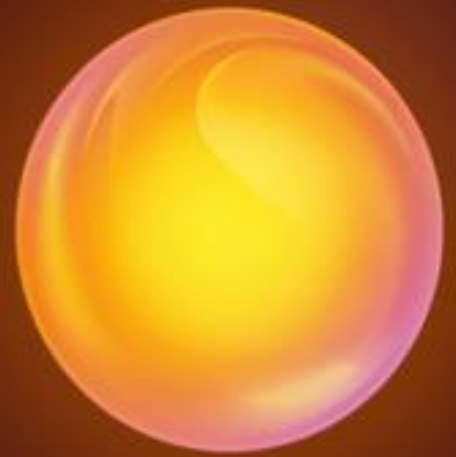
clang-tidy:

C++ с акцентом вашей команды

Андрей Белобров, SberDevices



План доклада

- Что такое clang-tidy?
- Как запускать и какие есть опции?
- Реальные кейсы: bugprone и другие проверки
- Стратегия внедрения: quality gate и zero-warning policy
- Как и зачем обновлять на свежие версии?
- Выводы

Платформа для устройств с виртуальным ассистентом

- Разработка на C++ 20
- Сборка с помощью CMake
- Множество поддерживаемых платформ, основные Android и Linux
- Множество конфигураций сборки под разные устройства

Собственный стандарт кодирования на основе Google C++ Style Guide

Инструменты для автоматического форматирования кода:

C/C++, protobuf	**->**	**clang-format**
CMake	**->**	**cmake-format**
JSON	**->**	**jq**

Автоматизация проверки Code Style

modernize-use-override

Перегруженные методы помечаем словом **override** без **virtual.**

google-explicit-constructor

Конструкторы с одним аргументом помечаем ключевым словом **explicit.**

modernize-use-auto

Не используем **auto** если выведенный тип не очевиден.

Что такое ***clang-tidy***?

*”**clang-tidy** is a clang-based C++ “linter” tool”*

— Находит ошибки: bugprone-*, clang-analyzer-*
— Помогает поддерживать современный стиль C++: modernize-*
— Проверяет правила C++ core guidelines: cppcoreguidelines-*
— Проверяет соответствие Google C++ style guide: google-*
— Помогает улучшить читаемость кода: readability-*
— Находит некоторые проблемы в производительности: performance-*

Запускаем clang-tidy

```
# Формируем compile_commands.json в build каталоге
$ cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=ON -B build


# Передаем build каталог и конфигурационный файл
$ clang-tidy -p build/ -config-file=.clang-tidy test.cc

# Результат
test.cc:1:1: warning: missing username/bug in TODO [google-readability-todo]

# Исходный файл test.cc
// TODO Add doxygen doc
```

Запускаем clang-tidy

```
# Проверяем заголовочные файлы
$ clang-tidy -p build/ --header-filter='(^|/)(libs|src)/' test.cc

# Исправляем замечания
$ clang-tidy -p build/ -fix test.cc

# Результат
testTidy.cc:1:1: warning: missing username/bug in TODO [google-
readability-todo]
1 | // TODO Add doxygen doc
| ^~~~~~~~~~~~~~~~~~~~~~
| // TODO(apbelobrov): Add doxygen doc
test.cc:1:1: note: FIX-IT applied suggested code changes
```

bugprone-use-after-move

```
void SpotterServiceImpl::initSpotter(std::shared_ptr<Spotter> spotter)
{
        spotter_ = std::move(spotter);
        SERVICE_LOG_INFO << "init spotter " << spotter->name();

        spotter_->setCharacter("sber");
        ...
        spotter_->addListener(shared_from_this());
}
```

bugprone-use-after-move

```
void SpotterServiceImpl::initSpotter(std::shared_ptr<Spotter> spotter)
{
        spotter_ = std::move(spotter);
        SERVICE_LOG_INFO << "init spotter " << spotter->name();

        spotter_->setCharacter("sber");
        ...
        spotter_->addListener(shared_from_this());
}
```

'spotter' used after it was moved [bugprone-use-after-move]

bugprone-unchecked-optional-access

```
std::optional<DownloadedModel>    DbProviderImpl::getActiveSpotterModel(SpotterType spotterType) const;


SpotterSettings FWManagerImpl::getSpotterSettings(SpotterType spotterType)
{
    if (auto activeConfig = dbProvider_->getActiveSpotterModel(spotterType)) {
        return {std::move(activeConfig->modelName),
            std::move(activeConfig->model.path),
            activeConfig ? std::move(activeConfig->modelConfig->path) : "",
            false};
    }
    return configProvider_->getDefaultSpotterConfig(spotterType);
}
```

bugprone-unchecked-optional-access

```
std::optional<DownloadedModel>    DbProviderImpl::getActiveSpotterModel(SpotterType spotterType) const;


SpotterSettings FWManagerImpl::getSpotterSettings(SpotterType spotterType)
{
    if (auto activeConfig = dbProvider_->getActiveSpotterModel(spotterType)) {
        return {std::move(activeConfig->modelName),
            std::move(activeConfig->model.path),
            activeConfig ? std::move(activeConfig->modelConfig->path) : "",
            false};
    }
    return configProvider_->getDefaultSpotterConfig(spotterType);
}
```

unchecked access to optional value [bugprone-unchecked-optional-access]

bugprone-unchecked-optional-access

```
if (event_queue_.front().has_value()) {
        notifyPhraseSpotted(event_queue_.front().value());
}
```

bugprone-unchecked-optional-access

```
if (event_queue_.front().has_value()) {
        notifyPhraseSpotted(event_queue_.front().value());
}
```

unchecked access to optional value [bugprone-unchecked-optional-access]

Function results are not assumed to be stable across calls, except for ***const accessor*** *methods.*

bugprone-unchecked-optional-access

```
if (const auto& event = event_queue_.front(); event.has_value()) {
        notifyPhraseSpotted(event.value());
}
```

~~unchecked access to optional value [bugprone-unchecked-optional-access]~~

«Распаковка» может привести к багам

bugprone-unchecked-optional-access:

- Помогает находить ошибки и опечатки при использовании std::optional

- Требует рефакторинга когда для использования безопасных паттернов обращения к optional переменным

- Остается проблема с классами, которые также имеют ***nullable*** состояние помимо std::optional (std::unique_ptr, std::shared_ptr)

LitFill, the Sanct. .unwrap() attracts bugs. DOUBLE-R

bugprone-suspicious-stringview-data-usage

```
std::string_view Asset::getStringView() const;

std::string jsonContent = jsonAsset->getStringView().data();
```

bugprone-suspicious-stringview-data-usage

```
std::string_view Asset::getStringView() const;

std::string jsonContent = jsonAsset->getStringView().data();
```

warning: result of a `data()` call may not be null terminated, provide size information to the callee to prevent potential issues [bugprone-suspicious-stringview-data-usage]

modernize-avoid-c-arrays

```
constexpr size_t bufferSize = 4096;
char buffer[bufferSize];
```

```
do not declare C-style arrays, use std::array<> instead
[modernize-avoid-c-arrays]
```

```
constexpr size_t bufferSize = 4096;
std::array<char, bufferSize> buffer;
```

modernize-avoid-c-arrays

```
const char LIKE[] = "like";
const char DISLIKE[] = "dislike";
```

```
do not declare C-style arrays, use std::array<> instead
[modernize-avoid-c-arrays]
```

```
CheckOptions:
  - key:             modernize-avoid-c-arrays.AllowStringArrays
    value:           'true'
```

cppcoreguidelines-virtual-class-destructor

```
class CollectListener
{
public:
	~CollectListener() = default;

	virtual void onCollect(const TelemetryData& data) = 0;
};
```

destructor of 'CollectListener' is public and non-virtual [cppcoreguidelines-virtual-class-destructor]

C.35: A base class destructor should be either public and virtual, or protected and non-virtual

performance-unnecessary-value-param

```
bool ServiceImpl::inWhiteList(std::string file) const
{
        return whiteList_.find(file) != whiteList_.end();
}
```

warning: the parameter 'file' is copied for each invocation but only used as a const reference; consider making it a const reference [performance-unnecessary-value-param]

performance-unnecessary-value-param

```
void SpeedChecker::setConfig(std::string url, int timeout)
{
	speedTestUrl_ = url;
	speedTestTimeout_ = timeout;
}
```

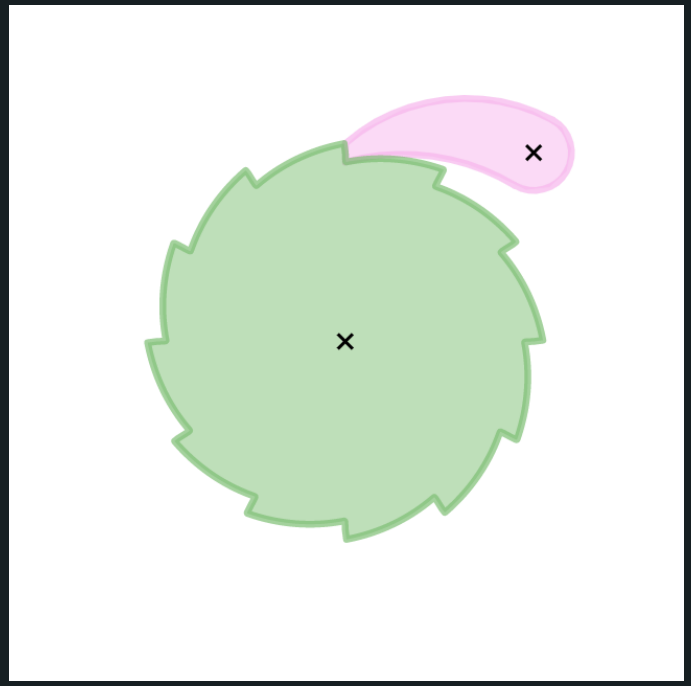
parameter 'url' is passed by value and only copied once; consider moving it to avoid unnecessary copies [performance-unnecessary-value-param]

Вопросы при внедрении статического анализа в существующий проект:

— Какой инструмент выбрать?
— Какие проверки использовать?
— Какой код проверять?

Вопросы при внедрении статического анализа в существующий проект:

- Какой инструмент выбрать?
- Какие проверки использовать?
- Какой код проверять?
- Как настроить quality gate?



Arglin Kampling, CC BY 4.0”
https://commons.wikimedia.org

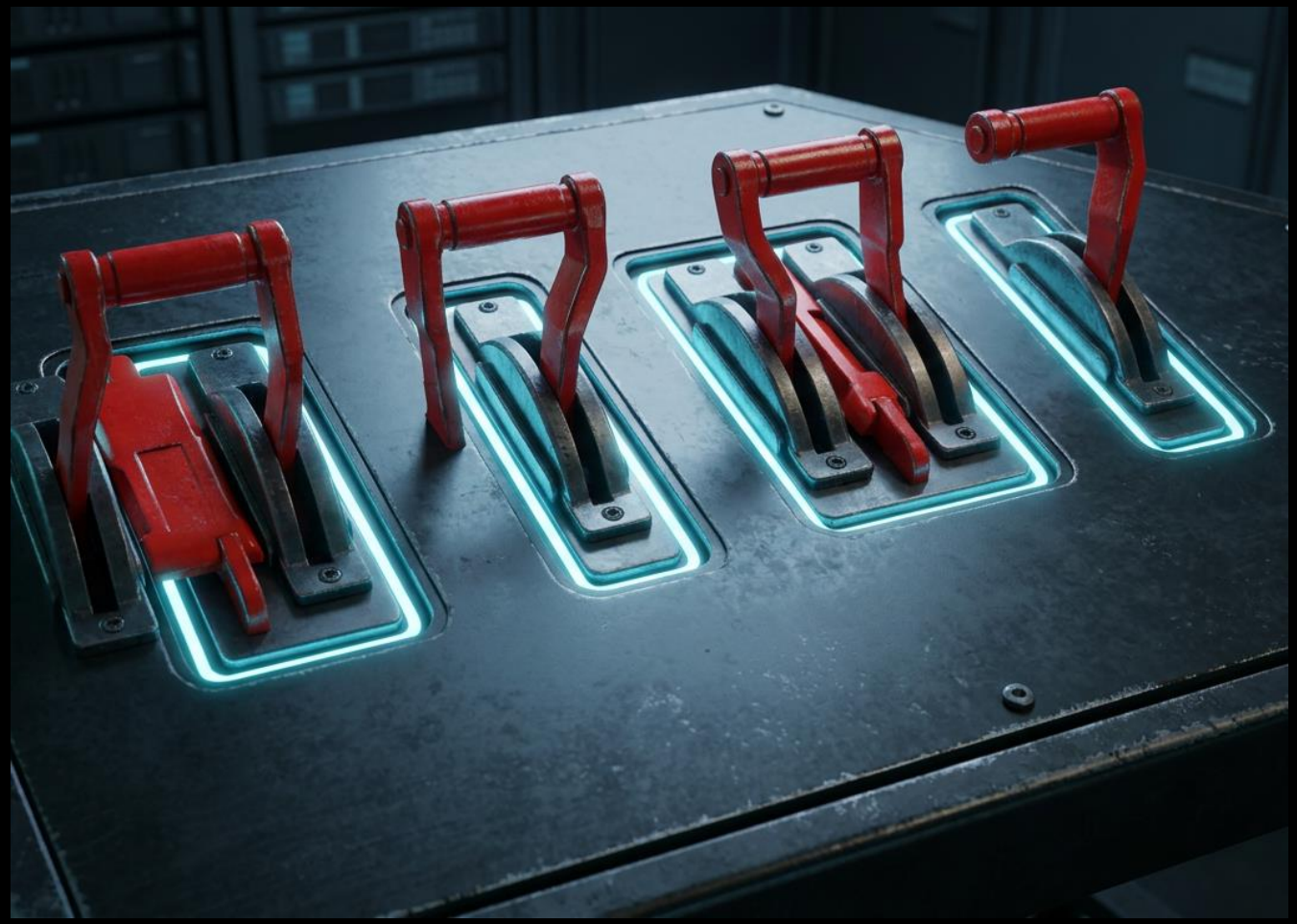
Внедрение в процесс разработки

Общие принципы:

- ***zero-warning policy*** на уровне отдельных файлов и компонентов
- ***единый*** конфигурационный файл **.clang-tidy** для «**чистой**» части проекта
- дополнительный конфиг **.clang-tidy-critical** для **остальной** части проекта

.clang-tidy

```
Checks: '-*,
bugprone-*,
clang-analyzer-*,
google-*,
modernize-*,
performance-*,
portability-*,
readability-*,
misc-*'
```

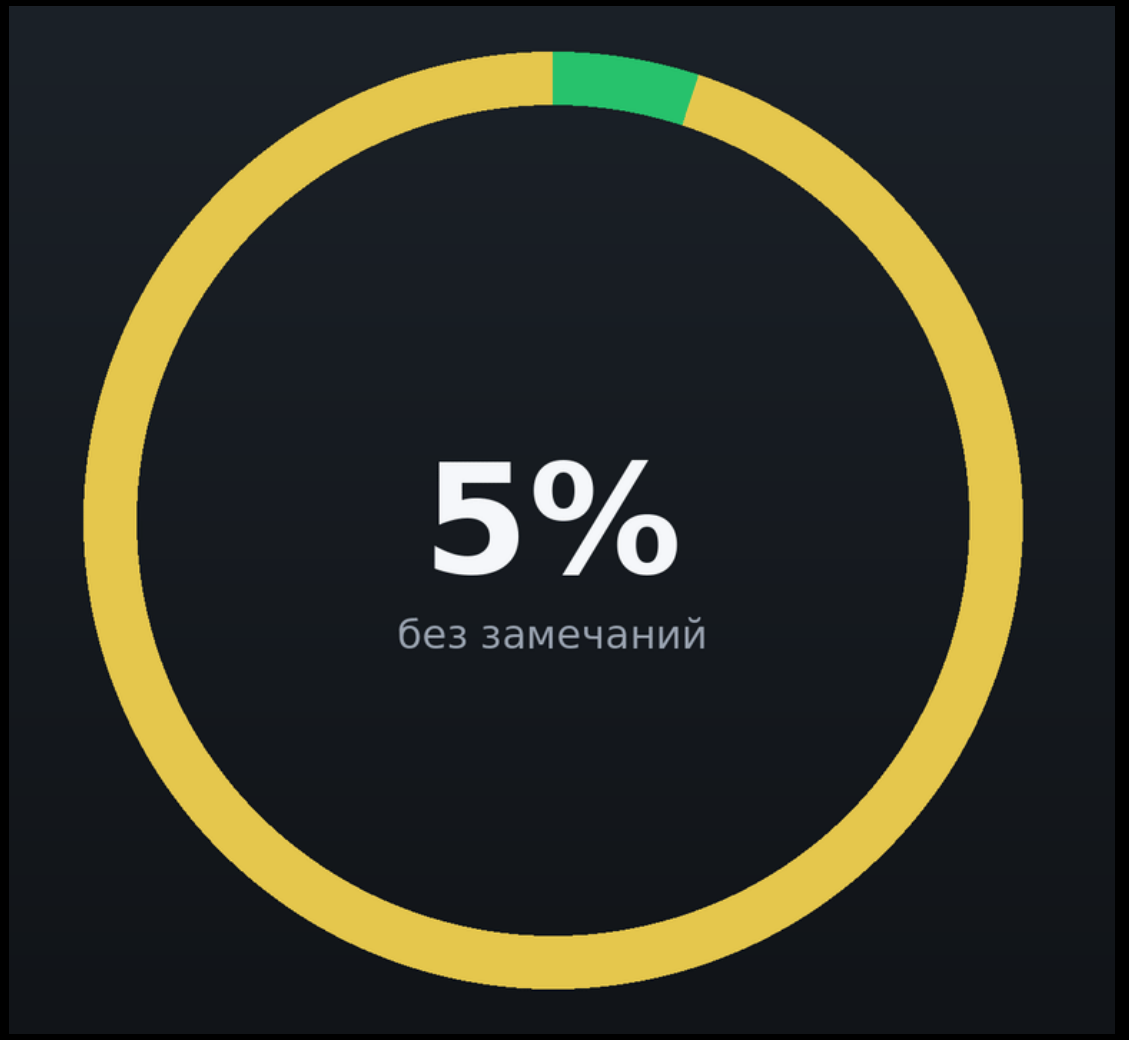


Сгенерировано нейросетью

Постепенное внедрение zero-warning policy

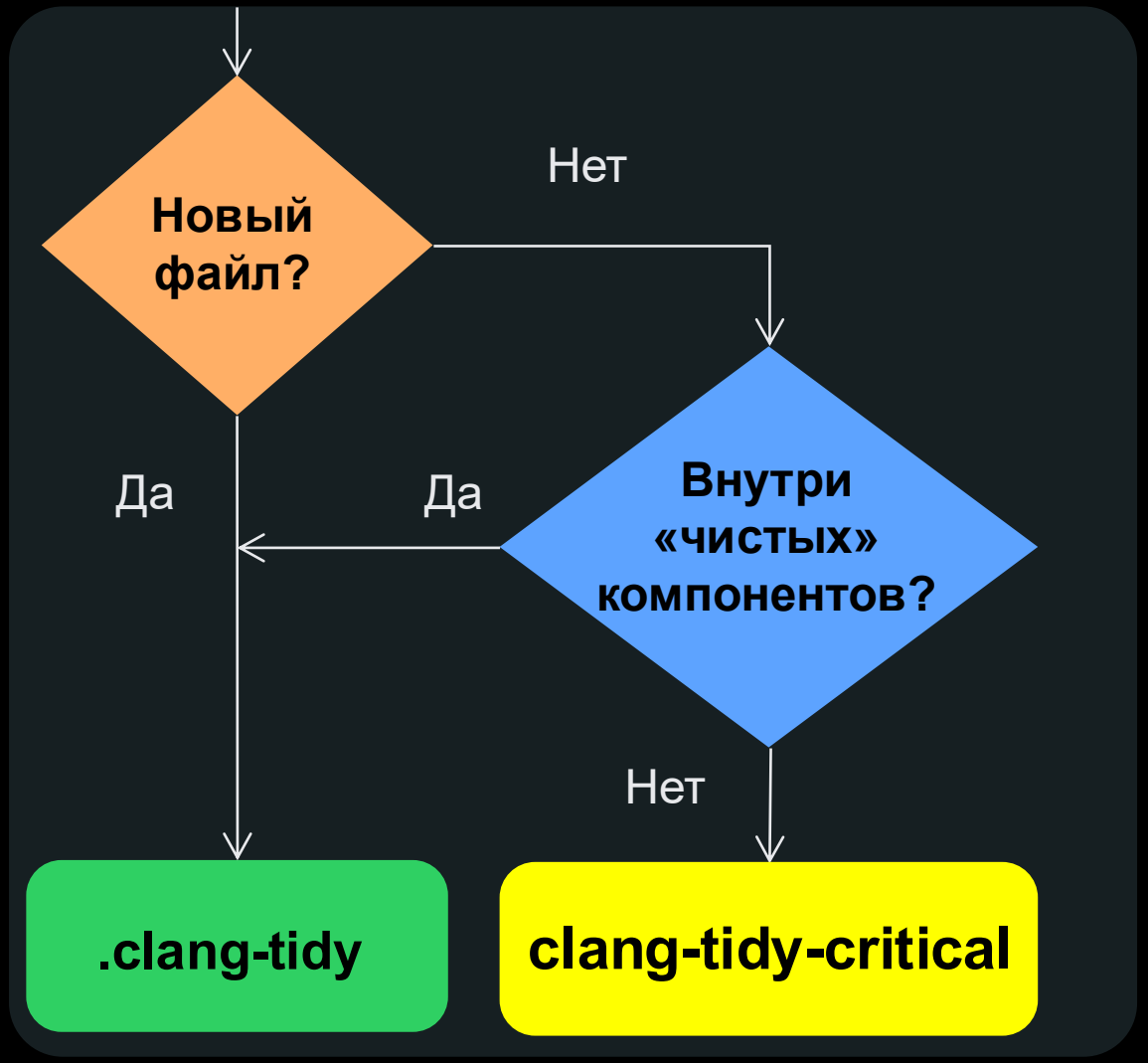
.clang-tidy-dirs

```
AssistantSDK/
SmartHomeSDK/
…
libs/audio/
libs/common/
…
services/assistant/
services/led/
services/network/
services/smartHome/
…
```



Сгенерировано нейросетью

Внедрение в процесс разработки




```
Checks: '-*,
bugprone-*
clang-analyzer-*,
modernize-*,
…

HeaderFilterRegex: "\
AssistantSDK/|\
SmartHomeSDK/|\
libs/audio/|\
libs/common/|\
..."
```

Внедрение в процесс разработки

.clang-tidy-critical

```
Checks: ‘-*,
clang-analyzer-*,

bugprone-use-after-move, bugprone-unchecked-optional-access,

readability-braces-around-statements, readability-isolate-declaration, readability-
redundant-declaration,

cppcoreguidelines-pro-type-const-cast, cppcoreguidelines-virtual-class-destructor’
```

Что изменилось после внедрения

- ✅ Более **95%** кода проекта в зеленой зоне
- ✅ Меньше споров о стиле в code review
- ✅ Быстрее обнаруживаем типовые ошибки
- ✅ Быстрее онбординг новых разработчиков
- ✅ Проще и быстрее внедрять другие анализаторы

Чему мы научились:

– Не нужно бояться выключать предупреждения, которые плохо работают

– Нужно проверять код до того, как он попал в репозиторий

– Исправление замечаний может внести новые баги, нельзя забывать про тестирование

Чему еще мы научились:

- clang-tidy может работать медленно, помогает профилирование
- Нужно регулярно проверять весь проект, запускаем ночную джобу
- Нужно внимательно читать документацию к проверкам
- Не бывает универсальной и стабильной конфигурации с набором проверок

readability-math-missing-parentheses

1

```
int64_t getRawMonotonicMs() noexcept
{
	timespec ts{};
	clock_gettime(CLOCK_MONOTONIC_RAW, &ts);
	return ts.tv_sec * 1000L + ts.tv_nsec / 1000000;
}
```

2

```
int64_t getRawMonotonicMs() noexcept
{
	timespec ts{};
	clock_gettime(CLOCK_MONOTONIC_RAW, &ts);
	return (ts.tv_sec * 1000L) + (ts.tv_nsec / 1000000);
}
```

modernize-avoid-bind

1

```
callbacks_.pingSent = std::bind(&Client::clientPingSent, this);
callbacks_.pongSent = std::bind(&Client::clientPongSent, this);
callbacks_.pingReceived = std::bind(&Client::clientPingReceived, this);
callbacks_.pongReceived = std::bind(&Client::clientPongReceived, this);
```

2

```
callbacks_.pingSent = [this] { clientPingSent(); };
callbacks_.pongSent = [this] { clientPongSent(); };
callbacks_.pingReceived = [this] { clientPingReceived(); };
callbacks_.pongReceived = [this] { clientPongReceived(); };
```

readability-use-anyofallof

1

```
for (const auto& ut : *children) {
        if (ut.u.id == unionId) {
                return true;
        }
}
```

2

```
using std::ranges::any_of;

if (any_of(children, [&](auto& ut) { return ut.u.id == unionId; })) {
        return true;
}
```

readability-implicit-bool-conversion

1

```
X509* cert = X509_STORE_CTX_get_current_cert(ctx);
if (!cert) {
        return false;
}
```

2

```
X509* cert = X509_STORE_CTX_get_current_cert(ctx);
if (cert == nullptr) {
        return false;
}
```

- key: readability-implicit-bool-conversion.AllowPointerConditions

Зачем обновлять clang-tidy?

Чтобы получить новые проверки:

- bugprone-unchecked-optional-access (15+)
- bugprone-suspicious-stringview-data-usage (19+)

Чтобы получить версию с исправлениями багов:

- bugprone-unchecked-optional-access (19+)

Чтобы получить новые опции для настройки проверок:

- bugprone-unchecked-optional-access (22+)
- cppcoreguidelines-avoid-goto (21+)
- modernize-avoid-c-arrays (19+)

modernize-use-ranges

1

```
bool stringContainsOnlyWhitespaces(const string& str)
{
        return std::all_of(str.cbegin(), str.cend(), [](const char c) {
                return std::isspace(c) != 0; });
}
```

2

```
bool stringContainsOnlyWhitespaces(const string& str)
{
        return std::ranges::all_of(str, [](const char c) {
                return std::isspace(c) != 0; });
}
```

modernize-use-constraints

1

```
template<typename F, typename... Args,
         typename = std::enable_if_t<std::is_same_v<
                    std::invoke_result_t<F, Args...>, bool>>>
bool executeUntilSucceed(int64_t maxWaitTimeoutMs, F&& f, Args&&... args)
```

2

```
template<typename F, typename... Args>
bool executeUntilSucceed(int64_t maxWaitTimeoutMs, F&& f, Args&&... args)
         requires(std::is_same_v<std::invoke_result_t<F, Args...>, bool>)
```

modernize-use-designated-initializers

1

```
HSV beginColor{0.0F, 1.0F, 0.0F};
HSV endColor{0.0F, 1.0F, 1.0F};
```

2

```
HSV beginColor{.hue = 0.0F, .sat = 1.0F, .val = 0.0F};
HSV endColor{.hue = 0.0F, .sat = 1.0F, .val = 1.0F};
```

modernize-use-std-format

1

```
const auto logTag = absl::StrFormat(
        "Task: %s, ID: %s, Left retry count: %d: ", context.taskName,
        context.taskId, context.leftRetryCount);
```

2

```
const auto logTag = std::format(
        "Task: {}, ID: {}, Left retry count: {}: ", context.taskName,        context.taskId,
context.leftRetryCount);
```

Как обновлять Clang-tidy?

.clang-tidy

```
Checks: ‘-*,

bugprone-*,-bugprone-easily-swappable-parameters,

google-*,-google-build-using-namespace,

modernize-*,-modernize-use-trailing-return-type,

cppcoreguidelines-pro-type-member-init,cppcoreguidelines-pro-type-const-cast,cppcoreguidelines-virtual-class-destructor,

misc-definitions-in-headers, misc-unused-parameters’
```

C++ с акцентом нашей команды

- Используем современный C++ 20: **modernize-***
- Используем Google C++ style guide: **google-***
- Допускаем меньше ошибок: **bugprone-*, clang-analyzer-***
- Следуем рекомендациям CERT C++ Coding Standard : **cert-***
- Следим за читаемостью кода: **readability-***
- Следуем рекомендациям C++ Core Guidelines: **cppcoreguidelines-***

C++ с акцентом нашей команды

- Пока не используем `std::format`: portability-restrict-system-includes
- Используем `#pragma once` : ~~portability-avoid-pragma-once~~
- Не запрещаем magic numbers: ~~readability-magic-numbers~~
- Не запрещаем директивы **`using namespace`** : ~~google-build-using-namespace~~
- Пока не проверяем имена переменных : ~~readability-identifier-naming~~

Make developers happy ~~again~~

Ключевые принципы эффективного статического анализа:

- Фокус на удобство разработчика
  - Минимизация ложных срабатываний
  - Сбор обратной связи и работа с ней
- Встраивание статического анализа в рабочий процесс
  - Интеграция в пайплайн код-ревью
  - Поддержка в IDE

https://abseil.io/resources/swe-book/html/ch20.html

Вопросы?

Андрей Белобров, SberDevices

belobrov.andrey@gmail.com

@belobrov_andrey