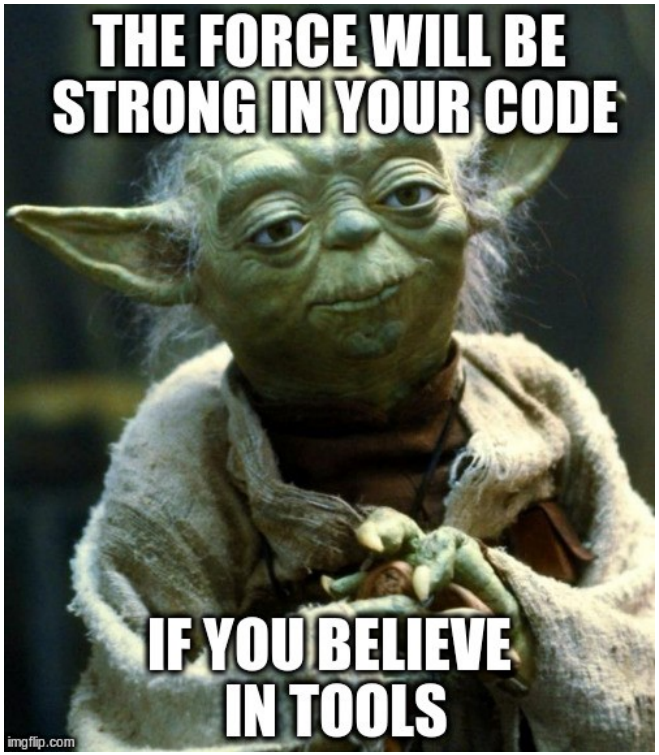
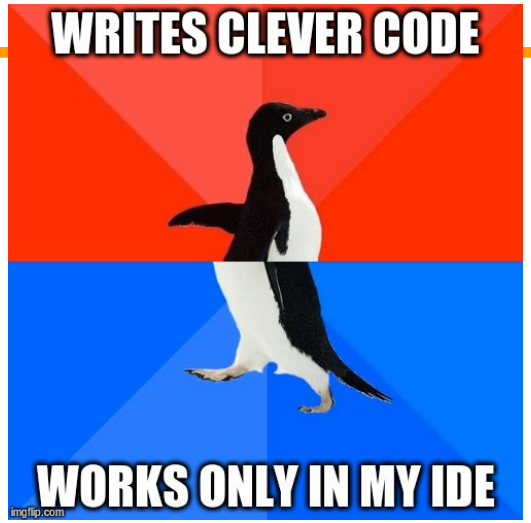
Tooling

- Build Tools
- Compiler Tools
- Static Analyzer
- Paket Manager
- Profiler
- Tests
- CI Pipeline
- Sonstiges



Build Tools



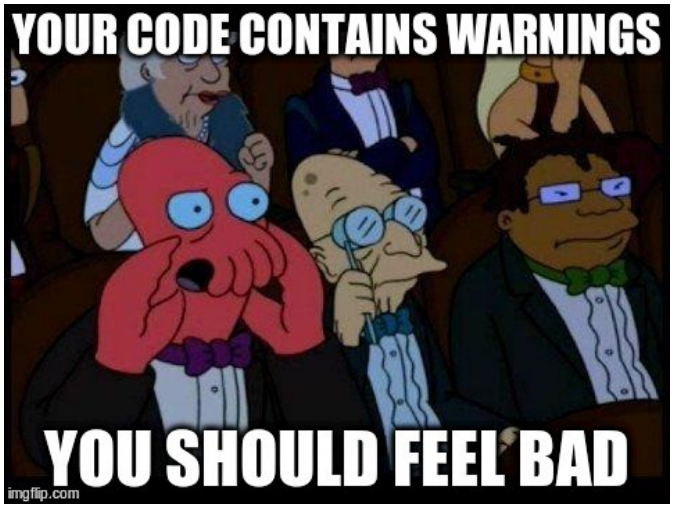
- Voraussetzung um Build im CI auszuführen
- Entwicklungsumgebung ist (meistens) auf der CI Plattform nicht vorhanden
- Garantiert eine gewisse Plattformunabhängigkeit

Build Tools

- Cmake (Industriestandard)
- Qmake (Cmake für QT)
- Make (sehr Komplex)
- Scons
- Autotools (veraltet)

Compiler Tools

- Compiler Flags
  - -Wall , -Wextra : aktiviert alle Compilerwarnungen
    - https://godbolt.org/g/aAKRWe
  - -Wpedantic : Warnt vor nicht Standard konformen Code
    - https://godbolt.org/g/6i83eX
  - -Wshadow : Weist auf mögliche Namenskonflikte hin
    - https://godbolt.org/g/17iXzX
  - -Werror : Macht alle Warnungen zu Fehlern
- GCC und Clang flags fast identisch
- MSVC leider mit kryptischen Nummerierungen
- Releasebuild sollte mit allen Flags
  ohne Warnungen compilieren!!



Compiler Tools

Sanitizer

- Compiler-Fehlermeldungen zur Laufzeit
  - Adress-Sanitizer überprüft z.B. Speicherzugriffsfehler
  - Leak-Sanitizer informiert über geleakten Speicher
  - Undefined behavior Sanitizer meldet undefiniertes Verhalten wie int/float Überläufe
  - Thread Sanitizer meldet Dataraces, Deadlocks etc.
  - Und noch einige mehr
- In GCC und Clang standardmäßig „verbaut“
- In MSVC nicht vorhanden
- Im CI nur für Tests Sinnvoll

Static Analyzer

- Static Analyzer prüfen Dateien auf semantische Korrektheit und Fehler.
  - Vorteil: Funktioniert ohne Binärdateien
  - Nachteil: Keine Laufzeitinformationen, öfters mal False-Positive
- Bekannteste Vertreter
  - CppCheck
  - Clang-tidy (enthält einen Core-Guideline Checker)

Paket Manager

- Da C++ Binärdateien braucht, laden Paketmanager die Sourcen runter und kompilieren die für das System
- Conan
  - Open Source C++ Paket Manager.
  - Bereits große Auswahl an Pakete. CMake Integration
  - Lokale Repos verwendbar
  - https://conan.io/
- C++ Aktive Network
  - Open Source C++ Paket Manager
  - CMake Integration
  - Riesige Paketauswahl
  - Einfache Bedienung
  - https://cppan.org/
- Cmake
  - Kann auch Abhängigkeiten installieren
  - https://github.com/Dobiasd/FunctionalPlus#way-2-using-cmakes-externalproject

Profiler

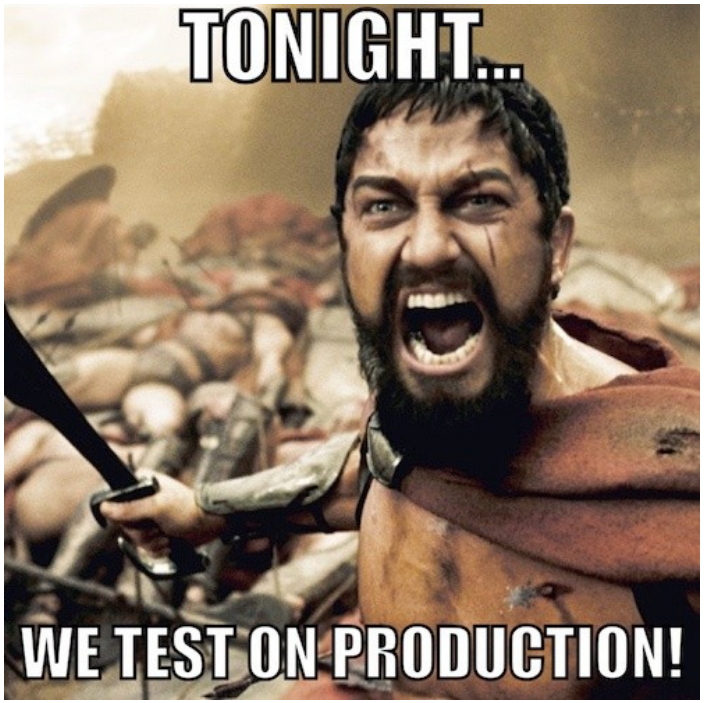
- Testet Laufzeitverhalten von Code
  - Wie lange ein Funktion braucht
  - Wie oft bestimmte Funktionen aufgerufen werden
- Bevor man auf Perfomance Optimiert unbedingt profilen!!!
- Richtiges Profilen oft schwierig, da oft nicht einsehbare Effekte den Code „bremsen“
- Sauberer, lesbarer Code oft wichtiger.
- http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rper-reason

Profiler

- Gprof (Linux)
  - Einfaches Profiling Tool im text Format
  - Muss zusätzliches LinkerFlag „-pg“ angegeben werden
  - Kaum Laufzeit Overhead, aber nur ungenaues Profilen
- Callgrind
  - Bestandteil von Valgrind, Visialisierung mit cachegrind
  - Keine zusätzlichen Compile oder Linkerflags
  - Massiver Performance-Overhead
  - Sehr ausführliche Visualisierungen
- Visual Studio
  - Bereits integrierter Profiler in der IDE
  - Zurzeit eins der besten Programme dafür

Tests

- CPPUTest
  - Test Framework für Unix Systeme
  - Sehr Schlank, ausgelegt für Embeddes Systeme
  - Fertige Eclipse Integration
- GoogleTest
  - Plattform Übergreifendes Test Framework
  - Industriestandart
  - Extrem viele Tools und Funktionen
- Catch 2
  - Leichtgewichtiges Single-Header Framework
  - Schlanke Syntax



Tests

- Mocking bei Klassenschwierig in C++
  - Ableiten funktioniert nur, wenn Funktion virtuell ist
  - Also alle gemockten Fuktionen
    - Als Interface
    - Als Virtual

CI Pipeline

- Pipelines garantieren eine gewisse Codequalität bei jedem Commit/Merge/Release
- Code sollte auf möglichst vielen Compilern getestet werden
  - Garantiert Plattformunabhängigkeit
  - Compiler prüfen oft unterschiedliche Fehler
- Pipeline sollte mindestens Enthalten
  - Compilieren
  - Testen
  - Statische Codeanalyse
- Vor allem auf Embedded Systemen!!!
  - Docker auf für ARM und andere Embedded Systeme verfügbar

CI Pipeline

- GitHub
  - Fertige Erweiterungen
  - Einfach zu Integrieren
  - Travis / AppVeyor für Buildsysteme unter Linux / Mac / Windows
  - Doxygen Integration: codedocs.xyz
  - Linting: linthub.io

CI Pipeline

- GitLab
  - Umfangreiche CI bereits integriert
  - Eigene Docker Registry
  - Notation in Komplexen Fällen etwas umständlich
  - Kann Privat aufgesetzt werden oder auf gitlab.com

CI Pipeline

- Jenkins
  - Extrem Mächtiges CI Tool
  - Mehrere 1000 Plugins, frei Konfigurierbar
  - Muss selbst aufgesetzt werden
  - Steile Lernkurve

Sonstiges

- https://github.com/lefticus/cppbestpractices/blob/master/02-Use_the_Tools_Available.md
- Reverse Debugging: undo.io (leider Kostenpflichtig)
- Clang-format für automatische Code Formatierung

Nächste Vorlesung:

Ausblick C++17 / C++20