Development and Integration Workflows for Large Complex Distributed CSE Software Efforts

Roscoe A. Bartlett, Ph.D.
Dept. Software Engineering & Research
https://bartlettroscoe.github.io

SAND2019-2092 C
Overview of CASL

- CASL: Consortium for the Advanced Simulation of Lightwater reactors
- DOE Innovation Hub including DOE labs, universities, and industry partners
- Goals:
  - Advance modeling and simulation of lightwater nuclear reactors
  - Produce a set of simulation tools to model lightwater nuclear reactor cores to provide to the nuclear industry: **VERA: Virtual Environment for Reactor Applications**.
- Phase 1: July 2010 – June 2015
- Phase 2: July 2015 – June 2020
- Organization and management:
  - ORNL is the hub of the Hub
  - Milestone driven (6 month plan-of-records (PoRs))
  - Focus areas: **Physics Integration (PHI)**, Thermal Hydraulic Methods (THM), Radiation Transport Methods (RTM), Advanced Modeling Applications (AMA), Materials Performance and Optimization (MPO), Validation and Uncertainty Quantification (VUQ)

Roscoe Bartlett was PHI software engineering and integration lead for CASL from 2010-2016.
VERA development is complicated!

VERA Currently Composed of:
- 21 different repositories on casl-dev.ornl.gov (some git clones of other repos) most with a different access list (NDAs, Export Control, IP, etc.)
- Integrating development efforts from many teams from 9+ institutions

Large single CMake build system using TriBITS CMake Framework:
- Very large full source code base:
  - 55K source and script files
  - 12M lines of code (not comments)
  - 2,700 CMakeLists.txt files
- 229 packages + subpackages enabled (out of 496 total) ≈ 46% of full code base
- Most CMake developer reconfigures take place in less than 30 seconds!

VERA Software Development Process:
- VERA integration maintained by continuous and nightly testing:
  - Pre-push CI testing: checkin-test-vera.sh, cloned VERA git repos
  - Post-push CI testing: CTest/CDash, all VERA git repos
  - Nightly testing: MPI and Serial builds, Debug and Release builds, ...
  - Maintain 100% passing builds and tests most days!
- Many internal and external repository integrations on daily basis
- VERA releases are taken off of stable ‘master’ branches on casl-dev git repos.
- Low maintenance cost of the infrastructure
- Primary/originating institution shown in Blue
- Most codes being contributed by multiple institutions
- All direct dependencies are NOT shown
- Dependencies between repos are though TriBITS package dependencies
- CAS maintains compatible git repo forks of all these repos
Overview of SNL ATDM (2019)

Advanced Technology Development & Mitigation (ATDM) Project
• Started in FY14 under DOE Advanced Simulation and Computing (ASC) Program
• Consumed into the larger DOE Exascale Computing Project (ECP) in FY16
• Background/Motivation:
  • Exascale computers coming in 2023 using new programming models and hardware that current generation of ASC CSE codes will not run.
  • Rapidly developing/changing pre-exascale hardware and system software
• Mission of ATDM:
  • Design next-generation exascale CSE codes unconstrained by software.
  • Leverage components and advanced algorithms for sensitivity analysis, design optimization, calibration, inversion, UQ/QMU, etc.

Sandia National Labs (SNL) ATDM Project:
• Two Primary SNL ATDM Codes:
  • EMPIRE - ElectroMagnetic Plasma In Radiation Environments
  • SPARC - Sandia Parallel Aerodynamics and Reentry Code
• Leveraging & co-developing 2nd Trilinos packages built on Kokkos abstraction layer for on-node and on-GPU performance, Tpetra for node/GPU aware distributed memory linear algebra data-structures, and solvers built on these.
**Development & Integration Overview**

- Core functionality provided by SNL 2\textsuperscript{nd} generation Trilinos (Kokkos, Tpetra, etc.)
- SNL ATDM APP requirements drive Trilinos development.
- Each SNL ATDM APP maintains its own fork of Trilinos that is updated periodically.

**Challenges**

- Very long and expensive builds for templated Kokkos-based C++11 code.
- Limited computer testing resources.
- APPs needing frequent updates of Trilinos without getting new defects.
- Keeping Trilinos & APPs working on changing ATDM/ECP platforms and environments.
- Defects in system software (e.g. compilers, MPI) slipping through system testing and instead being detected in Trilinos and APPs.
- Pushing for higher production-quality software from Ph.D. researchers.
Challenges in the Early Years of SNL ATDM (pre 2018)

• Trilinos Stability Problems:
  • No testing requirement before Trilinos developers pushed changes to the main ‘develop’ branch.
  • Many nightly builds submitting to CDash dashboard had many failing builds and failing tests that persisted for long periods of time.
    => Made it hard to see new defects
  • Little-to-no automated testing of Trilinos suite on ATDM pre-exascale platforms.

• ATDM APP developers and other staff members directly pulled from the main Trilinos ‘develop’ branch:
  • APP developers and other staff members often experienced broken builds.
  • Some important builds (e.g. CUDA on GPUs) often broken for significant lengths of time.

• Impact:
  ➢ Lower ATDM APP developer productivity.
  ➢ Lower confidence in Trilinos.
  ➢ Avoidance depending on more Trilinos packages that absolutely required.
Common challenges in CASL VERA and SNL ATDM:
• Balancing speed of integration vs. stability of updates
• Coordination of different development teams
• Keeping build and testing infrastructure working both in external repos/projects and internal to the project

Different primary challenges in CASL VERA vs. SNL ATDM:
• CASL VERA:
  • Coordination of different development teams for multiple institutions.
  • Maintaining integrated build, test, and deployment from many different external projects.
• SNL ATDM:
  • Productive development and integration on many unstable buggy changing pre-exascale platforms.
  • Maintaining portability on wide range of ATDM/ECP platforms
  • Fast integration.
Multi-Team Multi-Repository Testing & Integration Basics
What Not to Do

Why is this so bad?

- Lack of test coverage in the external repo’s native test suite to cover project’s needs.
- External repo developers not testing against the project’s code and tests.
- External repo may be broken w.r.t. to the project for long period of time.
- Project developers frequently pull code that does not even configure or build.
- Broken code frequently interrupting the work of project developers.
Managing Internal and External Development & Integration

Project must keep consistent clones of every external repo and carefully sync updates!

**Issues that need to be addressed:**

- Flexibility for development inside and outside of the project.
- Providing a flow of frequent stable updates of the software.
- Maintaining the stability of the software to keep project developers productive.
- Making non-backward compatible changes across many repos.
- Full tracking of changes and updates.
Basic Parts to Development & Integration Process

• **Git Workflows:**
  • How git repositories and branches are set up, how merges occur, what git commands are run, etc.
  • Different git workflows used for external repo developers, Project developers, and repo/project co-developers.

• **Testing gates for workflows:**
  • Gating test suites can/should be run before each “merge” in the workflow.
  • Gating tests can be run manually or automated, daily or “every-so-often”.
  • Important test suites:
    - **RepoX build & tests:** Gates updating the main RepoX development branch.
    - **Project builds & tests:** Gates all updates of the project’s repos.

• **Detection, triage and fixing of new failing builds and tests:**
  • Detection and notification of new failures.
  • Triage failures.
  • Address failures.
  • Manage & follow-up.
Single External Repo
Project Integration
Trilinos => ATDM APP
2) ATDM Trilinos Nightly Builds & Tests (CDash)

- Build and run native Trilinos test suite on all the ATDM platforms.
- First step in providing stable portability on many pre-exascale platforms.
- Builds are too expensive to run more than one set per 24-hour day.
- Frequent random system failures make detection of new code-related failures difficult.
Initial creation of APP fork of Trilinos

Must pass gating:
1) Auto PR Trilinos builds & tests
2) ATDM Trilinos builds & tests
3) APP nightly builds & tests

Legend for Git Workflow Diagrams

- direct commit on <main-branch>
- link to ancestor commit
- (explicit) merge commit
- commits on branch
- link to merge ancestor
- branch references
- Person creating commit
- Unspecified git graph/history

Time

<main-branch>

<topic-branch>
Injecting New Failures and Fixing Failures: A Race!

- **Mean-time to fail**: Average time (in days) for when a new failure shows up in ‘develop’ branch in one or more promoted ATDM Trilinos builds.
- **Mean-time to fix**: Average time (in days) to discover, triage and fix a failure on the Trilinos ‘develop’ branch in the promoted ATDM Trilinos builds.
- **The core problem**: If “mean-time to fail” is less than “mean-time to fix”, then the ATDM Trilinos builds on ‘develop’ on average will **ALWAYS be broken** (and therefore block updates of Trilinos to the APP customers)!

\[
\text{Mean-time to fix} < \text{Mean-time to fail}
\]

\[
\text{Mean-time to fix} > \text{Mean-time to fail}
\]
Reducing Time to Detect, Triage, and Address New Failures

ATDM Trilinos Builds & Tests
General SE Principles for Defects

- Cost of a defect goes up (significantly) the longer it takes to detect and correct a defect.

- Lean/Agile SE Practices for dealing with defects:
  - Strong automated testing (have tests help new detect defects)
  - Continuous testing (reduce the time to detect new defects caught by tests)
  - Continuous integration (reduce time to detect conflict defects)
  - STOP THE LINE when a new defect gets into the main development branch
    - Fixing defects in previously working software is higher priority than developing new features!
Detecting New Failures/Missing Results: CDash Email

**FAILED** (bm=1, twoif=2, twip=1, twif=2): Promoted ATDM Trilinos Builds on 2019-01-04

- **Builds on CDash** (num/expected=33/33)
- **Non-passing Tests on CDash** (num=4)

**Builds Missing: bm=1**
- Tests without issue trackers Failed: twoif=2
- Tests with issue trackers Passed: twip=1
- Tests with issue trackers Failed: twif=2

**Builds Missing: bm=1**

<table>
<thead>
<tr>
<th>Group</th>
<th>Site</th>
<th>Build Name</th>
<th>Missing Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATDM</td>
<td>waterman</td>
<td>Trilinos-atdm-waterman-cuda-9.2-release-debug</td>
<td>Build exists but no test results</td>
</tr>
</tbody>
</table>

**Tests without issue trackers Failed** (limited to 20): twoif=2

<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Test Name</th>
<th>Status</th>
<th>Details</th>
<th>Consecutive Non-pass Days</th>
<th>Non-pass Last 30 Days</th>
<th>Pass Last 30 Days</th>
<th>Issue Tracker</th>
</tr>
</thead>
<tbody>
<tr>
<td>sens-rhel6</td>
<td>Trilinos-atdm-sens-rhel6-intel-opt-openmp</td>
<td>Belos_BlockGmresPoly_Epetra :File_Ext_0_MPI_4</td>
<td>Failed</td>
<td>Completed</td>
<td>1</td>
<td>1</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

**Tests with issue trackers Failed: twif=2**

<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Test Name</th>
<th>Status</th>
<th>Details</th>
<th>Consecutive Non-pass Days</th>
<th>Non-pass Last 30 Days</th>
<th>Pass Last 30 Days</th>
<th>Issue Tracker</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutrino</td>
<td>Trilinos-atdm-mutrino-intel-opt-openmp-HSW</td>
<td>Anasazi_Epetra_BKS_norestart :test_MPI_4</td>
<td>Failed</td>
<td>Completed</td>
<td>21</td>
<td>21</td>
<td>3</td>
<td>#3499</td>
</tr>
</tbody>
</table>
Reproducing ATDM Trilinos Builds: GitHub Issue

Steps to Reproduce

One should be able to reproduce this failure on waterman as described in:

  More specifically, the commands given for waterman are provided at:

  The exact commands to reproduce this issue should be:

```
$ cd <some_build_dir>/

$ source $TRILINOS_DIR/cmake/std/atdm/load-env.sh cuda-9.2-release-debug

$ cmake \
  -GNinja \
  -DTrilinos_CONFIGURE_OPTIONS_FILE:STRING=cmake/std/atdm/ATDMDevEnv.cmake \
  -DTrilinos_ENABLE_TESTS=ON -DTrilinos_ENABLE_Intrepid2=ON \
  $TRILINOS_DIR

$ make NP=20

$ bsub -x -Is -n 20 ctest -j20
```
How to Address Failures?

Already cleaned-up promoted builds clean:
   a) Fix the failures \=> **Best option!**
   b) Mark failing tests as “allow to fail” and not trigger global FAIL:
      - Only for non-blocking issues
      - Allows us to watch test run but not block updates of Trilinos to APPs
      - Best for when someone is working to fix non-blocking failures.
   c) (Temporarily) disable failing tests:
      - Only for non-blocking issues
      - Best for cases where no-one is going to work on fixing the failures soon.
   d) Revert the commit(s) (or PR merge) causing the failure:
      - Best for critical/blocking failures that can’t be fixed ASAP.

Initial failures setting up new platforms:
   a) Fix the failures
   b) (Temporarily) disable failing tests
   c) Mark failing tests as “allow to fail” and not trigger global FAIL
      - NOTE: Reverting commits is NOT an option for cleaning up failures that occur when setting up new builds on new platforms or envs on existing platforms.
Conclusions and Lesions Learned CASL & SNL ATDM

- Projects must set up their own forks of external repos that must be frequently updated and define integration testing workflows.

- Detecting, Traiging, and Addressing New Failures:
  - Running tests using similar configurations on different systems and compilers helps to speed up detection of new software defects.
  - Effective detection and triaging requires an analysis tool that takes a broad view of build and tests results to show trends, commonality, and history.
  - Likely 90-95% of failing (Trilinos) tests don’t indicate a problem impacting a specific customer but they hide the 5-10% that do.
  - Must carefully scrutinize every failing test to detect new defects.
  - Must not allow existing failures to hide new failures!

- Build and Test Systems:
  - Heterogenous build and test systems significantly increase development and maintenance costs and slow/delay integrations.
  - Homogenous build and test systems across teams and software reduce development and maintenance costs and speeds integrations. (i.e. CASL)

One of biggest impediments to improving development and integration workflows is developer inability/unwillingness to learn git!