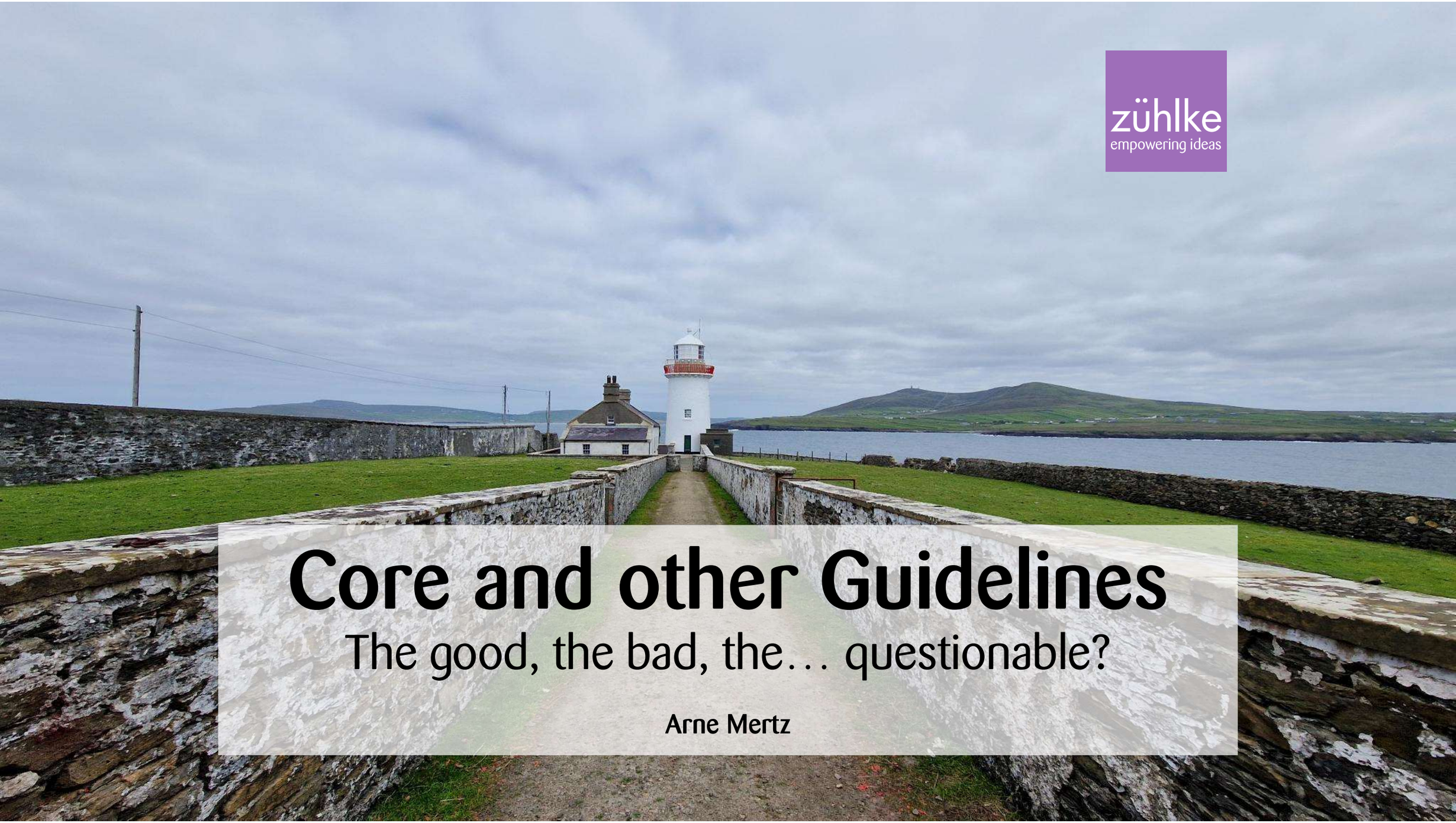

zühlke
empowering ideas

Core and other Guidelines

The good, the bad, the… questionable?

Arne Mertz

Arne Mertz
Software Engineer at Zühlke
(mostly embedded projects)
Working with C++ for ca. two decades
Trainer for C++ and maintainable code

Context

- ~12 projects, 9 years
- Different industries and environments
- Common baseline: “Style guide” documents
  - Provided by customer or project
  - More or less closely followed
  - Often inspired by C++ Core Guidelines (https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines)

Jack of all trades, master of none

Jack of all trades, master of none

"Can do everything, but nothing really well."

Jack of all trades, master of none

????

"Can do everything, but nothing really well."

⚠️ Jack of all trades, master of none

????

"Can do everything, but nothing really well."

Jack of all trades, master of none.
But ofttimes better than a master of one.

Jack of all trades, master of none.
But ofttimes better than a master of one.

We need generalists.

Agenda

A collection of guidelines found in projects

- Where they come from
- What their impact is
- What we can learn from them

Some (opinionated) guidelines for guidelines

A guideline found in the wild

zühlke
empowering ideas

⚠️ Define or delete all copy, move, and destructor functions

(Rule of 5)

C.21: **If** you define or =delete any copy, move, or destructor function, define or =delete them all

(Rule of 5)

C.20:If you can avoid defining default operations, do
(Rule of 0)

C.21: **If** you define or =delete any copy, move, or destructor function, define or =delete them all
(Rule of 5)

⚠️ Define or delete all copy, move, and destructor functions

(Rule of 5)

Impact

Always defining or deleting all special members can lead to:

- Needless boilerplate
- Needlessly nontrivial members
- Needlessly restricting copy/move operations
- *Wrong* boilerplate due to mindless repetition

Rules or Guidelines?

"…the code is more what you'd call *guidelines* than actual *rules*."

Captain Barbossa, Pirates of the Caribbean

Rules or Guidelines?

Developers love rules

- Strict & clear
- Less cognitive load
- We're used to them (compilers, linters, …)
- Often can be checked automatically

Rules or Guidelines?

Guidelines are less strict

- There are exceptions
- More freedom, more responsibility

Avoid interpreting guidelines as rules!

- Prefer to follow them
- Consider providing comments with rationale when breaking them

⚠️ Define or delete all copy, move, and destructor functions

(Rule of 5)

Automated tooling

Consider using automated tools to check for guidelines

- E.g. clang-tidy has a set of core guideline checks:

cppcoreguidelines-special-member-functions

Consider whether you really need a written rule when you have tooling covering it

A guideline found in the wild

zühlke
empowering ideas

⚠️ Don’t use Exceptions

We do not use Exceptions
at Google

We do not use Exceptions
at Google

- For existing code, the introduction of exceptions has implications on all dependent code.
- Most existing C++ code at Google is not prepared to deal with exceptions.
- Our advice against using exceptions is not predicated on philosophical or moral grounds, but practical ones.
- Things would probably be different if we had to do it all over again from scratch.

⚠️ Don’t use Exceptions

Impact

Exceptions are part of the language for a reason

- Handled reliably (return codes can be ignored)
- Handled where they *can* be handled (not necessarily in the calling function)

Not using exceptions has impact on code quality and effort

- Altered return channel or out parameters
- Writing and testing error propagation

Context matters

Guidelines are often not universal

- Different projects and teams require different guidelines
- Applies to rules as well

⚠️ Don't use Exceptions

*unless you are Google

A guideline found in the wild

zühlke
empowering ideas

⚠️ Every function shall have a single return statement. (SESE)

A function shall have a single point of exit at the end of the function.

A function shall have a single point of exit at the end of the function.

MISRA-C-15.5, MISRA-CPP-6.6.5
(IEC 61508, functional safety)

A function shall have a single point of exit at the end of the function.

- Counterargument: guard clauses (early return) are a common pattern that improves readability
- When dealing with resources, use RAII to enable the use of guard clauses

Context matters

Some projects require following external standards, but not every project does

- Consider not repeating those standards in your guideline documents
- Regulations often require a more formal approach
- Often supported by automated tools

⚠️ Every function shall have a single return statement. (SESE)

SESE: Single entry, single exit

SESE: Single ???? entry, single exit

SESE: Single entry, single exit

- Don’t jmp/goto/ENTRY alternate entry points in a function
- Don’t return **to** a different point than the one immediately after the call of your function

⚠️ Every function shall have a single return statement. (SESE)

Impact

zühlke
empowering ideas

```
std::string findProverbOrigin(std::string const& proverb) {
  std::string retVal = "-";
  bool isOK = true;
  if (proverb.empty()) {
    LOG_ERROR("Invalid input");
    isOK = false;
  }
  // more checks...
  if (isOK) {
    /* ... */;
  }
  return retVal;
}
```

Impact

Forcing a single exit reduces code quality

- “ret” value dragged through functions top to bottom
- Deep indentation
- “isOK” value to bypass rest of function in case of failure
- No exceptions

Context matters

Some guidelines are based on ancient texts

- Guidelines made for other languages may not apply
- That includes guidelines for “old” C++

⚠️ Every function shall have a single return statement. (SESE)

*… cite rules that do not apply

A guideline found in the wild

zühlke
empowering ideas

⚠️ Don’t use references as members

C.12: Don’t make data members const or references **in a copyable or movable type**

C.12: Don't make data members const or references **in a copyable or movable type**

- They make such types difficult to use by making them at least partly uncopyable/unmovable for subtle reasons
- Note: use pointers instead (gsl::not_null, if needed)

C.12: Don't make data members const or references **in a copyable or movable type**

i.e. in a type *designed* to be copyable or movable

- They make such types difficult to use by making them at least partly uncopyable/unmovable for subtle reasons
- Note: use pointers instead (gsl::not_null, if needed)

An otherwise copyable type

```
class QuoteRandomizerService
{
  QuoteRepository& m_repository;
  std::vector<QuoteRepository::QuoteId> usedQuotes;
  std::mt19937 randomEngine;
public:
  explicit QuoteRandomizerService(QuoteRepository& repository);
};
```

C.12: Don't make data members const or references

in a type *designed* to be copyable or movable

Example: Application services, factories, repositories, etc.:

- Long object lifetimes, no value semantics, object count rarely changing
- No need for copying or moving

⚠️ Don’t use references as members

Rationale matters

Guidelines are not universal

- The rationale informs us about exceptions

Blindly following guidelines can impair code quality

- E.g. `gsl::not_null<T*>` vs. `T&`

Don’t ignore a guideline to avoid another

- E.g. rule of 0 to avoid C.12

⚠ Don't use references as members

*… be dogmatic

A guideline found in the wild

zühlke
empowering ideas

⚠️ Const goes on the left of the type
(West const)

NL.26: Use conventional const notation

NL.26: Use conventional const notation

Reason: Conventional notation is more familiar to more programmers. Consistency in large code bases.
Note: This is a recommendation for when you have no constraints or better ideas. This rule was added after many requests for guidance.

Read the fine print

Some Core Guideline “rules” are neither

- It’s OK to make your own team conventions

NL: Naming and layout suggestions

- The Core Guidelines explicitly state:

  “These rules are suggested defaults to follow unless you have reasons not to.”

⚠️ Const goes on the left of the type
(West const)

Use automated tooling

Clang-format and other tools take care of formatting

- Use tooling to check them
- Reduce cognitive load by shortening the document

⚠️ Const goes on the left of the type (West const)

*… make me read formatting rules

A guideline found in the wild

zühlke
empowering ideas

⚠️ Use Singleton for objects that exist only once

Singleton

Intent: Ensure a class has only one instance, **and provide a global point of access to it**

Singleton

A global variable makes an object accessible, but it doesn't keep you from instantiating multiple objects.

Singleton

Use when there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.

⚠️ Use Singleton for objects that exist only once

Impact

Hundreds of singletons in one code base

- Tightly coupled monolith
- “Unit test” = 1.5 MLOC tied into every test

Interpretation matters

zühlke
empowering ideas

Guidelines are written and read by humans

- Text has to be interpreted
- Interpretations change
- Get a common understanding

Interpretation matters

Modern interpretation:

If you need global state (you should usually avoid it),
use Singleton.

Alternative: Dependency injection

⚠️ Use Singleton for objects that exist only once

*… unless for **very** good reasons

Guidelines for guidelines

zühlke
empowering ideas

Guidelines for your team

Understand the context & rationale

- When do your guidelines apply, when not?
- Why are they important for your project?

Guidelines for your team

Seek common understanding when and how to break guidelines, e.g.

- Explanatory comments
- Naming conventions
- Code reviews

Concentrate on writing down guidelines you can’t automate

- Use tooling to check for the rest

Automated tooling

Developers love automated tools

- They tell us what to do

Automated tooling

Developers love automated tools

- They tell us what to do
- … do they, though?

- They follow rules, not guidelines
  - They tell us what is (or might be) wrong
  - If the solution is clear, they can fix it automatically

Automated tooling

Get rid of guideline violation warnings

- Turning the waring off is sometimes the right thing
- Fixing the code is sometimes the right thing
- *How* to fix the code is not always easy

Turning off automated tools on a per-case basis

- e.g. C.12 can be checked by clang-tidy

```
QuoteRepository& m_repository;  // NOLINT(*ref-data-members)
```

Automated tooling

Guidelines = freedom = responsibility

- Don’t turn off your brain
- Don’t play the metrics
- Find the *correct* way to address warnings

Guidelines in code reviews

You *will* review code that violates guidelines

- It might be a mistake
- It might be on purpose, and justified
- It might be on purpose, but not justifiable

Find the *correct* way to address the violation, together

Guidelines for guidelines

zühlke
empowering ideas

Keep your guideline documents short

- Who can remember 20 pages of guidelines?

Prefer references over repetition

- Links to Regulations, Core Guidelines, etc. provide context

Guidelines for guidelines

Match your team's situation

- Experience and knowledge
- Surrounding software landscape
- Project specific guidelines

Do you need the Rule of 0/5 in your guideline document?

Guidelines for guidelines

zühlke
empowering ideas

Make it a living document

- Talk about the content
- Reevaluate from time to time

Do you *still* need the Rule of 0/5 in your guidelines after 2 years?

- Consider an archive for new joiners
- Needs to be maintained!

Guidelines for guidelines

zühlke
empowering ideas

What about customers' guidelines?

- The customer is always right

Guidelines for guidelines

zühlke
empowering ideas

What about customers' guidelines?

- The customer is always right – in the matter of taste
- You are the experts
- Guidelines can be challenged

Guidelines for guidelines

zühlke
empowering ideas

YMMV

our ileage ay ary

Different teams can have different situations.

What's yours?

Questions? Comments!

zühlke
empowering ideas

zühlke
empowering ideas

Thank you  Let‘s talk!

🌍 Simplify C++! – www.arne-mertz.de

@arne_mertz@mastodon.social

 arne.mertz@zuehlke.com

 #include<C++> Discord (includecpp.org)

Emojis by emojione.com