Pipeline Testing – on a Budget

Lars Kr. Lundin

- CPL developer
- NACO and VISIR developer
- SPHERE Contact Person

Three steps

- Static code analysis
- Unit tests
- Memory errors

Static Code Analysis:

The analysis of the source code without program execution.

Uncover programming errors and non-conforming code.

Analysis tool with no overhead:

- Use gcc with various options: -std=c99 -pedantic -Wall -Wextra
- `./configure CFLAGS='-std=c99 -pedantic -Wall -Wextra'`
- `make check TESTS=`
- Fix the compiler warnings right away!

Unit Test

Unit test:

Validate the individual units (i.e. functions) of the source code.

Benefits:

- Shows the intended usage of the tested function
- Expose errors coded in later changes
- Unit testing can be automated (e.g. nightly test runs)
- Ensures proper requirements for the function to test
- Facilitates Test Driven Development

CPL Framework for Unit Testing

Benefits:

- Standardize and reduce amount of test code
- Generates standardized, extensive reports on failure
- Secondary usage for performance evaluation

```
#include <cpl.h>
int main(void)
{
   cpl_test_init(PACKAGE_BUGREPORT, CPL_MSG_WARNING);

   /* Actual test code here */

   return cpl_test_end(0);
}
```

CPL Unit Test example

```
cpl_image * my_bias = exam_get_bias(my_default);

cpl_test_error(CPL_ERROR_NONE);

cpl_test_nonnull(my_bias);

cpl_test_leq(0.0, cpl_image_get_min(my_bias));

cpl_test_abs(expected, cpl_image_get_mean(my_bias), tolerance);

cpl_image_delete(my_bias);
```

Optional output

```
sh-3.1$ export CPL_MSG_LEVEL=info
sh-3.1$ make check
make  exam_dfs-test
[ INFO  ] User   time to test [s]: 0
[ INFO  ] System time to test [s]: 0
[ INFO  ] Number of MFLOPs in this test: 0.32768
[ INFO  ] All 4 test(s) succeeded
PASS: exam_dfs-test
```

Memory leak message

[ERROR] Memory leak detected:
#----- Memory Diagnostics -----
Maximum number of pointers: 3
#----- Memory Currently Allocated -----
Number of active pointers: 2
[ERROR] This failure may indicate a bug in the tested code
[ERROR] Please email the logfile exam_dfs-test.log to
Firstname.Lastname@consortium.org
[ERROR] System specifics:
CPL version: 4.3.0cvs
CFITSIO version is less than 3.0
WCSLIB installation is detected
This platform is not big-endian
Compile date: Sep 9 2008
Compile time: 14:27:59
__STDC__: 1
__STDC_HOSTED__: 1
__STDC_IEC_559__: 1
gcc version: 4.1.2 20070626 (Red Hat 4.1.2-13)

Example failure message

[ERROR] Failure at exam_dfs-test.c:74: |expected – cpl_image_get_mean(my_bias)|
= |1 - 1.00001| = |-1.19209e-05| <= 2.22045e-15 = tolerance
[ERROR] 1 of 4 test(s) failed
[ERROR] This failure may indicate a bug in the tested code
[ERROR] Please email the logfile exam_dfs-test.log to
Firstname.Lastname@consortium.org
[ERROR] System specifics:
CPL version: 4.3.0cvs
CFITSIO version is less than 3.0
WCSLIB installation is detected
This platform is not big-endian
Compile date: Sep 9 2008
Compile time: 14:27:59
__STDC__: 1
__STDC_HOSTED__: 1
__STDC_IEC_559__: 1
gcc version: 4.1.2 20070626 (Red Hat 4.1.2-13)

Finding Memory Errors

valgrind:

- An open source dynamic memory analysis tool.
- Finds a range of memory errors that can otherwise be difficult to debug.
- Use frequently on unit tests and full recipe execution.
- Use if memory leaks or segmentation violation occurs.
- `./configure –enable-debug` to get source code location in warnings.
- `export VALGRIND_OPTS='--trace-children=yes --leak-check=full –show-reachable=yes'`
- `make check TESTS_ENVIRONMENT=valgrind`
- Runs up to 50 times slower – do not overuse
- Can also track down bottlenecks due to poor cache usage

Exact location of memory leak

```
==2689== 262,164 bytes in 1 blocks are definitely lost in loss record 1 of 2
==2689==    at 0x401F6F2: malloc (vg_replace_malloc.c:149)
==2689==    by 0x43659D4: cx_malloc (cxmemory.c:236)
==2689==    by 0x41A7F9D: cpl_malloc (cpl_memory.c:148)
==2689==    by 0x4189C08: cpl_image_new (cpl_image_io.c:137)
==2689==    by 0x80487B1: exam_get_bias (exam_dfs-test.c:50)
==2689==    by 0x8048839: test_dfs (exam_dfs-test.c:66)
==2689==    by 0x804896C: main (exam_dfs-test.c:93)
```

Traceback of the allocation that was lost
with exact source code location

Exact location of invalid memory read (segfault)

```
==27057== Invalid read of size 4
==27057==    at 0x80489B2: test_dfs (exam_dfs-test.c:78)
==27057==    by 0x8048A04: main (exam_dfs-test.c:95)
==27057==  Address 0x450BC40 is 0 bytes inside a block of size 20 free'd
==27057==    at 0x4020289: free (vg_replace_malloc.c:233)
==27057==    by 0x4365BCD: cx_free (cxmemory.c:409)
==27057==    by 0x41A80B0: cpl_free (cpl_memory.c:251)
==27057==    by 0x418BF87: cpl_image_delete (cpl_image_io.c:1027)
==27057==    by 0x80489AE: test_dfs (exam_dfs-test.c:76)
==27057==    by 0x8048A04: main (exam_dfs-test.c:95)
```

Traceback of both execution and (stale) allocation!

Happy Coding!

Integration test:

Test the combination of individual software components as a group, i.e. a complete pipeline recipe with a front-end, i.e. esorex and gasgano.

Involves (manual) validation of the science products. Science validation is time consuming, thus done only after all automated tests are successful.

Creates reference set of validated pipeline output data for subsequent regression testing.

Regression Test

Regression test:

Uncover regression bugs, i.e. when functionality that used to work as required stops doing so.

Regression are typically introduced as an unintended consequence of a software change.

While for new projects Unit and Integration testing is the most important, Regression testing becomes essential once the development proceeds beyond the first working delivery.

Regression bugs in a recipe can happen due to changes in:

- The recipe itself, its internal support library, its calibration database
- Support libraries: CPL, QFITS, CFITSIO, FFTW, GSL, etc.
- The front-end, esorex or gasgano
- The run-time system (OS, compiler, etc).

Regression Test Continued

"Also as a consequence of the introduction of new bugs, program maintenance requires for more system testing per statement written than any other programming. Theoretically, after each fix one must run the entire batch of test cases previously run against the system, to ensure that it has not been damaged in an obscure way. In practice, such regression testing must indeed approach this theoretical limit, and it is very costly."

Source: Fred Brooks, The Mythical Man Month, 1975.

Partial automation of regression tests:

- Creation of new test cases from existing ones
- Execution of tests (during lunch, or over night/week-end)
- Detection of memory-errors (valgrind), crashes, failures, missing products
- Find differences in product-headers.
- Compute statistics on data differences.
- Filter out insignificant changes (PRO DATE keys), rounding errors.

Nightly Pipeline Builds

Nightly Pipeline Builds offer:

- Tests run every night using latest CPL release
- Unit tests
- Simple integration test with esorex
- Various static checks (splint, staticcheck)
- Memory errors in Unit and Integration tests (valgrind)
- Multi-platform tests (Linux, Solaris, MacOS, HP-UX)
- Multi-compiler tests (new and older gcc, Solaris/HP-UX cc)

- http://www.eso.org/~llundin/cpl/qc/