Integration Strategies for Computational Science & Engineering Software

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

Sandia National Laboratories

Second International Workshop on Software Engineering for Computational Science and Engineering
May 23, 2009
Software Integration in the CS&E Environment

• Need to integrate a large amount of CS&E software:
  – Meshing
  – Discretizations
  – Solvers
  – Adaptivity
  – Analysis capabilities
  – Visualization
  – ...

• Each CS&E discipline is highly specialized and requires PhD-level expertise

• The set of algorithms and software is too large for any single organization to produce

• Set of software is too large to be developed under a single blanket of Full Continuous Integration (CI)

=> Software Engineering and Software Integration are key bottlenecks for CS&E to have the fullest impact!
CS&E Environment at Sandia National Labs for Trilinos

- Sophisticated CS&E applications
  - Discretized PDEs (SIERRA, Alegra, Aleph, Charon)
  - Circuit network models (Xyce)
  - Other types of calculations (Titan/VTK, Tramonto)
- (Massively) parallel MPI (Gordon Bell Winners)
- Almost entirely developed by non-software people
- Wide range of research to production (i.e. from Aleph to SIERRA)

TPL: Third Party Lib
- Provides functionality to multiple APPs
- The “Supplier” to the APP

APP: Application
- Delivers end user functionality
- The “Customer” of the TPL
Standard Software Integration Approaches

• Continuous Integration (CI)
  – Code is expected to build and the tests are expected to run
  – Maintained through synchronous or asynchronous CI
  – Requires high levels of cooperation and communication
  – Requires code to (re)build fast and tests to run fast

• Customer/Supplier Relationships
  – Combined code too large to build under single CI system
  – Organizations can not cooperate close enough
  – Protect APP for future TPL updates through development of Acceptance Test Suite
  – May not work as well for may CS&E codes
  – Not as well suited for closer collaborations
Challenges to Software Integration in CS&E Environments

- CS&E is a mix of research and production work
  - How can you mix research and production software?
- CS&E practitioners have a wide mix of backgrounds in physics, math, computer science, engineering, etc.
  - How can these people communicate together and integrate their technologies?
- CS&E involved very complex, very specialized algorithms
  - Requires PhD in area to develop best algorithms/software
  - How can you integrate very different complex algorithms software?
- Great variability in knowledge and interests in basic software development knowledge and skills
  - How can you produce high quality trusted software with unskilled programmers?
- Close collaboration between different disciplines needed to solve the hard problems
  - How can different practitioners work together through their software?
- CS&E heavily relies on fast floating-point computations
  - Output from program varies between platforms and even with different compiler options!
  - How can you keep tests working on different platforms?
- CS&E involves complex nonlinear models
  - Examples: ill conditioning, multiple solutions, bifurcations, non-convexities ...
Special Challenges with CS&E Software

• CS&E heavily relies on fast floating-point computations
  – Output from program varies between platforms and even with different compiler options!
  – How to you keep tests working on different platforms?
• CS&E involves complex nonlinear models
  – Examples: ill conditioning, multiple solutions, bifurcations, non-convexities ...

These issues conspire together to make testing and maintaining CS&E software on multiple platforms very difficult!

Consequences:
• A new test status: The differing test!
  – Code runs to completion but some error tolerance was exceeded
  – Many CS&E practitioners convince themselves that a “diff” is not as bad as a “fail”!
• Changes to a numerical algorithm that improve performance in every measure can cause numerous tests to ‘diff’ or even ‘fail’!
• Upgrades of a TPL can break an APP even if no real defects have been introduced!
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
• Keep APP Dev and TPL Dev up to date! => Supported by TPL 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 and collaboration
  • APP developers can break “New” a lot when refactoring
• However, this is satisfactory for many APP+TPL efforts!
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 when refactoring
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!
• 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 their pre-checkin APP test suite.

• Code that builds and passes the pre-checkin test suite is safe to check in.

• Co-development of the APP + TPL needs to be productive and not discourage frequent checkins (at least to direct collaborators).

• Regular APP developers should be able to easily build and test “New” APP + TPL Dev code to avoid breaking it before checkin.
APP + TPL Almost Continuous Integration: Overview

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

- TPL Dev
  - Nightly Testing

APP Owned

- Main APP VC Repository (Dev)
- APP-owne 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

APP Dev + TPL Dev Co-Developers

TPL Dev Developers
APP + TPL Almost Continuous Integration: Co-Development

- 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!
APP + TPL Almost Continuous Integration: Releases

Nightly Testing: APP Dev + TPL Dev (pre-checkin tests only, TPL Dev- checkin)
Nightly Testing: APP Dev + TPL Dev- (complete test suites)
Supported with asynchronous continuous integration testing of APP Dev + TPL Dev

- 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!
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
Experience with Integration Approaches with Trilinos at SNL

Charon + Trilinos Integration:
- First implemented APP + TPL Release and Dev Daily Integration in 2007
- Maintained daily integration with little effort
- Supporting more ambitious collaborations and integration efforts
- However, has never gone through a full release process under this model

Alegra + Trilinos Integration:
- Started APP + TPL Release and Dev Daily Integration in 2008
- Maintained daily integration with little effort on multiple platforms
- Upgrade to Trilinos 9.0 was easy and risk free, less overall effort

SIERRA + Trilinos Integration:
- Started APP + TPL Release and Dev Daily Integration in mid 2008
- Before daily integration:
  - SIERRA 4.9 released against Trilinos 7.0 (a 1.5 year old release)
  - Upgrade of SIERRA VOTD to Trilinos 8.0 was a “disaster”
- After daily integration:
  - SIERRA 4.10 released against Trilinos 9.0 (2 months old) with no issues
  - SIERRA 4.11 released against snapshot branch of Trilinos (2 weeks old)
- Currently having lots of problems with broken code in “New” APP code
- APP + TPL Almost Continuous Integration Process currently being developed!
• 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?
  – ...
Conclusions

• Need to integrate a large amount of CS&E software:
  – Meshing
  – Discretizations
  – Solvers
  – Adaptivity
  – Analysis capabilities
  – Visualization
  – ...

• Software Engineering and Software Integration are key bottlenecks for CS&E to have the fullest impact!

• The CS&E R&D community needs to adopt better Lean/Agile software engineering methods:
  – Need a strategy to inject basic software engineering knowledge into CS&E
  – These methods must be adapted to the special properties of CS&E
The End
Summary of CS&E Software Integration Models

• 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.

• Integration Models:
  – APP + TPL Release with Punctuated TPL Upgrades
    • Little to no testing of APP + TPL Dev in between TPL releases
  – APP + TPL Release and Dev Daily Integration
    • Daily Integration testing done for both APP + TPL Release and Dev
    • Staggered releases of TPL and APP
  – APP + TPL 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!