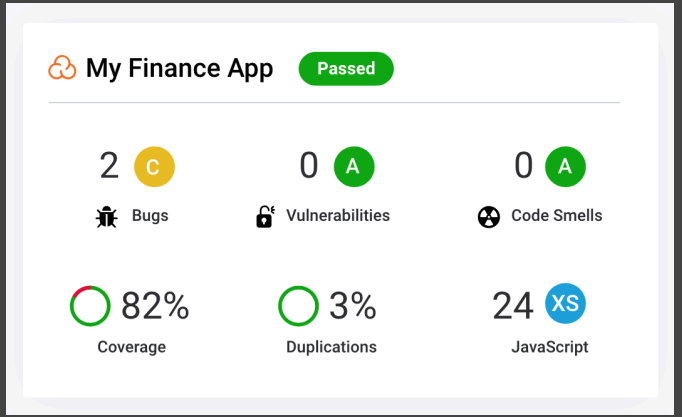
Gamifying Static Analysis



Gamifying Static Analysis

- Reduce noise
- Decompose
- Motivate
- Show progress





Gamifying Static Analysis

–

Microsoft

vscode-python

Bugs	0 A
Vulnerabilities	0 A
Code Smells	561 A
Coverage	–
Duplications	1.1%

54k lines of code / css, ts, ...

Wikimedia Foundation

mediawiki-core

Bugs	40 E
Vulnerabilities	1 E
Code Smells	731 A
Coverage	6.1%
Duplications	4.3%

426k lines of code / css, js, ...

jhipster

jhipsterSampleAp...

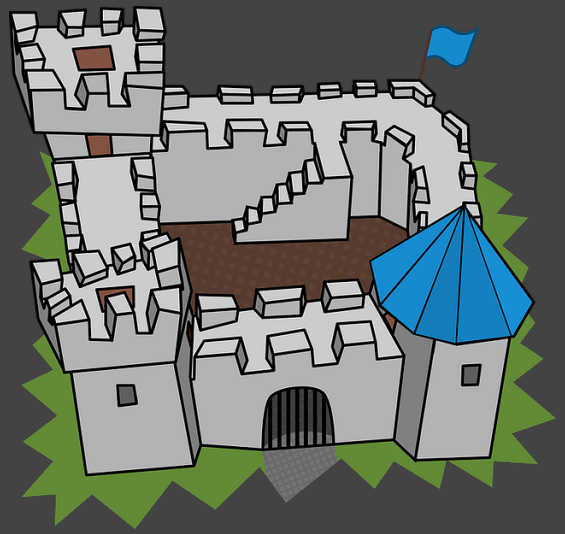
Bugs	0 A
Vulnerabilities	31 B
Code Smells	288 A
Coverage	81.2%
Duplications	2.3%

78k lines of code / css, java, ...

Gamifying Static Analysis

2018: http://www.bodden.de/pubs/db18gamifying.pdf

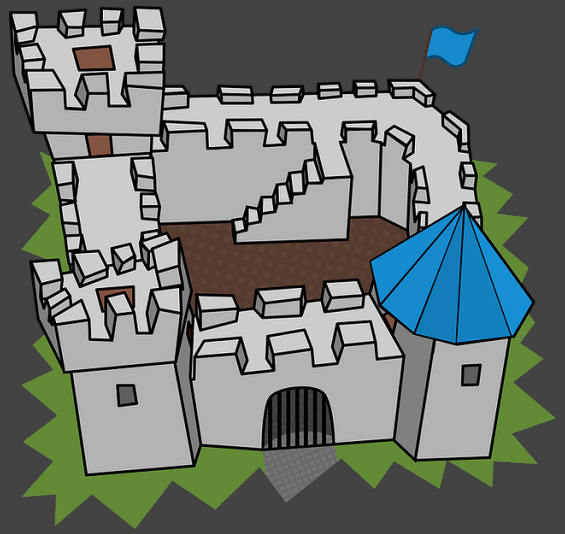
- Levels & decomposition
- Motivation & reward
- Using CTA instead of issues
- Team collaborative work



Gamifying Static Analysis
–

Why work gamification is a bad idea

- Short-term concept
- Helps with symptoms, not the root of the problem
- Treating people badly with disrespect
- Less serious attitude to work



Questions?

Thank you!

References

1. [Is High Quality Software Worth the Cost? By Martin Fowler](https://martinfowler.com/articles/is-quality-worth-cost.html)
2. [Aras Pranckevičius](https://twitter.com/aras_p/status/1076947443823136768)
3. [Tim Sweeney](https://twitter.com/TimSweeneyEpic/status/1409028887279984640)
4. [2021 Annual C++ Developer Survey "Lite"](https://isocpp.org/files/papers/CppDevSurvey-2021-04-summary.pdf)
5. [Lifetime safety: Preventing common dangling](http://wg21.link/p1179)
6. [Lifetime analysis in VS](https://devblogs.microsoft.com/cppblog/lifetime-profile-update-in-visual-studio-2019-preview-2/)
7. [Cross Translation Unit (CTU) Analysis](https://clang.llvm.org/docs/analyzer/user-docs/CrossTranslationUnit.html)
8. [CodeChecker by Ericsson](https://github.com/Ericsson/codechecker)
9. [ReSharper C++: Syntax Style](https://blog.jetbrains.com/rscpp/2021/03/30/resharper-cpp-2021-1-syntax-style/)
10. [Gamifying Static Analysis](http://www.bodden.de/pubs/db18gamifying.pdf)