Software Design for Scientific Applications

Karen Devine
Scalable Algorithms Department
Sandia National Laboratories

Erik Boman, Mike Heroux, Andrew Salinger (SNL)
Robert Heaphy (Respec Engineering)
Software Design for Scientific Applications

• Why distribute software?
• Challenges in software design and distribution
• Ideas for simplifying software design
  – Component design
  – Interface design
  – Software quality practices
• Challenges that remain
Distributing Software: Why bother?

• For your C.V.:
  – Greater visibility in research community.
  – Greater impact than only writing a paper.

• For the strength of your research program:
  – Wider group of reviewers of efforts.
  – Greater opportunity for collaboration.

• For altruistic reasons:
  – Contribution to scientific community.
  – Personal satisfaction in seeing your work used.
Distributing Software: Why bother?

• Because your funding source told you to.

  – “Successful applicants of Enabling Technologies must ensure that source code is fully and freely available for use and modification throughout the scientific computing community via a preapproved open source process.” -- DOE SciDAC CFP 2005.

  – “SDCI funds software activities for enhancing scientific productivity and for facilitating research and education collaborations through sharing of data, instruments, and computing and storage resources. The program requires open source software development.” -- NSF SDCI CFP 2007.
Challenges of Scientific Software Development

• Managing customer expectations.
  – Many funding sources.
  – Long-term vs. short-term goals.

• Users: a mixed blessing
  – Unpaid testers and debuggers.
  – Contribute to greater exposure, collaboration.
  – Have needs and their own deadlines.
Time Challenges

• Research vs. Software Development
  – Need to publish to gain tenure, advance in career.
  – Software development takes time, and production (or even semi-production) quality development takes LOTS of time.

• Staffing: “Let the grad students do it.”
  – High turnover results in intellectual loss.
    • “It takes a year to get back up to speed after a student graduates.”
  – Students often have little experience working in team environments.
  – Students need individual research achievements while contributing to team.
Technology Challenges

• Developing high-quality software isn’t easy.
  – Careful design needed.
  – Maintainability is important.
  – Robustness takes time, testing.
  – Portability to wide range of architectures.
  – Extensibility to new architectures.
Component Design

- Construct applications from smaller software “parts.”
- “Tinker-toy parallel computing” -- B. Hendrickson
- Components provide …
  - Services applications commonly need.
  - Support for wide range of applications.
  - Easy-to-use interfaces.
  - Data abstraction.
- Components avoid …
  - Prescribed data structures
  - Heavy framework
  - Limited freedom for application developers.

Hasbro, Inc.
Component Examples

**Linear/Nonlinear solvers**
- **Trilinos** (Sandia)
- **PETSc** (Argonne)

**ODE solvers**
- **Sundials** (LLNL)

**Dynamic load balancing**
- **Zoltan** (Sandia)

**Optimization**
- **DAKOTA** (Sandia)
- **TAO** (Argonne)

**Communication**
- **UPS** (LANL)

**Mesh adaptivity**
- **FMDB** (Rensselaer)
- **MeshSim** (Simmetrix, Inc.)
- **Pyramid** (JPL)

**Visualization**
- **VTK** (Kitware, Inc.)
Advantages of Component Design

• For users:
  – Smaller learning curve than large framework.
  – Easy to add to existing applications.
  – “Expert” implementations instead of “roll-your-own.”
  – Enable comparisons of different algorithms.
  – Time and cost saving.
• For developers:
  – Reduced dependencies between components eases effort of working in teams.
    • Separation of efforts for team members allows individual contribution with reduced conflicts.
  – Enable software reuse.
• For all: Migration path to emerging architectures.
  – Multicore, Cell, GPU, etc.
  – Implement new algorithms in components; users get benefits “for free.”
Connecting Components

• Connect either through a framework like CCA…
  – Common Component Architecture (Bernholdt, Armstrong, Kumfert, et al.)
  – Environment that facilitates software interoperability, programming language interoperability, and dynamic composability.
  – http://www.cca-forum.org

• … or manually through individual interfaces.
  – Classes, tools, interfaces.
Mesh Tools

Mesh I/O Interface

Composite Physics

Utilities

Rapid Development of Production Applications
Interfaces between Components

Input File

XML

Mesh File

Dakota Application Interface

ITAPS Interfaces

Mesh Database

Analysis Tools (black-box)

Analysis Tools (embedded)

Trilinos’ Stratimikos

Model Evaluator

Linear Algebra

Finite Element Interface

Derivative Tools

<Scalar>

Discretizations

Physics Fill

PostProcessing

Local Fill

ITAPS Interfaces

<Scalar>

Dakota Application Interface
Interface Design

• **Goals:**
  – Broad application support
  – Interoperability
  – Ease of use in new or existing applications

• **Abstraction:**
  – Identification of needed features
  – Removal of details
  – Management of complexity
  – Important for usability, maintainability, backward compatibility

• **Implementation strategies:**
  – Callback functions
  – Object-oriented design
  – Interoperable interfaces
Zoltan Toolkit

- Parallel dynamic load balancing and data services for dynamic, unstructured and/or adaptive applications.

Finite element simulations

Adaptive mesh refinement

Linear solvers & Preconditioners

Particle-based cell simulation

Contact detection & crash simulations
Zoltan Interface

• Zoltan data abstraction:
  – “Objects” with unique names (and, optionally, weights and/or coordinates) to be partitioned.
  – Objects can have (weighted) dependencies to other objects.

• Zoltan: Data-structure neutral design.
  – Supports wide range of applications and data structures.
  – Imposes no restrictions on application’s data structures.
  – Application does not have to build Zoltan’s data structures.
Zoltan Callback Interface

• Application interface:
  – Zoltan calls back to the application for needed info.
    • IDs of objects, coordinates, relationships to other objects.
  – Application provides simple functions to answer queries.
    • Registers function pointers with Zoltan.
  – Small extra costs in memory and function-call overhead.

• Once query functions are implemented, application can access all Zoltan functionality.

• Easy-to-use.
  – Provides separation between toolkit and application data structures.
  – Requires no extensive application coding.
  – Allows developers to retrofit old applications or build new ones.
Zoltan Application Interface

**APPLICATION**

- **Initialize Zoltan**
  - (Zoltan_Initialize, Zoltan_Create)

- **Select Method and Parameters**
  - (Zoltan_Set_Params)

- **Register callback function pointers**
  - (Zoltan_Set_Fn)

**ZOLTAN**

- **Zoltan_LB_Particle**:
  - Call callback functions.
  - Build data structures.
  - Compute new decomposition.
  - Return import/export lists.

- **Zoltan_Migrate**:
  - Call packing callback functions for exports.
  - Send exports.
  - Receive imports.
  - Call unpacking callback functions for imports.

- **Move data**
  - (Zoltan_Migrate)

- **Computation**
  - (Zoltan_LB_Particle)

- **Clean up**
  - (Zoltan_Destroy)
Another Example: Trilinos

- Trilinos is an evolving infrastructure to support component software development.
  - M. Heroux, PI, and many others.
  - Fundamental atomic unit of software is a package.
  - Provides a common abstract solver interface.
  - Specifies requirements and suggested practices for package Software Quality.
  - Allows package developers to focus only on things that are unique to their package.
Abstraction in Trilinos

• Core set of distributed vector, graph, matrix and communication classes (Epetra package). E.g.,
  – Epetra_CrsMatrix: Compressed-row sparse matrix.
  – Epetra_MultiVector: Dense matrix.
  – Epetra_Comm: Interprocessor communication.
• More than 25 packages of preconditioners and linear, nonlinear & eigensolvers built on top of Epetra.
• Consistent interface:
  – Applications using Epetra classes can access all solvers.
  – Class interfaces largely unchanged (except for additions) since initial design.
• Enabling research:
  – Underlying implementation almost completely rewritten since initial design.
  – Algorithms for emerging architectures (e.g., multicore) can be delivered through these interfaces.
**Evolving Trilinos Infrastructure**

- Beyond a “solvers” infrastructure
- Natural expansion of capabilities to satisfy application and research needs

**Numerical math**
Convert to models that can be solved on digital computers

**Algorithms**
Find faster and more efficient ways to solve numerical models

- New in Trilinos v9: Automatic differentiation, Discretization methods, Inline Meshing, …, Zoltan
Interoperable Interfaces:

**ITAPS**

- Interoperable Tools for Advanced Petascale Simulations (L. Diachin, PI)
- **ITAPS Goals**
  - Improve SciDAC applications’ ability to take advantage of state-of-the-art meshing and geometry tools.
  - Develop the next generation of meshing and geometry tools for petascale computing.
- “Standardization” of mesh and geometry interfaces.
Abstraction of PDE-simulation data hierarchy

- **Core Data Types**
  - Geometric Data: high-level description of the boundaries of the computational domain; e.g., CAD, image.
  - Mesh Data: geometric and topological information associated with the discrete representation of the domain.
  - Field Data: physics variables associated with application solution. Can be scalars, vectors, tensors; can be associated with any mesh entity.

- **Data Relation Manager**
  - Controls relationships among the core data types.
  - Resolves references between entities in different groups.
  - Performs operations depending on multiple core data types.

- Each core data type has an ITAPS interface.
  - Geometric data: iGeom
  - Mesh data: iMesh
  - Field data: iField
  - Relationship data: iRel
ITAPS Delivers Technology through ITAPS Interfaces

High-Level Petascale Integrated Tools

Build on Component Services

Are unified by Common Interfaces

Underlying Implementations

FMDB (Rensselaer) GRUMMP (U. British Columbia) MOAB (Argonne) FronTier (SUNY-SB)
Other Interface Examples

• MapReduce (J. Dean & S. Ghemawat, 2004): callback
  – Data abstracted to key/value pairs.
  – Operations done through Map and Reduce callbacks.
  – Map: processes key/value pair to produce intermediate data.
  – Reduce: merges values associated with intermediate keys.
  – Used for large-scale data manipulation, machine learning, clustering, indexing.

• Unix qsort, bsearch: callback mechanism
• PETSc (Gropp, Smith, et al.): object-oriented linear algebra
• C++ Standard Template Library: object-oriented
• MPI Message Passing Interface (Gropp et al.): interoperable interface
Which interface design to use?

- Abstraction is most important step.
  - Logical separations of data and operations.
  - Implementation approach is less important.

- Callback:
  - Simple to specify.
  - Easy to use.

- Object-oriented:
  - More elegant, robust abstraction mechanism.
  - Greater flexibility.

- Interoperability and standardization:
  - Always nice to have.
  - Can be implemented with callback or object-oriented approach.
Software Quality for Research

• Industry standards can be difficult to apply in research environments.
  – Level of formality too high.
• But we do many things prescribed in formal quality models.
  – Project planning: Proposals
    • Includes budget, staffing, project goals and milestones, requirements, approach.
  – Design review: peer review of publications.
• American National Standard: Quality Guidelines for Research
  – Section 3.5. The research proposal is the project plan.
  – Section 3.4. Peer review is “one of the primary mechanisms for assuring quality in science.”
A Minimalist’s Approach to Software Quality Engineering

• Seven Easy Steps to improved software development.
  – 1. Source Code Control
  – 2. Backup System
  – 3. Automated testing
  – 4. Bug Tracking
  – 5. Mailing Lists
  – 6. Documentation
  – 7. Release Process

• Small, low-burden tools that help manage projects.
• Overcome challenges of turnover, knowledge loss, working in teams.
1. Source Code Management

• Source management is essential.
  – Archive source code with change history.
  – Allow concurrent modifications to files.
  – Merge changes made by multiple developers.
  – Useful for personal files, too!
• CVS: Concurrent Versions System
  – Tried and True.
  – Not the fullest in features anymore.
  – Nice web-based interface: Bonsai.
• Other options:
  – Subversion
  – Mercurial
  – GIT
  – Bazaar
  – SVK
  – Monotone
  – darcs
  – Codeville
Bonsai “blame mode”

<table>
<thead>
<tr>
<th>Line</th>
<th>Author</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>uvcatal</td>
<td><code>uvcatal_1.198</code></td>
</tr>
<tr>
<td>74</td>
<td>egboman</td>
<td><code>egboman_1.126</code></td>
</tr>
<tr>
<td>75</td>
<td>rheapy</td>
<td><code>rheapy_1.1</code></td>
</tr>
<tr>
<td>76</td>
<td></td>
<td>Separated out the handling of isolated vertices in matching into a new routine Zoltan_PHG_match_isolated. No algorithm change.</td>
</tr>
<tr>
<td>78</td>
<td>uvcatal</td>
<td><code>uvcatal_1.13</code></td>
</tr>
<tr>
<td>80</td>
<td>rheapy</td>
<td><code>rheapy_1.45</code></td>
</tr>
<tr>
<td>81</td>
<td></td>
<td>if (!strcasestr(hgp-&gt;redm_str, &quot;no&quot;)) hgp-&gt;matching = NULL;</td>
</tr>
<tr>
<td>82</td>
<td>egboman</td>
<td><code>egboman_1.177</code></td>
</tr>
<tr>
<td>83</td>
<td></td>
<td>else if (!strcasestr(hgp-&gt;redm_str, &quot;none&quot;)) hgp-&gt;matching = NULL;</td>
</tr>
<tr>
<td>84</td>
<td>rheapy</td>
<td><code>rheapy_1.45</code></td>
</tr>
<tr>
<td>85</td>
<td></td>
<td>else if (!strcasestr(hgp-&gt;redm_str, &quot;ipm&quot;)) hgp-&gt;matching = pmatching_ipm;</td>
</tr>
<tr>
<td>86</td>
<td>lafisk</td>
<td><code>lafisk_1.209</code></td>
</tr>
<tr>
<td>87</td>
<td>uvcatal</td>
<td><code>uvcatal_1.185</code></td>
</tr>
<tr>
<td>88</td>
<td></td>
<td>else if (!strncasestr(hgp-&gt;redm_str, &quot;agg&quot;, 3)) {</td>
</tr>
<tr>
<td>89</td>
<td></td>
<td>hgp-&gt;matching = pmatching_agg_ipm;</td>
</tr>
<tr>
<td>90</td>
<td>uvcatal</td>
<td><code>uvcatal_1.166</code></td>
</tr>
<tr>
<td>91</td>
<td></td>
<td>else if (!strcasestr(hgp-&gt;redm_str, &quot;l-ipm&quot;)) hgp-&gt;matching = pmatching_local_ipm;</td>
</tr>
<tr>
<td>93</td>
<td>egboman</td>
<td><code>egboman_1.161</code></td>
</tr>
<tr>
<td>94</td>
<td>egboman</td>
<td><code>egboman_1.102</code></td>
</tr>
<tr>
<td>95</td>
<td>egboman</td>
<td><code>egboman_1.170</code></td>
</tr>
</tbody>
</table>
2. Backup System

- Make sure all systems are backed up.
  - In particular, make sure source management repository is backed up!
- *Test backups regularly.*
- Organization-wide solutions preferred.
  - Rely on expertise of system administrators.
  - Sharing backup resources can be more efficient for organization.
3. Automated Testing

• Suite of regression tests run nightly.
  – Easier to fix bugs if you catch them right away!
  – Especially important for teams.

• Example test systems:
  – ACTS (NIST)
  – DART; CTEST (Kitware)
  – Perl script and cron job.
  – FAST/Exact (SNL)

<table>
<thead>
<tr>
<th>SUMMARY: Config/Build/Test Results by Test Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>octopi.sandia.gov</td>
</tr>
<tr>
<td>software.sandia.gov</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUMMARY: Config/Build/Test Results by Test Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
</tr>
<tr>
<td>----------------------------------------</td>
</tr>
<tr>
<td>zoltan-codecheck</td>
</tr>
<tr>
<td>zoltan-octopi_CPP_v3_OMPI</td>
</tr>
<tr>
<td>zoltan-octopi_F90_v3_OMPI</td>
</tr>
<tr>
<td>zoltan-octopi_MAIN_LAMPURE</td>
</tr>
</tbody>
</table>
Unit Testing as a Way of Life

- TDD: Test Driven Design (Beck 2002; Astels 2003)
  - First, design the test; then write the software.
  - Design in small steps.
  - http://www.agiledata.org/essays/tdd.html

1. Write a test for desired functionality.
2. Run the test; make sure it fails.
3. Write code that allows test to pass.
4. Run all automated tests; make sure both old and new tests pass.
5. Clean up code; make sure all tests still pass.
4. Issue Tracking

• Maintain database of outstanding and resolved issues.
  – Includes descriptions, assignments, priority.
  – Provides persistent memory for you and/or your team.
• Bugzilla is free, searchable and easy to use.

<table>
<thead>
<tr>
<th>ID</th>
<th>Sev</th>
<th>Pri</th>
<th>OS</th>
<th>Assignee</th>
<th>Status</th>
<th>Resolution</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>3282</td>
<td>nor</td>
<td>P1</td>
<td>Linu</td>
<td><a href="mailto:zoltan-dev@software.sandia.gov">zoltan-dev@software.sandia.gov</a></td>
<td>CLOS</td>
<td>INVA</td>
<td>PHG: incorrect cutl, possibly bad partitions?</td>
</tr>
<tr>
<td>3389</td>
<td>nor</td>
<td>P1</td>
<td>Linu</td>
<td><a href="mailto:kddevin@sandia.gov">kddevin@sandia.gov</a></td>
<td>NEW</td>
<td></td>
<td>Zoltan PHG in main trunk segfaults for xyc680k with 1024 processors</td>
</tr>
<tr>
<td>3829</td>
<td>nor</td>
<td>P2</td>
<td>All</td>
<td><a href="mailto:kddevin@sandia.gov">kddevin@sandia.gov</a></td>
<td>CLOS</td>
<td>FIXE</td>
<td>dist_y initialized incorrectly for phg repartition, fast_repartition when k &lt; p</td>
</tr>
<tr>
<td>3293</td>
<td>maj</td>
<td>P1</td>
<td>All</td>
<td><a href="mailto:rheaphy@sandia.gov">rheaphy@sandia.gov</a></td>
<td>CLOS</td>
<td>FIXE</td>
<td>Many missing phg answer files</td>
</tr>
<tr>
<td>3641</td>
<td>nor</td>
<td>P1</td>
<td>Linu</td>
<td><a href="mailto:umit@bmi.osu.edu">umit@bmi.osu.edu</a></td>
<td>CLOS</td>
<td>FIXE</td>
<td>PHG identical edge removal gives ABR and FMR with OpenMPI on octopi</td>
</tr>
<tr>
<td>3973</td>
<td>nor</td>
<td>P2</td>
<td>Other</td>
<td><a href="mailto:zoltan-dev@software.sandia.gov">zoltan-dev@software.sandia.gov</a></td>
<td>NEW</td>
<td></td>
<td>PHG Build uses too many resources on RedStorm with &gt; 2K procs</td>
</tr>
<tr>
<td>3187</td>
<td>nor</td>
<td>P4</td>
<td>Other</td>
<td><a href="mailto:vjeung@sandia.gov">vjeung@sandia.gov</a></td>
<td>CLOS</td>
<td>FIXE</td>
<td>Code review for Hypergraph method PHG_REFINE</td>
</tr>
</tbody>
</table>
5. Mailing Lists

- Archive design discussions and decisions.
- Provide a “history” or “knowledge base” for the project that remains even when developers and students leave.
- Mailman is a simple option.

<table>
<thead>
<tr>
<th>Mailing List</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trilinos-Announce</td>
<td>Trilinos Announcements</td>
</tr>
<tr>
<td>Trilinos-Bugs</td>
<td>Secondary Trilinos bug reporting mechanism</td>
</tr>
<tr>
<td>Trilinos-Checkins</td>
<td>Trilinos-specific CVS commit log messages</td>
</tr>
<tr>
<td>Trilinos-Developers</td>
<td>Trilinos Development Discussions</td>
</tr>
<tr>
<td>Trilinos-Framework</td>
<td>Archive for Trilinos framework artefacts and</td>
</tr>
<tr>
<td>Trilinos-help</td>
<td>General Trilinos help address</td>
</tr>
<tr>
<td>Trilinos-Leaders</td>
<td>Trilinos Package Leaders</td>
</tr>
<tr>
<td>Trilinos-Regression</td>
<td>Captures output from general Trilinos autom</td>
</tr>
<tr>
<td>Trilinos-Users</td>
<td>Trilinos User Forum</td>
</tr>
</tbody>
</table>
6. Documentation

  - Simple examples are most requested feature.
  - Instructions for building and testing.
  - Details of usage, parameters, inputs, etc.

• Doxygen (van Heesch)
  – Automated generation of documentation.
  – C, C++, Java, Python, Fortran, PHP, C#, et al.
  – On-line hyperlinked documentation as well as off-line reference manuals.
  – Class structures, inheritance, methods, comments.
7. Release Process

- Maintain quality and stability of release.
- Can be a simple checklist.

Required Activities:

___ Update version number in source code:
  Date:                Release Version:

___ User's Manual OK for release (Date):

___ Developer's Manual OK for release (Date):

___ Final Testing completed (Date):

Test Platforms:

___ Linux            Date:                    Test Name:

___ QED cluster      Date:                    Test Name:

Test Results Archived (Where):

___ Tag code repository:
  Date:                   Tag:

___ Release package created (Date):

___ Update documentation and tar file on distribution web site.
  Date:

Checklist Complete (Date):

Decision to Release (Certification): (Who) ________, (Date) ________

Date of actual Release:
Unresolved Challenges: Post-Delivery Customer Support

• Customer Support after release.
  – Training and tutorials.
  – Answering questions (phone, email).

• Some programs (e.g., SciDAC) include outreach in funding.
  – Opportunity for impact and new collaborations.

• But after funding ends, how does customer support continue?
Unresolved Challenges: Post-Delivery Maintenance

“No good deed goes unpunished.”

• One of the biggest outstanding challenges in scientific software development.
• Research proposals focus (by necessity) on new algorithms and software development.
• How do we support software after release?
  – Porting to new architectures.
  – Bug fixes and minor enhancements.
• Currently, developers “donate” support time because they care about the project.
  – Not sustainable long term.
• Funding agencies need to develop maintenance strategy.
  – Who pays for it? Who does it?
  – How do we evaluate and reward those who do it?
For More Information…

• SIAM AN08 Minisymposium: 10:30am TODAY
  – Software Engineering Challenges in Scientific Computing
  – Mike Heroux, Sandia National Laboratories
  – Tim Davis, University of Florida
  – Penny Anderson, Mathworks
  – Thomas Tysinger, ANSYS, Inc.

• Trilinos web page: http://trilinos.sandia.gov
• ITAPS web page: http://www.itaps-scidac.org
• CSCAPES web page: http://www.cscapes.org
Thanks

SciDAC CSCAPES Institute (A. Pothen, Purdue U., PI)
SciDAC ITAPS Center (L. Diachin, LLNL, PI)
NNSA ASC Program

• S. Attaway (SNL)
• R. Bisseling (Utrecht U.)
• D. Bozdag (Ohio St. U.)
• C. Chevalier (SNL)
• T. Davis (U. Florida)
• J. Flaherty (RPI)
• W. Hart (SNL)
• B. Hendrickson (SNL)
• D. Keyes (Columbia)
• K. Ko (SLAC)
• T. Kolda (SNL)
• G. Kumfert (LLNL)

• L.-Q. Lee (SLAC)
• V. Leung (SNL)
• G. Lonsdale (NEC)
• X. Luo (RPI)
• W. Mitchell (NIST)
• L. Musson (SNL)
• S. Plimpton (SNL)
• L.A. Riesen (SNL)
• J. Shadid (SNL)
• M. Shephard (RPI)
• J. Teresco (Mount Holyoke)
• C. Vaughan (SNL)
• M. Wolf (U. Illinois)