Testing and Verification

Presented to
ATPESC 2017 Participants

Alicia Klinvex
Sandia National Laboratories

Q Center, St. Charles, IL (USA)
Date 08/09/2017

Outline

- Why testing is important
- Types of testing
- Best practices
- Verification
- Evaluating project needs
- How real DOE code is tested

Benefits of testing

- Promotes high-quality software that delivers correct results and improves confidence
- Increases quality and speed of development, reducing development and maintenance costs
- Maintains portability to a variety of systems and compilers
- Helps in refactoring
  - Avoid introducing new errors when adding new features
  - Avoid reintroducing old errors

How common are bugs?

Programs do not acquire bugs as people acquire germs, by hanging around other buggy programs. Programmers must insert them.
- Harlan Mills (Code Complete, by Steven McConnell)

- Bugs per 1000 lines of code (KLOC)
- Industry average for delivered software
  - 1-25 errors
- Microsoft Applications Division
  - 10-20 defects during in-house testing
  - 0.5 in released product

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
- Retracted 5 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
- 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
  - Usually written before or during code development
    - Prevent faults from being introduced
  - Example: Can I correctly compute a dot-product?

If a unit test fails, you should know *exactly* what is broken.

Granularity of tests

- Integration tests
  - Test interaction of larger pieces of software
  - Do not build or run as fast as unit tests
  - Example: Does the preconditioner class work with the Krylov solver class?

Granularity of tests

- System-level tests
  - Test the full software system at the user interaction level
  - Example: Does my CFD code compute the correct solution?

Types of tests

- Verification tests
  - Does the code implement the intended algorithm correctly?
  - Check for specific mathematical properties
  - Example
    - Solving Ax=b where A has 5 distinct eigenvalues
    - Does my Krylov solver converge in 5 iterations?
  - Can be any granularity

Types of tests

- Acceptance tests
  - Assert acceptable functioning for a specific customer
    - Different from other types of tests, which don't involve customers
  - Generally at the system-level
  - Example: Does my linear solver achieve the correct convergence rate for a particular customer's linear system?

Types of tests

- Regression (no-change) tests
  - Compare current observable output to a gold standard
    - Gold standard frequently comes from previous version of software
  - Similar to verification tests
    - Must independently verify that the gold standard is correct
  - Example
    - My Krylov solver took 10 iterations last week; does it still take 10 iterations?
    - Does it achieve the same solution?
  - Bounded change tests are better for floating point computations

Types of tests

- Performance tests
  - Focus on the runtime and resource utilization
  - Nothing to do with correctness
    - Orthogonal to other types of tests
  - Example: It took my code 10s to solve this linear system last week; does it take longer now?

Types of tests

- Installation tests
  - Verify that the configure-make-install is working as expected
  - Example: Can I build and run a simple driver using my library after the library is installed?

Good testing practices

- Test-driven development – acceptance tests are written before the software
  - Gain clarity on code
  - Guarantees tests will exist
  - Useful when testing is viewed as unsustainable tax on resources
- Provide users a regression test suite
- Test software regularly, preferably daily

Policies on testing practices

- Must have consistent policy on dealing with failed tests
  - Issue tracking
    - How quickly does it need to be fixed?
    - Who is responsible for fixing it?
  - Add regression test afterwards (to avoid reintroducing issue later)
- Someone needs to be in charge of watching the test suite

Policies on testing practices

- When refactoring or adding new features, run a regression suite before checkin
  - Be sure to add new regression tests for the new features
- Require a code review before releasing test suite
  - Another person may spot issues you didn't
  - Incredibly cost-effective

Policies on testing practices

- Avoid regression suites consisting of system-level no-change tests
  - Tests often need to be re-baselined
    - Often done without verification of new gold-standard
  - Hard to maintain across multiple platforms
  - Loose tolerances can allow subtle defects to appear

Motivating people to write tests

- Tests protect YOU from other people from breaking your work
  - If someone else's changes break your code, they are responsible for fixing it
- Testing is cheaper and easier than debugging
- You may already have some tests lying around
  - Drivers for generating conference or paper results
  - User submitted bugs
  - Examples

Verification

- 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.

Examples: Tpetra verification

- Distributed basic linear algebra subroutines
  - Sparse matrices
  - Dense matrices
- Check for correct linear algebra
- Check for correct errors
  - Does the program throw an exception if I try to multiply things with incompatible dimensions?

Belos verification

- Krylov solvers
- Use problems with known solutions
  - Given A and Y, generate B=AY
    - Ensures B is in the range of A
  - Solve AX=B
- Some tests use Belos matrix and vector classes
- Some tests use Epetra/Tpetra classes
- Test with and without preconditioning
  - Left and right

Anasazi verification

- Eigensolvers
- Use problems with known solutions
  - Generated using Matlab's sprand
  - Problems with analytic solutions
    - Discretization of the Laplace operator
- Measure the residual of the computed eigenvectors
  - $R = AX - BX\Lambda$
- Number of iterations are compared to a gold standard

Zoltan(2) verification

- Graph partitioning
  - Some Sandia-developed code
  - Some TPL wrappers
- Gold standard solutions
  - Labor intensive
  - Gold standard changes when algorithms change
  - Upgrades to a TPL such as ParMETIS require gold standard to be updated
- Uses metrics to determine whether the solution is correct
  - Edge cuts
  - Balance criteria

SuperLU verification

- SuperLU – sparse Gaussian elimination code
- Test suite
  - Many unit and integration level tests
  - Entire suite can be run in a few minutes
  - Demonstrates validation and acceptance testing, also no-change or bounded-change testing
  - Demonstrates how to deal with floating point issues

SuperLU test suite

- Suite has two main goals
  - Tests query functions to floating-point parameters
    - Machine epsilon, underflow and overflow thresholds, etc
  - Provide coverage of all routines
    - Tests all functions of the user-callable routines

SuperLU test suite

- Many input matrices are generated
  - Different numerical and structural properties
- Uses several numerical metrics to assert accuracy of solution
  - Stable LU factorization
  - Small forward and backward errors

Example: SuperLU test suite

- Performs exhaustive testing of a large number of input parameters

```
For each  set of valid values {
  For each  set of valid values {
    ...
    For each  set of valid values {
      For each matrix type {
        Generate the input matrix A and rhs b;
        Call a user-callable routine with input values {, ,…, };
        Compute the test metrics;
        Check whether each metric is smaller than a prescribed threshold;
      }
    }
    ...
  }
}
```

- Runs over 10,000 tests in a few minutes

Why not always use the most stringent testing?

- Effort spent in devising tests and testing regime are a tax on team resources
- When the tax is too high…
  - Team cannot meet code-use objectives
- When is the tax is too low…
  - Necessary oversight not provided
  - Defects in code sneak through

Evaluating project needs

- Objectives
  - Proof of concept
  - Limited research use
  - Library
  - Production – simulations and analysis
- Team
  - Number of developers
  - Background of developers
  - Geographical spread

Evaluating project needs

- Lifecycle stages
- Lifetime
  - How long a code is expected to live
  - New code versus some legacy components
- Complexity
  - Number of modules, models, data structures, solvers
  - Degree of coupling and interoperability requirements

Commonalities

- Unit testing is always good
  - It is unlikely to be sufficient
- Verification of expected behavior
- Understanding the range of validity and applicability is always important
  - Especially for individual solvers

Consider the project scope

- Proof of concept
  - Nothing more than the common testing of previous slide
- Limited use
  - Manually run test-suite before each use may suffice
    - Coverage is still important
- Library
  - Depends on team and complexity
- Regular simulation and analysis
  - Depends on team and complexity
  - Testing coverage needs system level integrated coverage

Customizing for project needs: Team

- One to two developers – periodic manual testing and verification
- Mid-size to large team – automated test suite running regularly
- Subgroups within the team – automated test suite with tests of different granularity
  - May also need multiple suites run on their own schedules

Other factors

- Frequency of testing depends upon lifecycle stage
  - Mid-size to large team working on the same code component doing rapid development – ideally continuous integration
  - Stable mature code - regular automated testing
  - Refactoring – needs its own strategy
- Complexity and lifetime
  - Affect the testing regime being devised
  - Testing needs and strategy differ when code incorporates legacy components

Maintenance of a test suite

- Testing regime is only useful if it is
  - Maintained
  - Monitored regularly
  - Has rapid response to failure
- Maintenance includes
  - Updating tests and benchmarks
  - Adjustments to software stack
  - Archiving and retrieval of test suite output
    - Helpful in tracing change in code behavior

Maintenance of a test suite

- Monitoring individual tests manually is unreasonable and should be automated
  - Manual inspection should be limited to failing tests
  - For repository code, failure can be correlated to check-ins within a particular time-frame
    - Only certain developers need to be involved

Maintenance of a test suite

- Tests should pass most of the time
  - Easy when code changes are infrequent
  - Harder when code is large and rapidly changing
    - Difficult to determine cause of failure
    - Pre-commit test suites are a good idea

Maintenance of a test suite

- Periodically review collection of tests
  - Look for gaps and redundancies
  - Pruning is important to conserve testing resources
  - Deprecated features can be removed
  - New tests may be necessary when new features are added

Selection of tests

- Important to aim for quick diagnosis of error
  - A mix of different granularities works well
    - Unit tests for isolating component or sub-component level faults
    - Integration tests with simple to complex configuration and system level
- Some rules of thumb
  - Enable quick pin-pointing
  - Coverage

For a large code experience with test selection see Dubey et al 2015

Examples

- From Alquimia, amanzi and Trilinos
- Focus on different team sizes and objectives
- Different lifetime spans

How is real DOE code tested?

	How many developers?	How much code?	How frequent are changes?
Alquimia	< 1 FTE	O(1,000) lines of code	Every few months
Amanzi	About a dozen	O(100,000) lines of code	A few commits every day
Trilinos	A few dozen	O(1,000,000) lines of code	About 12 per day

What is Alquimia?

- Biogeochemistry API and wrapper library
- Provides a unified interface to existing geochemistry engines
  - CrunchFlow
  - PFLOTRAN
- Allows subsurface flow and transport simulators to access a range of functionality
- NOT an implementation of a biogeochemistry reaction library
- Does NOT perform geochemical calculations

How is Alquimia tested?

- Continuous integration testing using Travis CI
- Works for them because
  - Alquimia builds fast
  - Test suite runs fast
  - Commits happen in short bursts

What is Amanzi/ATS?

- Amanzi
  - A parallel flow and reactive transport simulator
  - Used to analyze multiple DOE waste disposal sites
  - Example application: modeling hydrological and biogeochemical cycling in the Colorodo River System
    - Carbon cycling is especially important because of its role in regulating atmospheric CO_2

Amanzi/ATS testing practices

- 156 tests that can be run via “ctest”
  - No continuous integration, but developers are expected to run the test suite before committing
- New physics contributions are required to come with new system-level tests
- Various granularity tests

Amanzi/ATS testing granularity

- Unit tests
  - Code is highly componentized
- Medium-grained component tests
  - Discretizations
  - Solvers
- Coarse-grained system-level tests
  - Test full capability
  - Serve as example for new users

What is Trilinos?

- A collection of libraries intended to be used as building blocks for the development of scientific applications
- Organized into 66 packages
  - Linear solvers
  - Nonlinear solvers
  - Eigensolvers
  - Preconditioners (including multigrid)
  - And more!

How is Trilinos tested?

- Trilinos has 1500 tests between its 66 packages
- Developers are strongly advised to run a checkin test script when committing
- Automated testing on a variety of different platforms

Checkin test script

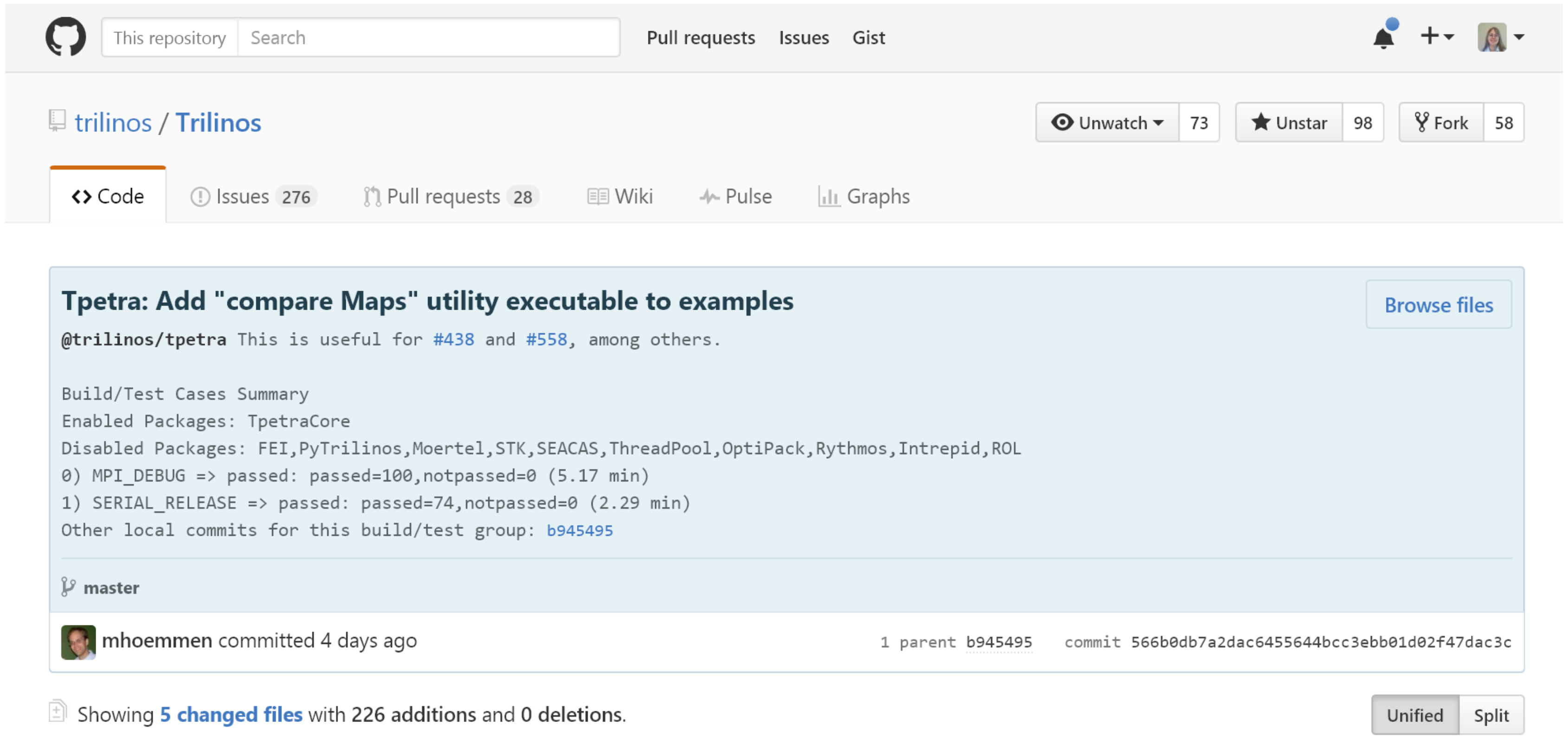
- Detects which packages were modified by your commits
- Determines which packages you potentially broke
- Configures, builds, and tests those packages
  - On success, pushes to repo
  - On failure, reports why it failed
- Useful for ensuring your changes don't break another package
- May take a while, but many people run it overnight

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
  - Identifies the person who knows how to un-break it
- Bugs are easier to fix if caught early

What if “bad people” don’t use the checkin script?

- Their commit doesn’t include the checkin script information



Tpetra: Add "compare Maps" utility executable to examples

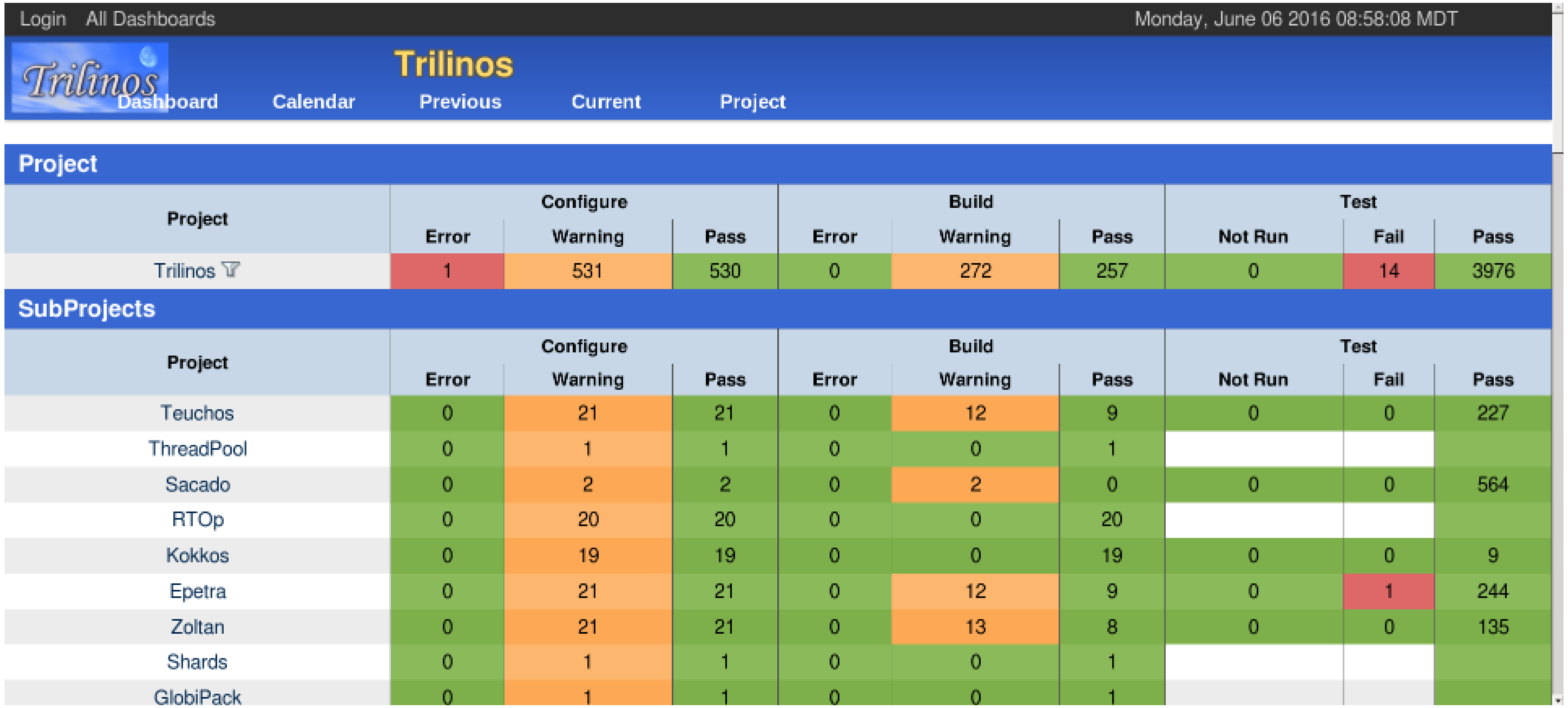
```
@trilinos/tpetra This is useful for #438 and #558, among others.

Build/Test Cases Summary
Enabled Packages: TpetraCore
Disabled Packages: FEI,PyTrilinos,Moertel,STK,SEACAS,ThreadPool,OptiPack,Rythmos,Intrepid,ROL
0) MPI_DEBUG => passed: passed=100,notpassed=0 (5.17 min)
1) SERIAL_RELEASE => passed: passed=74,notpassed=0 (2.29 min)
Other local commits for this build/test group: b945495
```

master

mhoemmen committed 4 days ago   1 parent b945495   commit 566b0db7a2dac6455644bcc3ebb01d02f47dac3c

Showing 5 changed files with 226 additions and 0 deletions.

Trilinos automated testing



Login All Dashboards Monday, June 06 2016 08:58:08 MDT

Trilinos

Dashboard Calendar Previous Current Project

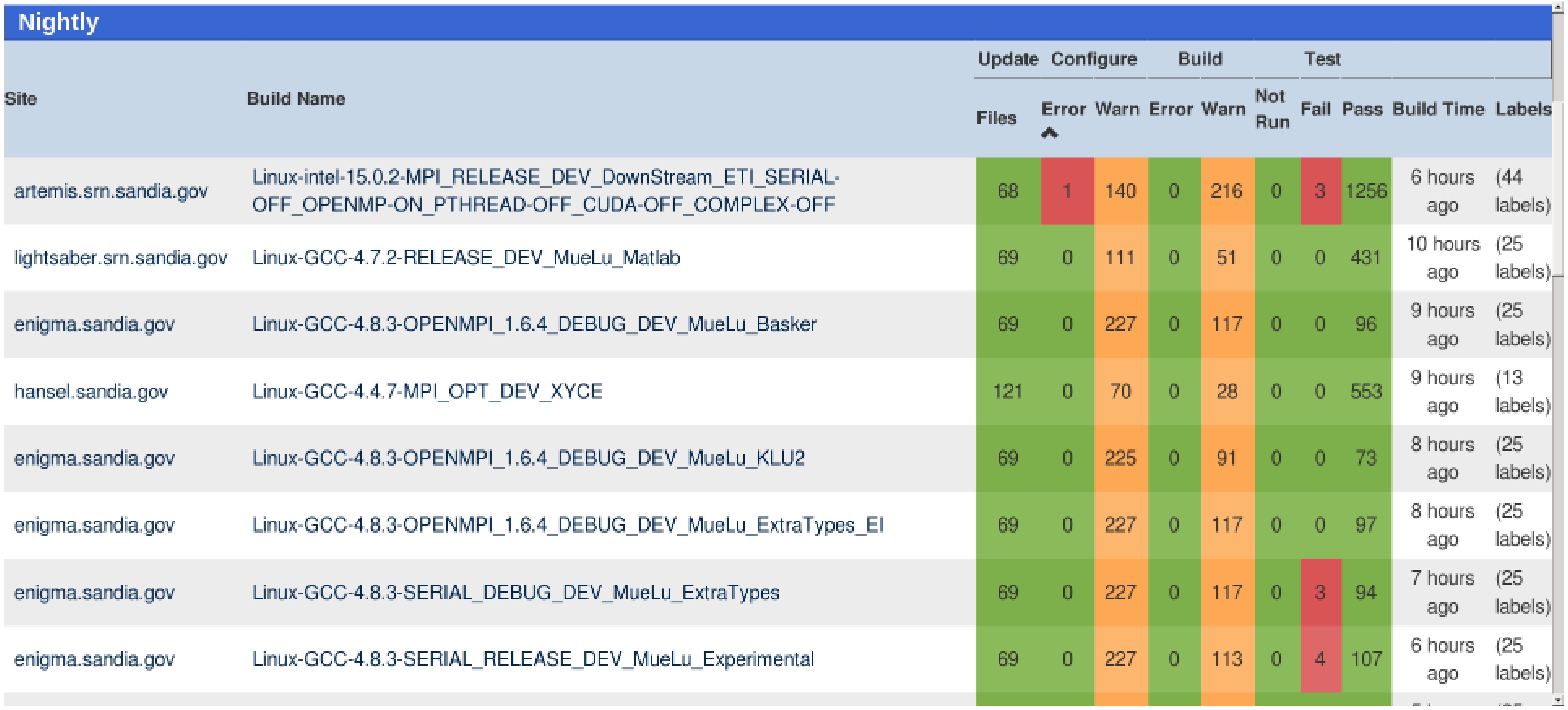
Project

Project	Configure			Build			Test		
	Error	Warning	Pass	Error	Warning	Pass	Not Run	Fail	Pass
Trilinos	1	531	530	0	272	257	0	14	3976

SubProjects

Project	Configure			Build			Test		
	Error	Warning	Pass	Error	Warning	Pass	Not Run	Fail	Pass
Teuchos	0	21	21	0	12	9	0	0	227
ThreadPool	0	1	1	0	0	1			
Sacado	0	2	2	0	2	0	0	0	564
RTOp	0	20	20	0	0	20			
Kokkos	0	19	19	0	0	19	0	0	9
Epetra	0	21	21	0	12	9	0	1	244
Zoltan	0	21	21	0	13	8	0	0	135
Shards	0	1	1	0	0	1			
GlobiPack	0	1	1	0	0	1			

Trilinos automated testing

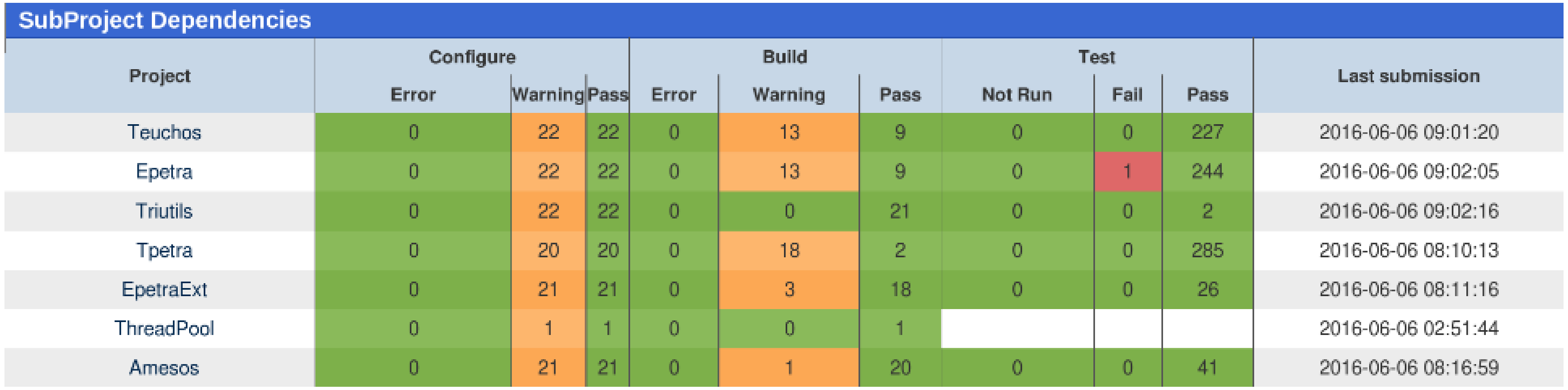


Nightly

Site	Build Name	Update	Configure		Build		Test				
		Files	Error ^	Warn	Error	Warn	Not Run	Fail	Pass	Build Time	Labels
artemis.srn.sandia.gov	Linux-intel-15.0.2-MPI_RELEASE_DEV_DownStream_ETI_SERIAL-OFF_OPENMP-ON_PTHREAD-OFF_CUDA-OFF_COMPLEX-OFF	68	1	140	0	216	0	3	1256	6 hours ago	(44 labels)
lightsaber.srn.sandia.gov	Linux-GCC-4.7.2-RELEASE_DEV_MueLu_Matlab	69	0	111	0	51	0	0	431	10 hours ago	(25 labels)
enigma.sandia.gov	Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_Basker	69	0	227	0	117	0	0	96	9 hours ago	(25 labels)
hansel.sandia.gov	Linux-GCC-4.4.7-MPI_OPT_DEV_XYCE	121	0	70	0	28	0	0	553	9 hours ago	(13 labels)
enigma.sandia.gov	Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_KLU2	69	0	225	0	91	0	0	73	8 hours ago	(25 labels)
enigma.sandia.gov	Linux-GCC-4.8.3-OPENMPI_1.6.4_DEBUG_DEV_MueLu_ExtraTypes_EI	69	0	227	0	117	0	0	97	8 hours ago	(25 labels)
enigma.sandia.gov	Linux-GCC-4.8.3-SERIAL_DEBUG_DEV_MueLu_ExtraTypes	69	0	227	0	117	0	3	94	7 hours ago	(25 labels)
enigma.sandia.gov	Linux-GCC-4.8.3-SERIAL_RELEASE_DEV_MueLu_Experimental	69	0	227	0	113	0	4	107	6 hours ago	(25 labels)

Trilinos automated testing

- Several Amesos2 (direct solver) tests are broken



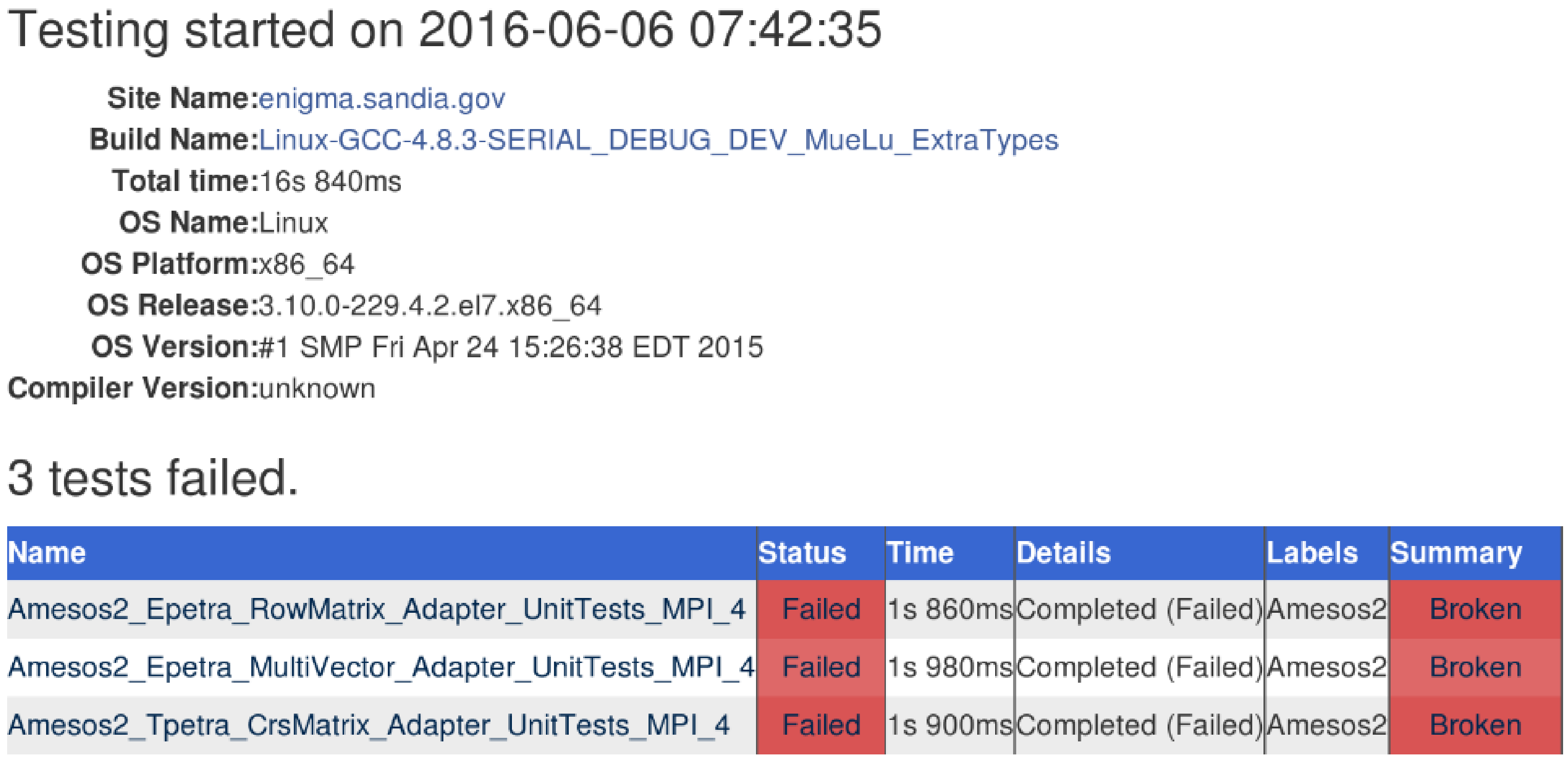
SubProject Dependencies

Project	Configure			Build			Test			Last submission
	Error	Warning	Pass	Error	Warning	Pass	Not Run	Fail	Pass	
Teuchos	0	22	22	0	13	9	0	0	227	2016-06-06 09:01:20
Epetra	0	22	22	0	13	9	0	1	244	2016-06-06 09:02:05
Triutils	0	22	22	0	0	21	0	0	2	2016-06-06 09:02:16
Tpetra	0	20	20	0	18	2	0	0	285	2016-06-06 08:10:13
EpetraExt	0	21	21	0	3	18	0	0	26	2016-06-06 08:11:16
ThreadPool	0	1	1	0	0	1				2016-06-06 02:51:44
Amesos	0	21	21	0	1	20	0	0	41	2016-06-06 08:16:59

- 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

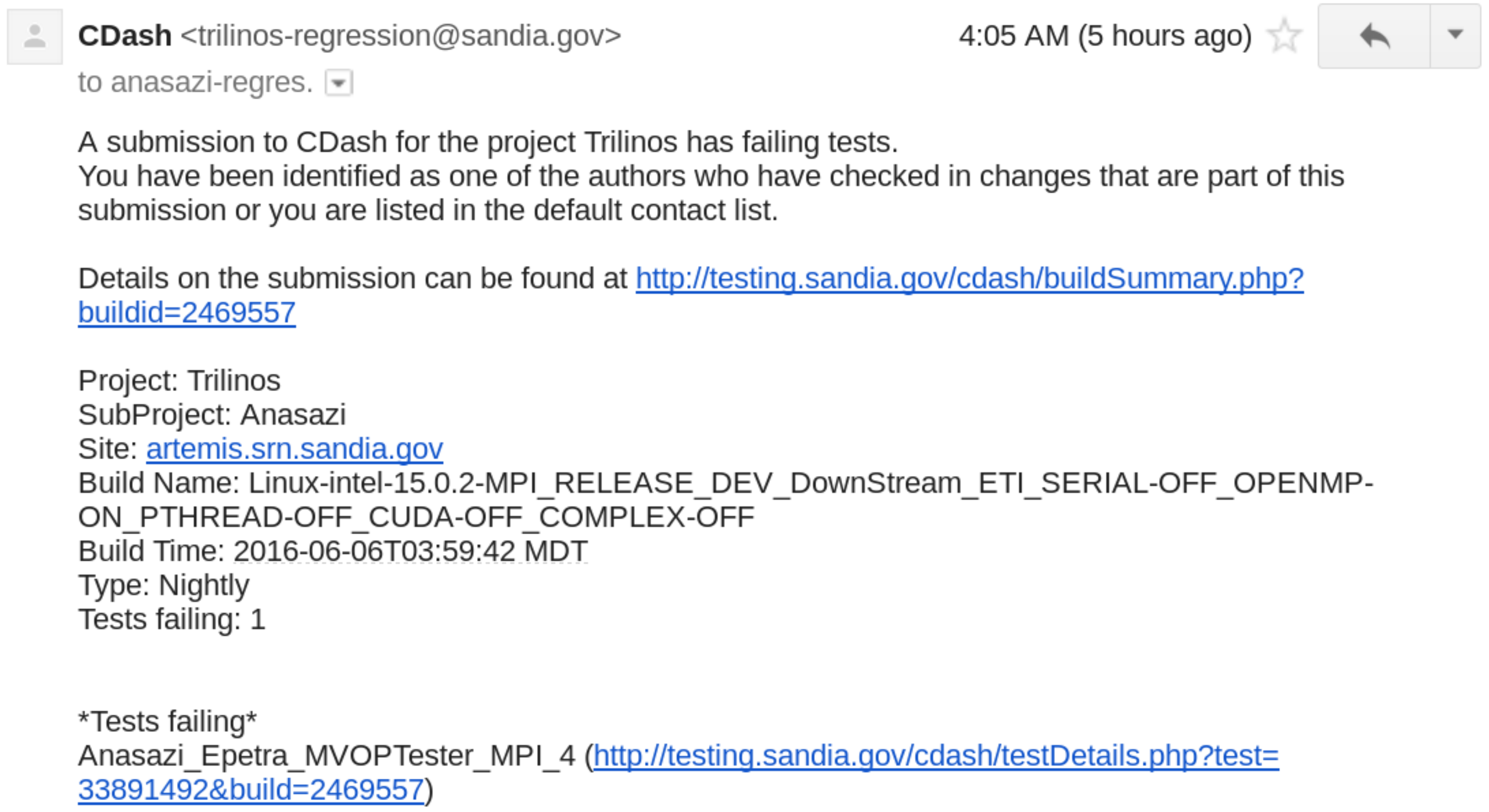
Site Name:	enigma.sandia.gov
Build Name:	Linux-GCC-4.8.3-SERIAL_DEBUG_DEV_MueLu_ExtraTypes
Total time:	16s 840ms
OS Name:	Linux
OS Platform:	x86_64
OS Release:	3.10.0-229.4.2.el7.x86_64
OS Version:	#1 SMP Fri Apr 24 15:26:38 EDT 2015
Compiler Version:	unknown

3 tests failed.

Name	Status	Time	Details	Labels	Summary
Amesos2_Epetra_RowMatrix_Adapter_UnitTests_MPI_4	Failed	1s 860ms	Completed (Failed)	Amesos2	Broken
Amesos2_Epetra_MultiVector_Adapter_UnitTests_MPI_4	Failed	1s 980ms	Completed (Failed)	Amesos2	Broken
Amesos2_Tpetra_CrsMatrix_Adapter_UnitTests_MPI_4	Failed	1s 900ms	Completed (Failed)	Amesos2	Broken

Trilinos automated testing

- 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
Anasazi_Epetra_MVOPTester_MPI_4 (http://testing.sandia.gov/cdash/testDetails.php?test=33891492&build=2469557)

New master/develop workflow

- Want master branch to remain stable
- All developer changes are now pushed to develop branch
- If changes are “okay”, merge develop to master
  - Currently a manual process for Trilinos framework team
  - If no new tests are failing on the dashboard, merge
  - Will eventually be automated

An important consideration: commits are so frequent, and the test suite is so large, it is impractical to run the test suite after each commit.

Summary

- Software testing is very important
- There are different types of tests
- Different projects will have different needs

- We will resume our talk of software testing after the break
  - Code coverage
  - Continuous integration testing

Thank you for your attention!