

The Trilinos Software Lifecycle Model

Michael A. Heroux, Presenter

James M. Willenbring

Robert T. Heaphy

Sandia National Laboratories



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under contract DE-AC04-94AL85000.



Trilinos Contributors

Chris Baker

Developer of Anasazi, RBGen

Ross Bartlett

Lead Developer of MOOCHO, Stratimikos, RTOp, Thyra
Developer of Rythmos

Paul Boggs

Developer of Thyra

Erik Boman

Lead Developer Isorropia

Todd Coffey

Lead Developer of Rythmos

Jason Cross

Developer of Jpetra

David Day

Developer of Komplex

Clark Dohrmann

Lead developer of CLAPS

Michael Gee

Developer of ML, Moertel, NOX

Bob Heaphy

Lead developer of Trilinos SQA

Mike Heroux

Trilinos Project Leader
Lead Developer of Epetra, AztecOO,
Kokkos, Komplex, IFPACK, Thyra, Tpetra
Developer of Amesos, Belos, EpetraExt, Jpetra,
Teuchos

Ulrich Hetmaniuk

Developer of Anasazi

Robert Hoekstra

Lead Developer of EpetraExt
Developer of Epetra, Thyra, Tpetra

Russell Hooper

Developer of NOX

Vicki Howle

Lead Developer of Meros
Developer of Belos and Thyra

Jonathan Hu

Developer of ML

Sarah Knepper

Developer of Komplex

Tammy Kolda

Lead Developer of NOX

Joe Kotulski

Lead Developer of Pliiris

Rich Lehoucq

Developer of Anasazi and Belos

Kevin Long

Lead Developer of Thyra
Developer of Belos and Teuchos

Roger Pawlowski

Lead Developer of NOX

Michael Phenow

Trilinos Webmaster
Lead Developer of New_Package
Developer WebTrilinos

Eric Phipps

Lead developer Sacado
Developer of LOCA, NOX

Dennis Ridzal

Lead Developer of Aristos

Marzio Sala

Lead Developer of Didasko, Galeri, IFPACK, WebTrilinos
Developer of ML, Amesos

Andrew Salinger

Lead Developer of LOCA, Capo

Paul Sexton

Developer of Epetra and Tpetra

Bob Shuttleworth

Developer of Meros.

Chris Siefert

Developer of ML

Bill Spatz

Lead Developer of PyTrilinos
Developer of Epetra, New_Package

Ken Stanley

Lead Developer of Amesos and New_Package

Heidi Thornquist

Lead Developer of Anasazi, Belos, RBGen and Teuchos

Ray Tuminaro

Lead Developer of ML and Meros

Jim Willenbring

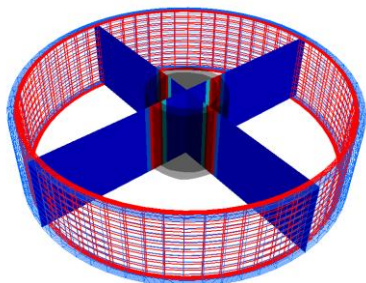
Developer of Epetra and New_Package.
Trilinos library manager

Alan Williams

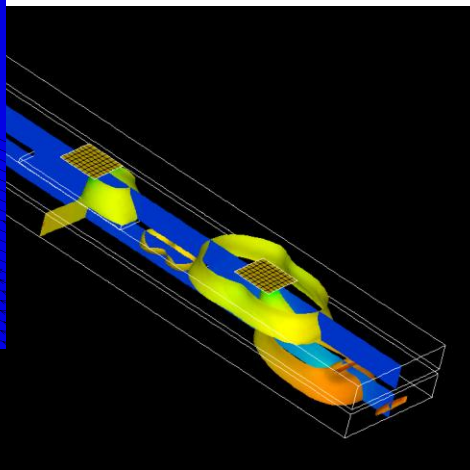
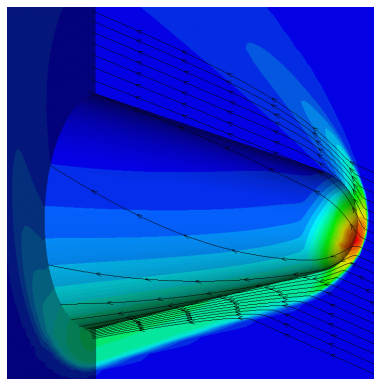
Lead Developer Isorropia
Developer of Epetra, EpetraExt, AztecOO, Tpetra

Background/Motivation

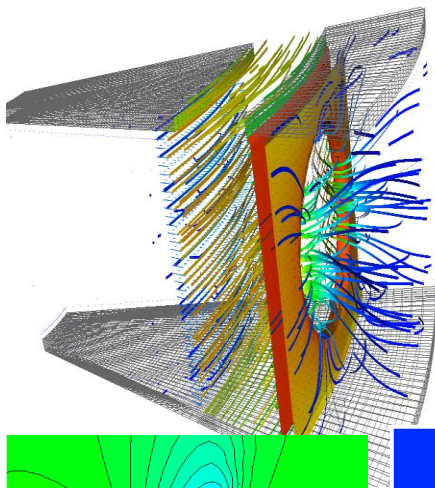
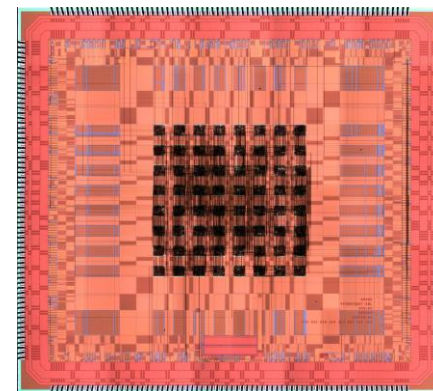
Target Problems: PDES and more...



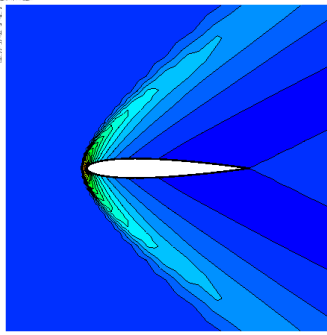
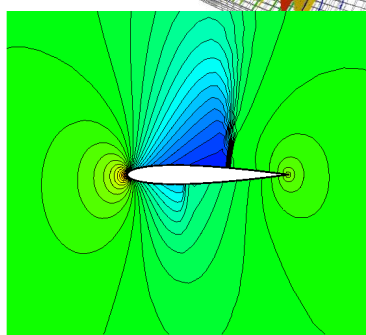
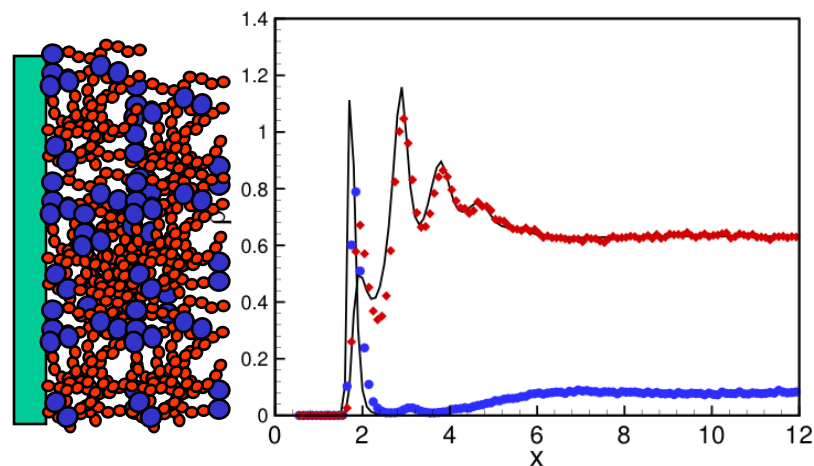
PDES



Circuits



Inhomogeneous Fluids

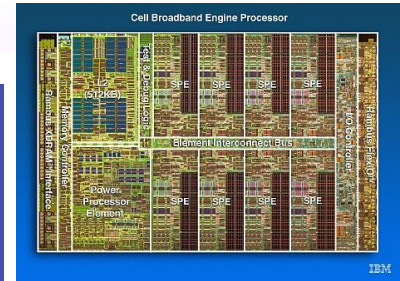
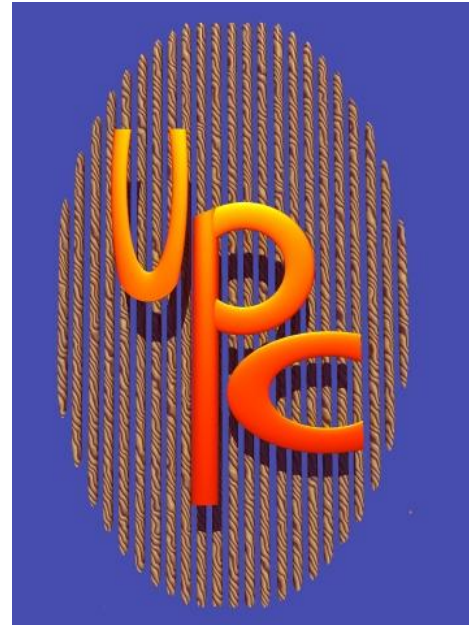


And More...

Target Platforms: Any and All (Now and in the Future)



- Desktop: Development and more...
- Capability machines:
 - ◆ Redstorm (XT3), Clusters
 - ◆ Roadrunner (Cell-based).
 - ◆ Large-count multicore nodes.
- Parallel software environments:
 - ◆ MPI of course.
 - ◆ UPC, CAF, threads, vectors,...
 - ◆ Combinations of the above.
- User “skins”:
 - ◆ C++/C, Python
 - ◆ Fortran.
 - ◆ Web, CCA.



Motivation For Trilinos

- Sandia does LOTS of solver work.
- When I started at Sandia in May 1998:
 - ◆ Aztec was a mature package. Used in many codes.
 - ◆ FETI, PETSc, DSCPack, Spooles, ARPACK, DASPCK, and many other codes were (and are) in use.
 - ◆ New projects were underway or planned in multi-level preconditioners, eigensolvers, non-linear solvers, etc...
- The challenges:
 - ◆ Little or no coordination was in place to:
 - Efficiently reuse existing solver technology.
 - Leverage new development across various projects.
 - Support solver software processes.
 - Provide consistent solver APIs for applications.
 - ◆ ASCI (now ASC) was forming software quality assurance/engineering (SQA/SQE) requirements:
 - Daunting requirements for any single solver effort to address alone.

Evolving Trilinos Solution

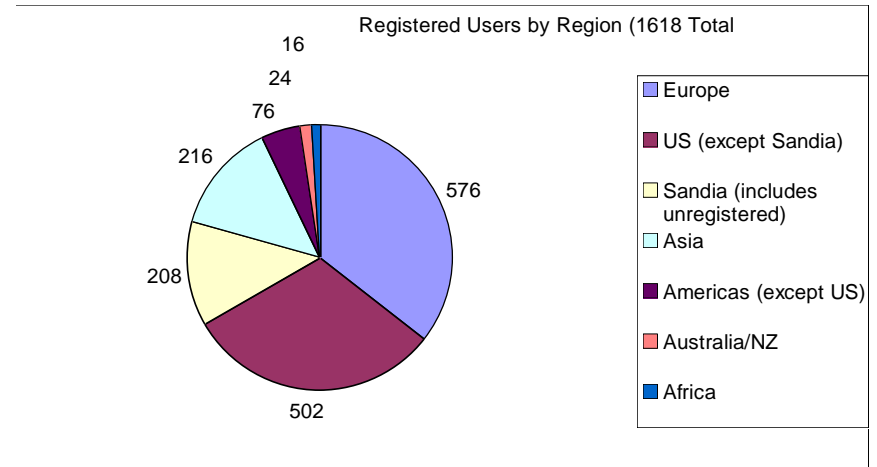
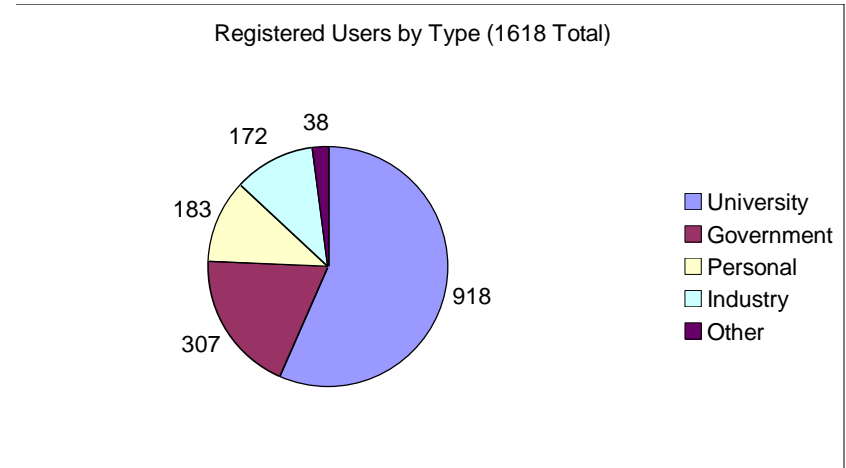
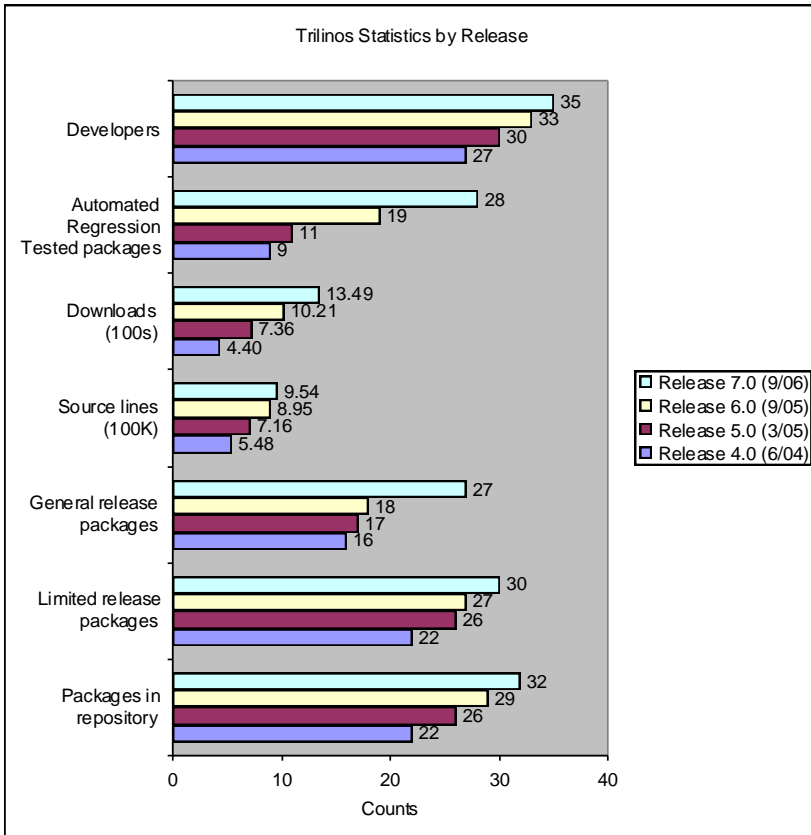
- Trilinos¹ is an evolving framework to address these challenges:
 - ◆ Fundamental atomic unit is a *package*.
 - ◆ Includes core set of vector, graph and matrix classes (Epetra/Tpetra packages).
 - ◆ Provides a common abstract solver API (Thyra package).
 - ◆ Provides a ready-made package infrastructure (new_package package):
 - Source code management (cvs, bonsai).
 - Build tools (autotools).
 - Automated regression testing (queue directories within repository).
 - Communication tools (mailman mail lists).
 - ◆ Specifies requirements and suggested practices for package SQA.
- In general allows us to categorize efforts:
 - ◆ Efforts best done at the Trilinos level (useful to most or all packages).
 - ◆ Efforts best done at a package level (peculiar or important to a package).
 - ◆ **Allows package developers to focus only on things that are unique to their package.**

1. Trilinos loose translation: “A string of pearls”

Trilinos Strategic Goals

- **Scalable Solvers:** As problem size and processor counts increase, the cost of the solver will remain a nearly fixed percentage of the total solution time.
 - **Hardened Solvers:** Never fail unless problem essentially unsolvable, in which case we diagnose and inform the user why the problem fails and provide a reliable measure of error.
 - **Full Vertical Coverage:** Provide leading edge capabilities from basic linear algebra to transient and optimization solvers.
- } Algorithmic Goals
- **Grand Universal Interoperability:** All Trilinos packages will be interoperable, so that any combination of solver packages that makes sense algorithmically will be possible within Trilinos.
 - **Universal Accessibility:** All Trilinos capabilities will be available to users of major computing environments: C++, Fortran, Python, Web
 - **Universal Solver RAS:** Trilinos will be:
 - ◆ Integrated into every major application at Sandia (**Availability**).
 - ◆ The leading edge hardened, efficient, scalable solutions for each of these applications (**Reliability**).
 - ◆ Easy to maintain and upgrade within the application environment (**Serviceability**).
- } Software Goals

Trilinos Statistics



Trinos Package Concepts

Package: The Atomic Unit

Trilinos Packages

- Trilinos is a collection of *Packages*.
- Each package is:
 - ◆ Focused on important, state-of-the-art algorithms in its problem regime.
 - ◆ Developed by a small team of domain experts.
 - ◆ Self-contained: No explicit dependencies on any other software packages (with some special exceptions).
 - ◆ Configurable/buildable/documented on its own.
- Sample packages: NOX, AztecOO, ML, IFPACK, Meros.
- Special package collections:
 - ◆ Petra (Epetra, Tpetra, Jpetra): Concrete Data Objects
 - ◆ Thyra: Abstract Conceptual Interfaces
 - ◆ Teuchos: Common Tools.
 - ◆ New_package: Jumpstart prototype.

Objective	Package(s)	Trilinos Package Summary
Linear algebra objects	Epetra, Jpetra, Tpetra	
Krylov solvers	AztecOO, Belos, Komplex	
ILU-type preconditioners	AztecOO, IFPACK	
Multilevel preconditioners	ML, CLAPS	
Eigenvalue problems	Anasazi	
Block preconditioners	Meros	
Direct sparse linear solvers	Amesos	
Direct dense solvers	Epetra, Teuchos, Pliris	
Abstract interfaces	Thyra	
Nonlinear system solvers	NOX, LOCA	
Time Integrators/DAEs	Rythmos	
C++ utilities, (some) I/O	Teuchos, EpetraExt, Kokkos	
Trilinos Tutorial	Didasko	
“Skins”	PyTrilinos, WebTrilinos , Star-P, Stratimikos , <i>ForTrilinos</i>	
Optimization	MOOCHO , Aristos	
Archetype package	NewPackage	
Other new in 7.0 (8.0)	Galeri , Isorropia , Moertel , RTOp , <i>Aristos</i> , <i>RBGen</i>	

Package Maturation Process

Asynchronicity

Day 1 of Package Life

- **CVS:** Each package is self-contained in Trilinos/package/ directory.
- **Bugzilla:** Each package has its own Bugzilla product.
- **Bonsai:** Each package is browsable via Bonsai interface.
- **Mailman:** Each Trilinos package, including Trilinos itself, has four mail lists:
 - ◆ package-checkins@software.sandia.gov
 - CVS commit emails. “Finger on the pulse” list.
 - ◆ package-developers@software.sandia.gov
 - Mailing list for developers.
 - ◆ package-users@software.sandia.gov
 - Issues for package users.
 - ◆ package-announce@software.sandia.gov
 - Releases and other announcements specific to the package.
- **New_package** (optional): Customizable boilerplate for
 - ◆ Autoconf/Automake/Doxygen/Python/Thyra/Epetra/TestHarness/Website

Sample Package Maturation Process

Step	Example
Package added to CVS: Import existing code or start with new_package.	ML CVS repository migrated into Trilinos (July 2002).
Mail lists, Bugzilla Product, Bonsai database created.	ml-announce, ml-users, ml-developers, ml-checkins, ml-regression @software.sandia.gov created, linked to CVS (July 2002).
Package builds with configure/make, Trilinos-compatible	ML adopts Autoconf, Automake starting from new_package (June 2003).
Epetra objects recognized by package.	ML accepts user data as Epetra matrices and vectors (October 2002).
Package accessible via Thyra interfaces.	ML adaptors written for TSFCore_LinOp (Thyra) interface (May 2003).
Package uses Epetra for internal data.	ML able to generate Epetra matrices. Allows use of AztecOO, Amesos, Ifpack, etc. as smoothers and coarse grid solvers (Feb-June 2004).
Package parameters settable via Teuchos ParameterList	ML gets manager class, driven via ParameterLists (June 2004).
Package usable from Python (PyTrilinos)	ML Python wrappers written using new_package template (April 2005).

Startup Steps

Maturation Steps


Maturation Jumpstart: NewPackage

- NewPackage provides jump start to develop/integrate a new package
- NewPackage is a “Hello World” program and website:
 - ◆ Simple but it does work with autotools.
 - ◆ Compiles and builds.
- NewPackage directory contains:
 - ◆ Commonly used directory structure: src, test, doc, example, config.
 - ◆ Working Autoconf/Automake files.
 - ◆ Documentation templates (doxygen).
 - ◆ Working regression test setup.
 - ◆ Working Python and Thyra adaptors.
- Substantially cuts down on:
 - ◆ Time to integrate new package.
 - ◆ Variation in package integration details.
 - ◆ Development of website.

NOTE: NewPackage can be used independent from Trilinos

SQA/SQE

- Software Quality Assurance/Engineering is important.
- Not sufficient to say, “We do a good job.”
- Trilinos facilitates SQA/SQE development/processes for packages:
 - ◆ 10 of 30 ASC SQE practices are directly handled by Trilinos (no requirements on packages).
 - ◆ Trilinos provides infrastructure support for the remaining 20.
 - ◆ Trilinos Dev Guide Part II: Specific to ASC requirements.
 - ◆ Trilinos software engineering policies provide a ready-made infrastructure for new packages.
 - ◆ Trilinos philosophy:
Few *requirements*. Instead mostly *suggested practices*. Provides package with option to provide alternate process.

Trilinos Service	SQE Practices Impact
<p>Yearly Trilinos User Group Meeting (TUG) and Developer Forum: Once a year gathering for tutorials, package feature updates, user/developer requirements discussion and developer training.</p>	<ul style="list-style-type: none"> — All Requirements steps: gathering, derivation, documentation, feasibility, etc. — User and Developer training.
<p>Monthly Trilinos leaders meetings: Trilinos leaders, including package development leaders, key managers, funding sources and other stakeholders participate in monthly phone meetings to discuss any timely issues related to the Trilinos Project.</p>	<ul style="list-style-type: none"> — Developer Training. — Design reviews. — Policy decisions across all development phases.
<p>Trilinos and package mail lists: Trilinos lists for leaders, announcements, developers, users, checkins and similar lists at the package level support a variety of communication. All lists are archived, providing critical artifacts for assessments and audits.</p>	<ul style="list-style-type: none"> — Developer/user/client communication. — Requirements/design/testing artifacts. — Announcement/documenting of releases.
<p>Trilinos and Trilinos3PL source repositories: All source code, development and user documentation is retained and tracked. In addition, reference versions of all external software, including BLAS, LAPACK, Umfpack, etc. are retained in Trilinos3PL.</p>	<ul style="list-style-type: none"> — Source management. — Versioning. — Third-party software management.
<p>Bugzilla Products: Each package has its own Bugzilla Product with standard components.</p>	<ul style="list-style-type: none"> — Requirements/faults capturing and tracking.
<p>Trilinos configure script and M4 macros: The Trilinos configure script and related macros support portable installation of Trilinos and its packages</p>	<ul style="list-style-type: none"> — Portability. — Software release.
<p>Trilinos test harness: Trilinos provides a base testing plan and automated testing across multiple platforms, plus creation of testing artifacts. Test harness results are used to derive a variety of metrics for SQE.</p>	<ul style="list-style-type: none"> — Pre-checkin and regression testing. — Software metrics. <div style="text-align: right;">  <p>Sandia National Laboratories</p> </div>

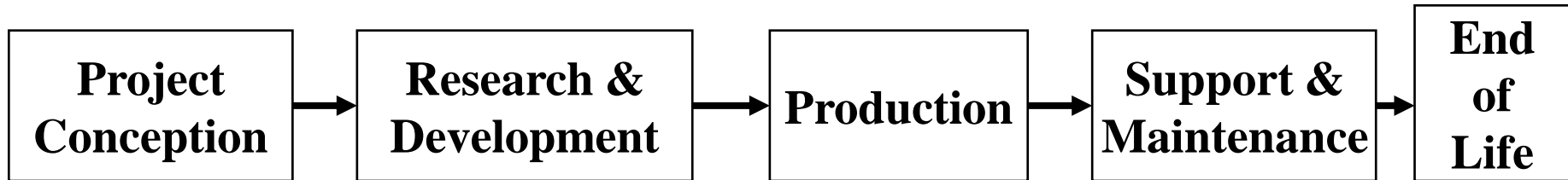
Trilinos Availability/Information

- Trilinos and related packages are available via LGPL.
- Current release (7.0) is “click release”. Unlimited availability.
- Trilinos Release 8: August 2007.
- Trilinos Awards:
 - ◆ 2004 R&D 100 Award.
 - ◆ SC2004 HPC Software Challenge Award.
 - ◆ Sandia Team Employee Recognition Award.
 - ◆ Lockheed-Martin Nova Award Nominee.
- More information:
 - ◆ <http://trilinos.sandia.gov>
 - ◆ <http://software.sandia.gov>
 - ◆ Additional documentation at my website:
<http://www.cs.sandia.gov/~mheroux>.
- 5th Annual Trilinos User Group Meeting: November 6-8, 2007 at Sandia National Laboratories, Albuquerque, NM, USA.

Software Lifecycles

(Typical) Project Lifecycle

Consider this lifecycle



Scientific Research and Life Cycle Models

- Life Cycle Models are generally developed from the point of view of business software.
- Little consideration is given to algorithmic development.
- Traditional business execution environment is traditional mainframe or desktop, not parallel computers.
- Traditional development “techniques” are assumed.

Research Software needs a different model

- Research should be “informal”:
 - ◆ Allow external collaborators, students, post-docs, etc.
 - ◆ Allow changes of direction without seeking permission
 - ◆ Should use modern software development paradigms
 - i.e. Lean/Agile methods
 - ◆ Must be verified more than validated

- Production code must:
 - ◆ Have formality appropriate to risks,
 - ◆ Be Complete (documentation, testing, ...),
 - ◆ Be “user proofed”,
 - ◆ Be platform independent (as necessary),
 - ◆ Be validated not just verified.

“Promotional” Model



- Lower formality
- Fewer Artifacts
- Lean/Agile

- Higher formality
- Sufficient Artifacts
- Bullet proof
- Maintainable

Trilinos Software Lifecycle Model

Lifecycle Models

- A major component of any software project.
- Exists, whether formal or *ad hoc*.
- Trilinos model:
 - ◆ Really a meta-model.
 - ◆ Attempts to captures the reality of our software engineering environment.

Trilinos Software Environment

- Many formal software lifecycle models exist.
- Trilinos environment seems somewhat unique:
(When compared to commercial environments, or not?)
- On one hand: Tasked to develop algorithms and software that are leading-edge, with the goal of solving problems that were previously intractable.
- On the other: Required to deliver software that can eventually be used to certify critically important engineering systems.

Further Complexities

- Trilinos composed of *packages*:
 - ◆ Self-contained pieces of software developed by semi-independent small teams.
 - ◆ Each package matures at its own pace:
 - Typically evolving from small algorithms study.
 - Becoming a widely-used piece of software.
 - Embedded in multiple applications.
- Requirements for rigor change as a particular Trilinos package matures.

Trilinos Lifecycle Phases

- Three phases:
 - ◆ Research.
 - ◆ Production Growth.
 - ◆ Production Maintenance.
- Each phase contains its own lifecycle model.
- Promotional events:
 - ◆ Required for transition from one phase to next.
 - ◆ Signify change in behaviors and attitude.
- Phase assigned individually to each package.

Lifecycle Phase 1: Research

- Conducting research is the primary goal.
- Producing software is potentially incidental to research.
- Any software that is produced is typically a “proof of concept” or prototype.
- Software that is in this phase may only be released to selected internal customers to support their research or development and should not be treated as production quality code.

Phase 1 Required Practices

- The research proposal is the project plan.
- Software is placed under configuration control as needed to prevent loss due to disaster.
- Peer reviewed published papers are primary verification and validation.
- The focus of testing is a proof of correctness, not software.
- Periodic status reports should be produced, presentation is sufficient.
- A lab notebook, project notebook, or equivalent is the primary artifact.

Phase 1 Remarks

- Phase 1 practices are common to efficient research in general.
- Research phase software:
 - ◆ Need not be written in the “target” language.
 - ◆ Nor support all target machines.
 - ◆ Usually has limited error checking and recovery.
- (Software) risk is low (primarily technical, not mitigated formality processes).
- Level of formality is low.

Phase 2: Production Growth

Goals:

1. Elevate package to releasable product.
2. Satisfy the Advanced Scientific Computing (ASC) Software Quality Plan, at a minimum.
3. Make software product suitable for use by highly skilled users.

Phase 1→2 Promotion Event

Risk Assessment:

1. What are the package's primary technical and project management risks?
2. How can these risks be mitigated?

Gap Analysis:

1. Which practices and processes must be added or improved to get the package into a releasable state?
2. What special actions or training will be required?
3. What is the target date for complying with the level of practices and processes required for release?

Promotion Decision:

1. Considering the results of the risk assessment, gap analysis, and other data, will the package be promoted to Phase 2?
2. What is the target date for releasing the package?

Phase 2 Required Practices

(Most Important)

1. Agile methods (with associated lifecycles) are encouraged, for example the practices and processes promoted by Extreme Programming .
2. All essential ASC SQE practices performed at an appropriate level (predetermined during promotion event from the research phase).
3. Artifacts should naturally “fall out” from SQE practices and periodic status reviews and management reports.
4. Process improvement and metrics are appropriate.

Phase 2 Remarks

- Phase may be cyclic (spiral, etc.) as new algorithms become incorporated.
- Software may not yet support all intended missions or platforms.
- Risk level is medium:
 - ◆ Technical risks are reduced.
 - ◆ Total risk is more project management oriented such as schedule, budget, staffing, etc.).
- Default level of formality is medium.

Phase 3: Production Maintenance

- Goal: Robust software suitable for typical end uses.
- At this point:
 - ◆ Requirements and prototype software foundation are stable.
 - ◆ However, Agile methods no long sufficient:
 - Product maintenance during the coming decades of software use where typically only adaptive maintenance
 - Response to computer system changes.
- More complete set of artifact is required.
- Software itself will require changes to improve maintainability.
- In extreme case: May make sense to rewrite large portions.

Phase 2 → 3 Promotional Event

Risk Assessment:

1. What are the package's primary technical and project management risks?
2. How can these risks be mitigated?

Gap Analysis:

1. Which practices and processes must be added or improved to get the package into a maintenance ready state?
2. What is the target date for complying with the required level of practices and processes?

Phase 2 → 3 Promotional Event

(cont)

Promotion Decision:

- What is the medium to long-term funding outlook for the package?
- Who is going to provide long-term maintenance services for the project? (One or more of the original developers, or a different group?)
- If funding is not likely to be available for future maintenance, current customers should be notified so they have a chance to offer continued funding if it is in their best interest. A list of these customers should be produced.
- If the customer base of the package is small or the package has been replaced with another code, the development team may consider retiring the code rather than moving to the third development phase. Any such decision should be approved by customers and management.
- Considering the answers to the above questions, and other available data, will the package be promoted to Phase 3?
- What is the target date (if any) for turning the package over to the long-term maintenance team?

Phase 3: Required Practices

1. After achieving maintenance ready status, the package may (as determined during the promotion event) be handed over to another party during this phase for continued support and development.
2. If the code is transferred to a different party, the ownership of the design is not necessarily transferred. The design owner must attend meetings concerning requirements changes and potential design changes.
3. A widely-accepted lifecycle model such as the Waterfall or Unified Process methods is used.
4. End of life planning is a key component during this phase. In particular, the software must have good compliance with SQE practices and internal documentation must be formal (UML is suggested).
5. SQE practice compliance and solid documentation will help to ensure a successful transfer of the code, and must be completed whether a transfer is currently planned or not.

Phase 3 Remarks

- The risk level is low (almost totally project management risks, which can be mitigated by appropriate process formality). The default level of formality is high (particularly if the project may be handed off to another party).
- This phase is untested so far.
- Moving to this phase is expensive.

Exceptional Cases

- Isolated Lower Phase:
 - ◆ Multiple techniques:
 - Subdirectories: All lower phase software is contained in specially-designated subdirectories and is only activated by special compilation procedures.
 - Interface adapters: Lower phase software is self-contained in new class files and accessed via polymorphism of abstract interface mechanisms or similar techniques that are not necessary for basic operation.
 - Conditional compilation directives: discouraged in general.
- Externally-developed Packages:
 - ◆ By default in Phase 1: Must go through same process.
 - ◆ BLAS/LAPACK different: Certified as part of dependent-package process.

Usage Status

- Lifecycle Model newly define: Gives us a target.
- 25% of package firmly in Phase 1.
- 75% of packages on the way to Phase 2.
- None are in Phase 3.
- Phase 3 is thus just speculation at this point.

Summary

- For years we have struggled to adapt to traditional lifecycle models, with little success.
- The Trilinos Software Lifecycle Model:
 - ◆ Provides tangible set of goals that seem to match our needs.
 - ◆ Is a work-in-progress.
- We actively seek interaction with others who have common environments and interests.