Exceptional service in the national interest

Sandia National Laboratories

Improving Scientific Software Productivity: The IDEAS Project

IDEAS productivity

Michael A. Heroux
Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Sandia National Laboratories

THE HPC SOFTWARE ENGINEERING "CRISIS".

A Confluence of Trends

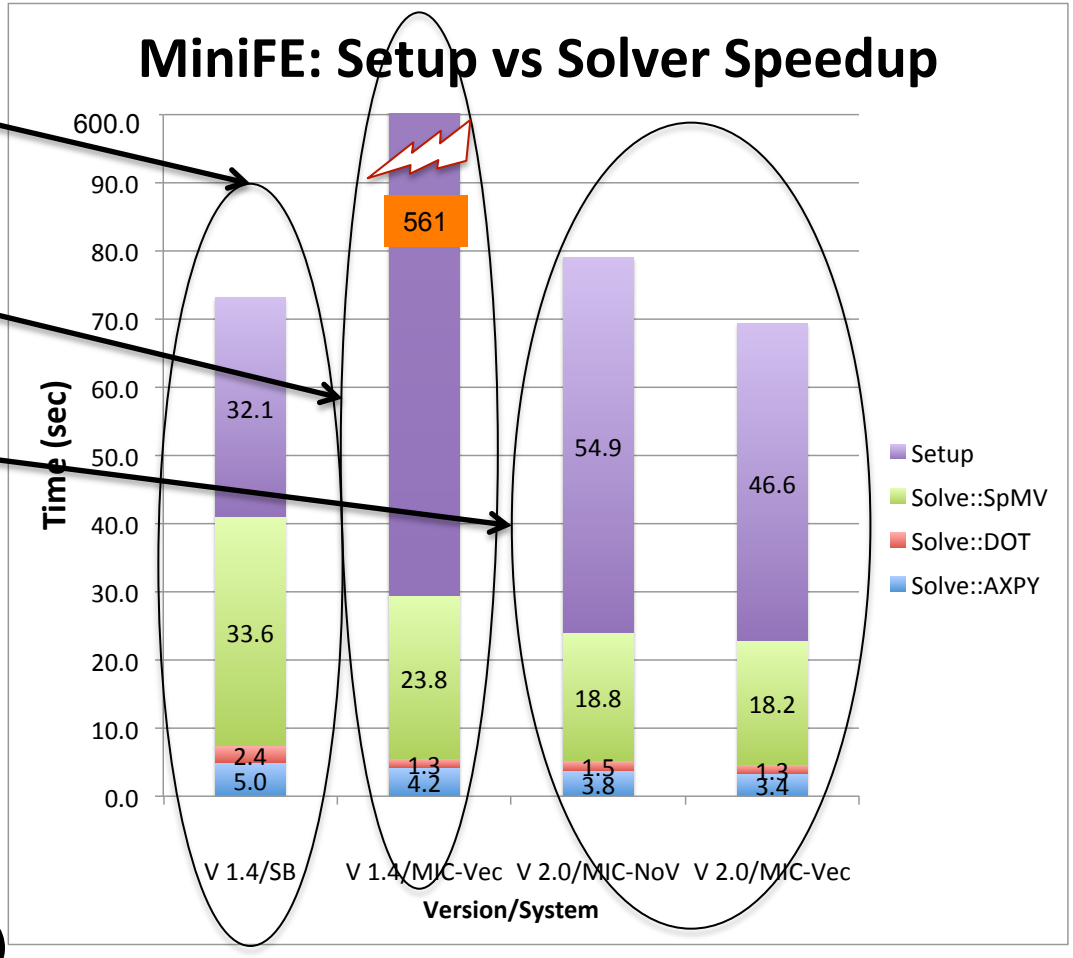
Sandia National Laboratories

- Fundamental trends:
  - Disruptive HW changes: Requires thorough alg/code refactoring.
  - Demands for coupling: Multiphysics, multiscale.
- Challenges:
  - Need 2 refactorings: 1+ε, not 2-ε. Really: Continuous change.
  - Modest app development funding: No monolithic apps.
  - Requirements are unfolding, evolving, not fully known *a priori*.
- Opportunities:
  - Better design and SW practices & tools are available.
  - Better SW architectures: Toolkits, libraries, frameworks.
- Basic strategy: Focus on productivity.

The work ahead of us: Threads and vectors

MiniFE 1.4 vs 2.0 as Harbingers

- Typical MPI-only run:
  - Balanced setup vs solve
- First MIC run:
  - Thread/vector solver
  - No-thread setup
- V 2.0: Thread/vector
  - Lots of work:
    - Data placement, const /restrict declarations, avoid shared writes, find race conditions, …
  - Unique to each app
- Opportunity: Look for new crosscutting patterns, libraries (e.g., libs of data containers)



Software Engineering and HPC: Efficiency vs Other Quality Metrics

Sandia National Laboratories

How focusing on the factor below affects the factor to the right	Correctness	Usability	Efficiency	Reliability	Integrity	Adaptability	Accuracy	Robustness
Correctness	↑		↑	↑			↑	↓
Usability		↑				↑	↑	
Efficiency	↓		↑	↓	↓	↓	↓	
Reliability	↑			↑	↑		↑	↓
Integrity			↓	↑	↑			
Adaptability					↓	↑		↑
Accuracy	↑		↓	↑		↓	↑	↓
Robustness	↓	↑	↓	↓	↓	↑	↓	↑

Source:
Code Complete
Steve McConnell

Helps it ↑

Hurts it ↓

Better, Faster, Cheaper: Pick 2 of the 3

Sandia National Laboratories

- Scenario:
  - You run a software company with 30 employees, 15 of whom are software engineers.
  - Among other products, you sell a computational mechanics FEM application to small businesses that enables them to prototype designs. 6 SW engineers work on the product.
  - The next release date for your customer portal app is Dec 1, 2015, but your team is running 2 weeks behind on its upgrade of the mesh quality improvement feature.
- What are your options to address the situation?

Must give up at least one thing.

Sandia National Laboratories

- ~~Better~~: Drop the mesh quality improvement feature set.
  - When would this be the best option?
  - When would you avoid this option?
- Or ~~Faster~~: Delay the product release by two weeks (or more?).
  - When would this be best?
  - When to avoid?
- Or ~~Cheaper~~: Assign additional engineers to the project (higher cost).
  - When is this best or not a good idea?

Sandia National Laboratories

If I had eight hours to chop down a tree, I would spend six sharpening my axe.

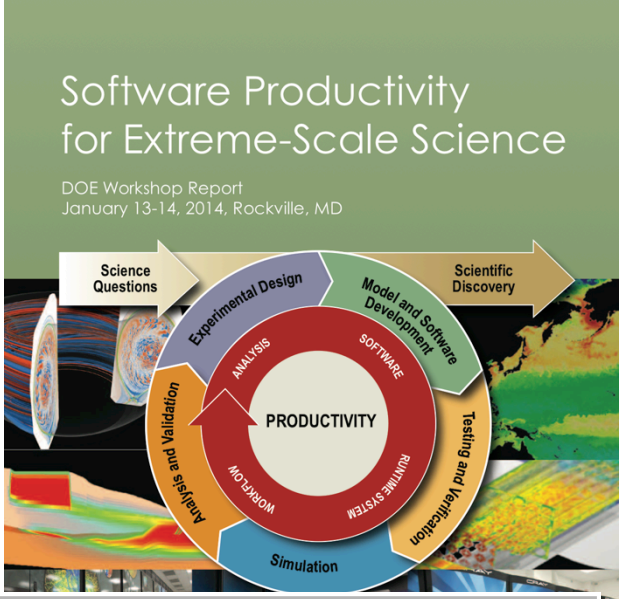
- Abraham Lincoln

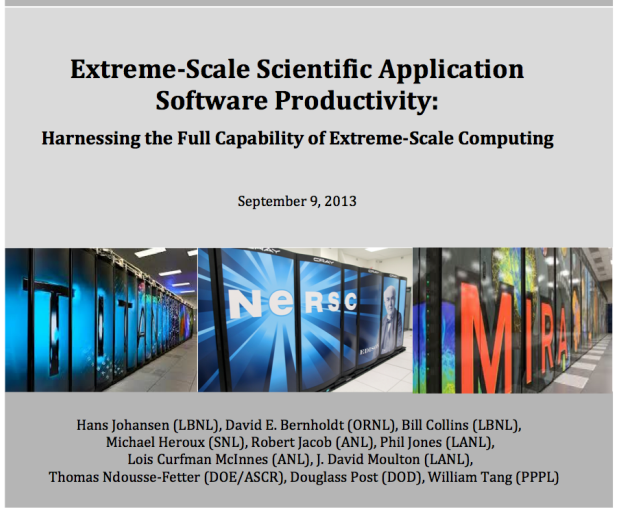
PRODUCTIVITY

BETTER, FASTER, CHEAPER: PICK ALL THREE

Productivity Emphasis

- *Scientific* Productivity.
- Many design choices ahead.
- Productivity emphasis:
  - Metrics.
  - Design choice process.
- Software ecosystems: Rational option
  - Not enough time to build monolithic.
  - Too many requirements.
  - Not enough funding.
- Focus on actionable productivity metrics.
  - Optometrist model: which is better?
  - Global model: For “paradigm shifts”.





Sandia National Laboratories

IDEAS: A NEW DOE PRODUCTIVITY-FOCUSED PROJECT

Institutional Leads (Pictured)
Full Team List

IDEAS productivity

Sandia National Laboratories

Science Use Cases

J. David Moulton

Tim Scheibe

Carl Steefel

Glenn Hammond

Reed Maxwell

Scott Painter

Ethan Coon

Xiaofan Yang

Project Leads

ASCR: M. Heroux and L.C. McInnes

BER: J. D. Moulton

Extreme-Scale Scientific Software Development Kit (xSDK)

Mike Heroux

Ulrike Meier Yang

Jed Brown

Irina Demeshko

Kirsten Kleese van Dam

Sherry Li

Daniel Osei-Kuffuor

Vijay Mahadevan

Barry Smith

Hans Johansen

Lois Curfman McInnes

Ross Bartlett

Todd Gamblin*

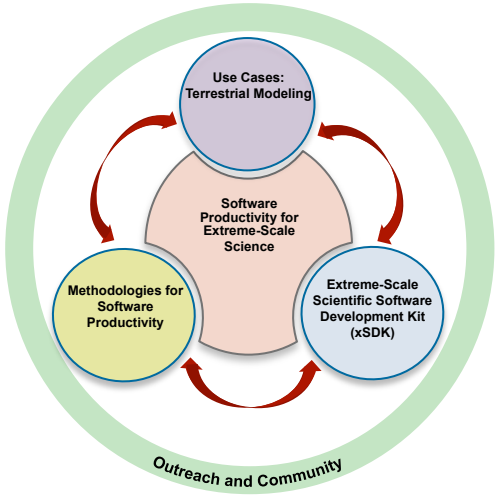
Andy Salinger*

Jason Sarich

Jim Willenbring

Pat McCormick

Methodologies for Software Productivity



Outreach

David Bernholdt

Katie Antypas*

Lisa Childers*

Judith Hill*

*Liaison

Interoperable Design of Extreme-scale Application Software (IDEAS)

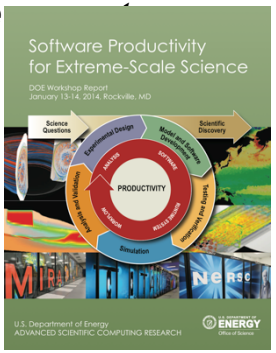
Motivation

Enable ***increased scientific productivity,*** realizing the potential of extreme- scale computing, through ***a new interdisciplinary and agile approach to the scientific software***

Objectives

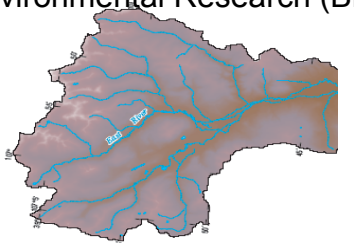
Address confluence of trends in hardware and increasing demands for predictive multiscale, multiphysics simulations.

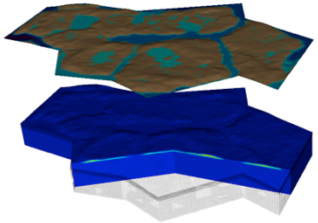
Respond to trend of continuous refactoring with efficient agile software engineering methodologies and improved software design

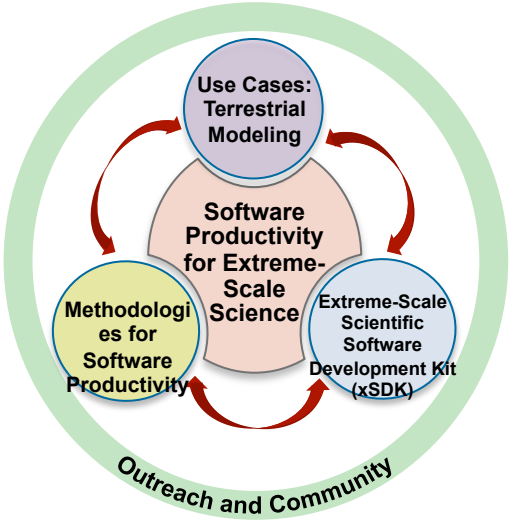


Impact on Applications & Programs

Terrestrial ecosystem ***use cases tie IDEAS to modeling and simulation goals*** in two Science Focus Area (SFA) programs and both Next Generation Ecosystem Experiment (NGEE) programs in DOE Biologic and Environmental Research (BER).







Approach

ASCR/BER partnership ensures delivery of both crosscutting methodologies and metrics with impact on real application and programs.

Interdisciplinary multi-lab team (ANL, LANL, LBNL, LLNL, ORNL, PNNL, SNL)

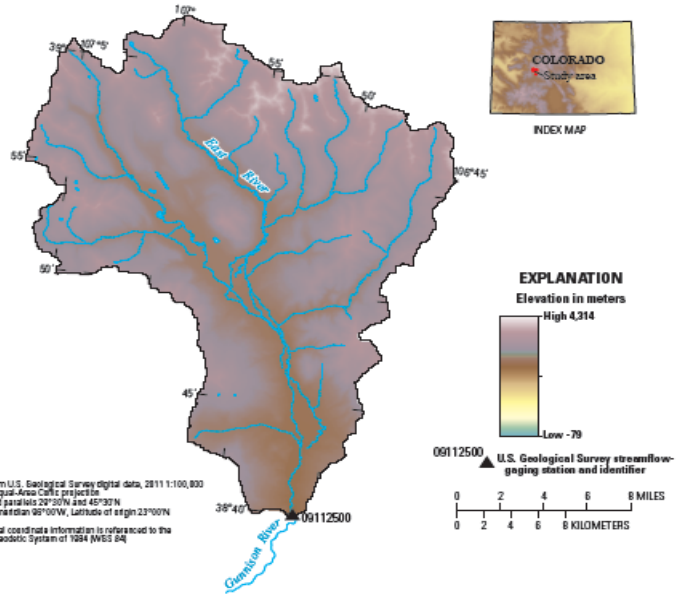
ASCR Co-Leads: Mike Heroux (SNL) and Lois Curfman McInnes (ANL)
BER Lead: David Moulton (LANL)
Topic Leads: David Bernholdt (ORNL) and Hans Johansen (LBNL)

Integration and synergistic advances in three communities deliver scientific productivity; outreach establishes a new holistic perspective for the broader scientific community.

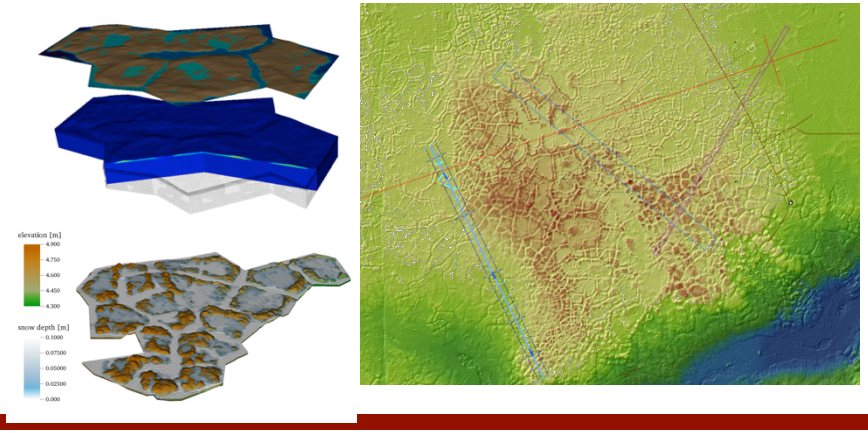
Use Cases: Multiscale, Multiphysics Representation of Watershed Dynamics

Sandia National Laboratories

- **Use Case 1:** Hydrological and biogeochemical cycling in the Colorado River System.



- **Use Case 2:** Thermal hydrology and carbon cycling in tundra at the Barrow Environmental Observatory.

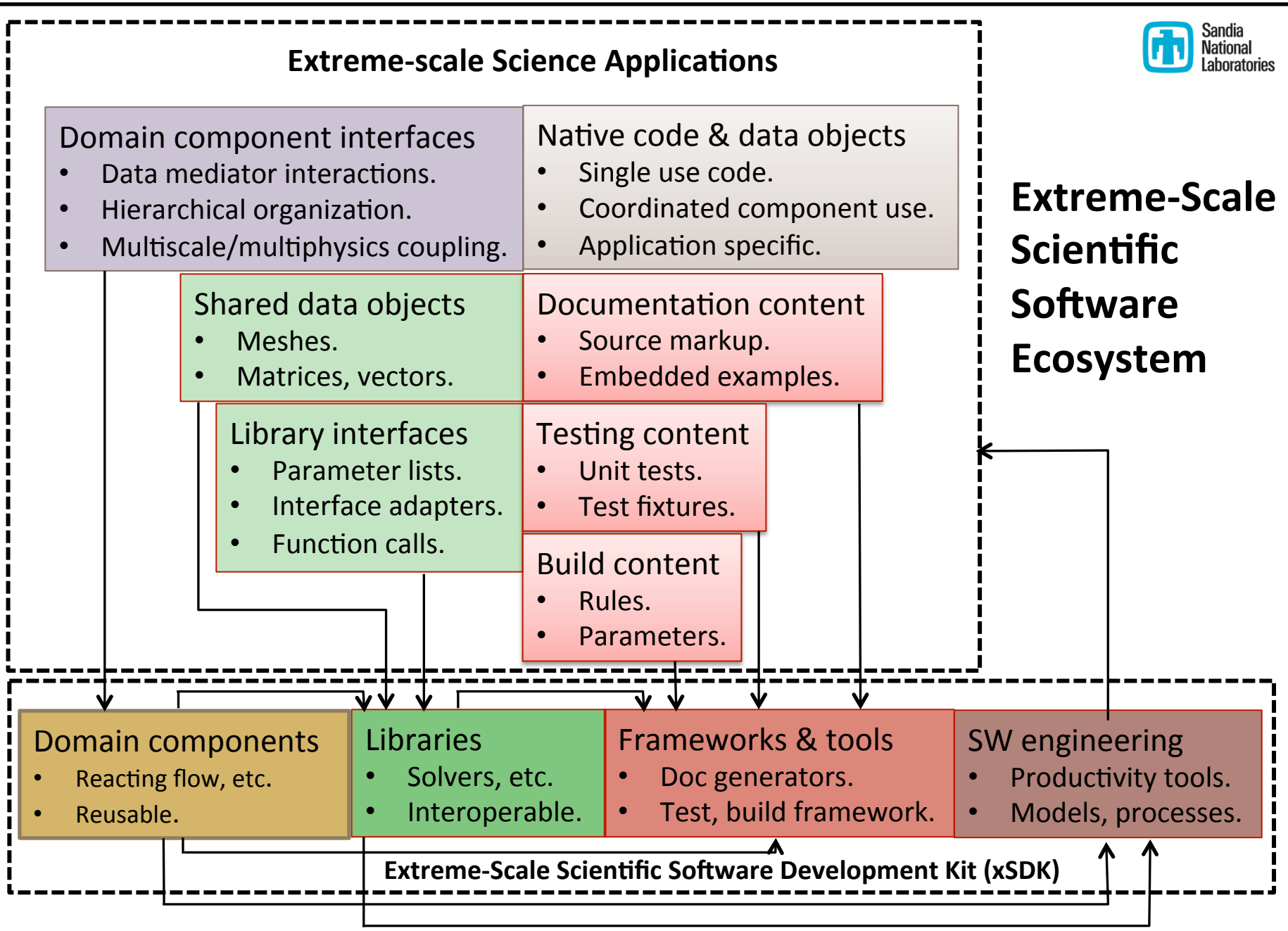


IDEAS Themes

Sandia National Laboratories

- **Use cases:** Drive efforts. Traceability from all efforts.
  - But generalized for future efforts.
- **Methodologies for software productivity**:
  - Metrics: Define for all levels of project. Track progress.
  - Workflows, lifecycles: Document and formalize. Identify best practices.
- **xSDK:** frameworks + components + libraries.
  - Build apps by aggregation and composition.
- **Outreach:** Foster communication, adoption, interaction.

- **First of a kind: Focus on software productivity.**


Extreme-scale Science Applications
Domain component interfaces
• Data mediator interactions.
• Hierarchical organization.
• Multiscale/multiphysics coupling.
Native code & data objects
• Single use code.
• Coordinated component use.
• Application specific.
Shared data objects
• Meshes.
• Matrices, vectors.
Documentation content
• Source markup.
• Embedded examples.
Library interfaces
• Parameter lists.
• Interface adapters.
• Function calls.
Testing content
• Unit tests.
• Test fixtures.
Build content
• Rules.
• Parameters.
Extreme-Scale Scientific Software Ecosystem
Domain components
• Reacting flow, etc.
• Reusable.
Libraries
• Solvers, etc.
• Interoperable.
Frameworks & tools
• Doc generators.
• Test, build framework.
SW engineering
• Productivity tools.
• Models, processes.
Extreme-Scale Scientific Software Development Kit (xSDK)

IDEAS Project Structure and Interactions

DOE Program Managers

ASCR: Thomas Ndousse-Fetter
BER: David Lesmes

IDEAS: Interoperable Design of Extreme-scale Application Software

ASCR Co-Leads: Mike Heroux (SNL) and Lois Curfman McInnes (ANL) **BER Lead:** J. David Moulton (LANL)

Executive Advisory Board

Use Cases for Terrestrial Modeling

Lead: J. David Moulton (LANL)
Tim Scheibe (PNNL)
Carl Steefel (LBNL)
Glenn Hammond (SNL)
Reed Maxwell (CSM)
Scott Painter (ORNL)
Ethan Coon (LANL)
Xiaofan Yang (PNNL)

Methodologies for Software Productivity

Lead: Hans Johansen (LBNL)
Roscoe Bartlett (ORNL)
Todd Gamblin* (LLNL)
Christos Kartsaklis (ORNL)
Pat McCormick (LANL)
Krishna Narayanan (ANL)
Andrew Salinger* (SNL)
Jason Sarich (ANL)
Dali Wang (ORNL)
Bill Spotz (SNL)
Jim Willenbring (SNL)

Extreme-Scale Scientific Software Development Kit

Lead: Mike Heroux (SNL)
Jed Brown (ANL)
Irina Demeshko (SNL)
Kerstin Kleese-Van Dam (PNNL)
Sherry Li (LBNL)
Vijay Mahadevan (ANL)
Daniel Osei-Kuffuor (LLNL)
Barry Smith (ANL)
Ulrike Yang (LLNL)

Outreach and Community

Lead: David Bernholdt (ORNL)
Katie Antypas* (NERSC)
Lisa Childers* (ALCF)
Judy Hill* (OLCF)

IDEAS productivity

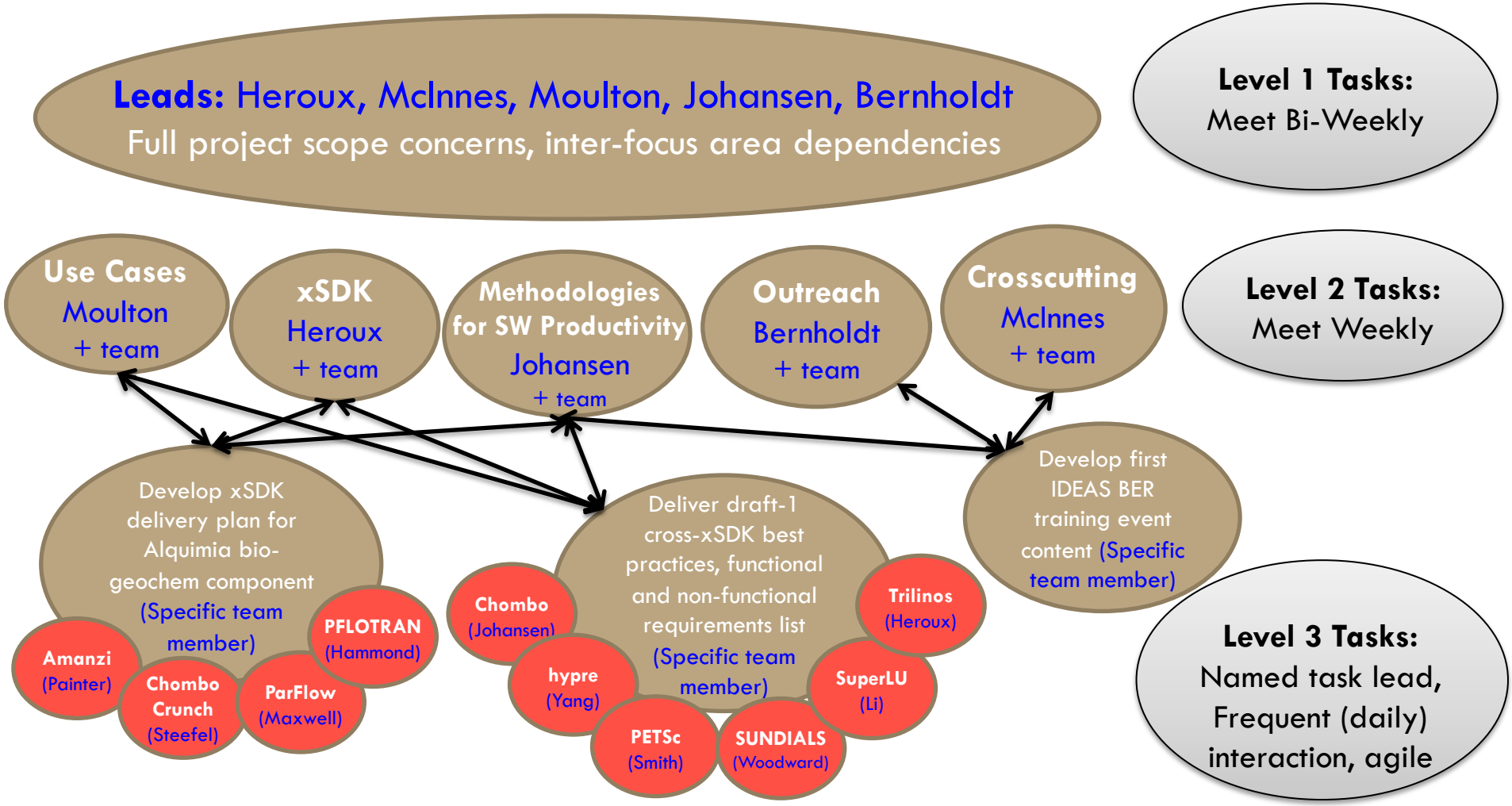
Crosscutting Lead: Lois Curfman McInnes (ANL)

BER Terrestrial Programs
SFAs
CLM
NGEE
ACME

DOE Extreme-scale Programs
Exascale Co-Design
Exascale Roadmap
ASCR Math & CS
SciDAC

DOE Computing Facilities
ALCF
NERSC
OLCF

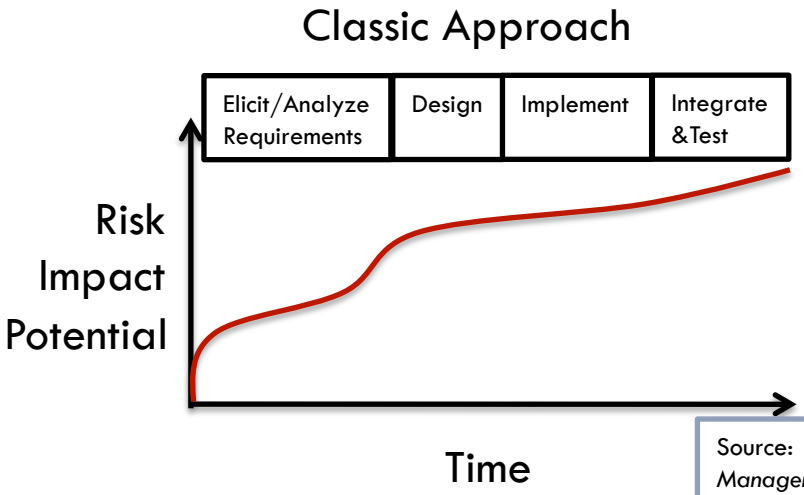
IDEAS Project Management: Three-tiered Structure

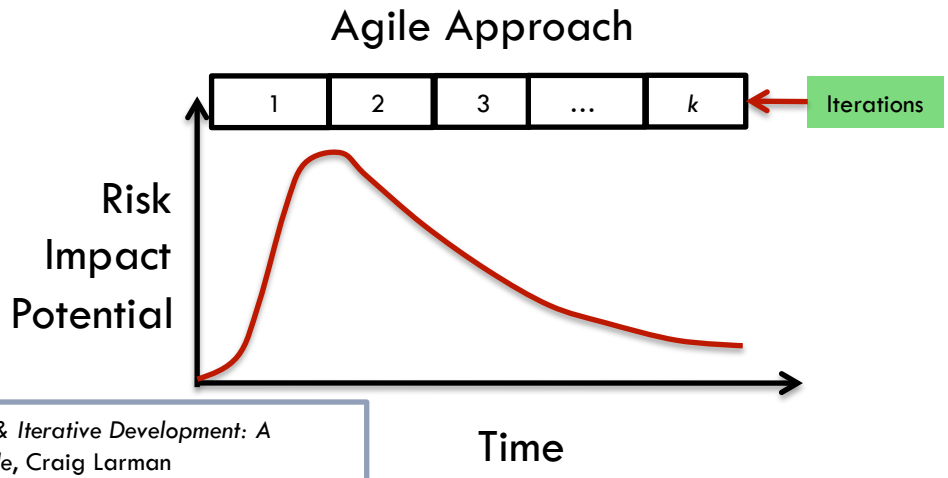


Risk Management: Classic vs Agile

IDEAS will use agile risk management workflows.

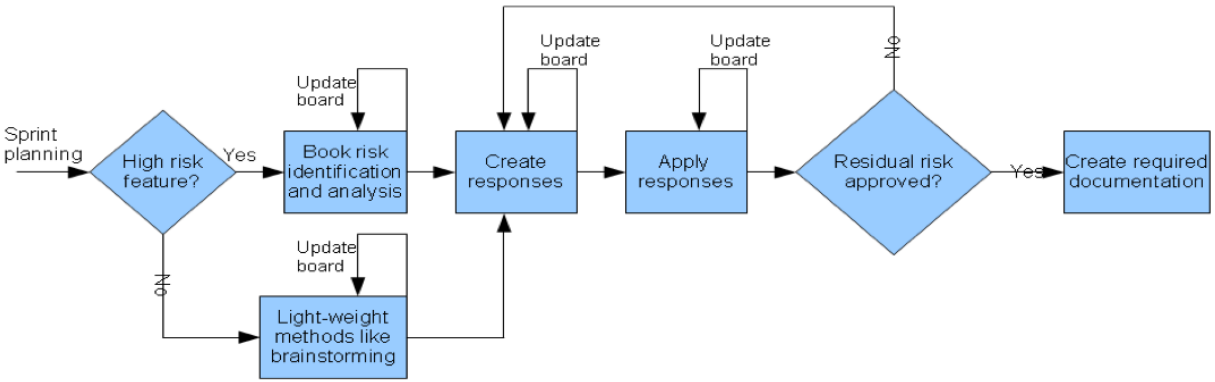
Sandia National Laboratories





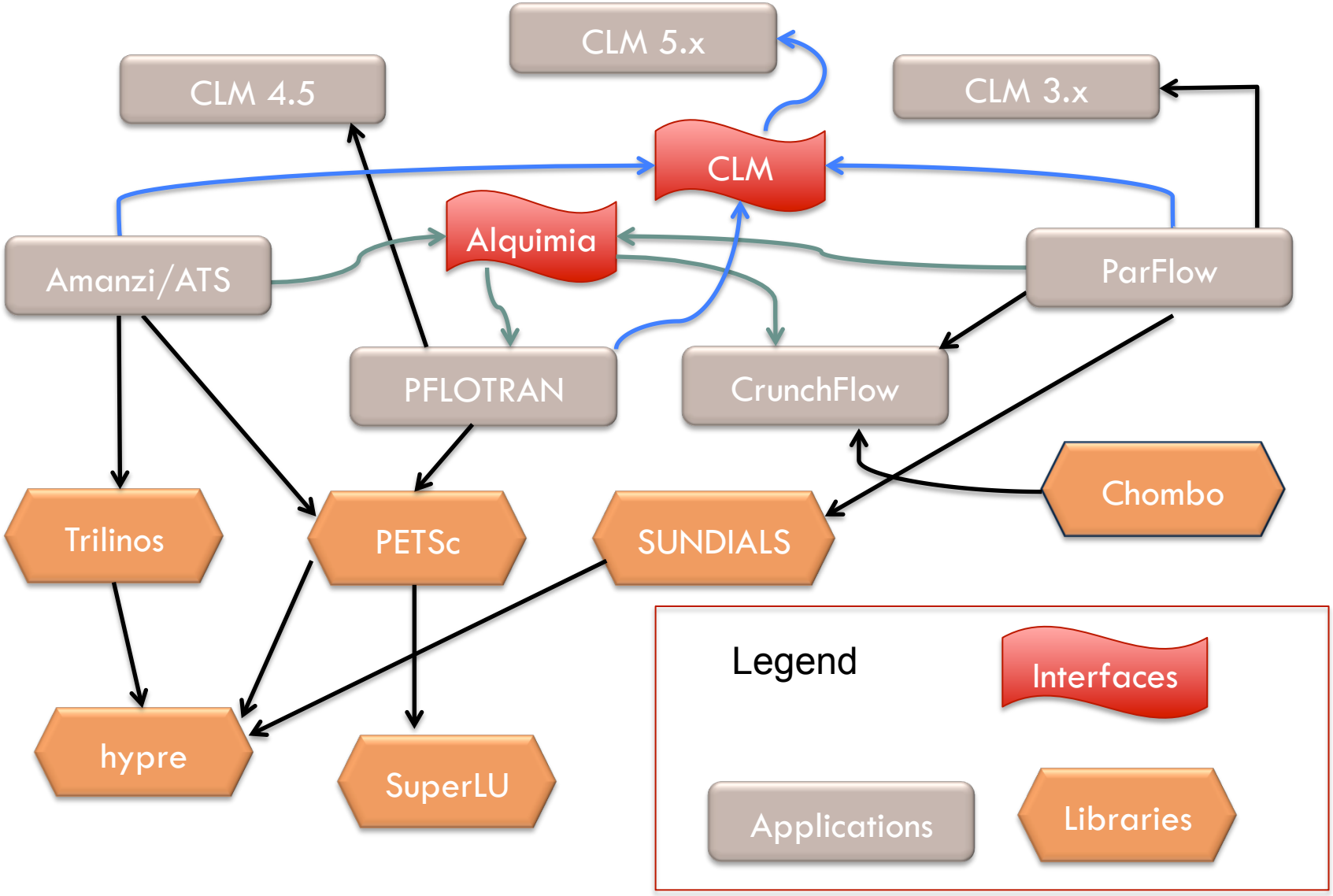
Source: *Agile & Iterative Development: A Manager's Guide*, Craig Larman

- Agile is better:
  - Risk managed incrementally.
  - Impact high early, lower later.
- Risk mgmt, mitigation easier:
  - Adapt: Less to refactor.
  - Drop: Less invested (lower loss).



Source: *A Model For Risk Management In Agile Software Development*, Ville Ylimannela

IDEAS Codes and Libraries



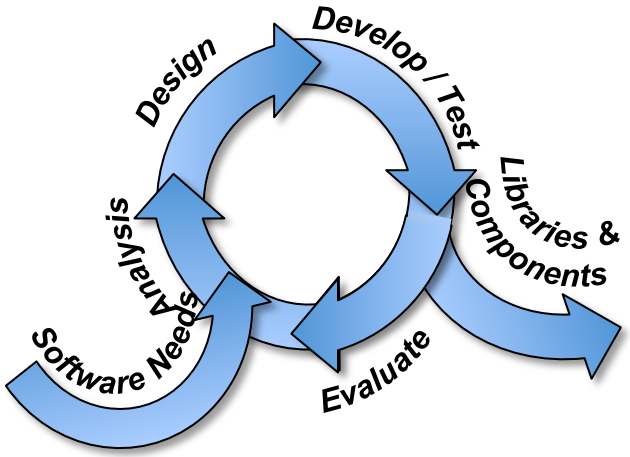
Sandia National Laboratories

SW LIFECYCLE MODELS & PRODUCTIVITY

Common Developer Workflows

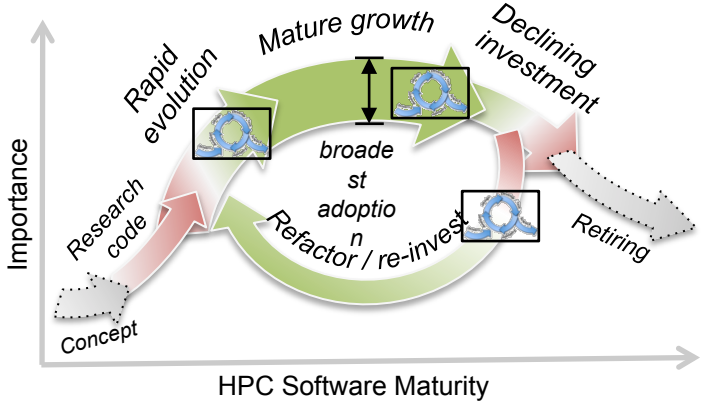
Sandia National Laboratories

- Define and elaborate key workflows:
  - **Software performance workflows:**
    - Performance analysis, refactoring.
  - **Development workflows:**
    - Clean-slate, augmentation, refactoring [of legacy code].
    - Sprints in an R&D culture.
  - **Repository collaboration workflow models:**
    - Centralized vs feature branch vs. forking, etc.
  - **Documentation workflows:**
    - Domain, user, reference.
  - **Test development & integration workflows:**
    - Test-driven development, test harnesses, auto-regression tests.
- Identify, promote effective tools & best practices:
  - Tool use, enhancement driven by methodology needs.

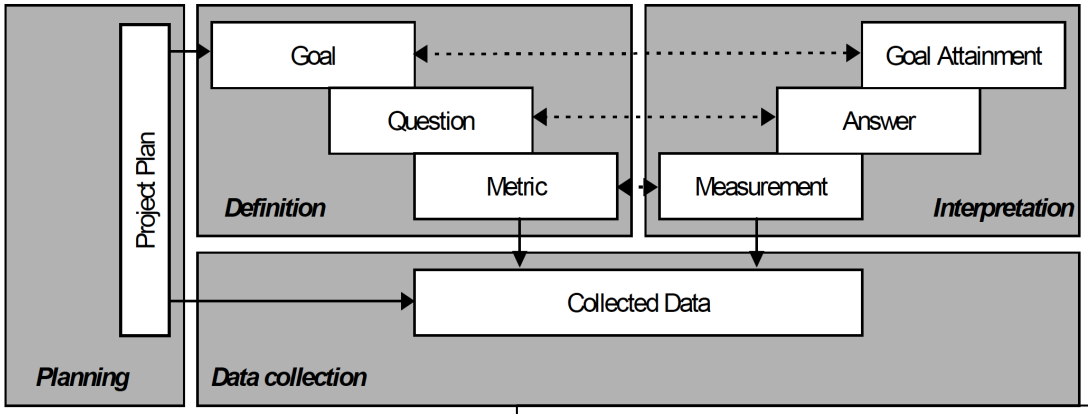


Methodologies: Lifecycles

- Establish & demonstrate use of effective lifecycles.
  - Phased expectations: Experimental to maintenance.
  - Expectations within each phase:
    - Experimental: Project plan – Funding proposal, artifacts: publications.
    - Maintenance: Domain document, automated regression testing, etc.
  - Promotion criteria, embedded phase regressions, etc.
  - Starting point: TriBITS, collaborations with Human Brain Project, EPFL.
- Training & adoption:
  - Materials, interaction with LCFs.



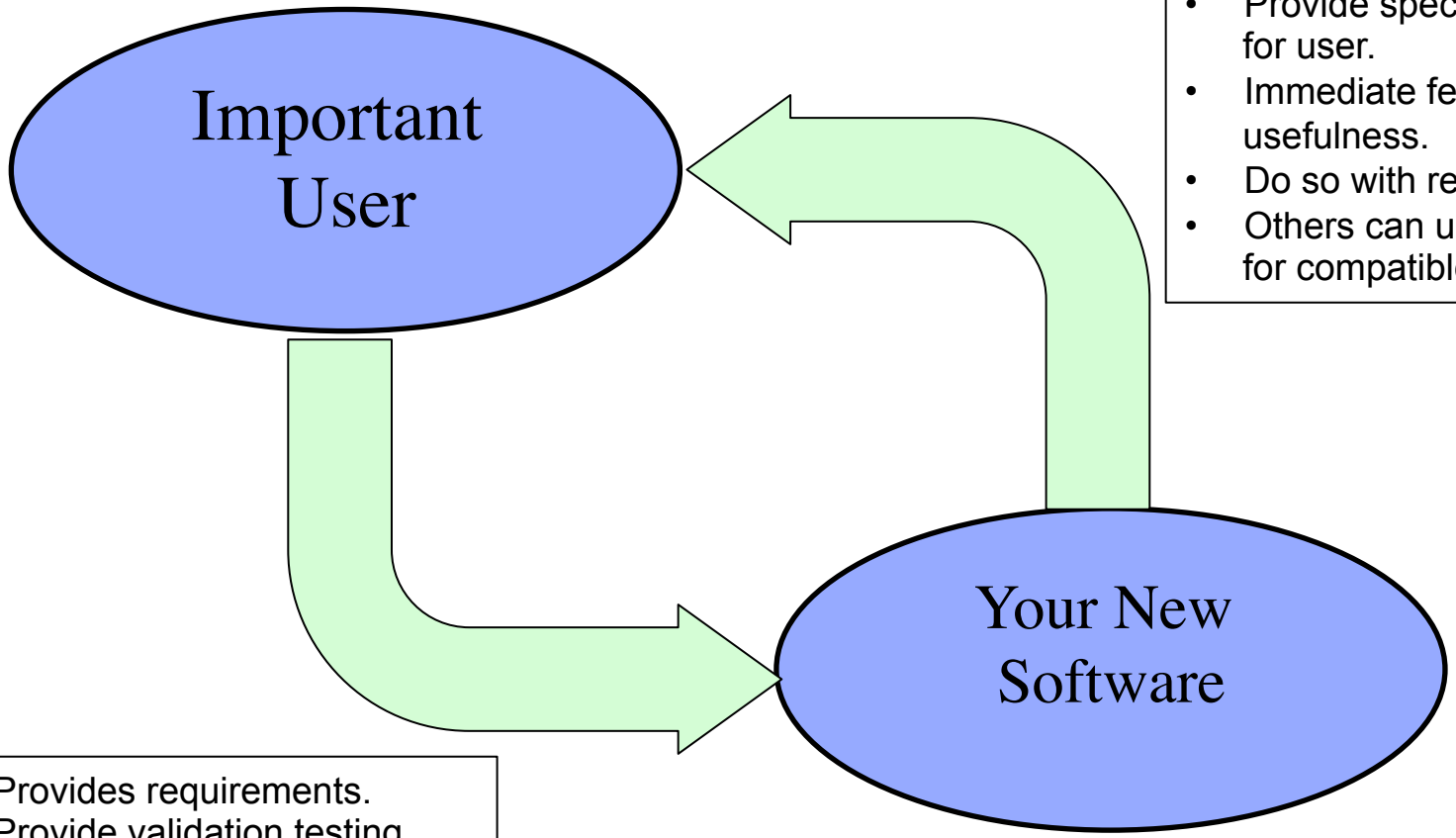
Methodologies: SW Productivity Metrics



Source: *The GQM Method: A Practical Gui for Quality Improvement of SW Developme* Solingen and Berghout.

- Define *processes* to define metrics.
  - Starting point: Goals, questions, metrics (GQM).
    - Define goals, ID questions to answer, define progress metrics.
- GQM Example:
  - Goal: xSDK Interoperability.
  - Question: Can IDEAS xSDK components & libs link?
  - Metric: Number of namespace collisions.
- Cultivate effective use of metrics:
  - Use metrics to drive and track use case progress.
- Promote use of metrics via Outreach.

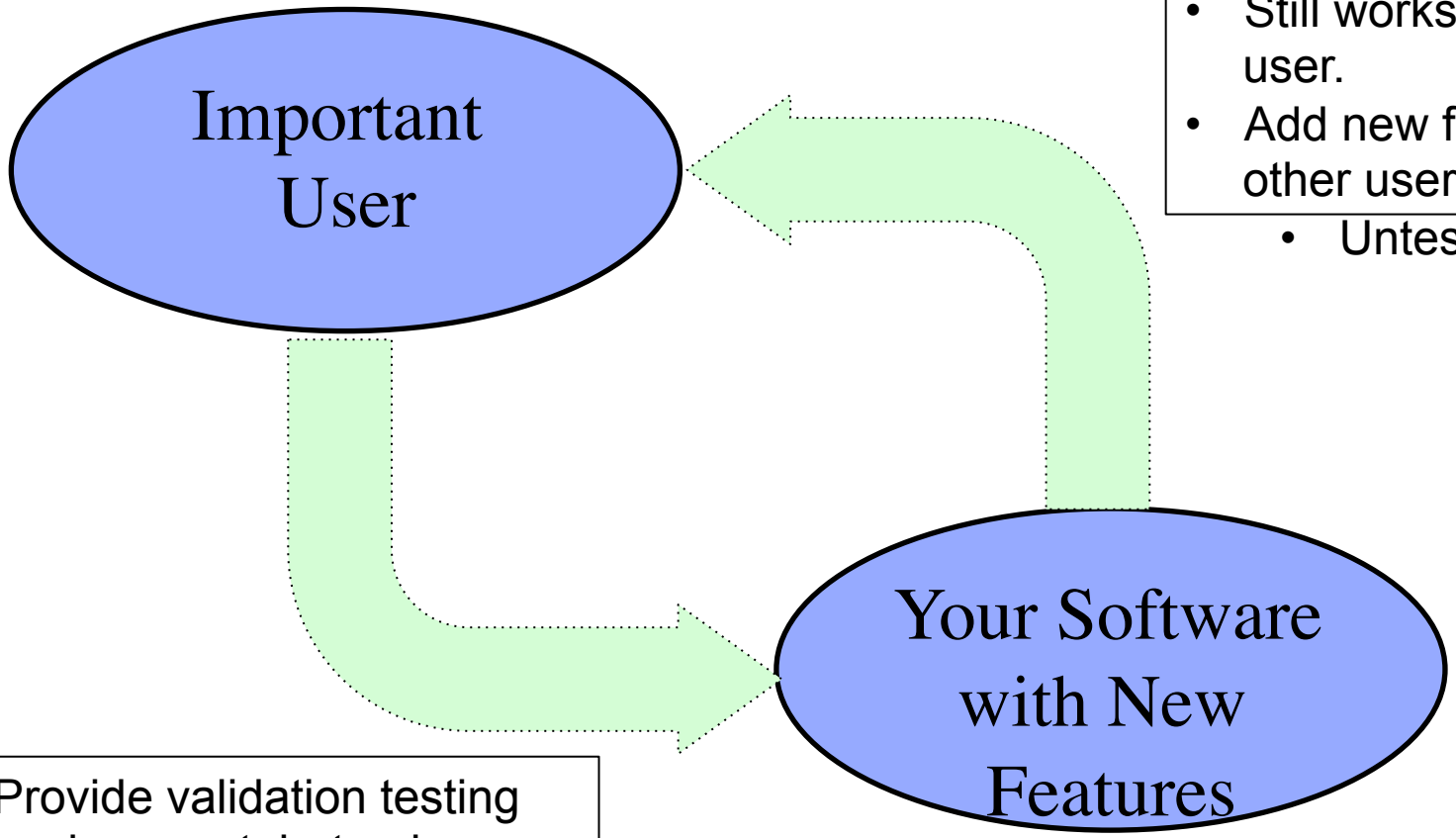
Common SW Development Scenario: Today



- Provide specific capabilities for user.
- Immediate feedback on usefulness.
- Do so with reuse in mind.
- Others can use your software for compatible needs.

- Provides requirements.
- Provide validation testing environment.
- Immediate feedback on correctness.

Common SW Development Scenario: Next Year

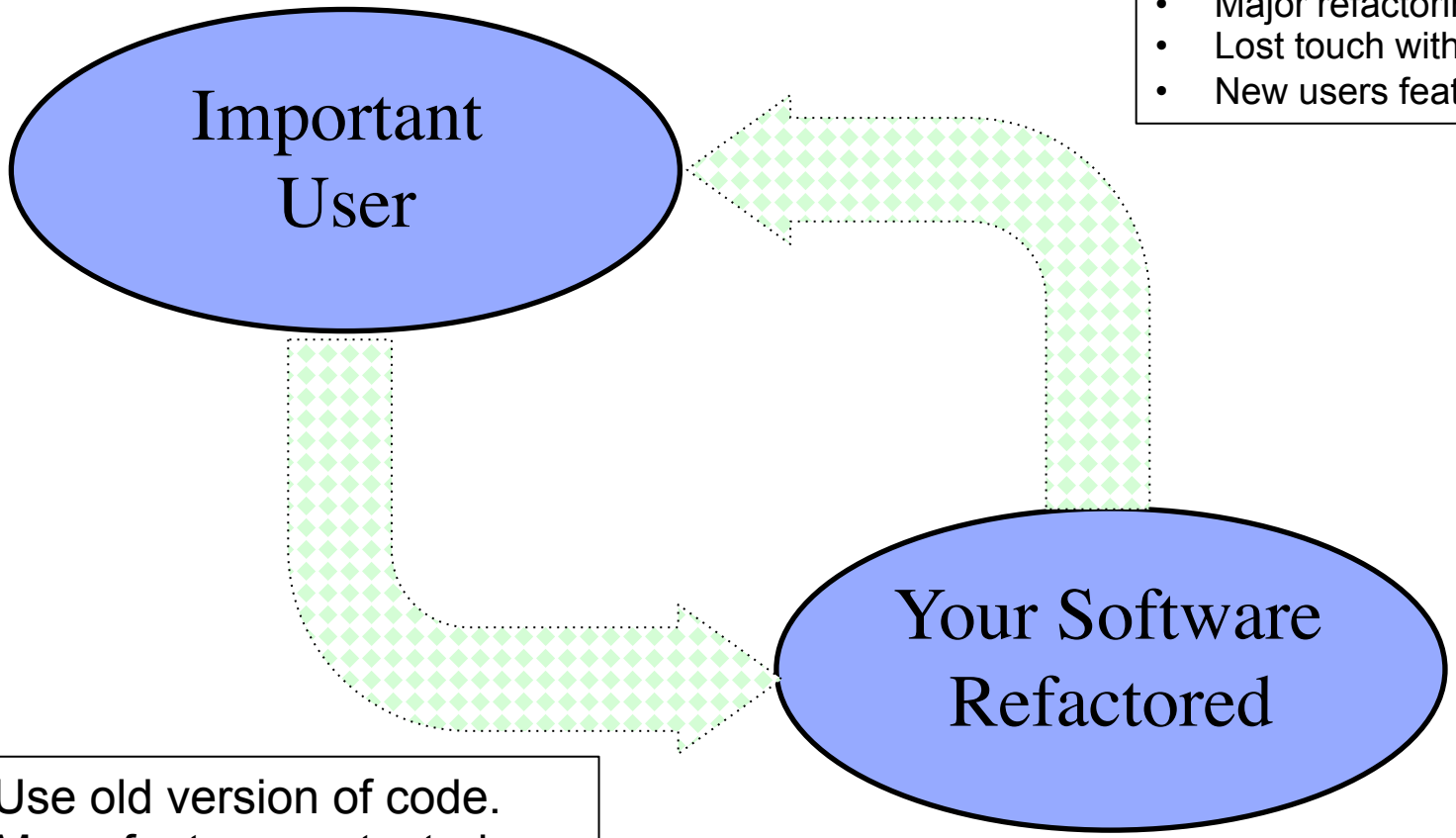


- Still works for original user.
- Add new features for other users.
  - Untested

- Provide validation testing environment, but only partial coverage.
- Other features untested.

Common SW Development Scenario: 5 Years

Sandia National Laboratories



- Major refactoring.
- Lost touch with original users.
- New users features untested.

- Use old version of code.
- Many features untested.

Result: Not enough test coverage for confident refactoring.

Validation-Centric Approach (VCA): Common Lifecycle Model for CSE Software

Sandia National Laboratories

Central elements of validation-centric approach (VCA) lifecycle model

- Develop the software by testing against real early-adopter customer applications.
- Manually verify the behavior against applications or other test cases.

Advantages of the VCA lifecycle model:

- Assuming customer validation of code is easy (i.e. linear or nonlinear algebraic equation solvers => compute the residual) …
- Can be very fast to initially create new code.
- Works for the customer's code right away.

Problems with the VCA lifecycle model:

- ***Does now work well when validation is hard*** (i.e. ODE/DAE solvers where no easy to compute global measure of error exists).
- ***Re-validating against existing customer codes is expensive or is often lost*** (i.e. the customer code becomes unavailable).
- ***Difficult and expensive to refactor***: Re-running customer validation tests is too expensive or such tests are too fragile or inflexible (e.g. binary compatibility tests).

VCA lifecycle model often leads to expensive or unmaintainable codes.

SE for CSE: Early years

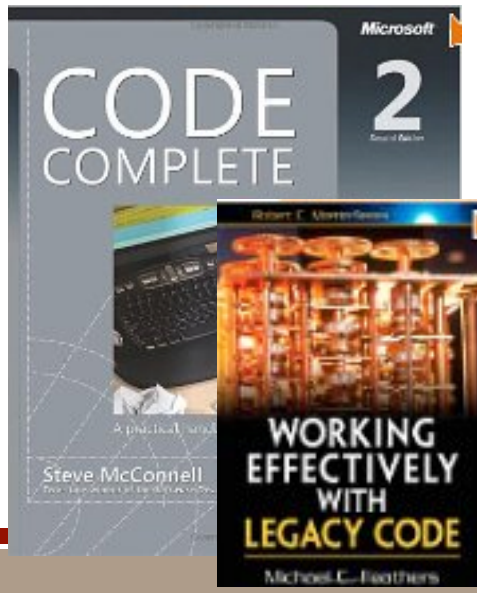
Sandia National Laboratories

- **Application validation-centric approach**
  - Write software within the context of use
  - Little stand-alone testing, efficient in short term
  - Over time: Components fragile, refactoring risky
- **SE *imposed* on CSE: Failure**
  - Theory: Commercial SW success => CSE SW success
  - Practice: Ignored first process phase: Gather requirements
  - Heavyweight, disconnected: artifacts costly, quickly irrelevant
  - Result: Bad impression lasting decades

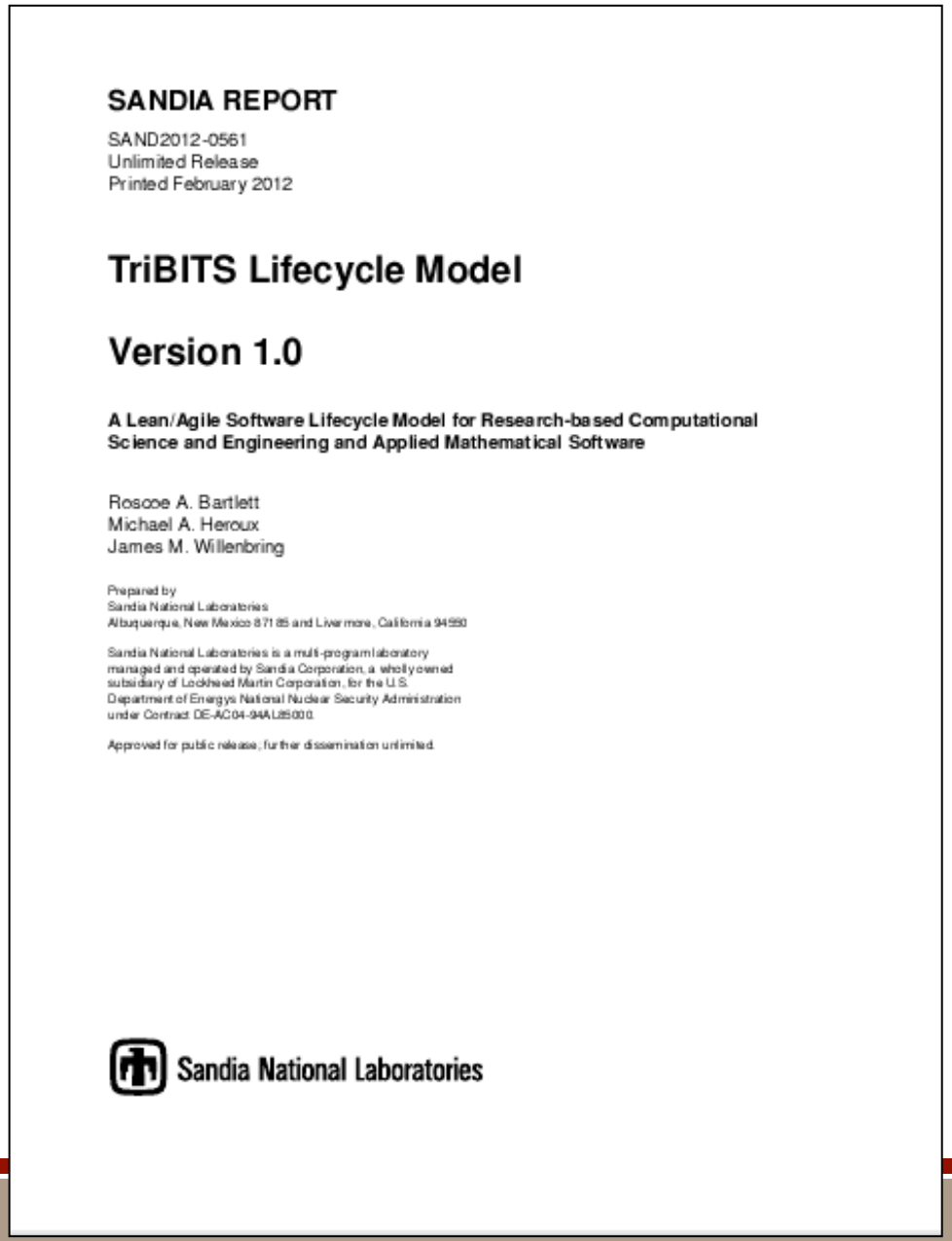
SE for CSE: Recent years, present

Sandia National Laboratories

- **Agile/Lean principles can work**
  - With discipline, accommodations
    - Sprints great for feature development
    - Must be balanced w/ R&D (longer time cycle)
    - Distributed teams: Extend team-room concept
  - Rigorous V&V required, esp. stand-alone tests
    - Long-lived products
    - Confidence to refactor
- **Community Education**
  - Widely-read material: Common Sensibility
  - Materials exist, not widely know, more needed



TriBITS Lifecycle Model 1.0 Document



SANDIA REPORT

SAND2012-0561
Unlimited Release
Printed February 2012

TriBITS Lifecycle Model

Version 1.0

A Lean/Agile Software Lifecycle Model for Research-based Computational Science and Engineering and Applied Mathematical Software

Roscoe A. Bartlett
Michael A. Heroux
James M. Willenbring

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Sandia National Laboratories

TriBITS: One Deliberate Approach to SE4CSE

Component-oriented SW Approach from Trilinos, CASL Projects, LifeV, …
Goal: "Self-sustaining" software

Sandia National Laboratories

TriBITS Lifecycle Maturity Levels
0: Exploratory
1: Research Stable
2: Production Growth
3: Production Maintenance
-1: Unspecified Maturity

Goals

- *Allow Exploratory Research to Remain Productive*: Minimal practices for basic research in early phases
- *Enable Reproducible Research*: Minimal software quality aspects needed for producing credible research, researchers will produce better research that will stand a better chance of being published in quality journals that require reproducible research
- *Improve Overall Development Productivity*: Focus on the right SE practices at the right times, and the right priorities for a given phase/maturity level, developers work more productively with acceptable overhead
- *Improve Production Software Quality*: Focus on foundational issues first in early-phase development, higher-quality software will be produced as other elements of software quality are added
- *Better Communicate Maturity Levels with Customers*: Clearly define maturity levels so customers and stakeholders will have the right expectations

Ultimate Goal: Produce "self-sustaining" software products.

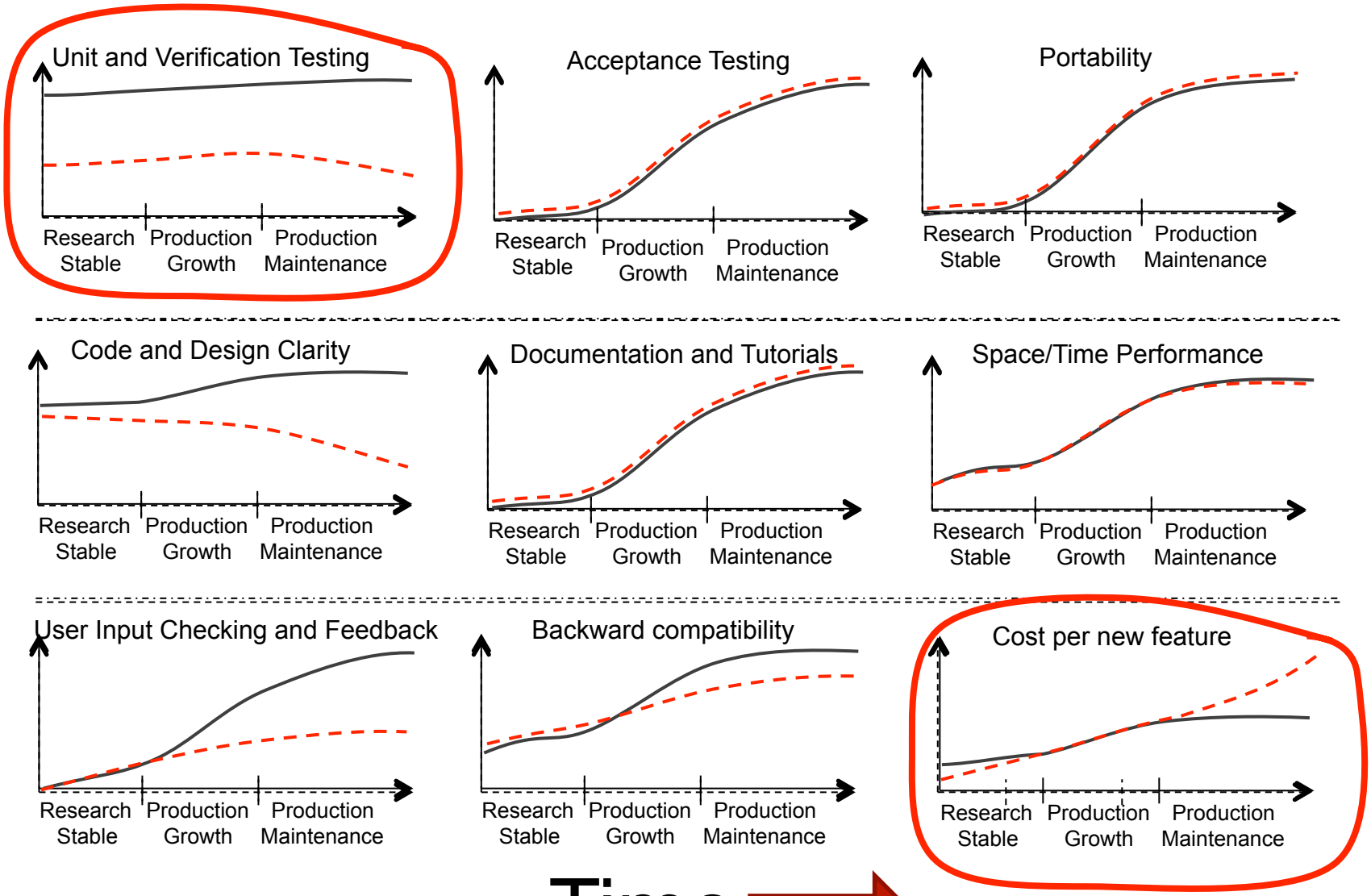
Defined: Self-Sustaining Software

Sandia National Laboratories

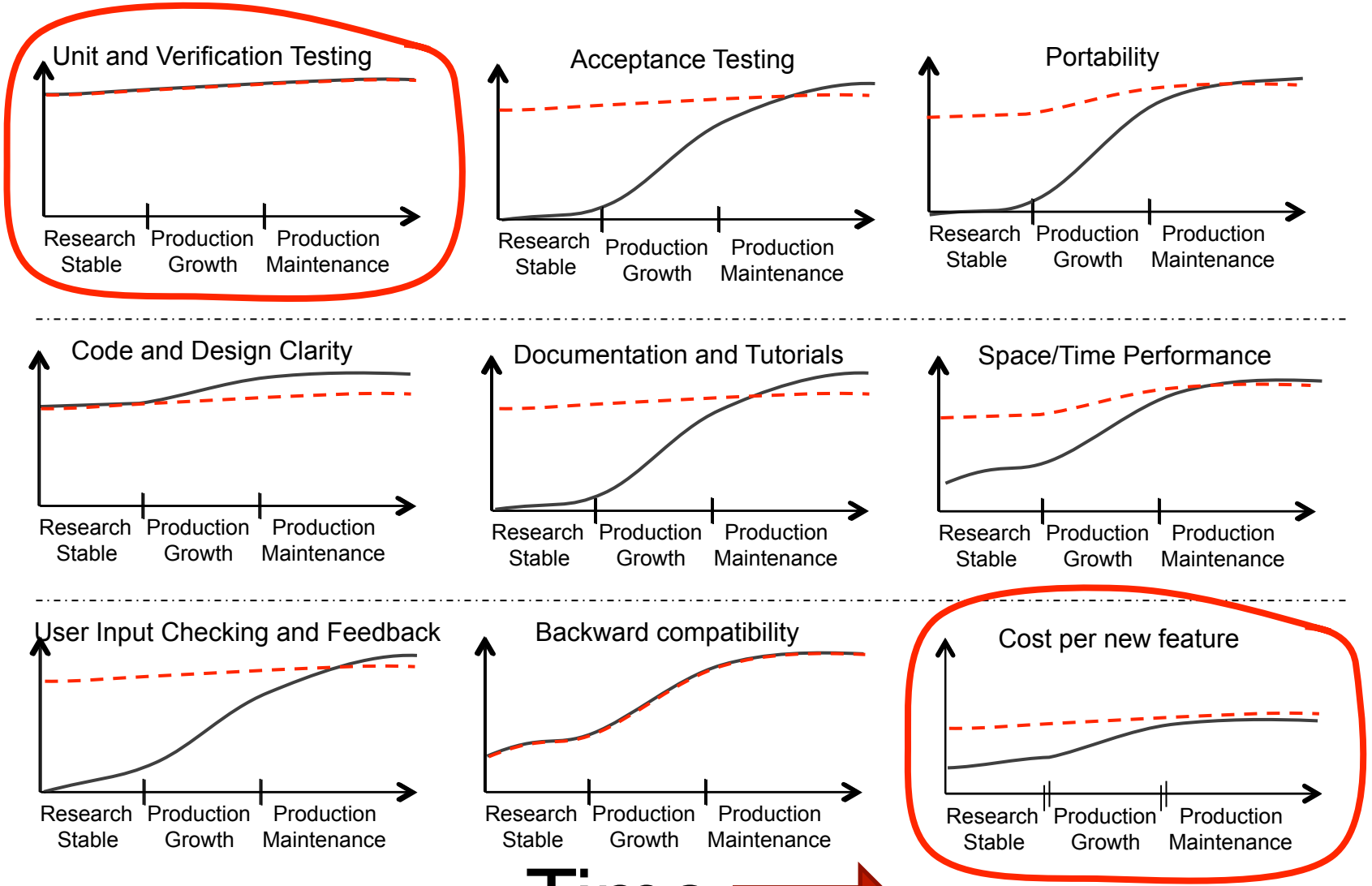
- ***Open-source*:** The software has a sufficiently loose open-source license allowing the source code to be arbitrarily modified and used and reused in a variety of contexts (including unrestricted usage in commercial codes).
- ***Core domain distillation document*:** The software is accompanied with a short focused high-level document describing the purpose of the software and its core domain model.
- ***Exceptionally well testing*:** The current functionality of the software and its behavior is rigorously defined and protected with strong automated unit and verification tests.
- ***Clean structure and code*:** The internal code structure and interfaces are clean and consistent.
- ***Minimal controlled internal and external dependencies*:** The software has well structured internal dependencies and minimal external upstream software dependencies and those dependencies are carefully managed.
- ***Properties apply recursively to upstream software*:** All of the dependent external upstream software are also themselves self-sustaining software.
- ***All properties are preserved under maintenance*:** All maintenance of the software preserves all of these properties of self-sustaining software (by applying Agile/Emergent Design and Continuous Refactoring and other good Lean/Agile software development practices).

Example: Reference LAPACK Implementation

TriBITS (–) vs. Validation-Centric Approach (--)



TriBITS(–) vs. Pure Lean/Agile Approach (--)



Test Driven Development

Sandia National Laboratories

- Write tests first:
  - Guarantees that tests will be written.
  - Debugs the API: First attempt to use SW as intended.
- Use tests during development:
  - All tests fail at first.
  - Pass incrementally as SW written.
  - Measure of progress.
- Use tests forever more:
  - Regression.
  - Backward compatibility.
  - Aggressive refactoring.
- Single most important activity:
  - Assures long, happy life for your product.

Addressing existing Legacy Software

Sandia National Laboratories

- One definition of "Legacy Software": Software that is too far from away from being Self-Sustaining Software, i.e:
  - Open-source
  - Core domain distillation document
  - Exceptionally well testing
  - Clean structure and code
  - Minimal controlled internal and external dependencies
  - Properties apply recursively to upstream software

- Question: What about all the existing "Legacy" Software that we have to continue to develop and maintain? How does this lifecycle model apply to such software?

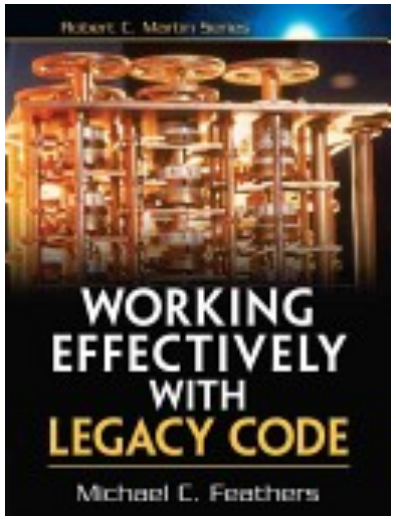
- Answer: Grandfather them into the TriBITS Lifecycle Model by applying the Legacy Software Change Algorithm.

Grandfathering of Existing Packages

Sandia National Laboratories

Agile Legacy Software Change Algorithm:

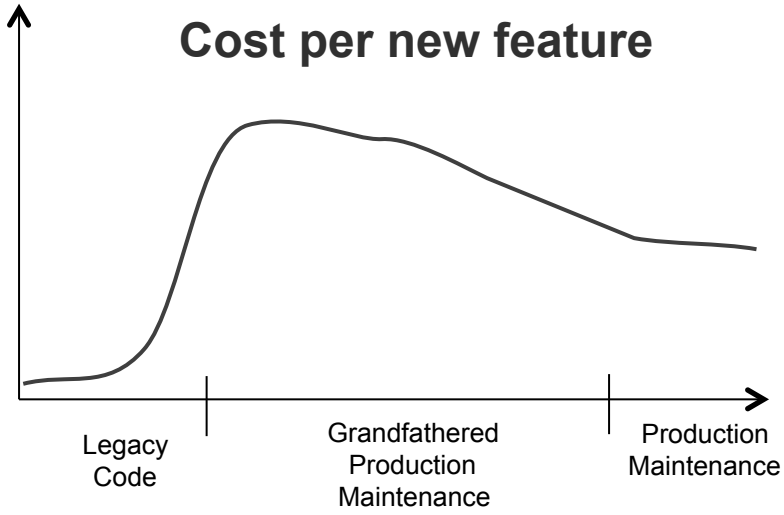
1. Identify Change Points
2. Break Dependencies
3. Cover with Unit Tests
4. Add New Functionality with Test Driven Development (TDD)
5. Refactor to removed duplication, clean up, etc.



Grandfathered Lifecycle Phases:

1. Grandfathered Research Stable (GRS) Code
2. Grandfathered Production Growth (GPG) Code
3. Grandfathered Production Maintenance (GPM) Code

NOTE: After enough iterations of the Legacy Software Change Algorithm the software may approach Self-Sustaining software and be able to remove the "Grandfathered" prefix.



End of Life?

Sandia National Laboratories

Long-term maintenance and end of life issues for Self-Sustaining Software:

- User community can help to maintain it (e.g., LAPACK).
- If the original development team is disbanded, users can take parts they are using and maintain it long term.
- Can stop being built and tested if not being currently used.
- However, if needed again, software can be resurrected, and continue to be maintained.

NOTE: Distributed version control using tools like Git greatly help in reducing risk and sustaining long lifetime.

Summary

Sandia National Laboratories

- HPC has Major Disruptions:
  - Disruptive architecture changes force disruptive software refactoring.
  - Capabilities drive ability to Couple physics and scales, need for modularity.
- A Productivity Focus is promising:
  - Walking back to first principles, iterating forward.
  - Provides guidance in time of disruptive changes.
- Libraries provide part of the answer to disruptions:
  - Provide leverage by promoting reusable software.
  - Provide portability by encapsulating HW architecture differences (GPUs vs. CPUs).
- Community changes are necessary:
  - Change of culture: Applications are *composed* within an ecosystem.
  - Professional SW processes and practices are fostered and adopted.