LA-UR-17-24877

Approved for public release; distribution is unlimited.

Title: Software Testing - Auditing Software Quality Assurance Plans

Author(s): Thompson, Kelly Glen

Intended for: Presentation at ISTI Parallel Computing Summer Research Internship

Issued: 2017-06-16

Disclaimer:

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Software Testing

Auditing Software Quality Assurance Plans

2017 June 21

Kelly Thompson kgt@lanl.gov
Computational Physics and Methods Group, CCS-2

EST. 1943
Operated by Los Alamos National Security, LLC for NNSA

UNCLASSIFIED

Slide 1

- Introduce myself
 - Kelly Thompson, blah blah
 - POC Jayenne team, Thermal Radiative Transfer project, 20 years of C++/Fortran at LANL
 - Also support Capsaicin.
 - Total: 600k LOC on all LANL HPC platforms, Roadrunner, Desktop Linux, OSX, Windows.
 - This is mostly a lightweight talk. There are full semester courses on this topic. There are 80 hour professional training classes on this topic. The focus here isn't to be comprehensive, but rather to present a view of testing that my team has found helpful.
 - In the bigger pictures of software, testing is usually one of the last phases (before a release in the waterfall approach), but can also appear at the beginning of a development cycle in XP.
 - As in everything else, we attempt to automate as much of the testing process as possible, but manual checks remain an essential component of testing.
- Hand out placards, pen, paper
- Bring prizes.
- Have a score pad or something similar.

Schedule of Lectures

- Lecture series:
 - June 16 Algorithms and Patterns
 - Hardware/Software Interaction – Part 2
 - Debugging Tools
 - June 19 Heat Equation
 - June 21 Visualization
 - **Software Testing**
 - June 22 Practical I/O
- This will be an interactive lecture with teams, points and prizes.

- Create teams? 3-5 people each. Team name.
- +1 means there will be questions, discussions and points awarded.
- Select panel to grade answers
- **Describe a famous bug (w/o respect to testing).**

F...A...I...L...U...R...E... (Famous bugs)

- Mars Climate Orbiter (1998)
 - \$655M space probe crashes due to wrong units system.

- Thrust values provided in Imperial instead of SI units caused the space craft to fly too close to the planet and disintegrate in the upper atmosphere.

F...A...I...L...U...R...E... (Famous bugs)

- WWII – NORAD reports US was under missile attack (1980) – caused by faulty circuit
- Pentium chip math FDIV bug (1993)
 - Only float precision for specific types of division
- Y2K bug (2000, 2038)
- Sony BMG rootkit exploit (2005)
- Heartbleed (2012)
 - OpenSSL vulnerability

- For these cases, better testing **might** have helped (or maybe not)
- Testing your own code may have less devastating or embarrassing results, but you still need a testing plan.
- **Personal stories: code that you have written. What went wrong? What testing was missing? (points)**
- NORAD
 - In 1980, NORAD reported that the US [was under missile attack](#). The problem was caused by a faulty circuit, a possibility the reporting software hadn't taken into account.
 - Bad inputs lead to bad reports.
 - https://en.wikipedia.org/wiki/North_American_Aerospace_Defense_Command#Cold_War_and_false_alarms
- FDIV
 - Intel attributed the error to missing entries in the lookup table used by the floating-point division circuitry.
 - The man who found the bug points out that since it went unnoticed for a year in a popular product, that likely indicates that the bug was less harmful than IBM suggested.
 - In December 1994, Intel [recalled](#) the defective processors. In January 1995, Intel

announced "a pre-tax charge of \$475 million against earnings, ostensibly the total cost associated with replacement of the flawed processors."^[1]

- https://en.wikipedia.org/wiki/Pentium_FDIV_bug
- Y2K
 - Problems were anticipated, and arose, because twentieth-century software often represented the four-digit year with only the final two digits—making the year 2000 indistinguishable from 1900. The assumption of a twentieth-century date in such programs caused various errors, such as the incorrect display of dates and the inaccurate ordering of automated dated records or real-time events.
 - Also:
 - Monotonically increasing assumption
 - Leap year logic
 - The total cost of the work done in preparation for Y2K is estimated at over US\$300 billion
 - https://en.wikipedia.org/wiki/Year_2000_problem
- Sony BMG rootkit
 - When inserted into a computer, the CDs installed one of two pieces of software which provided a form of [digital rights management](#) (DRM) by modifying the operating system to interfere with CD copying. Neither program could easily be uninstalled, and they created vulnerabilities that were exploited by unrelated [malware](#).
 - https://en.wikipedia.org/wiki/Sony_BMG_copy_protection_rootkit_scandal
- Heartbleed
 - It results from improper input validation (due to a missing [bounds check](#)) in the implementation of the TLS [heartbeat](#) extension,
 - confidential data exposed, private keys compromised, allowed impersonation.
 - <https://en.wikipedia.org/wiki/Heartbleed>

Testing as a part of the SQA Puzzle

- Characteristics of Quality Software
 - What can you test for?
 - Example: Portability across platforms or toolsets.

External and Internal Characteristics -> review list on next 2 slides.

Ask for stories

+3 points for best story related to topics on each of the next 2 slides.

Testing as a part of the SQA Puzzle

Credit: Steve McConnell, Code Complete

- **External Characteristics of Quality Software**
 - **Correctness** (no defects in specification, design, implementation)
 - Usability (how easy is it to use the software or API)
 - **Efficiency** (minimization of system resource use, execution time)
 - **Reliability** (performs required functions under stated conditions)
 - Integrity (prevents improper use, data protection, maintains valid state)
 - Adaptability (can it be used elsewhere)
 - **Accuracy** (free from error wrt quantitative output)
 - **Robustness** (correct function with invalid inputs/state or environmental stress)

- External and Internal Characteristics
- This talk on testing will focus on colored items – mostly because I want to focus on quick, automated testing.
- Some items might be related to opinion or arbitrary rule (i.e.: a coding standard).
- Must check both roots of square root!
- Concern about gamma-ray

Testing as a part of the SQA Puzzle

Credit: Steve McConnell, Code Complete

- *Internal Characteristics of Quality Software*
 - Maintainability (can be updated, enhanced, extended)
 - Flexibility (can be modified for other uses)
 - Portability (can be made to run on non-target systems)
 - Reusability (parts can be used by other software)
 - Readability (can be read and quickly understood)
 - Testability (features can be tested as units and as a whole)
 - Understandability (design/purpose is easily understood)

Software Construction

XP sometimes uses “spiral” model.

The cost of finding and fixing bugs

It is generally accepted that bugs found/fixed late in the software development life cycle become much more expensive to fix.
This should motivate us to test early and often

Testing as a part of the SQA Puzzle

- Types of testing...
 - How do we test for accuracy, robustness, correctness, portability, etc.?
 - When should you do the testing?
 - Example: Basic build checks while designing or constructing new code.

Actual testing methods that I use...

Bug first list, top 3 per team. If they match mine, +1 point each.

Types of testing used in my projects

- How
 - Code review
 - Design-by-Contract
 - Unit Testing
 - Regression Testing
 - Sanitizer builds
- When
 - Configure time
 - Compile time
 - Run time
 - Nightly
 - Continuous
 - On-demand
 - On release
 - Post-release

Used by Jayenne & Draco

Catching bugs early is best.

Contest to list the most items from my list. → points for confirming to my world view ☺

Types of testing used in my projects

- Special cases
 - Code correctness
 - Performance testing
 - Memory testing
 - Flavors
 - Code Reviews
- Other considerations
 - Vendor and machine dependencies
 - Defensive programming, error processing

Next, I will spend some time on each of these topics

Code Review

- Not usually thought of as 'testing'
 - Correctness in Specification, Design, Implementation.
 - Robustness (corner cases)
 - Readability, Understandability
- Defensive programming features
 - Overarching design
 - Level of complexity
 - Good variable, function names, style conformance
 - Debug friendly code?
 - Error processing mechanics
 - Future-proof?

The screenshot shows a code review tool interface. At the top, it says 'Filter: Human review | Manual | Automated' and 'SQA: Correctness | Robustness | Readability'. The code is displayed in a monospaced font with syntax highlighting. Various lines of code are highlighted in green, red, and blue, indicating different review categories or status. Comments and annotations are visible next to the code lines.

EST 1943
Operated by Los Alamos National Security, LLC for NNSA

UNCLASSIFIED

- Done via (1) pair-programming, (2) in person review, (3) via Github pull-request.
- Generally we try to avoid code that would be accepted to the Obfuscated C Contest.
- Sometimes, code is just hard to understand. It may require comments!

Design-by-Contract

- Built into some languages
 - Ada, D, Eiffel
 - Demonstrates intent of designer
 - Constrain complexity
- Commands:
 - `Require( cond )` Precondition
 - `Check( cond )` Logic check
 - `Ensure( cond )` Post condition
 - `Insist( cond, message )` Hard logic check
 - `Remember( code )` Only insert code if DbC is enabled.
 - `check_class_invariants()` Check state of object

Design-by-Contract

- Single most powerful test mechanism for CCS-2 thermal radiative transfer codes
- Implemented as CPP #ifdefs and can be compiled out for optimization.

```
inline double calc_fleck_d(  
    size_t const cell,  
    double const tau) const {  
    Require(cell <= num_cells());  
    Require(tau >= 0);  
    ...  
    Ensure(!(fleck_d < 0.0));  
    Ensure(!(fleck_d > 1.0));  
    return std::min(fleck_d,  
        opacity->get_fleck(cell));  
}
```

Unit Testing / TDD

- Test small sections of code, often at the function-level.
 - Check function or block specification
 - Check behavior if given invalid inputs
 - Do the building blocks for the project work as expected?
 - Often uses fixtures and fake/dummy functions or classes to help isolate the scope of each test (testability).
- Often used during the software construction phase
- Test harnesses often used (CppUnit, GoogleTest, etc.)

```
Test project /scratch/kellyt/gcc-mpid/d/src/ds++/test
Start 1: dsxx_tstScalarUnitTest_arg1
1/32 Test #1: dsxx_tstScalarUnitTest_arg1 ..... Passed    0.01 sec
Start 2: dsxx_tstScalarUnitTest_arg2
2/32 Test #2: dsxx_tstScalarUnitTest_arg2 ..... Passed    0.00 sec
Start 3: dsxx_tstAssert
3/32 Test #3: dsxx_tstAssert ..... Passed    0.00 sec
Start 4: dsxx_tstCheck_Strings
4/32 Test #4: dsxx_tstCheck_Strings ..... Passed    0.01 sec
```

Slide 16

- 1) Example from Draco, github.com/lanl/draco
- 2) In one of my projects, we have two lines of test code for each line of 'real' code.
- 3) Consider 'what if' scenarios – especially for ill-conditioned cases (input)
- 4) Consider ensuring that code will stop and print reasonable message if state is sufficiently ill-conditioned.

Unit Testing 2

```
4: Test: passed
4: Successfully found bad characters in string definition.
4: Found disallowed character(s) in string: "a#"
4: Test: passed
4: result.size() == 3
4: Test: failed
4: result did not have expected values.
4: Test: passed
4: *** Illegal character function test: PASSED ***
4: String length utility test:
4: Test: passed
4: Successfully found bad characters in string definition.
4: Size of string is not in allowable range: "space check"
4: Size of string is not in allowable range: ""
4: Strings lengths must be greater than 1 and less than 4.
4: Test: passed
4: result2 has the expected size.
4: Test: passed
4: result2 has expected content.
4: Test: passed
4: *** String length function test: PASSED ***
4: Unique strings utility test:
4: Test: passed
4: Successfully found bad characters in string definition.
4: Duplicate string found: "is"
4: All strings must be unique!
4: Test: passed
4: result3 has expected content.
4: Test: passed
4: Test: passed
```

```
if (result.size() == 3)
    PASSMSG("result.size() == 3");
if (ut.numFails == 0) {
    if (*result[0] == "a#" &&
        *result[1] == "xxx!" &&
        *result[2] == "space check")
        PASSMSG("result has expected values.");
    else
        FAILMSG("result did not have expected values.");
}
```

Slide 17

Unit Testing 3

- Works best when code is separated into libraries that have acyclic dependencies.
 - Each library can be tested independently of others. Test the lower level libraries first.
 - One collection of tests per object definition or set of flat functions.

Sanitizer Builds

- Compile time checks
 - Tests code correctness against language standards, style guides, and coding practice guides
 - Compiler options like `-Wall` `-Wunused-macros` `-Werror`
 - Effective C++ Guide: `-Weffc++`
 - Sanitize `-fsanitize=all`
 - Alternate STL: `-D_GLIBCXX_DEBUG`
`-D_GLIBCXX_DEBUG_PEDANTIC` or `STLPort`
 - Style: Use of *clang-format*, *uncrustify*, etc.
 - Use of multiple toolsets on multiple platforms (flavors)
 - GNU, Intel, LLVM, MS compilers
 - OpenMPI or MPICH; LAPACK or OpenBLAS

```
source/src/parser/Debug_Options.cc:2
4:6: warning: 'bool
(anonymous)::is_bit(unsigned int)'
defined but not used [-Wunused-
function]
```

1. `-Weffc++` - check against rules from a book: Scott Meyer's "Effective C++"
2. Does it compile?
3. Possibly hundreds of arguments, many are long arrays of data.

Sanitizer Builds

- Configure time
 - System introspection
- Run time
 - Valgrind
 - Code coverage tools
 - Performance profiler

```

valgrind -q --tool=memcheck --leak-check=full --trace-children=yes --gen-suppressions=all --c
MLK
==18516== 128 bytes in 4 blocks are definitely lost in loss record 266 of 366
==18516==   at 0x4C37A78: realloc (vg_replace_malloc.c:788)
==18516==   by 0x51FCDD9: pmix_bfrop_buffer_extend (in /scratch/vendors/spack/20170502/opt
==18516==   by 0x5200AA7: pmix_bfrop_int32 (in /scratch/vendors/spack/20170502/opt
==18516==   by 0x520194D: pmix_bfrop_pack_value (in /scratch/vendors/spack/20170502/opt
==18516==   by 0x52135D8: _esh_store (in /scratch/vendors/spack/20170502/opt/spack/linux
==18516==   by 0x5218883: _job_data_store (in /scratch/vendors/spack/20170602/opt/spack
==18516==   by 0x521F077: _register_nspace (in /scratch/vendors/spack/20170602/opt/spack
==18516==   by 0x51D5EB7: event_process_active_single_queue (event.c:1370)
==18516==   by 0x51D5EB7: event_process_active (event.c:1440)
==18516==   by 0x51D5EB7: opal_libevent2022_event_base_loop (event.c:1644)
==18516==   by 0x522DA6C: progress_engine (in /scratch/vendors/spack/20170502/opt/spack/linux-rhel7-x86_64/gcc-6.3.0/openmpi-2.1.0-vr56hf5solpnbty63xzh3jn4ky3k
==18516==   by 0x549FDC4: start_thread (in /usr/lib64/libpthread-2.17.so)
==18516==   by 0x6CAA73C: clone (in /usr/lib64/libc-2.17.so)
==18516==

```

My CDash All Dashboards Log Out

Jayenne

Dashboard Back Calendar Current Projects

Coverage started on Thursday, May 25 2017

Coverage Summary

Total Coverage	92.9%
Tested lines	312800
Untested lines	23900
Files Covered	412 of 412
Files Satisfactorily Covered	399
Files Unsatisfactorily Covered	13

Show coverage over time

Directories (10) | No Executable Code (0) | Zero (0) | Low (1) | Medium (12) | Satisfactory

Show 25 entries

Directory	Status	Percentage
src/wedgehog_test	Satisfactory	84.24%
src/wedgehog_managers	Satisfactory	87.50%
src/wedgehog_output	Satisfactory	91.18%
src/mc	Satisfactory	91.39%
src/mlagro_interfaces	Satisfactory	92.98%
src/mlagro_data	Satisfactory	93.25%
src/wedgehog_components	Satisfactory	94.34%
src/mc	Satisfactory	95.19%
src/mlagro_manager	Satisfactory	97.50%
src/wedgehog_interfaces	Satisfactory	98.25%
src/mlagro	Complete	100.00%
src/jayenne_release	Complete	100.00%
src/wedgehog	Complete	100.00%
src/mc/v86	Complete	100.00%

Showing 1 to 10 of 10 entries

Integration Testing

- Mode of testing that verifies interfaces between components and against software design
 - Use of stubs for quick, small tests
 - Real code for production testing
 - Scientific coding often has language interfaces (C/C++/Fortran)

```
Arguments arg(angf.make_Arguments(
    ptd, *mesh_data));
run_imc_cycle(arg.problem_type_data,
    arg.zonal_data,
    arg.mesh_data,
    arg.sunf_source_data,
    arg.control_data,
    arg.tally_data);
```

EST 1943
Operated by Los Alamos National Security, LLC for NNSA

UNCLASSIFIED

How: Human review | Manual | Automated
SQA: Correctness | Robustness | Accuracy

```
1222: *****
1222: RESULTS FOR CYCLE 10
1222: ELAPSED TIME 0.01
1222: *****
1222: CELL ELEC TEMP NUH CENSUS RAD TEMP
1222: -----
1222: 1 1.000000e-04 8.000000e+01 9.352222e-01
1222: 2 1.000000e-04 0.000000e+00 0.000000e+00
1222: 3 1.000000e-04 0.000000e+00 0.000000e+00
1222: 4 1.000000e-04 2.850000e+02 9.994105e-01
1222: 5 1.000000e-04 0.000000e+00 0.000000e+00
1222: 6 1.000000e-04 2.390000e+02 9.933819e-01
1222: 7 1.000000e-04 5.000000e+01 9.952801e-01
1222: 8 1.000000e-04 6.400000e+01 1.000016e+00
1222: 9 1.000000e-04 4.700000e+01 1.001056e+00
1222: 10 1.000000e-04 6.600000e+01 1.010560e+00
1222: 11 1.000000e-04 9.000000e+00 1.029585e+00
1222: 12 1.000000e-04 2.900000e+01 1.016122e+00
1222: 13 1.000000e-04 1.400000e+01 1.013809e+00
1222: 14 1.000000e-04 3.100000e+01 1.014159e+00
1222: 15 1.000000e-04 1.600000e+01 1.027957e+00
1222: 16 1.000000e-04 2.400000e+01 9.655552e-01
1222: 17 1.000000e-04 6.000000e+00 1.041900e+00
1222: 18 1.000000e-04 3.400000e+01 1.006479e+00
1222: *****
1222: Test: passed
1222: Electron temp output ok for tstdynamic_AVR on 0
1222: Test: passed
1222: Radiation temp output ok for tstdynamic_AVR on 0
1222: Test: passed
1222: Census numbers ok for tstdynamic_AVR on 0
1222: Test: passed
1222: Electron temp output ok for tstdynamic_AVR on 1
1222: Test: passed
1222: Radiation temp output ok for tstdynamic_AVR on 1
1222: Test: passed
1222: Census numbers ok for tstdynamic_AVR on 1
1222: *****
1222: **** tstdynamic_AVR Test: PASSED.
1222: *****
```

Slide 21

1. Does it compile?
2. Possibly hundreds of arguments, many are long arrays of data.

Verification/Physics Testing

- Usually run a full simulations and compare results to gold standard generated from previous run or another code.
 - Testing is often time consuming (hours?)
 - Typically only optimized 'release' builds are tested
 - Less frequent checks (weekly?)

1. Does it compile?
2. Possibly hundreds of arguments, many are long arrays of data.

Regression Testing

- Run a suite of tests on a regular basis to see if any behavior has changed
 - Automated nightly or continuous testing
 - Across platforms
 - On demand checks
- Catches:
 - Code change has an unintended or unexpected impact.
 - API contract violated
 - Assumptions about machines or environment are invalidated.
 - Conflicting changes committed
- Unit Tests, Integration Tests and Verification Tests

1. If it fails -> look for a bad commit or bad pull-request merge. OR look for a system change.
2. In this case, the fix is often to revert a commit. You might even have to use bisection to isolate the bad commit.

- Machines
- Compilers
- Alt-TPLs
- MemCheck
- Code Coverage
- Sanitizer builds

UNCLASSIFIED

Downside of Regression Testing

- Difficult to setup, maintain
 - What is the pass/fail criteria?
 - Compiles?
 - Gold files?
 - Analytic solutions?
 - Testing differences between platforms like FMA, endianness
- Error resolution often involves the time consuming bisection method.

```
// test the photon densities separately
vector<double> pden_ref(num_cells, 0.0);
pden_ref[0] = 4.699724e-02;
pden_ref[1] = 4.393237e-02;
pden_ref[2] = 3.903708e-02;
pden_ref[3] = 4.370066e-02;
pden_ref[4] = 4.591482e-02;
pden_ref[5] = 3.840650e-02;
pden_ref[6] = 4.007759e-02;
pden_ref[7] = 4.129024e-02;
pden_ref[8] = 4.169960e-02;
pden_ref[9] = 4.094268e-02;
pden_ref[10] = 4.377634e-02;
pden_ref[11] = 4.394518e-02;
pden_ref[12] = 5.084521e-02;
pden_ref[13] = 4.691219e-02;
pden_ref[14] = 4.543738e-02;
pden_ref[15] = 4.572345e-02;

const double eps = 1.0e-6;
if (soft_equiv(argf.photon_density.begin(),
    argf.photon_density.end(),
    pden_ref.begin(),
    pden_ref.end(), eps)) {
    message << "Photon density output ok for "
    << "tstCompton_MG"
    << " on MPI rank " << rtt_c4::node();
    PASSMSG(message.str());
}
```

1. Does it compile?
2. Possibly hundreds of arguments, many are long arrays of data.

Release Testing

- Ensures released application is valid.
- Often repeats existing unit, integration and verification tests
- Assumes source code and build directory are inaccessible.
- Sometimes includes regression testing of old installations.

1. System changes underneath an existing install (OS patches, new MPI, new scheduler system, etc.)

1. Does it compile?
2. Possibly hundreds of arguments, many are long arrays of data.

Summary of testing modes

- How
 - Code review
 - Design-by-Contract
 - Unit Testing
 - Regression Testing
 - Sanitizer builds
- Special cases
 - Code correctness
 - Performance testing
 - Memory testing
 - Flavors
 - Code Reviews
- When
 - Configure time
 - Compile time
 - Run time
 - Nightly
 - Continuous
 - On-demand
 - On release
 - Post-release

Used by Jayenne & Draco
Catching bugs early is best.
Contest to list the most items from my list.

What else is in the testing toolbox?

- Traceability Matrix
 - Match requirements to test cases to passing tests
- Write code that will fail early
 - Compile time checks
 - Run time checks
 - Many platforms, toolsets
- Automation
 - Use test frameworks, Dashboards, CI models.
 - This won't find all the bugs, but it is a good baseline.
- Human element:
 - Code reviews, history tracking, ticket system, code standards, documentation
 - Design-discussions, -prototypes, -language (UML)

- Optional slides:
- Use language features to fail early: correct data types, templates, policies, compile flags (bounds checks), etc.
- GCC has a bad reputation for being sloppy wrt language standards.

Example Traceability Matrix

[illegible]

Effectiveness of testing Percentage of defects detected

Credit: Steve McConnell, Code Complete

Testing mode	Modal Rate
Informal design review	35%
Formal design inspections	55%
Informal code review	25%
Formal code inspection	60%
Modeling or prototyping	65%
Personal desk-checking of code	40%
Unit test	30%
New function test	30%
Integration test	35%
Regression test	25%
System test	40%
Low volume beta test (<10 sites)	35%
High volume beta test (>1000 sites)	75%

- Average: 40% effective
- Best in class organizations: 95% effective
- Your code will have defects!
- Any single method has equal value, using multiple methods is best.
- Using less than three methods produces quality insufficient for production.

Slide 31

Humans better at finding some kinds of bugs.
Automation better at other kinds.

Testing can fail to deliver

- Corner cases
- Exhaustive error catching and recovery is almost impossible
- Round off issues (order of summation)
- Thread contention
- MPI dead locks
- Large code bases (how can you ensure 1 million lines of code are 'correct'?)
- Performance problems

```
fortrl: severe (174): SIGSEGV, segmentation fault occurred
Image PC Routine Line Source
libpthread.so.0 00007F7863F699CB Unknown Unknown Unknown
libiomp5.so 00007F786449E2A8 Unknown Unknown Unknown
```

Los Alamos
NATIONAL LABORATORY
EST 1943

Operated by Los Alamos National Security, LLC for NNSA

UNCLASSIFIED

The screenshot shows the CMake / CMake interface. At the top, there's a search bar with the text 'mingw32-m'. Below the search bar, there are statistics: 'Open: 1,360', 'Closed: 5,719', and 'All: 7,079'. The 'Open' count is circled in red. Below the statistics, there's a search bar with the text 'Search or filter results...'. Below the search bar, there's a message: 'Impossible to build single cpp file using mingw32-m #16922 - opened about 9 hours ago by Marek Bel'. Below the message, there's a blue error message box with the text: 'A problem has been detected and Windows has been shut down to prevent damage to your computer. The problem seems to be caused by the following file: kbhid.sys. MANUALLY_INITIATED_CRASH. If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps. Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need. If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use safe mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode. Technical Information: *** STOP: 0x00000002 (0x00000000, 0x00000000, 0x00000000, 0x00000000) *** kbhid.sys - Address 0x94f00000 base at 0x94f00000 DateStamp 0x41bc705'.

Testing is one piece of the SQA puzzle - An important piece!

- Useful for auditing software
 - Correctness
 - Efficiency
 - Reliability
 - Accuracy
 - Robustness
 - Portability
- Problems with testing
 - Time-consuming
 - Too expensive
 - Misses too many cases
- How:
 - Code reviews, DbC testing, Unit testing, regression testing, sanitizer builds/testing
- When:
 - Configure-time, Compile-time, Run-time
 - Continuous, Nightly, On-demand, post-release

Discussion and Questions

- Developer stories
- What did you find useful?
- What did you find excessive?
- Failure modes

Score Board & Prizes