Almost Continuous Integration for the Co-Development of Highly Integrated Applications and Third Party Libraries

Roscoe A. Bartlett
Department of Optimization & Uncertainty Estimation
Trilinos Software Engineering Technologies and Integration Lead

Sandia National Laboratories

Sandia Software Engineering Seminar Series
January 14, 2009
## Applications (APPs) and Third-Party Libraries (TPL) at SNL

<table>
<thead>
<tr>
<th>Applications (APPs)</th>
<th>Third Partly Libraries (TPL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aleph</td>
<td>Trilinos</td>
</tr>
<tr>
<td>Xyce</td>
<td>Trilinos</td>
</tr>
<tr>
<td>Titan (VTK)</td>
<td>Trilinos</td>
</tr>
<tr>
<td>Charon*</td>
<td>Trilinos*, Xyce, Nevada</td>
</tr>
<tr>
<td>Alegra*</td>
<td>Trilinos*, Xyce, Nevada</td>
</tr>
<tr>
<td>SIERRA*</td>
<td>Trilinos*</td>
</tr>
</tbody>
</table>

* Some experimentation with more frequent APP + TPL Integration

- Tighter level of APP + TPL integration is needed in many cases
- Co-development of APP + TPL(s) is often needed to drive new efforts
- Current software engineering infrastructure and practices are insufficient to support desired goals
- We need new software engineering infrastructure to support these integration efforts
Lean/Agile Software Engineering Principles

• High quality software is developed in small increments and with sufficient testing in between sets of changes.

• High quality defect-free software is most effectively developed by not putting defects into the software in the first place (i.e. TDD, code reviews, pair programming, etc.).

• Software should be delivered to real (or as real as we can make them) customers is short intervals.

• Ruthlessly remove duplication in all areas.

• Avoid points of synchronization. Allow people to work as independently as possible and have the system set up to automatically support this.

• Most mistakes that people make are due to a faulty process/system (W. Edwards Deming).

• Automation is needed to avoid mistakes and improve software quality.

Common Approach
NOT AGILE!

Problems
• Cost of fixing defects increases the longer they exist in the code
• Difficult to sustain development productivity
• Broken code begets broken code (i.e. broken window phenomenon)
• Long time between branch and release
  – Difficult to merge changes back into main development branch
  – Temptation to add “features” to the release branch before a release
• High risk of creating a regression
Lean/Agile Methods: Development Stability

The Agile way!

Advantages
- Defects are kept out of the code in the first place
- Code is kept in a near releasable state at all times
- Shorten time needed to put out a release
- Allow for more frequent releases
- Reduce risk of creating regressions
- Decrease overall development cost
APP Only Upgrades After Each Major Release of TPL

- Transition from TPL X to TPL X+1 can be difficult and open ended
- Large batches of changes between integrations
- Greater risk of experiencing real regressions
- Upgrades may need to be completely abandoned in extreme cases
- However, this is satisfactory for many APP+TPL efforts!
• APP (SIERRA) Dev Developers only build/test against TPL Release
• TPL (Trilinos) Dev Developers work independent from APP
• Changes between TPL Release and TPL Dev handed through a) Refactoring, b) minimal ifdefs (NO BRANCHES)! => Backward Compatibility!
• Use of staggered releases of TPL and APP
• APP + TPL Dev Developers drive new capabilities
• Difficult for APP to depend too much on TPL
• Does not support tighter levels of integration
• However, this is satisfactory for many APP+TPL efforts!
APP Dev Builds Against Both TPL Release and TPL Dev

- All changes are tested in small batches
- Low probability of experiencing a regression
- Extra computing resources to test against 2 (3) versions of TPL
- Some difficulty flagging regressions of APP + TPL Dev
- APP developers often break APP + TPL Dev
- Difficult for APP to rely on TPL too much
- Hard to verify TPL for APP before APP release
- However, this is satisfactory for many APP+TPL efforts!
APP + TPL Integration: Different Collaboration Models

- APP Dev only upgraded after each major release of TPL
  - Little to no testing of APP + TPL Dev in between TPL releases
- APP Dev builds against both TPL Release and TPL Dev
  - APP developers work against TPL Release
  - APP + TPL team(s) build against TPL Dev
  - Daily integration testing done for both APP + TPL Release and Dev
  - Staggered releases of TPL and APP
- APP Dev developed only against TPL Dev (with “Almost” Continuous Integration)
  - Regular APP developers work independently using very recent APP-owned VC copy of TPL Dev-
  - Regular TPL developers work independently
  - APP Dev + TPL Dev developers
    - Check-out and modify APP Dev
    - Check-out and modify TPL Dev
    - Run both APP and TPL pre-checkin test suites
    - Check into both APP-owned and main TPL VC repositories
  - Nightly testing of APP Dev + TPL Dev automatically updates APP-owned TPL Dev- VC Repository
  - Releases best handled as combined releases of APP and TPL
General Development & Testing Principles

- Regular TPL developers only build and run TPL pre-checkin test suite.

- Regular APP developers should only check out code that has already built and passed the pre-checkin APP test suite.

- Nightly APP regression (and other) tests should only be run on code that has already been shown to build and pass the pre-checkin APP test suite.

- Code that builds and passes the pre-checkin test suite is safe to check in.
Basic Setup for APP + TPL Almost Continuous Integration

- **APP Dev**
  - Nightly Testing
  - APP Dev + TPL Dev

- **TPL Dev**
  - Nightly Testing

**APP Owned**
- Main APP VC Repository (Dev)
- APP-owned TPL VC Repository (Dev-)
- APP Pre-Checkin Test Suite
- APP Regression Test Suite

**TPL Owned**
- Main TPL VC Repository (Dev)
- TPL Pre-Checkin Test Suite
- TPL Regression Test Suite
Standard APP Development Process

1.a) Check out

Main APP VC Repository (Dev) → 5.a) Check in

APP Local Working Directory (Dev) → 2.a) Modify & extend

1.b) Check out

APP-owned TPL VC Repository (Dev-) → 5.c) Check in?

APP Pre-Checkin Test Suite → 5.b) Check in

• TPL (Dev-) code is typically not modified by average APP developers!
• However, small changes can be made and can be good!
APP Dev + TPL Dev Development Process

1.a) Check out

1.b) Check out

1.c) Check out

1.d) Check out (and merge)

5.a) Check in

5.b) Check in

5.c) Check in

5.d) Check in

5.e) Check in

2.a) Modify & extend

2.b) Modify & extend

2.c) Modify & extend

2.d) Modify & extend

3) Build

4.a) Run test suite

4.b) Run test suite

Main APP VC Repository (Dev)

APP-owneTPL VC Repository (Dev)

APP Pre-Checkin Test Suite

APP Local Working Directory (Dev)

TPL Local Working Directory (Dev- and Dev)

TPL Pre-Checkin Test Suite Working Directory

Main TPL VC Repository (Dev)

TPL Pre-Checkin Test Suite

• Pre-checkin test suites for APP and TPL are both run before checkin
• Simultaneous checks into APP-owned TPL Dev- and Main TPL Dev VC Repositories!
  – Changes in APP-owned TPL VC Dev- Repos get back into Main TPL VC Dev Repos!
4) [passed] Check in

- Only runs pre-checkin test suite and then only on the primary development platform! (just like a regular APP developer)
- TPL Dev- VC Repository is automatically updated by nightly testing process if a) merge, b) build, and c) pre-checkin test suite all pass!
  - This is the same criteria we have for any regular APP developer checkin!
- Integration build is checked throughout the day with continuous integration (but without the auto-updates of TPL Dev- VC repository to avoid conflicts)
APP + TPL Development and Testing Details and Policies

• Nightly Testing:
  – Nightly APP Dev + TPL Dev testing and checking in only run on primary development platform and only runs pre-checkin test suite
    => Minimizes extra testing computer resources!
  – Nightly APP regression (and other stronger) tests are only run on APP Dev + TPL Dev- and *not* with TPL Dev (but on the same day after upgrade of APP Dev-)
    • Only one version of Dev code goes through extended testing (e.g. porting, regression, performance, scalability).
    • If APP Dev + TPL Dev testing and updating of TPL Dev- succeeds, then extended testing will involve all changes to APP Dev and TPL Dev in the last 24 hours.

• Continuous Integration Testing:
  – Build and test APP Dev + TPL Dev throughout the day to flag problems and to help support co-development of APP Dev + TPL Dev

• Open Questions:
  – How are multiple TPL handled in nightly testing ?
    • Are all TPL updated at the same time in nightly testing process?
    • Are TPL updated and testing separately in a chain (TPL 1 followed by TPL 2, etc.)?
  – What about intra-TPL dependencies (i.e. Nevada and Xyce => Trilinos)?
    • Do all TPL need to follow this process as well?
**APP + TPL Almost Continuous Integration and Releases**

**The Future of APP + TPL Integration?**

- All changes are tested in small batches
- Low probability of experiencing a regression between major releases
- Less computing resources for detailed nightly testing (only one TPL version)
- All tested regressions are flagged in less than 24 hours
- Allows code to flow freely between the APP and TPL
- Supports rapid development of new capabilities from top to bottom
- All code checked out by APP Dev developers has passed pre-checkin build/test
- More complex processes (i.e. requires some tools?)
- APP Dev developers spend more time spent recompiling TPL code
- Recommended for projects requiring high levels of integration & collaboration!
Challenges with APP-Specific TPL Releases

Multiple releases of TPL (Trilinos) presents a possible problem with complex APPs

Solution:
=> Provide perfect backward compatibility of Trilinos (TPL) X through Trilinos SIERRA Y+1
Assorted Ideas for APP Dev + TPL Dev Nightly Testing

• Nightly and continuous updating, testing, and checkin algorithm
  – Check out APP Dev and + TPL Dev- from APP-owned TPL Dev- VC Repository(s)
  – Build and run pre-checkin APP test suite (for APP Dev + TPL Dev-)
  – For each TPL (i = 0 ... N-1) [ In order of increasing dependencies ]
    • Perform update of TPL i Dev from main TPL i VC Dev repository
    • Build and run pre-checkin APP test suite
    • If all passed, check into APP-owned TPL i Dev- VC repository [Nightly only]
    • Otherwise, skip checkin into APP-owned TPL i Dev- VC repository

• Advantages
  – Failures with one TPL do not automatically bring down integration with all TPL
    • Example: If Trilinos Dev works with Charon but Xyce Dev does not, at least Trilinos Dev
      would get updated and used by Charon Dev.
  – Provides additional information on where regressions are coming from
    • Example: A test passes with APP Dev + TPL Dev- but fails with APP Dev + TPL Dev
Maintenance of APP + TPL Integration

- **APP + TPL Monitor:**
  - Member of the APP development team
  - Has good familiarity with the TPLs
  - Performs first-round triage (APP or TPL?)
  - Forwards issues to APP or TPL Reps
  - Ultimate responsibility to make sure issues are resolved

- **APP Representative:**
  - Member of the APP development team
  - Second-round triage of APP issues
  - Forwards hard APP issues to APP developers

- **TPL Representative:**
  - Member of the TPL development team
  - Has some familiarity with the APPs
  - Second-round triage for TPL issues
  - Forwards hard TPL issues to TPL developers

- **General principles:**
  - Roles of authority and accountability (Ordained by management)
  - At least two people serve in each role
  - Rotate people in roles
Summary #1

• Nightly building and testing of the development versions of the application and TPLs:
  – results in better production capabilities and better research,
  – brings TPL developers and APP developers closer together allowing for a better exchange of ideas and concerns,
  – refocuses TPL developers on customer efforts,
  – helps drive continued research-quality TPL development, and
  – reduces barriers for new TPL algorithms to have impact on production applications.

• APP Dev developed only against TPL Dev (with “Almost” Continuous Integration)
  – Regular APP developers work independently using very recent APP-owned VC copy of TPL Dev-
  – Regular TPL developers work independently
  – APP Dev + TPL Dev developers
    • Check-out and modify APP Dev
    • Check-out and modify TPL Dev
    • Run both APP and TPL pre-checkin test suites
      • Check into both APP-owned and main TPL VC repositories
  – Nightly testing of APP Dev + TPL Dev automatically updates APP-owned TPL Dev- VC Repository
  – Releases best handled as combined releases of APP and TPL
Summary #2

• Integration Models:
  – APP Dev only upgraded after each major release of TPL
    • Little to no testing of APP + TPL Dev in between TPL releases
  – APP Dev builds against both TPL Release and TPL Dev
    • Daily Integration testing done for both APP + TPL Release and Dev
    • Staggered releases of TPL and APP
  – APP Dev developed only against TPL Dev (with “Almost” Continuous Integration)
    • APP Dev + TPL Dev developers update both APP-owned and main TPL repositories
    • Nightly testing of APP Dev + TPL Dev automatically updates APP-owned TPL Dev- VC Repository
    • Releases best handled as combined releases of APP and TPL
    • TPL Dev- checkins can be dialed back approaching TPL Release and Dev Integration!

• Final thoughts
  – Each of these different integration models will be appropriate for a particular APP+TPL situation.
  – The particular integration model can be switched during the life-cycles of the APP and TPL depending on several factors:
    • How critical is the TPL functionality currently to the APP?
    • Are there alternatives to a particular TPL that can duplicate functionality?
    • How actively is the TPL being developed?
    • Is it critical for the APP to continue to accept new releases of the TPL?
    • How active is the collaboration between APP and TPL developers?
    • Is the TPL a fundamental part of the infrastructure of the APP?
    • ...