Trilinos Software Engineering Technologies and Integration Capability Area Overview

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

Sandia National Laboratories

Trilinos User Group Meeting, November 3, 2009
Trilinos Software Engineering Technologies and Integration

• Numerical Algorithm Interoperability and Vertical Integration
  – Abstract Numerical Algorithms (ANAs)
  – Thyra (Interoperability and vertical integration of ANAs)
  – Epetra (Interoperability of element-based numerical algorithms)

• General Software Interoperability and Integration
  – Memory management (Teuchos::RCP, ...)
  – User input and configuration control (Teuchos::ParameterList, ...)
  – User introspection (Teuchos::FancyOStream, ...)

• Skin packages (wrappers for other languages)
  – PyTrilinos, ForTrilinos, Ctrilinos

• General Software Quality and Design
  – Separation of “Stable” vs. “Experimental” code
  – Day-to-day stability of “Stable” code

• Lean/Agile Software Engineering Principles and Practices
  – Internal Trilinos issues
  – External customer issues
Lean/Agile Software Engineering Principles and Practices

• Internal Trilinos development tools principles and practices
  – Scalability and robustness of build system and test tools
  – Continuous integration development principles and practices
  – Release process principles and practices

• Integration with customer application codes
  – Coordination of co-development with customer application codes (i.e. daily integration and asynchronous continuous integration)
  – Coordination of release schedules with customer application codes
  – Regulated backward compatibility and smooth upgrades
Backward Compatibility Considerations

- Backward compatibility is critical for:
  - Safe upgrades of Trilinos releases
  - Composability and compatibility of different software collections
Example of the Need for Backward Compatibility

Multiple releases of Trilinos presents a possible problem with complex applications

Solution:
=> Provide perfect backward compatibility of Trilinos X through Trilinos SIERRA Y+1
Backward Compatibility Considerations

• Backward compatibility is critical for:
  • Safe upgrades of Trilinos releases
  • Composability and compatibility of different software collections

• Maintaining backward compatibility for all time has downsides:
  • Testing/proving backward compatibility is expensive and costly
  • Encourages not changing (refactoring) existing interfaces etc.
    • => Leads to software “entropy” which kills a software product

• A compromise: Regulated backward compatibility (Tentative)
  • Maintain a window of perfect backward compatibility over major version numbers (e.g. 1-2 years)
  • Provide “Deprecated” compiler warnings
    • Example: GCC’s __deprecated__ attribute enabled with
      –DTrilinos_SHOW_DEPRECATED_WARNINGS:BOOL=ON
  • Provide strong automated testing of Trilinos backward compatibility
  • Drop backward compatibility between major version numbers
Regulated Backward Compatibility for Trilinos (Tentative)

- Releases of Trilinos X guarantee backward comparability between releases X.Y and X.Z where Z > Y
  - Example: Trilinos 10.5 is backward compatible with 10.0 through 10.4
  - Example: Trilinos 11.X is not compatible with Trilinos 10.Y
- Major Trilinos version numbers change every 1-2 years
  - Example: Major Trilinos versions change every 2 years with 2 releases per year

Maintain backward compatibility of 11.0 with only 10.3 but drop all other deprecated code!

Backward compatibility test Dev and current release every night!

- Actual Target (Tentative):
  - Keep major Trilinos version number for two years
  - Put out releases quarterly (with minor releases X.Y.Z as needed)
http://trilinos.sandia.gov/capability_areas.html