Unit testing

- Konzeptionelles Aufteilen von geschriebenem Code in *units* (oft Funktionen bzw. Methoden, gelegentlich auch alle Methoden einer ganzen Klasse)
- Formulieren von Beispielen mit bekannten Ergebnissen (oder zumindest Eigenschaften) als Code
- Ausführen von Test-Code möglichst oft (z.B. bei jedem build) ⟶ liefert Daten/Eindruck bzgl. *Korrektheit* des Codes

Unit testing frameworks

- *Frameworks* bieten Werkzeuge um formulieren und ausführen von unit tests bequemer zu machen
- Hier speziell betrachtet: Google Test

Beispiel: Addition

gtest_add/add.h

```
#pragma once

int add(int a, int b);
```

gtest_add/add.cpp

```
#include "add.h"

int add(int a, int b) {
  return a + b;
}
```

gtest_add/add_unittest.cpp

```
#include "add.h"
#include <gtest/gtest.h>

namespace {
  TEST(Add, Negative) {
    EXPECT_EQ(add(2, -2), 0);
  }
  TEST(Add, Four) {
    EXPECT_EQ(add(2, 2), 4);
  }
}
```

gtest_add/main.cpp

```
#include "add.h"
#include <iostream>

using namespace std;

int main() {
  int a, b;
  while (cin >> a >> b)
    cout << add(a, b) << endl;
}
```

gtest_add/meson.build

```
project('add', 'cpp')

add_lib = library('add_lib', 'add.cpp')
executable('add', 'main.cpp', link_with : add_lib)

gtest = subproject('gtest')
test('gtest',
  executable('gtest_add', 'add_unittest.cpp', 'add.cpp',
  ↪  dependencies : gtest.get_variable('gtest_main_dep'))
)
```

```
meson test -C build
```

Test Fixtures

- Häufig mehrere Tests für selbe „Situation“ gewünscht; z.B. Eigenschaften einer bestimmten Datenstruktur
- Ansatz von Google Test ist RAII
- *Fixture* ist Klasse, Konstruktor etabliert „Situation“, jeder Test kann automatisch erzeugtes Objekt der Klasse verwenden
- Fixture Klassen müssen (`public`) Erben von `testing::Test`
- Implementierung von Google Test verwendet Klasse die wiederrum von Fixture erbt; Konstruktor und relevante Attribute müssen mindestens `protected` sein

Beispiel: Test Fixture

gtest_vector/vector_unittest.cpp

```cpp
#include <vector>
#include <algorithm>
#include <functional>
#include <gtest/gtest.h>

namespace {
  using namespace std;

  class VectorFixture : public testing::Test {
    protected:
      vector<int> vs;

      VectorFixture(): vs({1, 4, 3, 2, 5}) {}
  };

  TEST_F(VectorFixture, DoubleReverse) {
    vector<int> vs_orig(vs);
    reverse(vs.begin(), vs.end());
    reverse(vs.begin(), vs.end());
    EXPECT_EQ(vs_orig, vs);
  }
  TEST_F(VectorFixture, SortReverse) {
    sort(vs.begin(), vs.end(), greater<int>());

    vector<int> vs_comm(vs);
    sort(vs_comm.begin(), vs_comm.end(), less<int>());
    reverse(vs_comm.begin(), vs_comm.end());

    EXPECT_EQ(vs, vs_comm);
  }
}
```