Testing your Code

Alicia Klinvex
Sandia National Labs
August 8, 2016
TESTING
Outline

- Why testing is important
- Types of tests
- Testing tips
- How Trilinos is tested
- Code coverage
- Verification
Why testing is important: the protein structures of Geoffrey Chang

- Some inherited code flipped two columns of data, inverting an electron-density map
- Resulted in an incorrect protein structure
- Resulted in 5 retracted publications
  - One was cited 364 times
- Many papers and grant applications conflicting with his results were rejected
Why testing is important: the 40 second flight of the Ariane 5

- Ariane 5: a European orbital launch vehicle meant to lift 20 tons into low Earth orbit
- Initial rocket went off course, started to disintegrate, then self-destructed less than a minute after launch
- Seven variables were at risk of leading to an Operand Error (due to conversion of floating point to integer)
  - Four were protected
- Investigation concluded insufficient test coverage as one of the causes for this accident
- Resulted in a loss of $370,000,000.
Why testing is important: the Therac-25 accidents

- Therac-25: a computer-controlled radiation therapy machine
- Minimal software testing
- Race condition in the code went undetected
- Unlucky patients were struck with approximately 100 times the intended dose of radiation, ~ 15,000 rads
- Error code indicated that no dose of radiation was given, so operator instructed machine to proceed
  - Documentation gave no indication that the frequent malfunctions of the machine could place a patient at risk
  - See also: why documentation is important
- Recalled after six accidents resulting in death and serious injuries
Granularity of tests

- **Unit tests**
  - Test individual functions or classes
  - Build and run fast
  - Localize errors

- **Integration tests**
  - Test interaction of larger pieces of software

- **System-level tests**
  - Test the full software system at the user interaction level
Types of tests

- **Verification tests**
  - Does the code implement the intended algorithm correctly?
  - Check for specific mathematical properties

- **Acceptance tests**
  - Assert acceptable functioning for a specific customer
  - Generally at the system-level

- **Regression (no-change) tests**
  - Compare current observable output to a gold standard
  - Must independently verify that the gold standard is correct

- **Performance tests**
  - Focus on the runtime and resource utilization
  - Nothing to do with correctness

- **Installation tests**
  - Verify that the configure-make-install is working as expected
CSE testing challenges

- Floating point issues
  - Different results
    - On different platforms
    - On different runs (due to multi-processor computation)
  - Ill-conditioning can magnify these small differences
    - Final solution may be different
    - Number of iterations may be different
  - Performing a diff is bad

- Non-unique solutions
CSE testing challenges

- Scalability testing
  - Difficult to get accurate data on a shared machine
  - Getting access to many processors on a parallel machine is expensive
    - Many supercomputing facilities discourage routine scalability testing
    - Large jobs may sit in the queue for quite some time
  - How do you scale a problem for weak scaling studies?
    - A more refined problem may not have the same condition number
Testing tips

- Ideal time to build a test suite is during development
  - Ensures that new code does not break existing functionality
- Failing tests should help you identify what part of the code needs to be fixed
- Software should be tested regularly
- Develop a consistent policy on dealing with failed tests
  - Use an issue tracking system
  - Add a regression test after the issue is fixed
- Run a regression test suite when checking in new code
- Avoid zero-diffing tests against gold standard output
  - spiff ([https://github.com/dontcallmedom/spiff](https://github.com/dontcallmedom/spiff))
How is Trilinos tested?

- Trilinos has 1500 tests between its 66 packages
- Developers are strongly advised to run a checkin test script when committing
  - Detects which packages were modified by your commits
  - Determines which packages you potentially broke
  - Configures, builds, and tests those packages
    - On success, pushes to repo
    - On failure, reports why it failed
  - Useful for ensuring your changes don’t break another package
  - May take a while, but many people run it overnight
- Automated testing on a variety of different platforms
Why do we do automated testing if everyone uses the checkin script?

- May test a different set of packages
- May test different environments
  - Do your changes work with Intel compilers as well as GNU?
  - Do your changes work on a mac?
  - Do your changes work with CUDA?
- Identifies a small set of commits that could have broken a build or test
  - Average 12 commits per day
  - Identifies the person who knows how to un-break it
- Bugs are easier to fix if caught early
Checkin test script examples

- Example 1: a harmless change to a comment
- Example 2: breaking the build
- Example 3: breaking some tests
Example 1: a harmless change

#include "AnasaziConfigDefs.hpp"
#include "AnasaziEigenSolver.hpp"
#include "AnasaziMultiVecTraits.hpp"

--- INSERT ---

/*! \file AnasaziTraceMinDavidson.hpp
 \brief Implementation of the TraceMin-Davidson eigensolver */

#ifndef ANASAZI_TRACEMIN_DAVIDSON_HPP
#define ANASAZI_TRACEMIN_DAVIDSON_HPP

--- HEADER ---
Example 1: a harmless change

```
Example 1: a harmless change

```

```
$./checkin-test-amklinv --do-all

Script: checkin-test.py
   --src-dir="/home/amklinv/TrilinosDir/github/Trilinos"
   --default-builds="MPI_DEBUG, SERIAL_RELEASE"
   --extra-repos-file=""
   --extra-repos-type=""
   --extra-repos=""
   --require-extra-repos-exist
   --enable-packages=""
   --disable-packages="FEI, Moertel, STK, Phalanx, PyTrilinos"
   --enable-all-packages="auto"
   --enable-fwd-packages
   --continue-if-no-updates
   --continue-if-no-changes-to-push
   --continue-if-no-enables
   --extra-cmake-options=""
   --test-categories="BASIC"
   --make-options="-j 48"
   --ctest-options="-E */(Anasazi_Epetra_BlockDavidson_test|Anasazi_Epetra_BlockDavidson_auxtest_MPI_4|Anasazi_Epetra_GeneralizedDavidson_solvtest_MPI_4|Anasazi_BlockDavidsonThyra_test_MPI_4|NOX_Thyra_Heg_MPI_1|Rhinos_CentralSolution_MPI_1|Rhinos_BackwardEuler_ConvergenceTest_MPI_1)"
   --ctest-timeout=60.0
   --no-show-all-tests
   --st-extra-builds=""
   --extra-builds=""
   --send-email-to='amklinv@sandia.gov'
   --skip-case-send-email
   --send-build-case-email=always
   --send-email-for-all
   --send-email-to-on-push='trilinos-checkin-tests@software.sandia.gov'
   --no-force-push
   --do-push-readiness-check
   --rebase
   --append-test-results
   --do-all

Starting time: Thu Apr 21 16:11:38 MDT 2016
```

```
Example 1: a harmless change

Note that the checkin script correctly identified what was modified.
Example 1: a harmless change

Configure, build, and test passed for MPI_DEBUG
Example 1: a harmless change

We are ready to push because all tests passed
Example 2: broken build

Missing semicolon at the end of the class. This will break the build.
Example 2: broken build

The checkin script detected that I broke the build.
Example 2: broken build

Checkin script also creates a log file with the error
Example 3: broken tests

Added a logic error to the code.
Example 3: broken tests

The checkin script detected that I broke several tests.
Example 3: broken tests

The log file tells us which tests were broken.
Trilinos automated testing

<table>
<thead>
<tr>
<th>Project</th>
<th>Error</th>
<th>Configure</th>
<th>Warning</th>
<th>Pass</th>
<th>Error</th>
<th>Build</th>
<th>Warning</th>
<th>Pass</th>
<th>Not Run</th>
<th>Fail</th>
<th>Test</th>
<th>Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trilinos</td>
<td>1</td>
<td>531</td>
<td>530</td>
<td>0</td>
<td>272</td>
<td>257</td>
<td>0</td>
<td>14</td>
<td>3976</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SubProjects**

<table>
<thead>
<tr>
<th>Project</th>
<th>Error</th>
<th>Configure</th>
<th>Warning</th>
<th>Pass</th>
<th>Error</th>
<th>Build</th>
<th>Warning</th>
<th>Pass</th>
<th>Not Run</th>
<th>Fail</th>
<th>Test</th>
<th>Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teuchos</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>12</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>227</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThreadPool</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>564</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sacado</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTOp</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kokkos</td>
<td>0</td>
<td>19</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Epetra</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>12</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>244</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zoltan</td>
<td>0</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>13</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>135</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shards</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GlobiPack</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Trilinos automated testing

<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Update</th>
<th>Configure</th>
<th>Build</th>
<th>Test</th>
<th>Build Time</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>artemis.srn.sandia.gov</td>
<td>Linux-intel-15.0.2-MPI_RELEASE_DEV_DownStream_ETI_SERIAL-OFF_OPENMP-ON_PTHREAD-OFF_CUDA-OFF_COMPLEX-OFF</td>
<td>68</td>
<td>1</td>
<td>140</td>
<td>0</td>
<td>216</td>
<td>3</td>
</tr>
<tr>
<td>lightsaber.srn.sandia.gov</td>
<td>Linux-GCC-4.7.2-RELEASE_DEV_MueLu_Matlab</td>
<td>69</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_Basker</td>
<td>69</td>
<td>0</td>
<td>227</td>
<td>0</td>
<td>117</td>
<td>0</td>
</tr>
<tr>
<td>hansel.sandia.gov</td>
<td>Linux-GCC-4.4.7-MPI_OPT_DEV_XYCE</td>
<td>121</td>
<td>0</td>
<td>70</td>
<td>0</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_KLU2</td>
<td>69</td>
<td>0</td>
<td>225</td>
<td>0</td>
<td>91</td>
<td>0</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_ExtraTypes_EL</td>
<td>69</td>
<td>0</td>
<td>227</td>
<td>0</td>
<td>117</td>
<td>0</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-serial_Debug_DEV_MueLu_ExtraTypes</td>
<td>69</td>
<td>0</td>
<td>227</td>
<td>0</td>
<td>117</td>
<td>0</td>
</tr>
<tr>
<td>enigma.sandia.gov</td>
<td>Linux-GCC-4.8.3-SERIAL_RELEASE_DEV_MueLu_Experimental</td>
<td>69</td>
<td>0</td>
<td>227</td>
<td>0</td>
<td>113</td>
<td>0</td>
</tr>
</tbody>
</table>
Several Amesos2 (direct solver) tests are broken.

Are any of its dependencies broken?
- Yes, there is a broken Epetra (basic linear algebra) test
- Maybe this broke Amesos2
Trilinos automated testing

- Which tests were broken in Amesos2?

Testing started on 2016-06-06 07:42:35

- Site Name: enigma.sandia.gov
- Build Name: Linux-GCC-4.8.3-serial_DEBUG_DEV_MueLu_ExtraTypes
- Total time: 16s 840ms
- OS Name: Linux
- OS Platform: x86_64
- OS Release: 3.10.0-229.4.2.el7.x86_64
- OS Version: #1 SMP Fri Apr 24 15:26:38 EDT 2015
- Compiler Version: unknown

3 tests failed.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Time</th>
<th>Details</th>
<th>Labels</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amesos2_Epetra_RowMatrix_Adapter_UnitTests_MPI_4</td>
<td>Failed</td>
<td>1s 860ms</td>
<td>Completed (Failed)</td>
<td>Amesos2</td>
<td>Broken</td>
</tr>
<tr>
<td>Amesos2_Epetra_MultiVector_Adapter_UnitTests_MPI_4</td>
<td>Failed</td>
<td>1s 980ms</td>
<td>Completed (Failed)</td>
<td>Amesos2</td>
<td>Broken</td>
</tr>
<tr>
<td>Amesos2_Tpetra_CrsMatrix_Adapter_UnitTests_MPI_4</td>
<td>Failed</td>
<td>1s 900ms</td>
<td>Completed (Failed)</td>
<td>Amesos2</td>
<td>Broken</td>
</tr>
</tbody>
</table>
If you may have broken something, you will get an email about it

CDash <trilinos-regression@sandia.gov> 4:05 AM (5 hours ago) ★
to anasazi-regres.

A submission to CDash for the project Trilinos has failing tests.
You have been identified as one of the authors who have checked in changes that are part of this submission or you are listed in the default contact list.

Details on the submission can be found at http://testing.sandia.gov/cdash/buildSummary.php?buildid=2469557

Project: Trilinos
SubProject: Anasazi
Site: artemis.srn.sandia.gov
Build Name: Linux-intel-15.0.2-MPI_RELEASE_DEV_DownStream_ETI_SERIAL-OFF_OPENMP-ON_PTHREAD-OFF_CUDA-OFF_COMPLEX-OFF
Build Time: 2016-06-06T03:59:42 MDT
Type: Nightly
Tests failing: 1

*Tests failing*
How do you motivate somebody to write all those tests?

- Tests protect YOU from other people breaking your work
  - If someone else’s changes break your code, they are responsible for fixing it
- You may already have some
  - Drivers for generating conference or paper results
    - Just reduce the problem size
  - User submitted bugs
    - Ask for a file that reproduces the issue
    - These make great regression tests
- Examples
  - Add a pass/fail condition and you have a test
How do I determine what other tests I need?

- **Code coverage tools**
  - Expose parts of the code that aren’t being tested
  - **gcov**
    - standard utility with the GNU compiler collection suite
    - counts the number of times each statement is executed
  - **lcov**
    - a graphical front-end for gcov

Similar tools exist for mac and windows.
How to use lcov

- Compile and link your code with --coverage flag
  - It’s a good idea to disable optimization
- Run your test suite
- Collect coverage data using lcov
- Generate html output using genhtml
A simple example

```cpp
#include<iostream>
#include "isEven.hpp"

int main()
{
    int num = 8;

    if(isEven(num))
        std::cout << num << " is an even number.\nTEST PASSED";
    else
        std::cout << num << " is an odd number.\nTEST FAILED";

    return 0;
}

bool isEven(int x)
{
    if(x%2 == 0)
        return true;
    return false;
}
```
A simple example

- Compile and link with --coverage flag
  - `g++ --coverage evenExample.cpp -o evenExample`
  - This creates a file called `evenExample.gcno`

- Run the test
  - `./evenExample`
  - This creates a file called `evenExample.gcda`

- Collect coverage data using lcov
  - `lcov --capture --directory . --output-file evenExample.info`
  - This creates `evenExample.info`

- Generate html output using genhtml
  - `genhtml evenExample.info --output-directory evenHTML`
  - This generates html files in the directory `evenHTML`
**A simple example**

**LCOV - code coverage report**

Current view: top level - /home/amklinv/IDEAS/testingTalk/examples/simpleExample  
Test: evenExample.info  
Date: 2016-05-24 14:13:07

<table>
<thead>
<tr>
<th>Filename</th>
<th>Line Coverage</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>evenExample.cpp</td>
<td>85.7 %</td>
<td>100.0 %</td>
</tr>
<tr>
<td>isEven.hpp</td>
<td>75.0 %</td>
<td>100.0 %</td>
</tr>
</tbody>
</table>

Generated by: LCOV version 1.12-4-g04a3c0e

This is the file we’re testing
A simple example

LCOV - code coverage report

Current view: top level - home/amkiln/IDEAS/testingTalk/examples/simpleExample - IsEven.hpp (source / functions)
Test: evenExample.info
Date: 2016-05-24 14:13:07

<table>
<thead>
<tr>
<th>Line data</th>
<th>Source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 : bool IsEven(int x)</td>
</tr>
<tr>
<td>2</td>
<td>: {</td>
</tr>
<tr>
<td>3</td>
<td>1 : if(x%2 == 0)</td>
</tr>
<tr>
<td>4</td>
<td>1 : return true;</td>
</tr>
<tr>
<td>5</td>
<td>:</td>
</tr>
<tr>
<td>6</td>
<td>0 : return false;</td>
</tr>
<tr>
<td>7</td>
<td>: }</td>
</tr>
</tbody>
</table>

We never tested this line of code (which activates when x is odd)
Let’s add another test

```cpp
#include <iostream>
#include "isEven.hpp"

bool isEven(int x)
{
    if(x%2 == 0)
        return true;
    return false;
}

int main()
{
    int num = 7;

    if(isEven(num))
        std::cout << num << " is an even number.\nTEST FAILED";
    else
        std::cout << num << " is an odd number.\nTEST PASSED";

    return 0;
}
```
A simple example

- Compile and link with --coverage flag
  - `g++ --coverage oddExample.cpp -o oddExample`
  - This creates a file called `oddExample.gcno`
- Run the test
  - `./oddExample`
  - This creates a file called `oddExample.gcda`
- Collect coverage data for BOTH TESTS using `lcov`
  - `lcov --capture --directory . --output-file twoExamples.info`
  - This creates `twoExamples.info`
- Generate html output using `genhtml`
  - `genhtml twoExamples.info --output-directory totalHTML`
  - This generates html files in the directory `totalHTML`
A simple example

**LCOV - code coverage report**

Current view: **top level** - /home/amklinv/IDEAS/testingTalk/examples/simpleExample

Test: twoExamples.info

Date: 2016-05-24 15:17:38

<table>
<thead>
<tr>
<th>Filename</th>
<th>Line Coverage</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>evenExample.cpp</td>
<td><strong>85.7 %</strong></td>
<td>6 / 7</td>
</tr>
<tr>
<td>isEven.hpp</td>
<td><strong>100.0 %</strong></td>
<td>4 / 4</td>
</tr>
<tr>
<td>oddExample.cpp</td>
<td><strong>85.7 %</strong></td>
<td>6 / 7</td>
</tr>
</tbody>
</table>

Generated by: *LCOV version 1.12-4-g04a3c0e*

This is the file we’re testing
A simple example

LCOV - code coverage report

Current view: top level - home/amklinv/IDEAS/testingTalk/examples/simpleExample - isEven.hpp (source / functions)
Test: twoExamples.info
Date: 2016-05-24 15:17:38

<table>
<thead>
<tr>
<th>Line data</th>
<th>Source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2: bool isEven(int x)</td>
</tr>
<tr>
<td>3</td>
<td>: {</td>
</tr>
<tr>
<td>4</td>
<td>2: if(x%2 == 0)</td>
</tr>
<tr>
<td>5</td>
<td>1: return true;</td>
</tr>
<tr>
<td>6</td>
<td>:</td>
</tr>
<tr>
<td>7</td>
<td>1: return false;</td>
</tr>
<tr>
<td>8</td>
<td>: }</td>
</tr>
</tbody>
</table>

We tested every line of this function
A real example - xSDKTrilinos

- Part of the Trilinos library, developed at SNL as part of the IDEAS project
- Contains the interfaces between Trilinos, PETSc, and hypre
- Available at [https://github.com/trilinos/xSDKTrilinos](https://github.com/trilinos/xSDKTrilinos)
- Ten automated tests are run nightly
  - Six are actually examples that were converted into tests
- Did we leave anything out?
A real example - xSDKTrilinos

- Step 1: Modify our CMake configuration file to use the --coverage flag to compile and link

```cpp
-D TPL_ENABLE_PETSC:BOOL=ON \
-D PETSC_LIBRARY_DIRS:FILEPATH="${PETSC_LIB_DIR}" \
-D PETSC_INCLUDE_DIRS:FILEPATH="${PETSC_INCLUDE_DIR}" \

-D TPL_ENABLE_ParMETIS:BOOL=ON \
-D ParMETIS_LIBRARY_DIRS:FILEPATH="${SUPERLU_LIB_DIR}" \
-D ParMETIS_INCLUDE_DIRS:FILEPATH="${SUPERLU_INCLUDE_DIR}" \

-D TPL_ENABLE_HYPRE:BOOL=ON \
-D HYPRE_LIBRARY_DIRS:FILEPATH="${HYPRE_LIB_DIR}" \
-D HYPRE_INCLUDE_DIRS:FILEPATH="${HYPRE_INCLUDE_DIR}" \

-D TPL_ENABLE_SuperLUDist:BOOL=ON \
-D SuperLUDist_LIBRARY_DIRS:FILEPATH="${SUPERLU_LIB_DIR}" \
-D SuperLUDist_INCLUDE_DIRS:FILEPATH="${SUPERLU_INCLUDE_DIR}" \

-D Trilinos_ENABLE_Amesos2:BOOL=ON \
-D Trilinos_ENABLE_xSDKTrilinos:BOOL=ON \

-D CMAKE_CXX_FLAGS:STRING="--coverage" \
-D CMAKE_C_FLAGS:STRING="--coverage" \
-D CMAKE_EXE_LINKER_FLAGS:STRING="--coverage" \
-D Trilinos_ENABLE_Fortran:BOOL=OFF \

${TRILINOS_HOME} -- INSERT --
```
A real example - xSDKTrilinos

- Build Trilinos (including xSDKTrilinos)
  - ./do-configure
  - make -j
- This will create a whole bunch of .gcno files
- This will also build the xSDKTrilinos tests because the configure file included
  - -D Trilinos_ENABLE_TESTS:BOOL=ON
  - -D Trilinos_ENABLE_EXAMPLES:BOOL=ON
  - -D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES=ON
A real example - xSDKTrilinos

- Run the tests using ctest
  - Note that this is not prohibitively slow
A real example - xSDKTrilinos

- All tests passed. Yay!
  - This also created a bunch of .gcda files

```
trilinos-build : ctest

Start 10: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
10/18 Test #10: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 1.42 sec
Start 11: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
11/18 Test #11: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 2.71 sec
Start 12: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
12/18 Test #12: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 1.47 sec
Start 13: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
13/18 Test #13: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 2.34 sec
Start 14: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
14/18 Test #14: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 1.50 sec
Start 15: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
15/18 Test #15: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 1.37 sec
Start 16: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
16/18 Test #16: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 1.42 sec
Start 17: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
17/18 Test #17: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 1.38 sec
Start 18: xSDKTrilinos_PETSc_Amesos2_example_MPI_4
18/18 Test #18: xSDKTrilinos_PETSc_Amesos2_example_MPI_4 ............... Passed 1.36 sec

100% tests passed, 0 tests failed out of 18

Label Time Summary:
  Amesos2 = 12.67 sec (8 tests)
  xSDKTrilinos = 16.39 sec (10 tests)

Total Test time (real) = 29.11 sec
```

[amklinv@s995692 trilinos-build]$
A real example - xSDKTrilinos

- Collect coverage data for the tests using lcov
  - `lcov --capture --directory . --output-file xSDKTrilinos.info`
  - This creates `xSDKTrilinos.info`
  - Lcov processes 634 gcda files in this step, so this does take a few minutes
A real example - xSDKTrilinos

- Generate html output using genhtml
  - genhtml xSDKTrilinos.info --output-directory xSDKTrilinos
  - This generates html files in the directory xSDKTrilinos
  - This step takes a few minutes too
Let’s take a look at the solver interface.
template<
class ScalarType,
class MV,
class OP>

PetscErrorCode PETScSolMgr<
ScalarType,MV,OP>::applyPrec(PCI M, Vec x, Vec Mx)
{
    using Teuchos::RCP;
    typedef PETScSolMgrHelper<
ScalarType,MV,OP> Helper;
    PetscErrorCode ierr;
    const PetscScalar * xData;
    PetscScalar * MxData;
    void * ptr;

    // Get the problem out of the context
    ierr = PCH::getContext(M,&ptr); CHKERRQ(ierr);
    LinearProblem<
ScalarType,MV,OP> * problem = (LinearProblem<
ScalarType,MV,OP>*)ptr;

    // Rip the raw data out of the PETSc vectors
    ierr = VecGetArrayRead(x, &xData); CHKERRQ(ierr);
    ierr = VecGetArray(Mx, &MxData); CHKERRQ(ierr);

    // Wrap the PETSc data in a Trilinos Vector
    RCP<MV> trilinosX, trilinosMX;
    Helper::wrapVector(const_cast<PetscScalar*>(xData), *problem->getLHS(), trilinosX);
    Helper::wrapVector(MxData, *problem->getLHS(), trilinosMX);

    // Perform the multiplication
    if(problem->isLeftPrec()) {
        problem->applyLeftPrec(*trilinosX, *trilinosMX);
    }
    else {
        problem->applyRightPrec(*trilinosX, *trilinosMX);
    }

    // Unwrap the vectors; this is necessary if we copied data in the wrap step
    Helper::unwrapVector(MxData, trilinosMX);

    // Restore the PETSc vectors
    ierr = VecRestoreArrayRead(x,&xData); CHKERRQ(ierr);
    ierr = VecRestoreArray(Mx,&MxData); CHKERRQ(ierr);

    // return 0;
}
A real example - xSDKTrilinos

// Perform the multiplication
if(problem->isLeftPrec()) {
    problem->applyLeftPrec(*trilinosX, *trilinosMX);
} else {
    problem->applyRightPrec(*trilinosX, *trilinosMX);
}

Oops. I never tested the RIGHT preconditioning branch.
Code coverage disclaimer

100% code coverage does not ensure bug-free code
VERIFICATION
Why is verification a separate topic?

- Code verification *uses* tests
- It is much more than a collection of tests
- It is the holistic process through which you ensure that
  - your implementation shows expected behavior,
  - your implementation is consistent with your model,
  - science you are trying to do with the code can be done.
Many stages and types of verification

- During initial code development
  - accuracy and stability during development of the algorithm
  - matching the algorithm to the model
  - interoperability of algorithms

- In later stages
  - Ongoing maintenance
  - while adding new major capabilities or modifying existing capabilities
  - Preparing for production

- If refactoring
  - Ensuring that behavior remains consistent and expected

- All stages have a mix of automation and human-intervention
Development Phase

- Development of tests and diagnostics goes hand-in-hand with code development
  - Non-trivial to devise good tests, but extremely important
  - A code is only as good as its tests
  - Compare against simpler analytical or semi-analytical solutions
    - They can also form a basis for unit testing
- In addition to testing for “correct” behavior, also test for stability, convergence, or other such desirable characteristics
- Many of these tests will be worth preserving for the maintenance phase
Mature Phase

- A subset of tests developed during the development phase become part of the regular testing regime
  - Focus on both code and functionality coverage
    - Code coverage by itself does not guarantee correctness if the many moving parts also do not interoperate well
    - Tweak in one part may result in correct behavior of individual parts, but not when they work together

- When new features or capabilities are added to the code -
  - Some tests will have been generated during development
  - The capability should have been verified for interoperability with existing code
  - A subset of all those tests should get included in the test-suite
    - Following the same principles of code and functionality coverage
Other Phases

- **Preparing for simulations**
  - Targeted testing of production configuration
  - Performance testing and tuning
  - More on this tomorrow...

- **Refactoring**
  - Possible expansion of test-suite
  - Numerical drift in results
  - Ramp-on planning
  - More on this tomorrow...
Selecting Tests

- Selection of tests is non-trivial
  - Tools exist for code coverage
  - Not for interoperability coverage

- One approach: use a matrix
  - Put infrastructure components in rows, science components in columns
  - List interoperability constraints, and pick tests
    - Tests for ongoing productions
    - Tests known to be sensitive to perturbations
    - Least complex tests that can cover the empty spots
    - Least complex tests that meet the missing interoperability constraints

Dubey et al, *Ongoing verification of a multiphysics community code: FLASH, Software: Practice and Experience* Vol 45(2) pp. 233-244
Example from FLASH

<table>
<thead>
<tr>
<th>Tests</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sedov</td>
<td>SV</td>
<td>A test on the same row indicates interoperability between corresponding physics</td>
</tr>
<tr>
<td>Cellular</td>
<td>CL</td>
<td>Similar logic would apply to tests on the same column for infrastructure</td>
</tr>
<tr>
<td>Poisson</td>
<td>PT</td>
<td>More goes on, but this is the primary methodology</td>
</tr>
<tr>
<td>White Dwarf</td>
<td>WD</td>
<td></td>
</tr>
</tbody>
</table>

You can pick rows and columns in many different ways
Hands on session tonight

- A small linear algebra package called Morpheus
  - Contains matrix and vector classes and functions

- We will...
  - Examine its Doxygen documentation
  - Determine the code coverage using gcov
  - Discuss how to improve its test suite