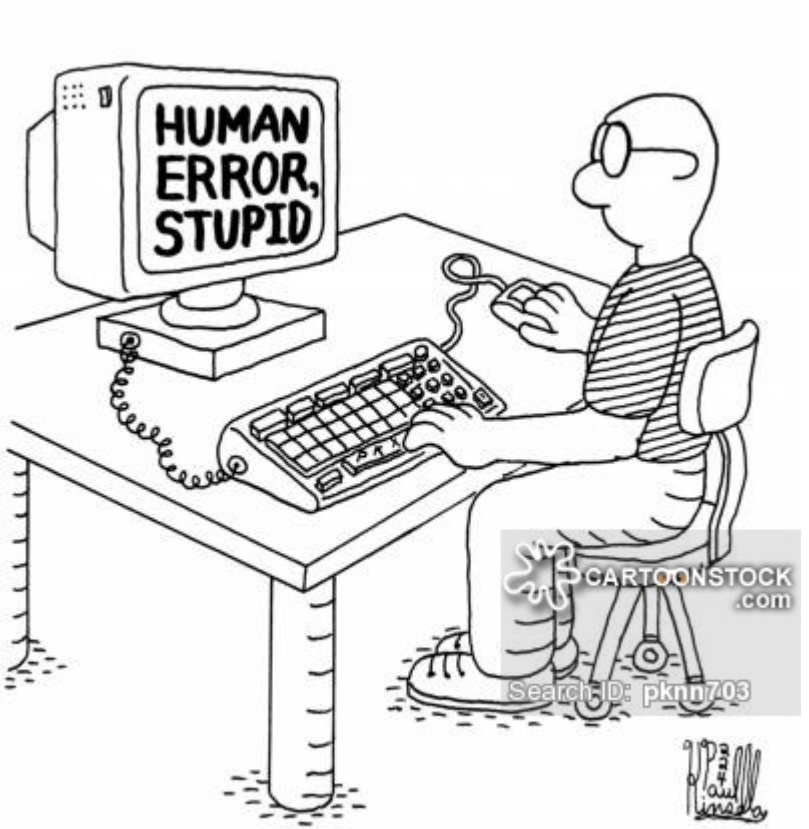
Code Quality Tools

GRK 1504 Spring Block Course (March 18, 2015)

Lorenz Hauswald

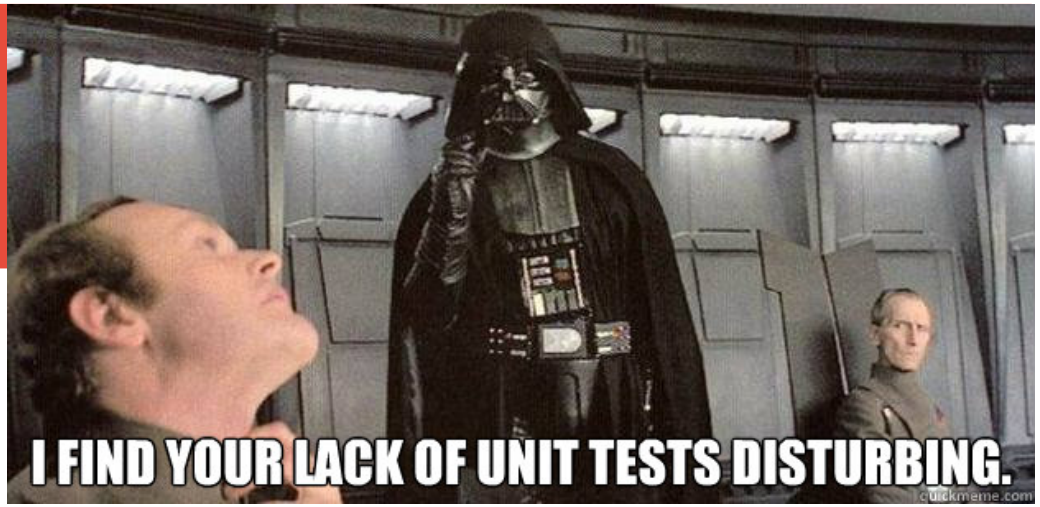
Motivation

- **We heavily rely on our software to produce good results**
- **We make mistakes that will probably…**
  - cost us time later
  - screw up our results
  - ruin somebody else's day
- **mostly avoidable**



(c) http://www.cartoonstock.com/directory/p/programing.asp

Unit Testing

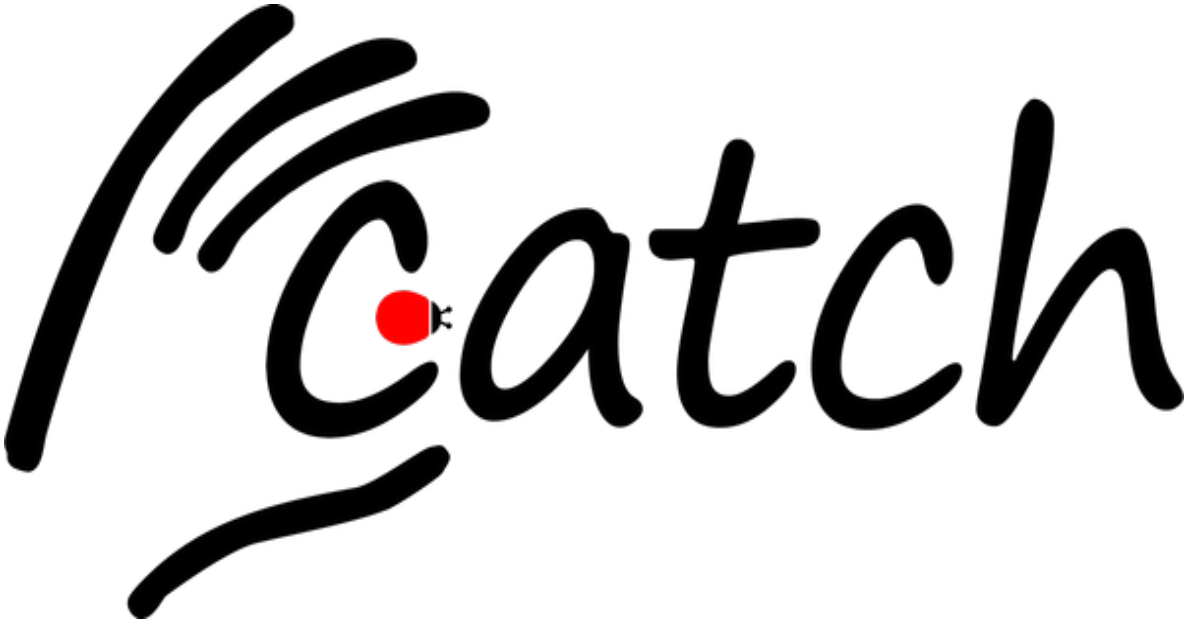


(c) https://raygun.io/blog/2015/02/unit-tests-great-peace-mind/

- **test each part of the program separately**
- **can be rerun routinely (e.g. test driven development)**
- **give confidence and ability to do additions/refactoring w/o fear**
- **Define test cases:**
  - What is the function/class supposed to do?
- **Test on example input**
  - Are there interesting corner cases?
  - What about error conditions?

CATCH – a C++ Unit Testing Framework

- single header, no ext. dependencies
- easy to learn
- nice output
- BDD style
- no fixtures
- let's have a look at some examples



https://github.com/philsquared/Catch

CATCH: Example

```
double deltaPhi(double phi1, double phi2){
  return phi1 - phi2;
}
```

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

TEST_CASE("DeltaPhi is computed correctly"){
  REQUIRE( deltaPhi(0, M_PI/6) == M_PI/6 );
}
```

CATCH: Example

```
double deltaPhi(double phi1, double phi2){
   return phi1 - phi2;
}
```

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

TEST_CASE("DeltaPhi is computed correctly"){
   REQUIRE( deltaPhi(0, M_PI/6) == M_PI/6 );
}
```

```
test.cxx:7: FAILED:
  REQUIRE( deltaPhi(0, 3.14159265358979323846/6) == 3.14159265358979323846/6 )
with expansion:
  -0.5235987756 == 0.5235987756

===============================================================================
test cases: 1 | 1 failed
assertions: 1 | 1 failed
```

CATCH: Example

```
double deltaPhi(double phi1, double phi2){
  return std::abs(phi1 - phi2);
}
```

```
TEST_CASE("DeltaPhi is computed correctly"){
  REQUIRE( deltaPhi(0, M_PI/6) == M_PI/6 );
}
```

```
test.cxx:7:
PASSED:
  REQUIRE( deltaPhi(0, 3.14159265358979323846/6) == 3.14159265358979323846/6 )
with expansion:
  0.5235987756 == 0.5235987756

===============================================================================
All tests passed (1 assertion in 1 test case)
```

CATCH: Example

```
double deltaPhi(double phi1, double phi2){
  return std::abs(phi1 - phi2);
}
```

```
TEST_CASE("DeltaPhi is computed correctly"){
  REQUIRE( deltaPhi(0, M_PI/6) == M_PI/6 );
  REQUIRE( deltaPhi(0, 2*M_PI) == Approx(0).epsilon(0.001) );
}
```

CATCH: Example

```
double deltaPhi(double phi1, double phi2){
  return std::abs(phi1 - phi2);
}
```

```
TEST_CASE("DeltaPhi is computed correctly"){
  REQUIRE( deltaPhi(0, M_PI/6) == M_PI/6 );
  REQUIRE( deltaPhi(0, 2*M_PI) == Approx(0).epsilon(0.001) );
}
```

```
test.cxx:8: FAILED:
  REQUIRE( deltaPhi(0, 2*3.14159265358979323846) == Approx(0).epsilon(0.001) )
with expansion:
  6.2831853072 == Approx( 0.0 )

===============================================================================
test cases: 1 | 1 failed
assertions: 2 | 1 passed | 1 failed
```

CATCH: Example

```
double deltaPhi(double phi1, double phi2){
  double diff = std::abs(phi1 - phi2);
  while(diff >= 2*M_PI) diff -= 2*M_PI;
  return diff;
}
```

```
TEST_CASE("DeltaPhi is computed correctly"){
  REQUIRE( deltaPhi(0, M_PI/6) == M_PI/6 );
  REQUIRE( deltaPhi(0, 2*M_PI) == Approx(0).epsilon(0.001) );
}
```

```
===============================================================================
All tests passed (2 assertions in 1 test case)
```

Other Assertion Macros

```
//inverted macro:
REQUIRE_FALSE(/*...*/);

//continue checking assertions in test case:
CHECK(/*...*/);

//exception related:
REQUIRE_THROWS( deltaPhi(NAN, 3) );
REQUIRE_THROWS_AS(/*...*/, exception_type);
REQUIRE_NOTHROW(/*...*/);
```

Logging Macros

```
INFO("This message is logged and shown if
something fails");

WARN("Warnings are always shown, but don't fail
the test");

FAIL("Shows and fails the test case");

CAPTURE( interestingVariable );
//is shown as:
//interestingVariable := 2.718281828
```

Test Sections

```
TEST_CASE("HistogramManager test"){
  auto hm = std::make_shared<HistogramManager>();

  SECTION("add and retrieve TH1F"){/*...*/}
  SECTION("add and retrieve TH2F"){/*...*/}
  SECTION("setting histogram set name"){/*...*/}
  SECTION("getting hist in folder"){/*...*/}

  SECTION("set binding and autofill set"){
    TH1F* hist = new TH1F("testname", "test", 20, 0, 2);
    hm->AddHist(hist, "set");

    SECTION("set binding to pointer to scalar"){/*...*/}
    SECTION("set binding to pointer to vector"){/*...*/}
    SECTION("set binding to scalar function"){/*...*/}
    SECTION("set binding to vector function"){/*...*/}
    SECTION("autofill all sets with weight"){/*...*/}
  }
}
```

Test Sections

```
TEST_CASE("HistogramManager test"){
  auto hm = std::make_shared<HistogramManager>();

  SECTION("add and retrieve TH1F"){/*...*/}
  SECTION("add and retrieve TH2F"){/*...*/}
  SECTION("setting histogram set name"){/*...*/}
  SECTION("getting hist in folder"){/*...*/}

  SECTION("set binding and autofill set"){
    TH1F* hist = new TH1F("testname", "test", 20, 0, 2);
    hm->AddHist(hist, "set");

    SECTION("set binding to pointer to scalar"){/*...*/}
    SECTION("set binding to pointer to vector"){/*...*/}
    SECTION("set binding to scalar function"){/*...*/}
    SECTION("set binding to vector function"){/*...*/}
    SECTION("autofill all sets with weight"){/*...*/}
  }
}
```

Behaviour Driven Design Test Style

```
SCENARIO( "vectors can be sized and resized", "[vector]" ) {
  GIVEN( "A vector with some items" ) {
    std::vector<int> v( 5 );
    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 5 );
    WHEN( "more capacity is reserved" ) {
      v.reserve( 10 );
      THEN( "the capacity changes but not the size" ) {
        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 10 );
      }
    }
  }
}
/* reported as:
   Scenario: vectors can be sized and resized
     Given: A vector with some items
      When: more capacity is reserved
      Then: the capacity changes but not the size      */
```

Static Code Analysis

- analyze code without execution
- reasoning about all possible paths
- can enforce style guidelines

(c) https://blog.logentries.com/wp-content/uploads/2014/07/static-analysis-tool1.jpg

- null pointer dereference
- conditional initialization
- infinite recursive loops
- memory leaks
- out of bounds checks
- obsolete/unsafe functions
- ...

cppcheck

- cross-platform
- free (GNU GPL)

```
class TestObject{
public:
  TestObject(int t) : m_test(t) {}
  inline int testf() { return m_test; }
private:
  int m_test;
};
```

```
(style, inconclusive) Technically the member function
                      'TestObject::testf' can be const.
```

cppcheck Example

```
TestObject* x;
if(value > 0){
  if(value == 1) x = nullptr;
  else if(value == 2) x = new TestObject(4);
  std::cout << x->testf() << std::endl;
  delete x;
  x->testf();
}
```

cppcheck Example

```
TestObject* x;
if(value > 0){
  if(value == 1) x = nullptr;
  else if(value == 2) x = new TestObject(4);
  std::cout << x->testf() << std::endl;
  delete x;
  x->testf();
}
```

```
(error) Memory is allocated but not initialized: x
```

```
(error, inconclusive) Possible null pointer dereference: x
```

```
(style) The scope of the variable 'x' can be reduced.
```

```
(error) Dereferencing 'x' after it is deallocated / released
```

shellcheck

- static code analysis for shell scripts
- also runs online at: www.shellcheck.net

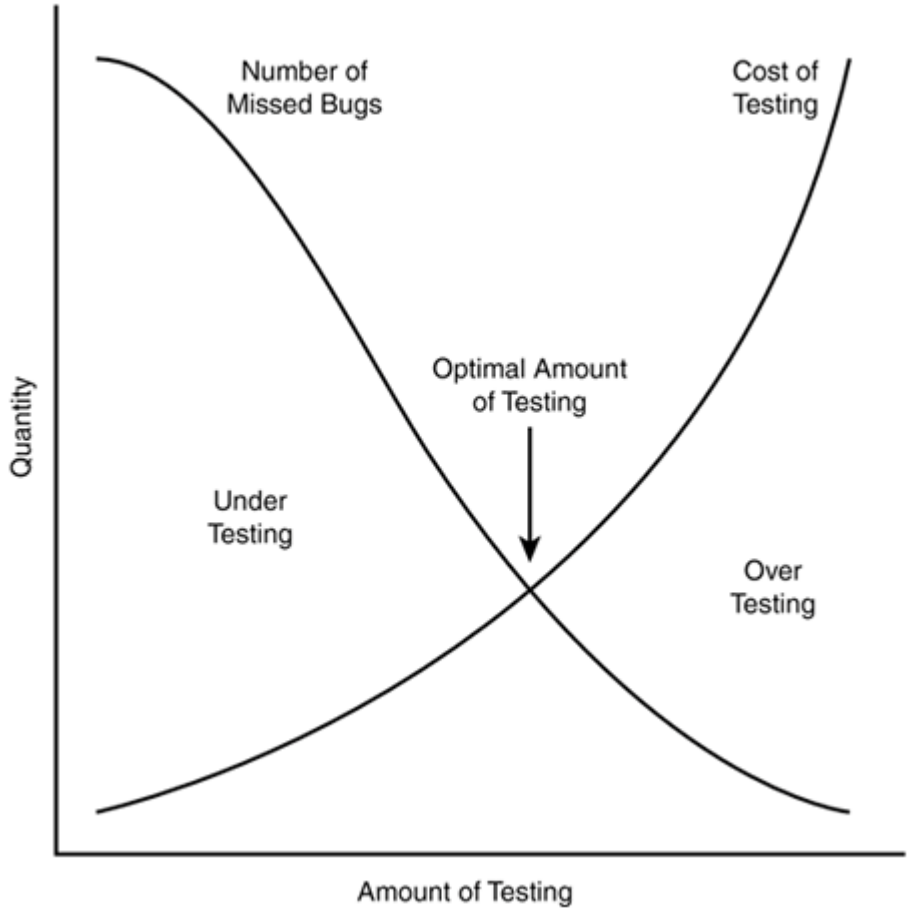
```
1  #!/bin/sh
2  ## Example of a broken script. Hit the Down Arrow button to ShellCheck it!
3  for f in $(ls *.m3u)
           ^-- SC2045 Iterating over ls output is fragile. Use globs.
               ^-- SC2035 Use ./*.m3u so names with dashes won't become options.
4  do
5    grep -qi hq.*mp3 $f \
             ^-- SC2062 Quote the grep pattern so the shell won't interpret it.
                 ^-- SC2086 Double quote to prevent globbing and word splitting.
6      && echo -e 'Playlist $f contains a HQ file in mp3 format'
             ^-- SC2039 In POSIX sh, echo flags are not supported.
                 ^-- SC2016 Expressions don't expand in single quotes, use double quotes for that.
7  done
8
```

Pylint

- static analysis for Python
- error detection
- generate UML diagrams
- Python PEP8 style guidelines
- gives a total score out of 10.0
- integration in editors and IDEs

Conclusion

- **unit testing:**
  - takes some time
  - many well-known frameworks available
  - CATCH: relatively new, easy but powerful
- **static code analysis**
  - almost no effort needed to get results
  - cppcheck, shellcheck, pylint
- **Thanks. Questions?**



(c) http://unica.altervista.org/javascript-unit-testing-lecture/src/images/optimalTestEffort.png