

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

Anastasia Kazakova

JetBrains

@anastasiak2512

Time for a quote

*“C makes it easy to shoot yourself in the foot;
C++ makes it harder, but when you do it blows your whole leg off”*

- *Bjarne Stroustrup*

http://www.stroustrup.com/bs_faq.html#really-say-that

Why C++ is hard in general?

- C heritage and C/C++ compatibility
- Legacy code & modern code co-exist
- New C++ standards: too quick or too slow?
- Functional and math concepts in modern C++
- Tools missing

Time for a story

Each time someone was leaving the team, the C++ code written by that person was aggressively rewritten by those who stayed.



Time for a sample: goto fail

About the security content of iOS 7.0.6

This document describes the security content of iOS 7.0.6.

iOS 7.0.6

- **Data Security**

Available for: iPhone 4 and later, iPod touch (5th generation), iPad 2 and later

Impact: An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS

Description: Secure Transport failed to validate the authenticity of the connection.
This issue was addressed by restoring missing validation steps.

CVE-ID

CVE-2014-1266

Time for a sample: goto fail

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                  uint8_t *signature, UInt16 signatureLen)
{
    OSStatus err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
    goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...
}
```

Time for a sample: know the type

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

void fun_type() {
    auto operation_result = doOperation(3.0, 0);
    //...
}
```

Time for a sample: where is type

```
#ifdef MAGIC
template<int>
struct x {
    x(int i) { }
};

#else
int x = 100;
#endif

void test(int y) {
    const int a = 100;
    auto foo = x<a>(0);
}
```

Define code quality

Code quality =

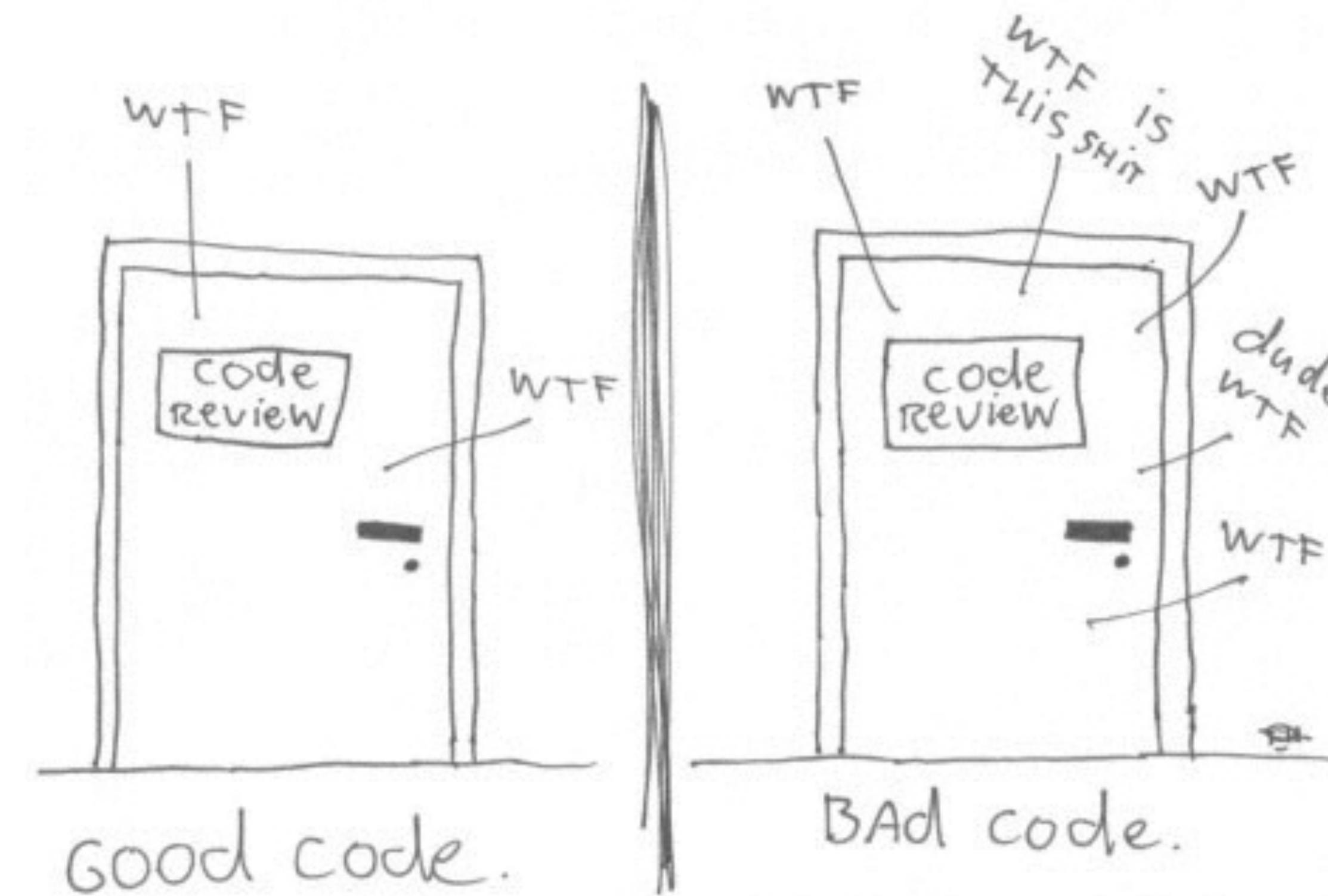
- easy to read
- easy to maintain
- easy to use
- works correct

Code quality (CISQ) =

- reliability
- efficiency
- security
- maintainability
- size

Define code quality

The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/minute



We need tools

“Tool support is essential... We need tools”

- *Bjarne Stroustrup*

Tools for code quality

1. Follow the code style, formatter
2. Generate code
3. Run code analysis
4. Refactor
5. Check with unit tests

Tools for code quality

1. Follow the code style, formatter
2. Generate code
3. Run code analysis
4. Refactor
5. Check with unit tests

Tools: Formatter

C++ specific

C++ specific

- Formatting with/without parsing/resolve

```
void test() {
    struct x {
        x(int) { };
    };
    int y = 100;
    auto a = (x)-5;
    auto b = (y)-5;
}
```

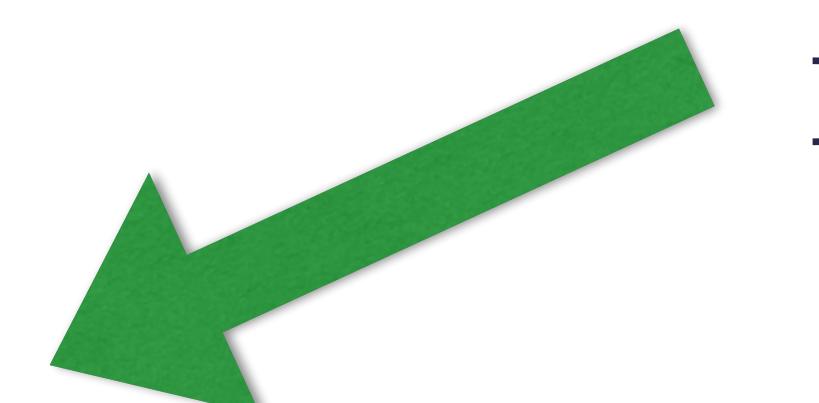
Tools: Formatter

C++ specific

C++ specific

- Formatting with/without parsing/resolve

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



```
void test() {  
    struct x {  
        x(int) { };  
    };  
  
    int y = 100;  
  
    auto a = (x)-5;  
    auto b = (y)-5;
```

Tools: Formatter C++ specific

C++ specific

- Formatting with/without parsing/resolve
- Variety
 - Variety of standards
 - Google, LLVM/LLDB, Qt, GNU, Chromium, Mozilla, WebKit, etc.
 - Heritage: Custom style guides in companies, big projects, etc.
 - Style guides are different in essence

Tools: Formatter Workflow

When to reformat?

- On file save
- On-the-fly formatting
- Pre-commit hook
- Explicit actions: format line, selection, file

Tools: Formatter

- Clang-Format <http://clang.llvm.org/docs/ClangFormat.html>
 - Command-line tool
 - `// clang-format off/on`
 - Library profiles
 - Integration
 - Vim, Emacs, VS, VS Code, Qt Creator, NetBeans, ReSharper C++

Tools: Formatter

- Clang-Format <http://clang.llvm.org/docs/ClangFormat.html>
 - Command-line tool
 - `// clang-format off/on`
 - Library profiles
 - Integration
 - Vim, Emacs, VS, VS Code, Qt Creator, NetBeans, ReSharper C++
 - Fuzzy parser

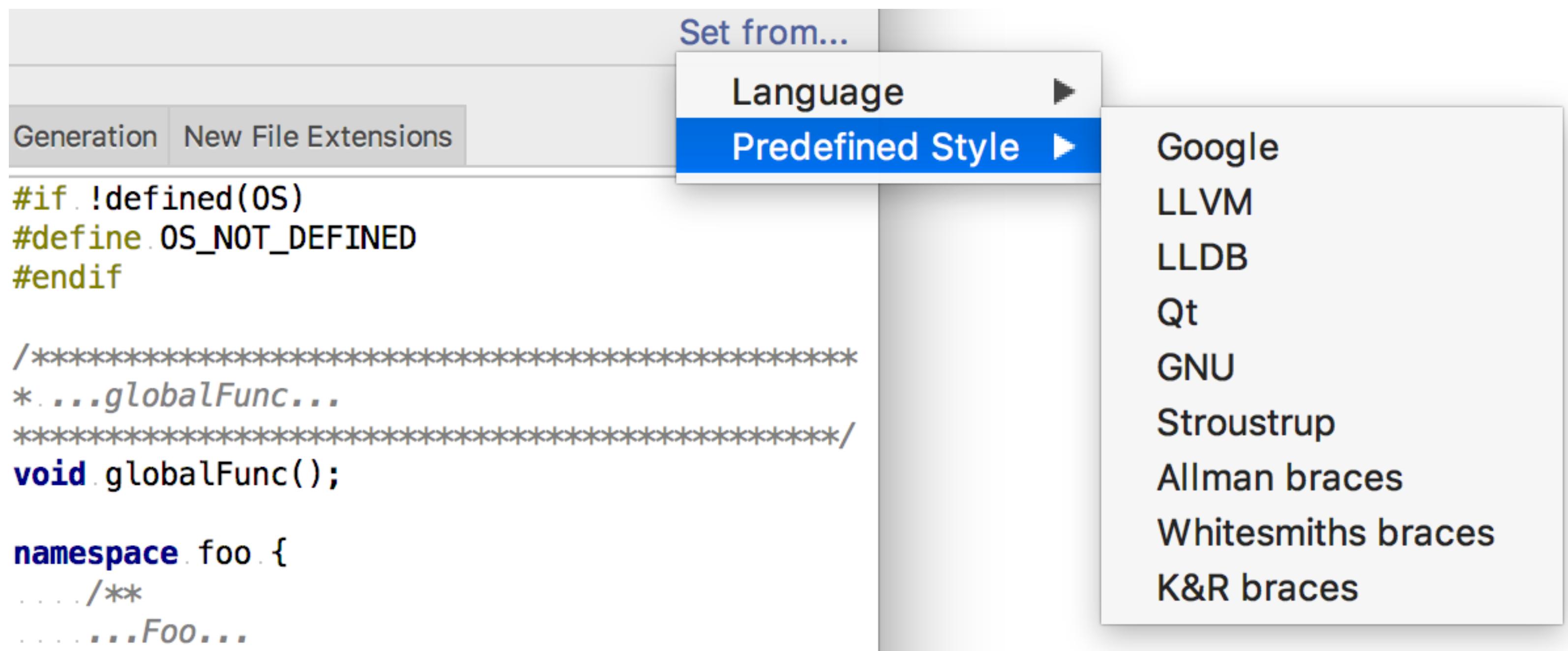
Tools: Formatter

- Clang-Format <http://clang.llvm.org/docs/ClangFormat.html>
 - Command-line tool
 - `// clang-format off/on`
 - Library profiles
 - Integration
 - Vim, Emacs, VS, VS Code, Qt Creator, NetBeans, ReSharper C++
 - Fuzzy parser
- IDEs
 - Built-in formatter tool
 - Import/export/convert issues

Tools: Formatter

Extract style

- Idea: get style from the actual code
- Problem: lots of settings, dependant parameters
- Genetic algorithm



Tools for code quality

1. Follow the code style, formatter
2. Generate code
3. Run code analysis
4. Refactor
5. Check with unit tests

Generate C++ code

Why?

- Keep to style guidelines
- Use common design patterns
- Avoid errors

What?

- File templates
- Comments/Docs
- Code snippets

Tools: Code generation

- IDEs

- class/file templates
- getters/setters
- other

```
class Distance {  
private:  
    int feet = 0;  
    int inches = 0;  
public:  
    Distance(int feet, int inches) : feet(feet), inches(inches) {}  
  
    int getFeet() const { return feet; }  
  
    void setFeet(int feet) { Distance::feet = feet; }  
  
    int getInches() const { return inches; }  
  
    void setInches(int inches) { Distance::inches = inches; }  
  
    bool operator==(const Distance &rhs) const {  
        return feet == rhs.feet && inches == rhs.inches;  
    }  
  
    bool operator!=(const Distance &rhs) const { return !(rhs == *this); }  
  
    bool operator<(const Distance &rhs) const {  
        if (feet < rhs.feet)  
            return true;  
        if (rhs.feet < feet)  
            return false;  
        return inches < rhs.inches;  
    }  
  
    bool operator>(const Distance &rhs) const { return rhs < *this; }  
  
    bool operator<=(const Distance &rhs) const { return !(rhs < *this); }  
  
    bool operator>=(const Distance &rhs) const { return !(*this < rhs); }  
  
    friend std::ostream &operator<<(std::ostream &os, const Distance &distance) {  
        os << "feet: " << distance.feet << " inches: " << distance.inches;  
        return os;  
    }  
};
```

Tools: Code generation

```
class Distance {  
private:  
    int feet = 0;  
    int inches = 0;  
public:  
};
```

```
#endif //DEMOMORE_DISTANCE_H
```

Demo

Tools: Code generation

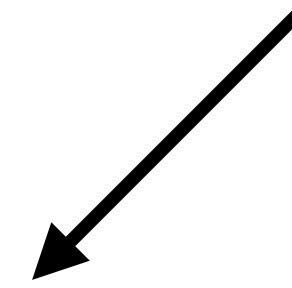
- IDEs
- Specialized tools
 - protobuf,
 - cog and other transformation tools,
 - clang based tools

Tools: Code generation

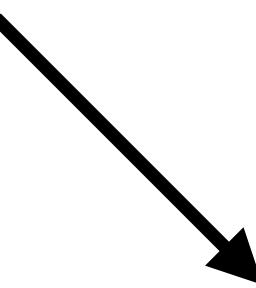
- IDEs
- Specialized tools
- C++ Metaclasses

Code generation Metaclasses

```
$class interface {
    constexpr {
        compiler.require($interface.variables().empty(),
                        "interfaces may not contain data");
        for... (auto f : $interface.functions()) {
            compiler.require(!f.is_copy() && !f.is_move(),
                            "interfaces may not copy or move; consider a"
                            " virtual clone() instead");
            if (!f.has_access()) f.make_public();
            compiler.require(f.is_public(),
                            "interface functions must be public");
            f.make_pure_virtual();
        }
    }
    virtual ~interface() noexcept { }
};
```



```
interface Shape {
    int area() const;
    void scale_by(double factor);
};
```



```
struct Shape {
    virtual int area() const = 0;
    virtual void scale_by(double factor) = 0;
    virtual ~Shape() noexcept {
    }
};
```

Tools for code quality

1. Follow the code style, formatter
2. Generate code
3. Run code analysis
4. Refactor
5. Check with unit tests

Tools: Code analysis

- Static code analysis tools
- Dynamic code analysis tools

Tools: Static analysis

- Compiler errors and warnings
- Extra checks (DFA, style, etc.)

Tools: More static code analysis

CppCheck: <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/>

- 10 years
- approx 170 checks
- variable scope, out-of-bounds, memory leaks, sizeof args, etc.

Tools: Clang Analyzer

Clang Analyzer: https://clang-analyzer.llvm.org/available_checks.html

- clang -analyze -analyzer-checker=...
- Clang-Tidy: clang-analyzer-*
- Xcode
- C, C++, Objective-C
- 40 checks for C/C++
 - core checks
 - new/delete leaks
 - dead code
 - nullability
 - security checks



Tools: (Modernize with) Clang-Tidy

Clang-Tidy: <http://clang.llvm.org/extra/clang-tidy/>

- `-checks=-*,modernize-*`
 - replace `std::bind` with lambdas
 - use `auto` in iterators, new expressions, cast expressions
 - replace C standard library headers with their C++ alternatives
 - replace a default constructor's member initializers with the new default member initializers in C++11
 - more (24 checks)
- In total >200 checks!

Tools: (Modernize with) Clang-Tidy

The screenshot shows a code editor window with a tab labeled "modernize.cpp". The code is written in C++ and includes several standard library headers: #include <functional>, #include <vector>, and #include <iostream>. The code defines a function add that returns the sum of its two integer parameters. It also contains a bind_to_lambda function that creates a lambda expression binding the value 2 to the variable x and another lambda expression that prints the elements of a vector. A loop_convert function iterates over the elements of a vector and prints them. A MyClass is defined with a constructor that takes two string parameters and initializes them. The class has a private member string Copied and a const member string &ReadOnly.

```
5  #include <functional>
6  #include <vector>
7  #include <iostream>
8
9  int add(int x, int y) { return x + y; }
10
11 void bind_to_lambda(int num) {
12     int x = 2;
13     auto clj = std::bind(add, x, num);
14 }
15
16 void loop_convert(const std::vector<int>& vec) {
17     for(auto iter = vec.begin(); iter != vec.end(); ++iter) {
18         std::cout << *iter;
19     }
20 }
21
22 class MyClass {
23 public:
24     MyClass(const std::string &Copied,
25             const std::string &ReadOnly)
26             : Copied(Copied), ReadOnly(ReadOnly) {}
27
28 private:
29     std::string Copied;
30     const std::string &ReadOnly;
31 };
32
```

Tools: C++ Core Guidelines

C++ Core Guidelines: <https://github.com/isocpp/CppCoreGuidelines>

- Support in Clang-Tidy (16 checks)
- Via Clang-Tidy
 - Eclipse, CLion, NetBeans
- Native support
 - CppCoreCheck in Visual Studio
 - Cevelop



Tools: More static code analysis

Coverity: <http://www.coverity.com>

- C, C++, C#, Objective-C and more
- Business process:
 - incremental & full analysis, parallel analysis, CI plugins
- approx 80 checks
 - out-of-bounds reads double free, use after free
 - incorrect buffer size
 - integer overflow
 - uninitialised variable
 - null dereference

Tools: More static code analysis

- CppCheck
- Clang Analyzer
- Clang-Tidy
- C++ Core Guidelines
- Coverity
- PVS-Studio (<https://www.viva64.com/en/w/>)
- Infer (Facebook) (<http://fbinfer.com/docs/infer-bug-types.html>)
- IDEs checks

Tools: Data Flow Analysis in CLion

```
typedef enum class Color { Red, Blue, Green, Yellow };

void do_shadow_color(int shadow) {
    Color cl1, cl2;

    if (shadow)
        cl1 = Color::Red, cl2 = Color::Blue;
    else
        cl1 = Color::Green, cl2 = Color::Yellow;

    if (cl1 == Color::Red || cl2 == Color::Yellow) {
    } Condition is always true when reached more... (⌘F1)
}
```

```
void Sample2(int flag) {
    Color c;

    switch (flag) {
        case 0:
            c = Color::Red;
            break;
        case 1:
            c = Color::Blue;
            break;
        default:
            c = Color::Green;
    }

    switch (c) {
        case Color::Red:
            break;
        case Color::Blue:
            break;
        case Color::Green:
            break;
        case Color::Yellow:
            break;
    } Unreachable code more... (⌘F1)
}
```

Tools: Dynamic code analysis

- Valgrind
- Google LLVM sanitizers
- GNU gprof, gperftools
- Oracle Developer Tools (SunStudio)
- Dr. Memory
- and others

Tools: Dynamic code analysis

Valgrind	Sanitizers
JIT/VM	Requires recompilation
any compiler / nearly any executable	Clang (3.1-3.2), GCC (4.8)
Runs one thread at a time	All threads running at-speed
20-50 slow down	2-5 slow down
Linux, Solaris, macOS, Android	ASan: Linux, macOS, iOS Simulator, Android, FreeBSD MSan: Linux x86_64 TSan: Linux x86_64
Memcheck, Cachegrind, Callgrind, Hellgrind, massif, etc.	Address-, Leak-, Memory-, Thread-, Undefined Behavior- sanitizers Data Flow API

Tools: Valgrind

- Memcheck <http://valgrind.org/docs/manual/mc-manual.html>
 - illegal read/write/frees
 - using uninitialized or derived from uninitialized value
 - overlapping src and dst in memcpy, strcpy, strncpy, strcat, strncat
 - fishy size arguments in allocators
 - memory leaks

Tools: Valgrind

Run memdefault

Console Valgrind

The screenshot shows the Clion IDE interface with the Valgrind tool selected. The left panel displays a tree view of Valgrind errors, and the right panel shows the source code with the error location highlighted.

Valgrind Error Tree:

- InvalidWrite 1 warning
 - main.cpp 1 warning
 - Invalid write of size 1
 - 0x100000F63 main main.cpp:24
 - Address 0x100b58c3a is 0 bytes after a block of size 10 alloc'd
 - 0x100098616 malloc
 - 0x100183E2D operator new(unsigned long)
 - 0x100000F5A main main.cpp:23
 - Leak_DefinitelyLost 11 warnings
 - Leak_PossiblyLost 1 warning
 - SyscallParam 1 warning
 - UninitCondition 1 warning
 - main.cpp 1 warning
 - Conditional jump or move depends on uninitialized value(s)
 - 0x100000D8B foo(int) main.cpp:8
 - 0x100000F6E main main.cpp:27

Uninitialised value was created by a stack allocation
0x100004036 _dyld_start

InstructionPointer: 0x100000F6E

Object: /Users/anastasiakazakova/ClionProjects/mem_sample/cmake-build-debug/mem_sample

Function: main

Directory: /Users/anastasiakazakova/ClionProjects/mem_sample

File: main.cpp

Line: 27

```
int main() {
    char *name = new char[10];
    name[10] = 'x';

    int var;
    foo(var);

    constructTree();

    return 0;
}
```

Tools: Valgrind

- Memcheck <http://valgrind.org/docs/manual/mc-manual.html>
- Cachegrind <http://valgrind.org/docs/manual/cg-manual.html>
 - cache profiler
 - accuracy issues

```
==31751== I refs: 27,742,716
==31751== I1 misses: 276
==31751== LLi misses: 275
==31751== I1 miss rate: 0.0%
==31751== LLi miss rate: 0.0%
==31751==

==31751== D refs: 15,430,290 (10,955,517 rd + 4,474,773 wr)
==31751== D1 misses: 41,185 ( 21,905 rd + 19,280 wr)
==31751== LLd misses: 23,085 ( 3,987 rd + 19,098 wr)
==31751== D1 miss rate: 0.2% ( 0.1% + 0.4%)
==31751== LLd miss rate: 0.1% ( 0.0% + 0.4%)
==31751==

==31751== LL misses: 23,360 ( 4,262 rd + 19,098 wr)
==31751== LL miss rate: 0.0% ( 0.0% + 0.4%)
```

Tools: Valgrind

- Memcheck <http://valgrind.org/docs/manual/mc-manual.html>
- Cachegrind <http://valgrind.org/docs/manual/cg-manual.html>
- Callgrind <http://valgrind.org/docs/manual/cl-manual.html>
 - records the call history in the program
 - recursive calls requires attention
 - KCachegrind is a viewing tool

Tools: Valgrind

- Memcheck <http://valgrind.org/docs/manual/mc-manual.html>
- Cachegrind <http://valgrind.org/docs/manual/cg-manual.html>
- Callgrind <http://valgrind.org/docs/manual/cl-manual.html>
- Hellgrind, massif, etc.
 - Hellgrind = thread error detector
 - Massif = heap profiler

Tools: Sanitizers

- Address Sanitizer (ASan) -fsanitize=address
 - out-of-bound access
 - use-after-free
 - use-after-return
 - use-after-scope
 - double-free, invalid-free
 - memory leaks (LeakSanitizer)

Tools: Sanitizers

```
=====
==58237==ERROR: AddressSanitizer: stack-use-after-scope on address 0x7ffc4d830880 at
pc 0x0000005097ed bp 0x7ffc4d830850 sp 0x7ffc4d830848
WRITE of size 4 at 0x7ffc4d830880 thread T0
#0 0x5097ec  (/tmp/use-after-scope+0x5097ec)
#1 0x7ff85fa6bf44  (/lib/x86_64-linux-gnu/libc.so.6+0x21f44)
#2 0x41a005  (/tmp/use-after-scope+0x41a005)
```

Address 0x7ffc4d830880 is located in stack of thread T0 at offset 32 in frame
#0 0x5096ef (/tmp/use-after-scope+0x5096ef)

This frame has 1 object(s):

[32, 36) 'x' <== Memory access at offset 32 is inside this variable

HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
(longjmp and C++ exceptions *are* supported)

SUMMARY: AddressSanitizer: stack-use-after-scope (/tmp/use-after-scope+0x5097ec)

Shadow bytes around the buggy address:

```
0x100009afe0c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe0d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe0e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe0f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe100: 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
=>0x100009afe110:[f8]f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100009afe160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Tools: Sanitizers

- Address Sanitizer (ASan)
- Thread sanitizers (TSan) -fsanitize=thread
 - data race detector

Tools: Sanitizers

- Address Sanitizer (ASan)
- Thread sanitizers (TSan)
- Memory sanitizer (MSan) -fsanitize=memory
 - detects uninitialized memory reads

Tools: Sanitizers

- Address Sanitizer (ASan)
- Thread sanitizers (TSan)
- Memory sanitizer (MSan)
- Undefined Behavior sanitizer (UBSan) -fsanitize=undefined
 - using null pointers
 - signed integer overflow
 - floats conversion with overflow
 - out-of-bounds array index
 - division by zero
 - more

Tools: Sanitizers

- Address Sanitizer (ASan)
- Thread sanitizers (TSan)
- Memory sanitizer (MSan)
- Undefined Behavior sanitizer (UBSan)
- Data Flow sanitizer API

Tools for code quality

1. Follow the code style, formatter
2. Generate code
3. Run code analysis
4. Refactor
5. Check with unit tests

Tools: Refactoring

- Basic set
 - rename
 - extract Function
 - inline
- Profound set
 - change signature
 - extract variables / parameters / constants / typedefs / defines
 - pull/push members up/down the hierarchy
 - modernize
 - etc.

Tools: Refactoring

Requirements:

- cover tricky modern C++ cases
- distinguish C and C++
- scope: TU vs whole project
- non-code usages

Tools: Refactoring

Clang-rename

- clang family
- integrated into Vim, Emacs
- renaming inside TU

Tools: Refactoring

Test suite for C++ refactoring tools:

- <https://github.com/LegalizeAdulthood/refactor-test-suite>
- ~60 test suites on:
 - change signature
 - create setters, getters
 - extract constant, function, parameter, variable
 - inline macro, variable, etc.
 - rename
 - simplify boolean expression
 - etc.

Tools for code quality

1. Follow the code style, formatter
2. Generate code
3. Run code analysis
4. Refactor
5. Check with unit tests

Tools: unit testing

- Google Test - 45%
- Boost.Test - 26%
- CppUnit - 11%, CppUTest - 5%, Catch - 5%, others

Tools for code quality

1. Follow the code style, formatter
2. Generate code
3. Run code analysis
4. Refactor
5. Check with unit tests
6. Bonus! Package managers

Tools: package manager

Problems:

- Pre-compiled libraries may violate One-Definition Rule
- Usual way
 - download sources -> build -> include into project's build
- A variety of build systems for C++ projects

Tools: package manager

Language level

vs

External

Build system agnostic

vs

Build system built-in

Source-based

vs

Pre-built binaries

System

vs

Language specific

Tools: package manager

A Packaging System for C++ proposal (P0235R0)

- Language level
- By Blizzard
- *using, MANIFEST, options* syntax
- Implementation:
 - <https://github.com/Blizzard/clang>
 - <https://github.com/berkus/clang-packaging-P0235R0>
- cppget first, default public repo later

```
#using package "foo" \
("foo/foo.h", "bar/bar.h") \
[foo, bar.baz] \
version "1.2.3"
```

```
#if PLATFORM == WINDOWS
    #using option PLATFORM = "Win32"
#elif PLATFORM == LINUX
    #using option PLATFORM = "Linux"
#else
    #error Unknown platform
#endif

#if BUILD == DEBUG
    #using option TARGET = "Debug"
#else
    #using option TARGET = "Release"
#endif

#using package "mypackage"
```

Tools: package manager

Hunter: <https://github.com/ruslo/hunter>

- CMake-based
- Linux, Mac, Windows, iOS, Android, Raspberry Pi
- HunterGate
- Shareable directories with packages
- >20 supported packages
- GTest/GMock, Boost, Catch, Android, LLVM, OpenCL, Qt, etc

```
hunter_add_package(Boost COMPONENTS system filesystem iostreams)
find_package(Boost CONFIG REQUIRED system filesystem iostreams)
```

Tools: package manager

Conan: <https://github.com/conan-io/conan>

- External, build-system agnostic
- Predefined helpers for CMake
- Win, Linux, Mac, FreeBSD, SunOS
- C, C++, Python, Go, Fortran
- Download binaries or build from sources
- Web server or local storage
- VS, Xcode, Android Studio, CLion
- Detailed doc: <http://docs.conan.io/en/latest/>

```
[requires]          conanfile
Poco/1.7.8p3@pocoproject/stable

[generators]
cmake

[options]
Poco:shared=True # PACKAGE:OPTION=VALUE
OpenSSL:shared=True
```

```
[settings]          conanprofile
compiler=clang
compiler.version=3.5
compiler.libcxx=libstdc++11

[options]
MyLib:shared=True

[env]
CC=/usr/bin/clang
CXX=/usr/bin/clang++
```

References

- Bjarne Stroustrup, Writing Good C++14
 - [CppCon 2015] <https://www.youtube.com/watch?v=1OEu9C51K2A>
- Timur Doumler, Readable Modern C++
 - [C++ Russia 2017] <https://www.youtube.com/watch?v=6AoifPeOAXM>
- Peter Sommerlad, C++ Core Guidelines - Modernize your C++ Code Base
 - [ACCU 2017] <https://www.youtube.com/watch?v=fQ926v4ZzAM>
- Richard Thomson, Test suite for C++ refactoring tools
 - <https://github.com/LegalizeAdulthood/refactor-test-suite>
- JetBrains, Developer Ecosystem Survey 2017
 - <https://www.jetbrains.com/research/devecosystem-2017/>

**Thank you
for your attention**

Questions?