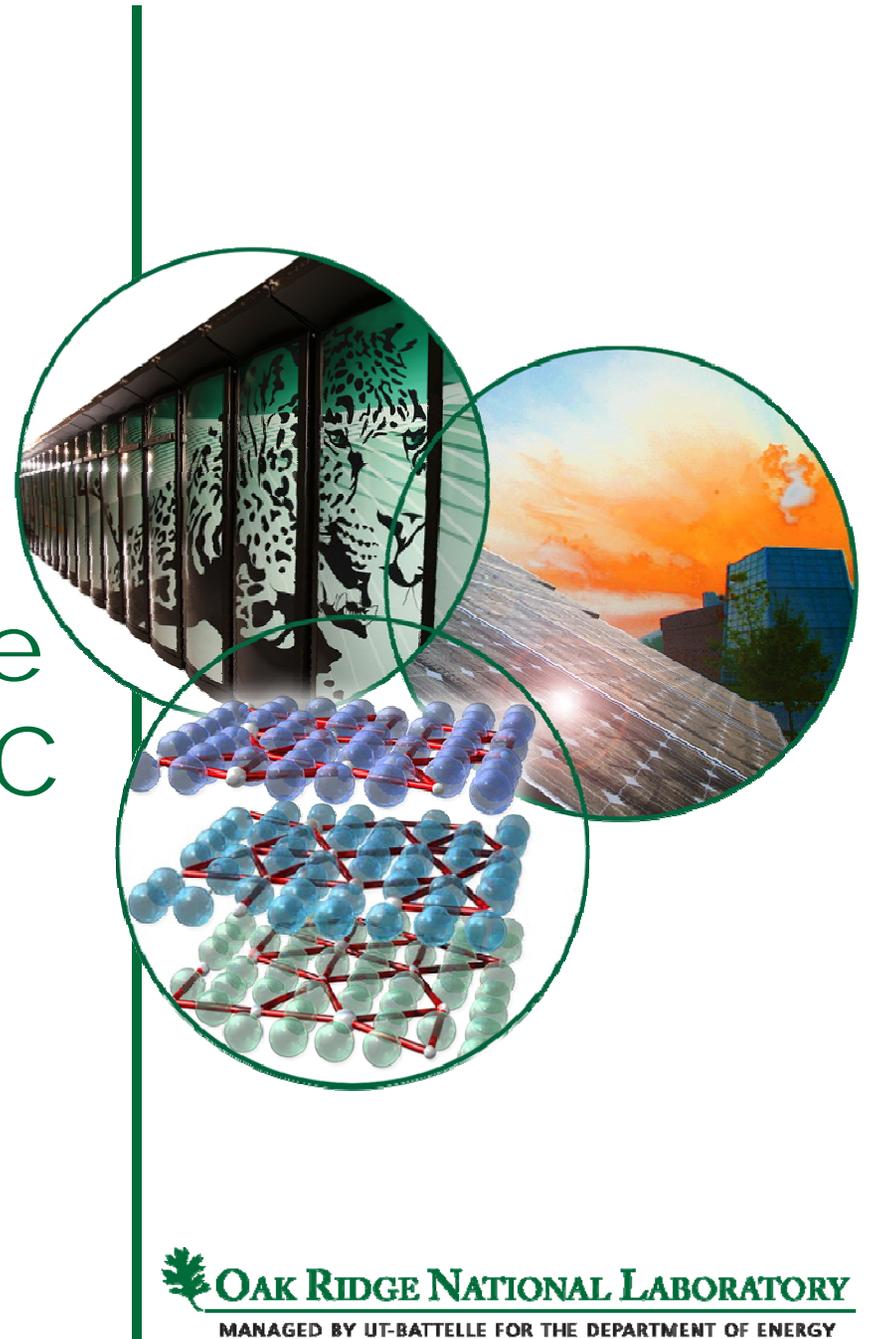


Repeating Our Mistakes? Software Engineering for HPC

David E. Bernholdt

Oak Ridge National Laboratory

bernholdtde@ornl.gov



Observations on the Sociology of Computational Science

- Most computational scientists learn computing from mentors and peers within their disciplines
 - Little or no formal training in computer science
 - Informal “self-study” of computer science is spotty
- Reward structure favors communicating **scientific** results
 - Not the **software** that enabled the science
 - Worse: the software gives competitive advantage!
 - Even less the **engineering** that enabled the software
- *Silos of excellence* from a domain perspective
 - Horizontal knowledge transfer diffusion is much slower



Observations on Software Engineering in Computational Science

- Many think software engineering techniques aren't applicable to research (software)
- Many think their code is small and simple, and doesn't need "fancy" engineering
- Many researchers have a short term perspective on their software
 - Probably commensurate with reward system
 - Software engineering is often a long-term investment – up-front costs for often nebulous future payoffs
- Many see software engineering as something that takes time and effort away from scientific progress

Is Software Engineering Really So Bad?

- Software engineering is ... dedicated to designing, implementing, and modifying software so that it is of **higher quality, more affordable, maintainable, and faster to build.**
 - *Wikipedia [emphasis mine]*
- The application of a **systematic, disciplined, quantifiable** approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
 - *Guide to the Software Engineering Body of Knowledge, IEEE 2001 [emphasis mine]*

Why the Disconnect?

- Lack of understanding of software engineering by computational scientists
- Lack of understanding of computational science by software engineers
- Software engineering for the wrong reasons
- The wrong software engineering techniques
- Software engineering has matured significantly
- *Those who cannot remember the past are condemned to repeat it.*
 - *George Santayana*

focus
developing scientific software.....

Understanding the High-Performance-Computing Community: A Software Engineer's Perspective

Victor R. Bacili and Daniela Cruzes, *University of Maryland, College Park, and Fraunhofer Center for Experimental Software Engineering-Maryland*

Jeffrey C. Carver, *Mississippi State University*

Lorin M. Hochstein, *University of Nebraska-Lincoln*

Jeffrey K. Hollingsworth and Marvin V. Zelkowitz, *University of Maryland, College Park*

Forrest Shull, *Fraunhofer Center for Experimental Software Engineering-Maryland*

Computational scientists developing software for HPC systems face unique software engineering issues. Attempts to transfer SE technologies to this domain must take these issues into account.

For the past few years, we've had the opportunity, as software engineers, to observe the development of computational-science software (called *codes*) built for high-performance-computing (HPC) machines in many different contexts. Although we haven't studied all types of HPC development, we've encountered a wide cross-section of projects. Despite these projects' diversity, several common traits exist:

- Many developers receive their software training from other scientists. Although the scientists often have been writing software for many years, they generally lack formal software engineering (SE) training, especially in managing multiperson development teams and complex software artifacts.
- Many of the codes aren't initially designed to be large. They start small and then grow on the basis of their scientific success.
- Many development teams use their own code (or code developed as part of their research group).

For these reasons (and many others), development

practices in this community differ considerably from those in more "traditional" SE.

We aim here to distill our experience about how software engineers can productively engage the HPC community. Several SE practices generally considered good ideas in other development environments are quite mismatched to the HPC community's needs. For SE researchers, the keys to successful interactions include a healthy sense of humility and the avoidance of assumptions that SE expertise applies equally in all contexts.

Background

A list of the 500 fastest supercomputers (www.top500.org) shows that, as of November 2007, the

And Can We Recover From It?

What Can Software Engineering Do For Computational Science?

Who's Doing It Better?

Is Research Needed in SE for HPC and Computational Science?

Configure & Build is Hard! But That Doesn't Mean We Can't Do Better (If We Try)!

make Apache Buildr Deb Creator OpenMake Meister
Autoconf Apache Maven Debian Package Maker OMake
Automake A.A.P. ElectricCommander Derferec Jam
CMake
Scmake
Apacemaker
Clea
r
nm
R:
MPW
Antr

The screenshot shows the Freshmeat website interface. At the top, there are navigation links: Login, Signup, Lost password?. Below is the Freshmeat logo and a search bar. A red circle highlights the text "510 projects tagged 'Build Tools'". Below this, there are search results for projects tagged "Build Tools".

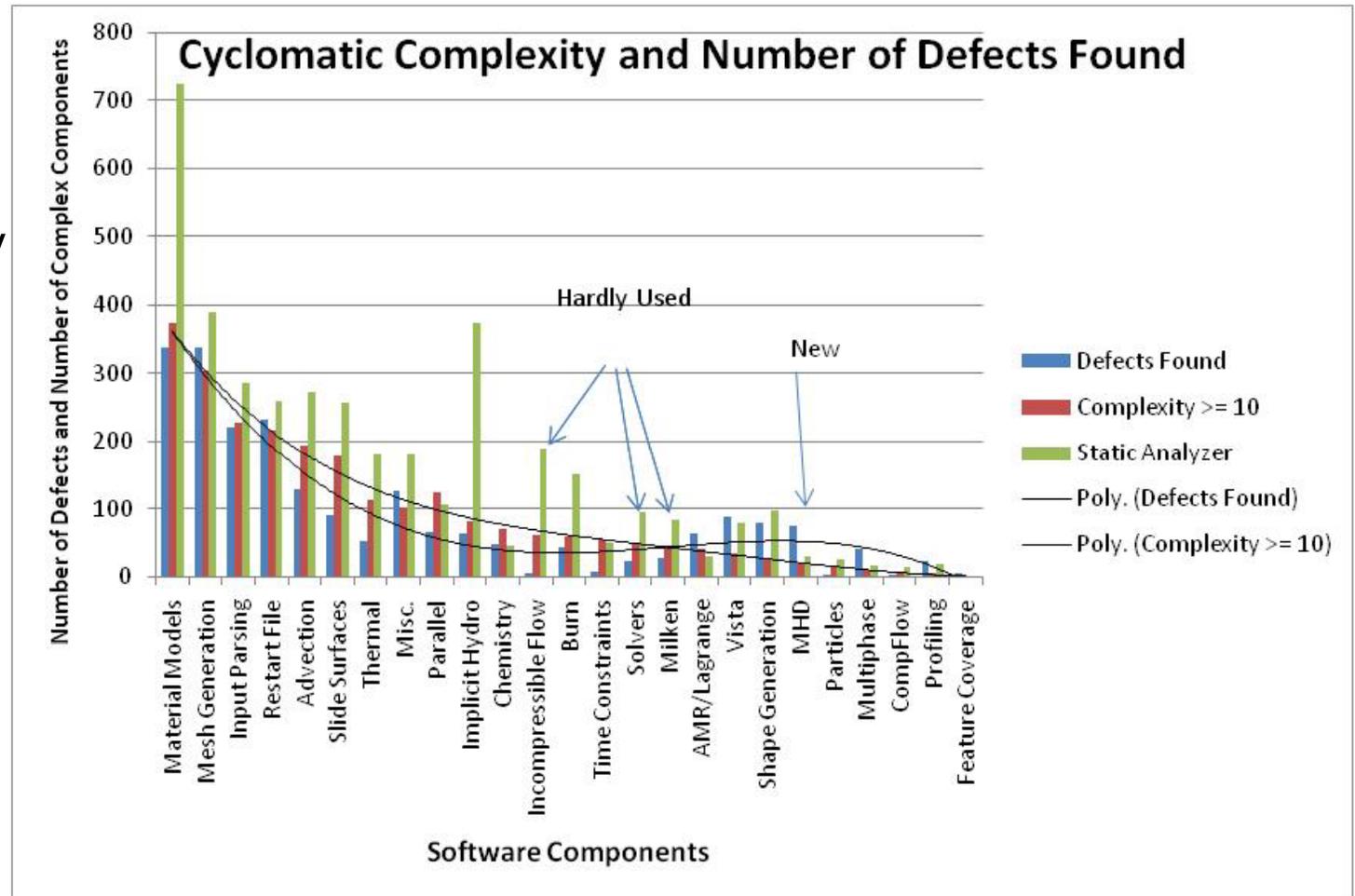
Project Name	Popularity	Vitality
Aegis Aegis is a transaction-based software configuration management system. It provides a framework within which a team of developers may work on many changes to a program independently, and Aegis coordinates integrating these changes back into the master source of the program, with as little disruption as possible. Aegis supports geographically distributed development. Tags: GPL, Software Development, Testing, Build Tools, Version Control Updated 09 Mar 2008 Pop: 278.42, Vit: 13.35		
AMC (ATOM Module Compiler) AMC is a programmable compiler/preprocessor. It has a built-in programming language called CGL (Code Generation Language) that lets you add new syntactical elements to the source files that AMC processes. In addition, AMC has a module structure reminiscent of the UCSD p-System compiler. AMC comes with a default package that adds a dynamic form of OOP to C. Tags: Software Development, Code Generators, Build Tools, Compilers Updated 25 Jul 2001 Pop: 40.15, Vit: 2.20		
Apache Toolbox Apache Toolbox provides a means to easily compile Apache (IPv4/6) SSL, PHP(v3/v4), MySQL, Updated 07 Oct 2004		

On the right side of the screenshot, there are sections for Tags, Licenses, and Programming languages.

- Tags:** Build Tools (510), Software Develo... (502), Libraries (63), Code Generators (62), Utilities (61), Show all 50
- Licenses:** GPL (200), LGPL (48), BSD Revised (31), Apache 2.0 (29), MITX (27), Show all 32
- Programming languages:** Java (167), C (98), C++ (80), Python (52)

Five Year Study - Complexity and Defects in R&D Codes

- Code complexity correlates strongly with actual defects, static analyzer issues

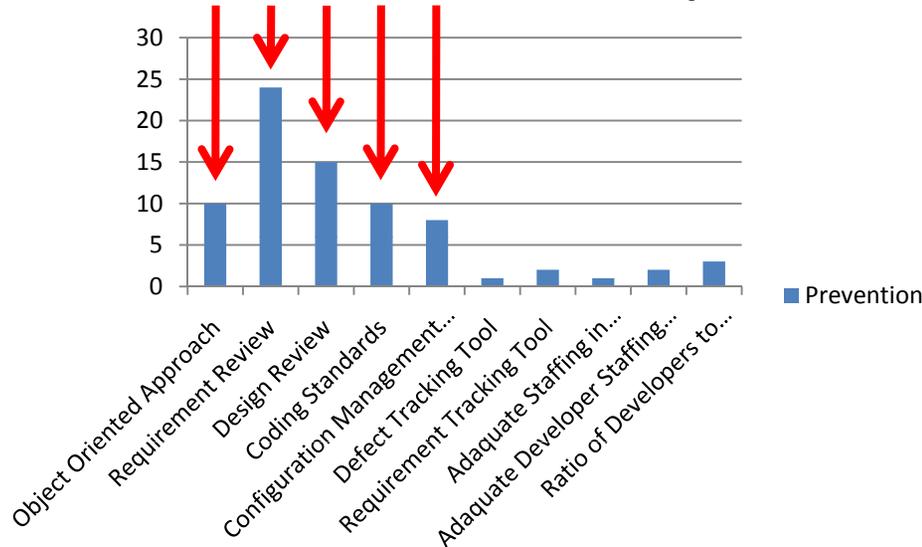


Courtesy of Greg Pope (LLNL)



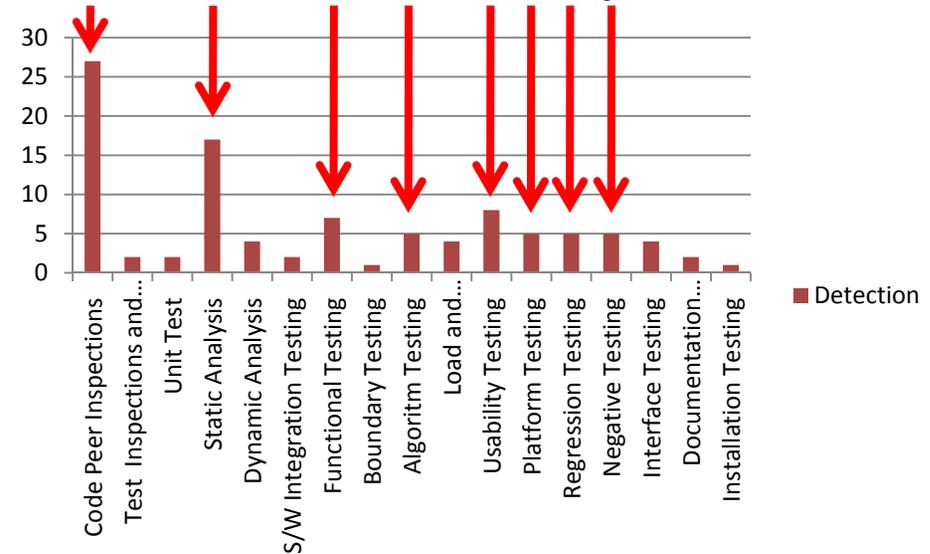
Techniques for Detecting and Preventing Bad Code

Prevention Techniques



- Requirements review
- Design review
- OO programming
- Coding standards
- Configuration management

Detection Techniques

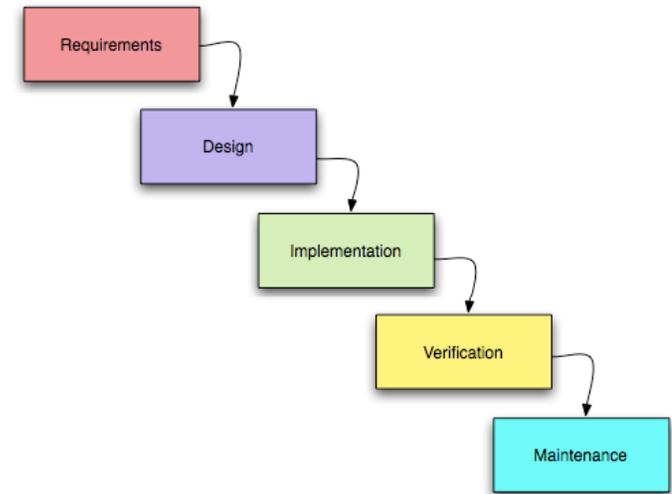


- Code inspections
- Static analysis
- Usability testing
- Functional testing
- Algorithm, platform, regression, negative testing

Courtesy of Greg Pope (LLNL)

Development Processes

- The software engineering community has gotten a lot more sophisticated about development processes
- Common in industry to have distributed teams developing large, complex software systems with changing requirements
- Active area of research
 - Software, people, management
- New approaches often over-hyped



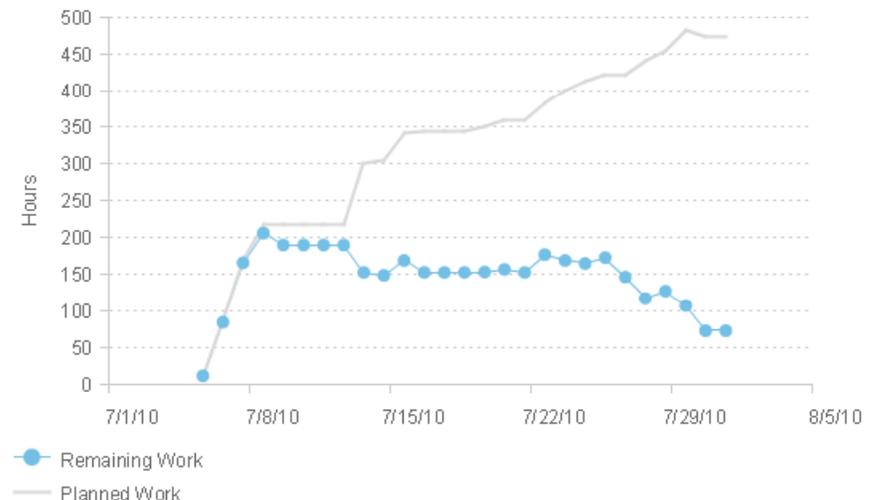
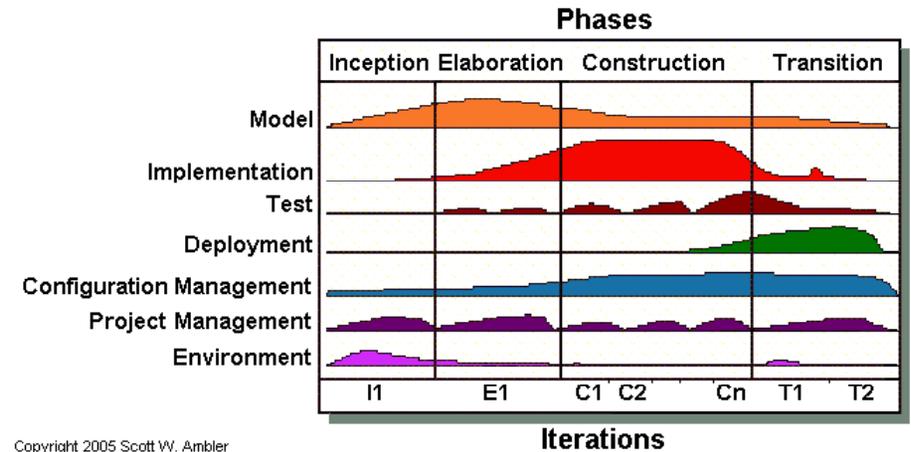
Classic “Waterfall” Development Process

Some current processes:

- SCRUM, Agile, XP, Kanban, TDD, FDD, RUP, ...

Agile Model Driven Design (AMDD)

- Combination of Agile & Rational Unified Process
- Model driven (UML, not M&S)
- Incorporates Test-Driven Development
- Development is iterative
- Releases at the end of each iteration
- Delivers functionality incrementally, which allows requirements to change



Burndown chart for July 2010 iteration

Courtesy of Jay Billings (ORNL)

Predictive Capability Maturity Model (for Computational M&S)

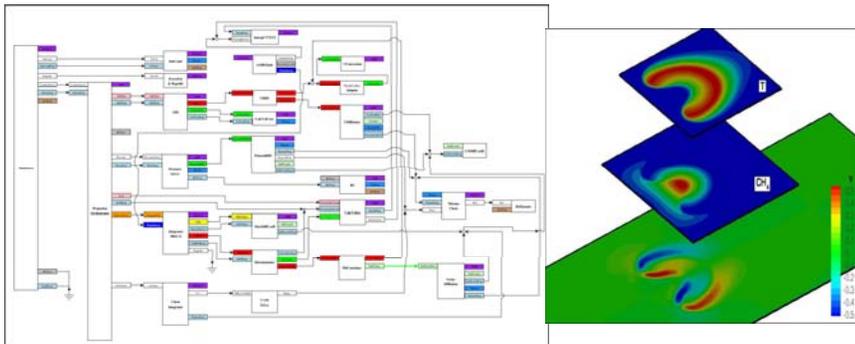
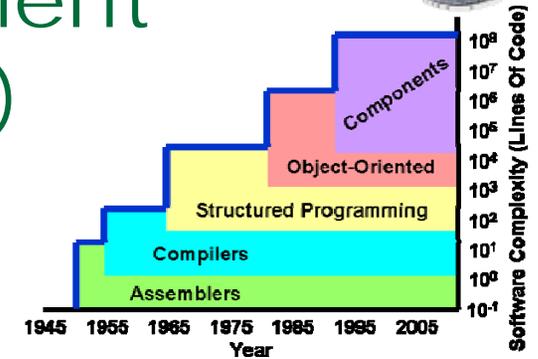
- A tool for assessing and communicating progress in predictive capability
 - Help organize and systematize VV & UQ
- PCMM components:
 - M&S elements
 - Maturity levels
 - Assessment criteria
- It is *application specific*
- See report SAND2007-5948 by Oberkampff, Pilch, and Trucano

MATURITY \ ELEMENT	Maturity Level 0 Low Consequence, Minimal M&S Impact, e.g. Scoping Studies	Maturity Level 1 Moderate Consequence, Some M&S Impact, e.g. Design Support	Maturity Level 2 High-Consequence, High M&S Impact, e.g. Qualification Support	Maturity Level 3 High-Consequence, Decision-Making Based on M&S, e.g. Qualification or Certification
Representation and Geometric Fidelity What features are neglected because of simplifications or stylizations?	<ul style="list-style-type: none"> • Judgment only • Little or no representational or geometric fidelity for the system and BCs 	<ul style="list-style-type: none"> • Significant simplification or stylization of the system and BCs • Geometry or representation of major components is defined 	<ul style="list-style-type: none"> • Limited simplification or stylization of major components and BCs • Geometry or representation is well defined for major components and some minor components • Some peer review conducted 	<ul style="list-style-type: none"> • Essentially no simplification or stylization of components in the system and BCs • Geometry or representation of all components is at the detail of "as built", e.g., gaps, material interfaces, fasteners • Independent peer review conducted
Physics and Material Model Fidelity How fundamental are the physics and material models and what is the level of model calibration?	<ul style="list-style-type: none"> • Judgment only • Model forms are either unknown or fully empirical • Few, if any, physics-informed models • No coupling of models 	<ul style="list-style-type: none"> • Some models are physics based and are calibrated using data from related systems • Minimal or ad hoc coupling of models 	<ul style="list-style-type: none"> • Physics-based models for all important processes • Significant calibration needed using separate effects tests (SETs) and integral effects tests (IETs) • One-way coupling of models • Some peer review conducted 	<ul style="list-style-type: none"> • All models are physics based • Minimal need for calibration using SETs and IETs • Sound physical basis for extrapolation and coupling of models • Full, two-way coupling of models • Independent peer review conducted
Code Verification Are algorithm deficiencies, software errors, and poor SQE practices corrupting the simulation results?	<ul style="list-style-type: none"> • Judgment only • Minimal testing of any software elements • Little or no SQE procedures specified or followed 	<ul style="list-style-type: none"> • Code is managed by SQE procedures • Unit and regression testing conducted • Some comparisons made with benchmarks 	<ul style="list-style-type: none"> • Some algorithms are tested to determine the observed order of numerical convergence • Some features & capabilities (F&C) are tested with benchmark solutions • Some peer review conducted 	<ul style="list-style-type: none"> • All important algorithms are tested to determine the observed order of numerical convergence • All important F&Cs are tested with rigorous benchmark solutions • Independent peer review conducted
Solution Verification Are numerical solution errors and human procedural errors corrupting the simulation results?	<ul style="list-style-type: none"> • Judgment only • Numerical errors have an unknown or large effect on simulation results 	<ul style="list-style-type: none"> • Numerical effects on relevant SRQs are qualitatively estimated • Input/output (I/O) verified only by the analysts 	<ul style="list-style-type: none"> • Numerical effects are quantitatively estimated to be small on some SRQs • I/O independently verified • Some peer review conducted 	<ul style="list-style-type: none"> • Numerical effects are determined to be small on all important SRQs • Important simulations are independently reproduced • Independent peer review conducted
Model Validation How carefully is the accuracy of the simulation and experimental results assessed at various tiers in a validation hierarchy?	<ul style="list-style-type: none"> • Judgment only • Few, if any, comparisons with measurements from similar systems or applications 	<ul style="list-style-type: none"> • Quantitative assessment of accuracy of SRQs not directly relevant to the application of interest • Large or unknown experimental uncertainties 	<ul style="list-style-type: none"> • Quantitative assessment of predictive accuracy for some key SRQs from IETs and SETs • Experimental uncertainties are well characterized for most SETs, but poorly known for IETs • Some peer review conducted 	<ul style="list-style-type: none"> • Quantitative assessment of predictive accuracy for all important SRQs from IETs and SETs at conditions/geometries directly relevant to the application • Experimental uncertainties are well characterized for all IETs and SETs • Independent peer review conducted
Uncertainty Quantification and Sensitivity Analysis How thoroughly are uncertainties and sensitivities characterized and propagated?	<ul style="list-style-type: none"> • Judgment only • Only deterministic analyses are conducted • Uncertainties and sensitivities are not addressed 	<ul style="list-style-type: none"> • Aleatory and epistemic (A&E) uncertainties propagated, but without distinction • Informal sensitivity studies conducted • Many strong UQ/SA assumptions made 	<ul style="list-style-type: none"> • A&E uncertainties segregated, propagated and identified in SRQs • Quantitative sensitivity analyses conducted for most parameters • Numerical propagation errors are estimated and their effect known • Some strong assumptions made • Some peer review conducted 	<ul style="list-style-type: none"> • A&E uncertainties comprehensively treated and properly interpreted • Comprehensive sensitivity analyses conducted for parameters and models • Numerical propagation errors are demonstrated to be small • No significant UQ/SA assumptions made • Independent peer review conducted

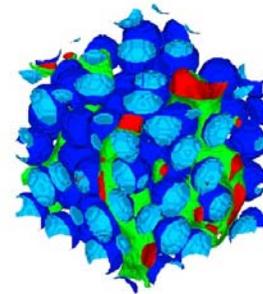
Courtesy of Angel Urbina (SNL)

The Common Component Architecture (CCA)

- Grassroots effort, started in 1998
 - Lab and university application and computer scientists
- Bring **component-based software development** to HPC for computational science and engineering
- Increase awareness of **software architecture** and how it effects the entire software lifecycle



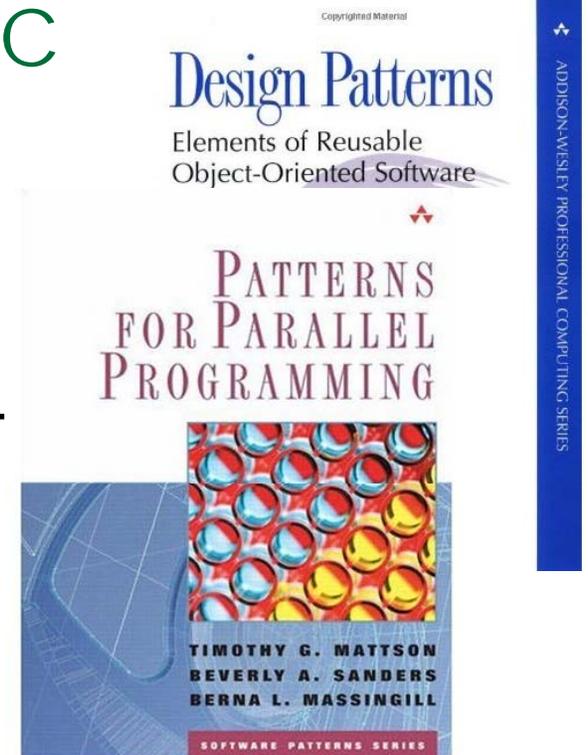
CCA-based combustion application “wiring diagram” and results. *Courtesy Cosmin Safta, (SNL)*



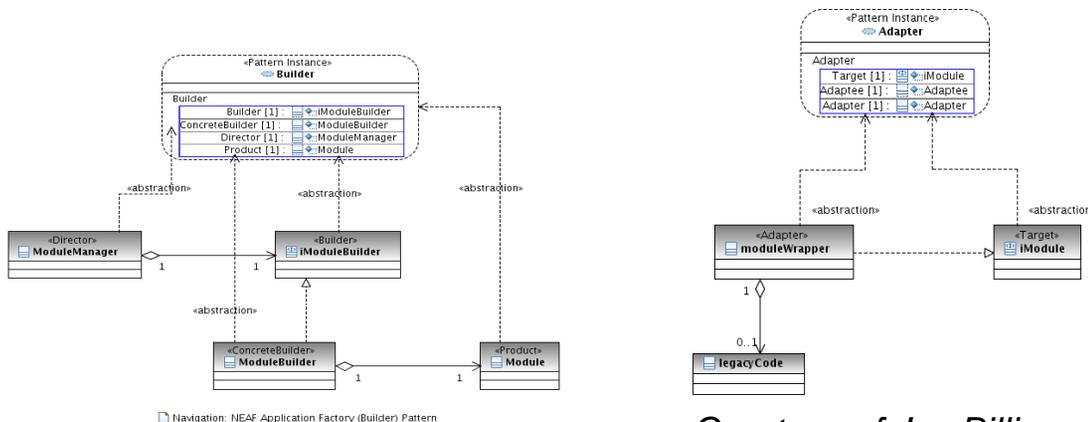
GWACCAMOLE smooth-particle hydrodynamics subsurface modeling code and results. *Courtesy Bruce Palmer (PNNL)*

Patterns for HPC

- Patterns are **general, reusable** solutions to commonly occurring problems in software development
- Patterns for architecture, design, algorithms, ...
- Templates, not finished code
- Pattern languages
- Used extensively in industry
- Little used in HPC

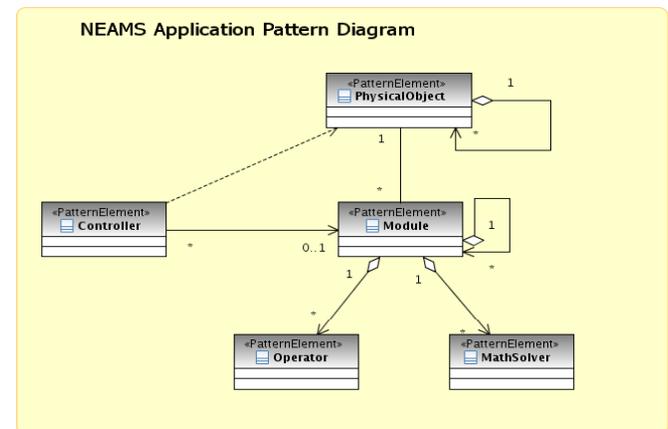


NEAMS Module Manager (Builder) Diagram



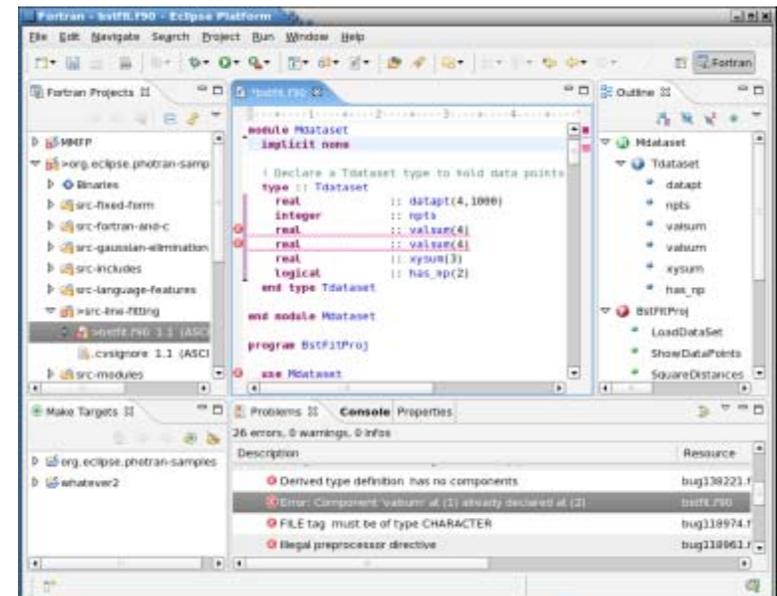
Navigation: NEAF Application Factory (Builder) Pattern

Courtesy of Jay Billings (ORNL)



Refactoring

- Code refactoring is the process of changing a computer program's source code **without modifying its external functional behavior** in order to improve some of the nonfunctional attributes of the software
 - Improve readability, reduce complexity, restructure architecture or object model
- Refactoring tools widely used in industry
- In HPC refactoring is usually done manually or sed scripts
 - Intimidating, so rarely done
 - Error prone
- Photran refactoring for Fortran
 - Rename, Encapsulate variable, Interchange loops, Introduce Implicit None, Move Saved Variables to Common Block, Replace Obsolete Operators, Standardize Statements, Remove Unused Variables, Data to Parameter, Extract Procedure, Extract Local Variable, Canonicalize Keyword Capitalization, Make COMMON Variable Names Consistent, Add ONLY Clause to USE Statement, Minimize ONLY List, Make Private Entity Public

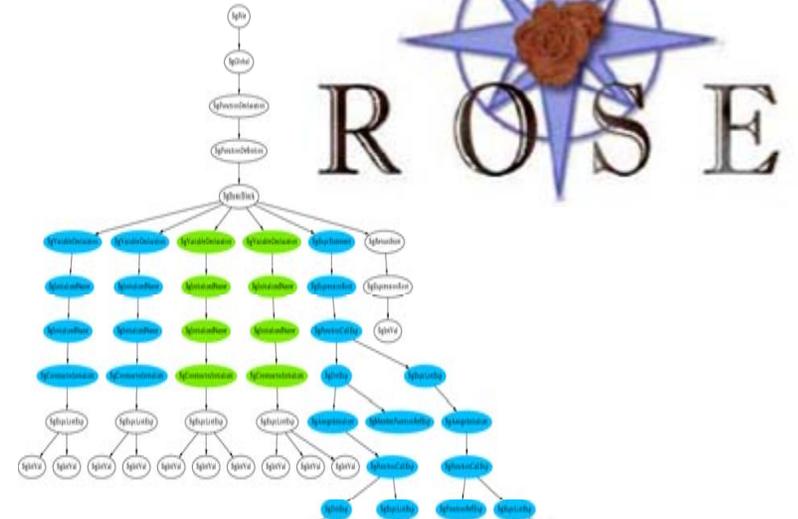


Photran: an IDE and refactoring tool for Fortran. Part of the Eclipse Parallel Tools Platform project.
 From www.eclipse.org/photran/

Behind the Scenes of Refactoring Tools

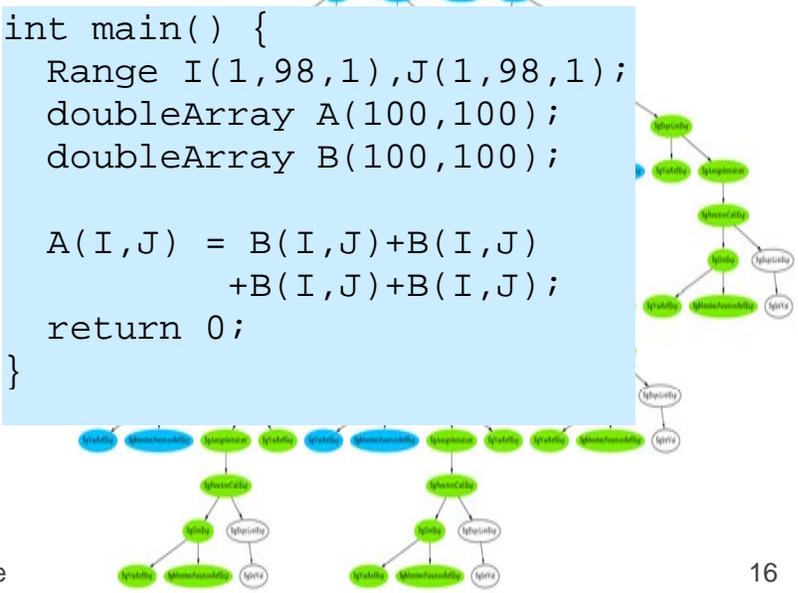


- Refactoring is a source-to-source (s2s) transformation of the source code
- Like a compiler, but...
 - Refactoring transformations rather than optimizations
 - Emits code in a programming language instead of object code
- ROSE is a compiler infrastructure designed for s2s
 - Led by Dan Quinlan (LLNL), DOE supported
 - Primary use is for performance optimizations
 - Also used for automatic differentiation
 - Writing transformations is hard



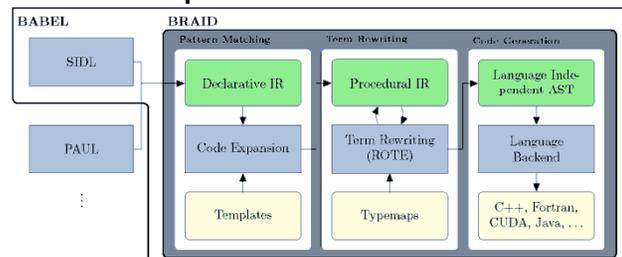
```
int main() {
    Range I(1,98,1),J(1,98,1);
    doubleArray A(100,100);
    doubleArray B(100,100);

    A(I,J) = B(I,J)+B(I,J)
            +B(I,J)+B(I,J);
    return 0;
}
```



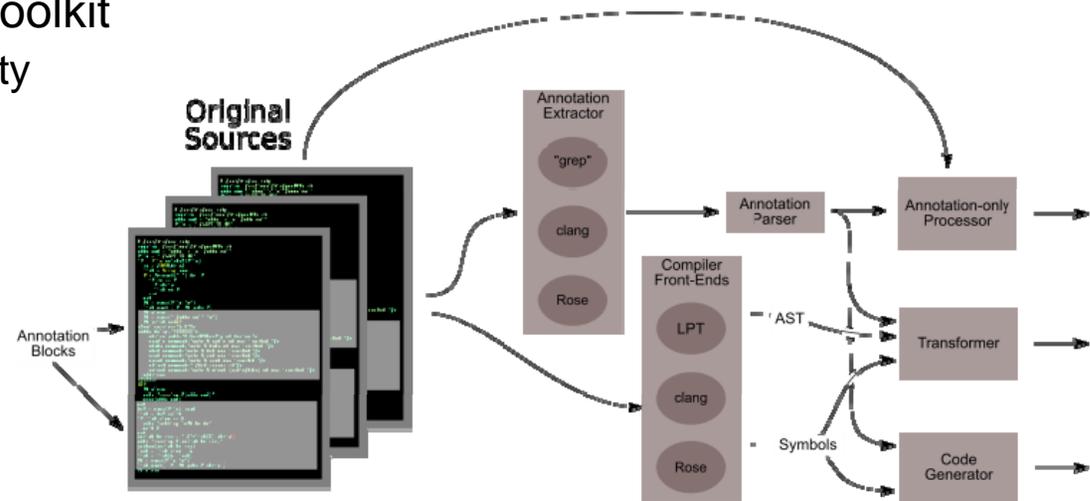
COMPOSE-HPC

- Recent X-Stack award
 - Galois, LLNL, ORNL, PNNL, SNL
- Facilitate composition of software (in many forms)
 - And related challenges (i.e. refactoring)
- Building tools for annotation languages, s2s transformation, and code generation
 - KNOT: Nimble Orchestration Toolkit
 - PAUL: annotation parsing facility
 - ROTE: Retargettable Open Transformation Engine
 - BRAID: code generation and optimization



Examples of compositions of interest...

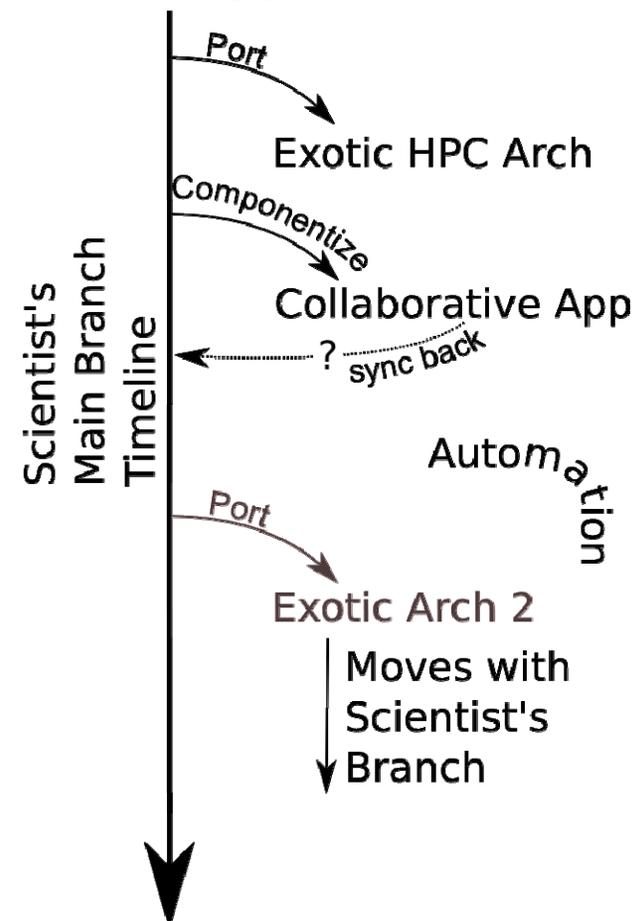
- Performance instrumentation
- CPUs and accelerators (i.e. GPUs)
- Verification and trust (contracts)
- Programming languages
- Threads
- Concurrency



A Long-Term Vision for Software Engineering in a Rapidly Changing World

- HPC application developers are facing a period of significant uncertainties and rapid changes in the underlying hardware
- How are they going to cope?
- Software engineering tools are going to be central to maintaining developer sanity
 - Systematize, automate, check, transform

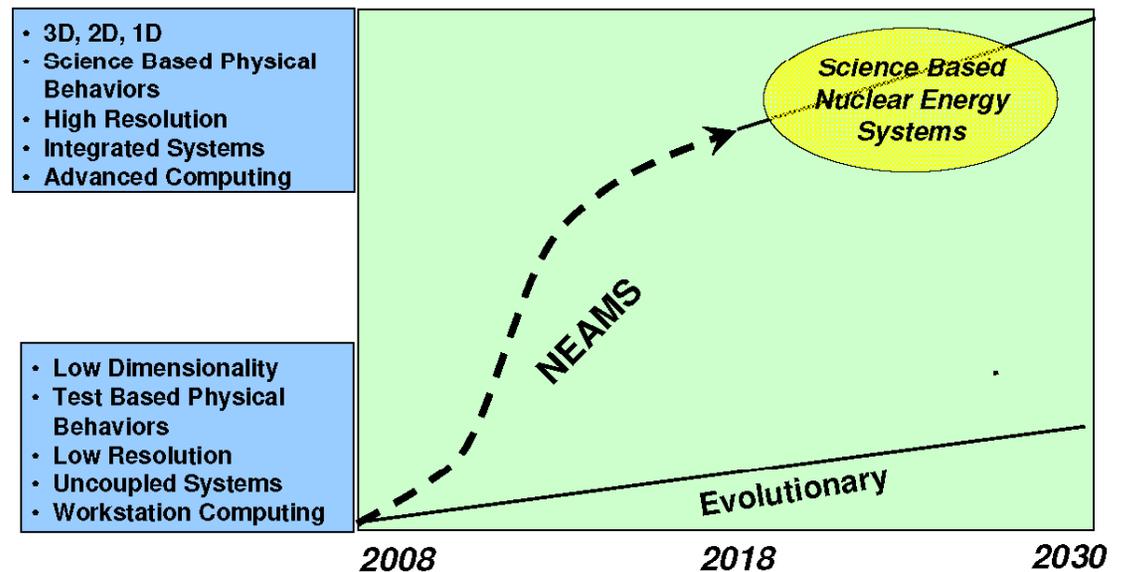
Science App Timeline



Courtesy of Rob Armstrong (SNL)

Nuclear Energy Advanced Modeling and Simulation (NEAMS)

- Program of the Office of Nuclear Energy, Advanced Modeling and Simulation Office
- NEAMS will produce truly predictive simulation tools to accelerate growth of the nuclear enterprise in ways that are too costly or time consuming to achieve by experimentation alone
- VV&UQ, and software engineering built into the program



Conclusions

- There is a long-standing disconnect between computational science and software engineering
- ASCI and NEAMS are examples of large-scale research software efforts which are benefitting from software engineering
 - Software engineering built into the programs!
- More software engineering research is needed to support HPC and computational science
 - Currently spotty, many gaps, lacks critical mass
- Are we repeating our mistakes?
 - Yes
- Do we have to continue repeating them?
 - No

Acknowledgements

- Angel Urbina, SNL
- Greg Pope, LLNL
- Jay Billings, ORNL
- Cosmin Safta, SNL
- Bruce Palmer, PNNL
- Rob Armstrong, SNL
- *Apologies to relevant projects not cited!*

Supported by...



X-Stack Research