LLNL-TR-418067

LAWRENCE LIVERMORE NATIONAL LABORATORY

The Nuclear Energy Advanced Modeling and Simulation Enabling Computational Technologies FY09 Report

L. F. Diachin, F. X. Garaizar, V. E. Henson, G. Pope

October 15, 2009

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

The Nuclear Energy Advanced Modeling and Simulation Enabling Computational Technologies FY09 Report

October 2009

Lori Diachin
Xabier Gariazar
Van Emden Henson
Gregory Pope

Lawrence Livermore National Laboratory

Lawrence Livermore National Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344.

Table of Contents

Abstract

In this document we report on the status of the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Enabling Computational Technologies (ECT) effort. In particular, we provide the context for ECT In the broader NEAMS program and describe the three pillars of the ECT effort, namely, (1) tools and libraries, (2) software quality assurance, and (3) computational facility (computers, storage, etc) needs. We report on our FY09 deliverables to determine the needs of the integrated performance and safety codes (IPSCs) in these three areas and lay out the general plan for software quality assurance to meet the requirements of DOE and the DOE Advanced Fuel Cycle Initiative (AFCI). We conclude with a brief description of our interactions with the Idaho National Laboratory computer center to determine what is needed to expand their role as a NEAMS user facility.

Introduction

The Nuclear Energy Advanced Modeling and Simulation (NEAMS) Campaign was instituted to provide essential computational simulation capabilities enabling the Department of Energy to partner with designers and engineers from the nuclear energy industry for the cost-effective, efficient, and safe development of the next generation of nuclear energy production. Nuclear energy is considered a key ingredient in any effective strategy to ensure the security and independence of our future energy supplies.

In the face of an aging nuclear power infrastructure now relying on an older, unsustainable model of energy production, with spent fuel piling up in repositories, the Department of Energy is embarking on an end-to-end redesign of the entire fuel cycle, from the acquisition of fissile material, through its processing into nuclear fuel, through the generation of power, reprocessing and recycling of used fuel, and finally through the final disposition of spent nuclear materials.

This is an enormous undertaking, a science and engineering task on a scale that the Department of Energy (or its predecessors) has embarked upon only a few times, most notably in the Manhattan Project. There is tremendous work to be done in the design of new methods and facilities; included in this work is a great amount of fundamental science.

The scientific community has come to recognize over the past fifteen years that science at this scale can only be feasible through heavy reliance on the use of computational simulation as a predictive science. The experiments necessary to prove or refine the fundamental theories are too expensive, or on too grand a scale, or requiring such extraordinary instrumentation to work at an atomistic scale, that very few of the necessary experiments can be conducted.

Mitigating this situation, however, has been the rise of computational science capability, reaching the point that computational simulation is now widely considered a peer to theory and experiment in a new triumvirate of science. Simulation has long been used to validate experiment and verify theory; now it is used to predict fundamental physics, to inform the design of experiment, and to guide the overall end-to-end process of scientific discovery, design, and engineering development.

For this state of affairs, the Department of Energy can take a fair share of credit. Beginning in the mid 1990s with the Accelerated Strategic Computing Initiative (ASCI), which matured into the Advanced Simulation and Computing (ASC) Program, and progressing into the early 2000s with the Scientific Discovery through the Advanced Computing (SciDAC) initiative, the DOE has in many ways led the way. ASCI, and later ASC, paved the way by harnessing National Laboratories, universities, and industry into a coherent team that produced unprecedented massively parallel supercomputer architectures, computing and development environments to employ these machines, and the detailed multiphysics codes operating at unheard of resolutions to create an accurate, verifiable, and trustworthy assessment of the state of the nation's nuclear weapons arsenal. SciDAC followed in the ASC footsteps by mobilizing multi-institutional teams to apply this same style of computational capability to a broad range of important scientific problems in a wide range of topics, from fundamental physics to climatology, biology, planetary science, and astrophysics, among others.

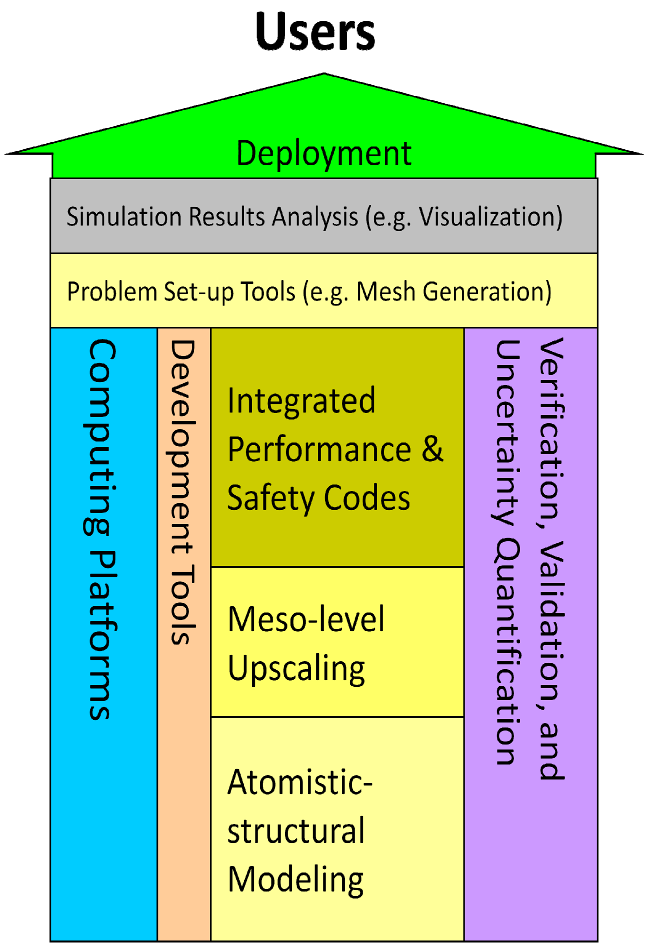
In just such a spirit was NEAMS founded. Recognizing that the future nuclear energy industry would have to overcome scientific obstacles and challenges on a scale similar to those faced a decade earlier by ASCI, the intention is that NEAMS become the source of the computational simulation capability that nuclear scientists and designers will require to overhaul the entire nuclear power production process from start to finish.

Overview of NEAMS and ECT's role

NEAMS is intended to deliver to its ultimate customers a comprehensive, integrated capability for performing large-scale multiphysics simulation to be used as a crucial tool in the design, engineering, licensing, and operation of the next-generation nuclear power system, including the entire fuel cycle from processing to utilization to separation and reprocessing, reuse, and ultimately, permanent storage or disposal. The ultimate users will be nuclear power plant and fuel cycle designers, engineers, and regulators.

For NEAMS to succeed, it must succeed in each of several distinct but highly interdependent functionalities, as delineated in Figure 1. Certain pieces of the puzzle are relatively obvious; for example, no high-performance simulations can be performed without access to large-scale computing platforms, so one function of NEAMS is to ensure that such facilities are available and that the codes and computer science tools and environments are developed and deployed to make use of these tools.

Looking at the overall structure from the lowest level, NEAMS must ensure that modeling capabilities at the atomistic level exist, as this fundamental understanding of the basic physics enables the simulation codes to be created with the appropriate materials behavior built into the codes directly. To accomplish this, an "upscaling" function must be developed to transfer what is learned about the sub-grid behavior of the materials to the macroscale modeling. These capabilities are used by the integrated safety and performance code teams, who lie at the heart of NEAMS. There are four such teams, specializing respectively in Reactors, Fuels, Waste Forms, and Safeguards & Separations. Several critical functionalities must be developed in support of these code teams. As noted above, the computing facilities that allow the codes to be developed, tested, and, ultimately, employed must be developed, acquired, or identified. Similarly, the code teams require the most modern code development tools and environments to perform effectively. A crucial component of creating large-scale simulations is the verification and validation of the codes: roughly speaking, *verification* is the process of insuring that the codes correctly solve the mathematical governing equations (are we solving the problem right?) while *validation* is the process of insuring that the governing mathematical equations are modeling the correct physical problem (are we solving the right problem?). Also critical to the process are tools that enable the problem to be set up correctly, in a way that can be solved, in a reasonable amount of time. For multiphysics simulation on complex geometries this implies the development of sophisticated meshing tools, including the capability of transferring data from meshes at meso-scales to the macro scale as part of the upscaling process. Once the simulations are run (and increasingly, *as* the simulations are being run, tools enabling scientists to analyze and interpret the results are


Users
Deployment
Simulation Results Analysis (e.g. Visualization)
Problem Set-up Tools (e.g. Mesh Generation)
Computing Platforms
Development Tools
Integrated Performance & Safety Codes
Meso-level Upscaling
Atomistic-structural Modeling
Verification, Validation, and Uncertainty Quantification

S functionality

necessary; hence an effort in visualization and analysis for large data sets is essential. Finally, NEAMS must develop a plan for deployment: a means of transferring the technology they create to the ultimate end users.

To accomplish these tasks, NEAMS is organized into five program elements. The most important, the "business end" of NEAMS, is made up of four multi-institutional Integrated Performance and Safety Code (IPSC) teams:

1. **Reactors:** the reactor core IPSC code team is striving to produce three-dimensional simulations of the reactor core for the design of next generation reactors and to understand their behavior in various accident scenarios and thereby ensure their safety.

2. **Fuels:** the fuels IPSC team aims to develop a three dimensional predictive tool for the simulation of pins and assemblies applicable to existing and future design nuclear reactors in both normal and abnormal operating conditions.

3. **Safeguards and Separations:** the objective of the Safeguards and Separations IPSC is to develop facility-level, first-principles, fully integrated simulation of separation and safeguard technologies that enable a sustainable next-generation fuel cycle with minimal processing, waste generation, and potential for material diversion.

4. **Waste Forms and Systems:** the objective of the Waste Forms IPSC is to provide an integrated suite of computational capabilities for simulating the long-term performance of waste forms in the engineered environment of a waste storage or disposal repository.

While each of these code teams faces specialized and unique problems to surmount, they share many common needs. As a result, four "cross-cutting elements" were established to provide services, tools, and expertise that would be needed by all of the IPSC code teams. The four cross-cutting program elements are:

1. **Fundamental Models and Methods (FMM)**: Among the challenges common to several of the IPSC teams is the need to fully understand and model physical behavior of materials at smaller length scales; this program element is intended to provide tools and techniques to understand how materials behave and interact at multiple scales – enabling the so-called Atomistic-to-Continuum (AtC) multi-scale simulation by providing the IPSC teams with understanding and improved properties to be used in their model. This element also identifies and drives small scale experimentation necessary to generate the data needed for physical and engineering models.

2. **Verification & Validation and Uncertainty Quantification (VU)**: Any computational simulation code, to be of any value, must be verified and validated as defined above. This program element is intended to provide tools that allow computer scientists, physicists, and engineers to measure and evaluate whether the code satisfies these two criteria. Moreover, in any computational model, there are parameters and variables whose values are, in varying degree, uncertain. Uncertainty quantification is designed to understand the possible effects on the final calculation due to these uncertainties. This program element is also established to develop and implement methodologies to understand the uncertainties and ensure that their effects fall within the design margins of the codes. As a consequence of the nature of this activity, this program element will function as NEAMS' principal interface to the Nuclear Regulatory Commission.

3. **Capability Transfer (CT)**: The success of NEAMS is entirely dependent on the degree to which the capabilities NEAMS creates are adopted and employed by the nuclear energy industry and licensing bodies. It is crucial, therefore that NEAMS include an element tasked to make the NEAMS product attractive, available, and readily useful to the scientists, designers, engineers, operators, and regulators of the nation's nuclear industry. The program element accordingly must develop methods for accomplishing this transfer, including strategies for turning scientific and research codes into engineering and production tools to be used by industry, and to work toward "hardening" and "productizing" the high performance computing codes and systems, making them easily accessed and improving their "usability."

4. **Enabling Computational Technology (ECT)**: One of the most important lessons of the ASCI and SciDAC experiences is that there is great commonality in the fundamental technological needs seen by high-performance code teams, independent of the specific scientific questions the teams are attempting to answer. All code teams need to have code development and programming environments that make tractable the writing, testing, debugging, and modifying of massive codes that easily run to hundreds of thousands of code lines, are written by teams of scores of programmers implementing the ideas of tens of scientists, over many months or years. An essential requirement of NEAMS is to ensure that technologies enabling code work at this scale are available to each of the IPSC teams.

For NEAMS to be successful, each of these cross-cutting program elements must succeed, in order that the IPSC teams can create the simulation capability incorporating the fundamental, extraordinary physics necessary to provide the designers and engineers in the nuclear industry with the confidence to employ NEAMS technology in the actual design of a new, reliable, efficient, and safe nuclear energy generating capability.

How (and what) ECT will deliver

A credible scientific computational simulation capability must have certain characteristics. First, it must be *science based*; that is, it must be based, to every extent possible, on first principals of physics rather than relying on heuristics, anecdotal evidence, or extrapolation from a small number of limited experiences. It must feature *high dimensionality*, including at least two and more often all three spatial dimensions, as well as the dimension of time. It must be *high resolution*, implying that the codes, and systems on which they run, must be capable of handling simulations involving many millions or even many billions of independent variables, as many as are necessary to resolve the problem with *adequate modeling of space and time*. The codes must be part of *integrated systems*, implying that codes modeling all components of the system (e.g., a reactor, a waste form processing stream, a fuel cycle) must be integrated together, sharing file formats (or having full-function converters), I/O processes, etc., so that all components can work together, in sequence or in parallel, throughout the end-to-end modeling process. An essential characteristic is that such codes must include the means for *appropriate verification, validation and uncertainty quantification*, necessary to ensure that the models give reliable and scientifically correct answers. It is expected that these codes will run on some of the world's *most powerful computing platforms* and that they will use the *best programming practices* and sophisticated tools for displaying and analyzing the results.

It is important to observe that ECT is charged with ensuring that the IPSC teams have access to the tools and techniques that enable the creation of the high-performance codes described in the preceding paragraph. This does *not* imply that ECT will be primarily in the business of creating these tools; rather, ECT will identify tools for a given functionality needed by the code teams (and there will likely be multiple tools for a given functionality), determine what is necessary to make that tool available to the code teams, and provide that capability. For the most part, this will consist of identifying one or several software tools needed for the functionality (e.g., a debugger, a memory profiler, or a visualization suite), and providing a central location where the code teams can acquire the source and/or binaries for the tool, together with any licensing or registration requirements. ECT's primary mode of functioning will be in the identification and propagation of appropriate tools. However, there may be areas in which a technological gap must be bridged; where the necessary tools simply do not yet exist or must be customized to meet the particular needs of the NEAMs program. In these cases, ECT may use its discretionary funds to underwrite, in part or in total, the development of the needed tools by one or more of the NEAMS partners.

Scope of ECT

The ECT program element is built around three fundamental pillars: a) codes, libraries, and tools, b) software quality assurance (SQA), and c) computing platforms, cycles, and data management. The goal of ECT is to provide the code teams with sufficient resources in each area to facilitate the construction and use of effective simulation codes and analysis tools.

Pillar1: Libraries and Tools:

Roughly speaking, the codes, libraries, and tools that ECT must provide fall into four broad categories:

- ***Programming tools***: this category includes all the tools necessary to create a productive programming environment that enables physical scientists and computer scientists to work effectively together to build an accurate, reliable high-performance simulation capability. This includes editors and editing environments, such as *emacs*; version control systems (e.g., *CVS*) that enable multiple team members to simultaneously edit the same code routines, controlling access and maintaining a fully recoverable historical succession of code versions; advanced compilers and linkers, including particularly tools that provide for advanced interoperability of code components even when they are written in different languages (mostly *Fortran*, *C*, *C++*, *Python*, *Java*, etc.) and on different architectures; and debugging, code profiling, and memory analysis tools (e.g., *TotalView*, *Vampir*, and *Insure++*, to name one of each).

- ***Model setup***: In this category are found the tools required to create a computational simulation problem from a mathematical or physical description of the real-world problem being simulated. This area includes *CAD* (computer-aided design) routines for mathematically describing the geometry of a problem, *mesh generators* , which replace the mathematical or geometrical description with a computational grid upon which the continuum description of the problem is replaced with a discrete set of gridpoints (or functions) upon which the calculation will proceed; *discretizers* that use the geometric information from the mesh generator and the mathematical formulation of the problem to create one or many systems of equations (linear or nonlinear) that must be solved to find a solution. Included in this category are the *adaptive mesh refinement (AMR)* tools that enable the simulation to calculate at extremely detailed resolution over relatively small areas where a great deal of complicated physics occurs (e.g., in the vicinity of a propagating crack) while using a coarser resolution over the majority of the computational domain, where the physics remains much simpler.

- ***Solvers and time-steppers***: It is commonly the case that at the heart of the simulation lies a large linear system of equations that must be solved at each time step. Most often this takes the form of a linear system of equations, that is, a matrix equation. Modern simulation codes frequently must solve linear systems with millions of equations and millions of variables, at each time step (of which there may be millions). In some problems the systems of equations are non-linear, greatly complicating the finding of a solution. Indeed, these problems are Increasingly complex and require more accuracy as available tools mature. As a result, sophisticated modeling has come to require highly accurate and efficient non-linear solvers. The use of optimization and minimizations as solution techniques is growing in popularity as well, generating an increasing need for effective tools for those functionalities.

- ***Results analysis***: Upon completion of the simulation (and increasingly often periodically during the simulation) it is essential that the scientist have at his or her fingertips tools to make sense of the mass of data the simulation creates. This category includes families of *statistical analysis codes*, applying statistics to the data generated in the simulation and used to deduce important relationships among the variables, the ability to compute and examine derived data values from existing data (e.g., derivative information), *visualization* tools that enable the scientist to "browse" through the results, which are often files of such enormity that they cannot be comprehended at all in their initial form as raw streams of numbers.

The results of our interactions with the other program elements in NEAMS to determine their needs in tools and libraries will be described in more detail in the section entitled "Tools and Libraries for the IPSC Teams".

Pillar 2: *Software Quality Assurance (SQA):*

Software Quality Assurance forms the second pillar of ECT, and is a critical component of any modern software architecture. In this context, SQA consists of providing an overarching philosophy about the construction, debugging, and maintenance of software, providing the code teams with sufficient indoctrination in the overarching philosophy that they want to employ best SQA practices, and providing the code teams with access to SQA tools enabling them to easily write codes that are compliant with those practices.

The NEAMS AFCI Quality Assurance Program Document (QAPD) has chosen a graded approach for quality. There are three quality rigor levels. Quality Rigor Level 3 (QRL3) is for routine R&D, feasibility studies, conceptual designs, exploratory tradeoffs and conceptual modeling. Quality Level Rigor 2 (QRL2) is for technical analysis used to inform policy reporting to congress or stakeholders, analysis for national environmental policy, and critical or controversial decisions. Quality Rigor Level 1 (QRL1) is for codes involved with facility safety, NRC licensing, or benchmarking of a methodology that has potential for NRC approval. Most of the development done in the NEAMS program falls into QRL3 for research and development codes.

The recommended SQA approach for NEAMS is based on experience with ASC research simulation codes. Both programs have the following key characteristics:

- A research and development environment.
- Multi-physics simulation codes.
- Staff consisting of Physicists, Material Scientists, and Computer Scientists.
- Languages used are commonly C, C++, Fortran, and Python.
- Platforms are parallel clusters and High Performance Computing (HPC).

A characteristic of R&D software is that, following the scientific approach of hypothesis → experiment → observation → conclusion, it is iterative in nature and involves trial and error. The software requirements (i.e., desired physics features) require the ability to experiment with the code to achieve an optimum design and implementation, both from a numerical precision and performance perspective. Knowledge gained experimenting with the design often changes the design approach to meet the requirements. Often a researcher is writing code to implement a design that has never before been done. Thus the need to balance the discipline of sound software quality engineering practices with the agility needed to develop a research code can become a major tension within the NEAMS program.

Similar challenges were faced by the DOE Advanced Simulation and Computing (ASC, formerly ASCI) program. In order to meet the requirements of the AFCI QAPD, the SQA approach proposed in the section entitled "Software Quality Assurance for NEAMS" describes and recommends nine key concepts for NEAMS. These concepts incorporate lessons learned and best practices for ASC scientific research codes focusing on "what worked" and minimizing "what did not work" in achieving the goal of balancing agility with software engineering discipline. In addition, we will highlight ASC best practices, industry best practices, and software productivity tools that have proven useful. Finally, in this section, we describe the results of our preliminary discussions with the IPSC teams regarding their current use of SQA tools and their needs in this area.

Pillar 3: Platforms, computing cycles, and data management:

A key element of the enabling technologies program element is the deployment of the necessary compute cycles and storage facilities for high performance computing simulations in each of the other program elements. This pillar comprises two components. First, the NEAMS program must have a clear understanding of the requirements of the IPSCs in this area. For example, what type of simulations are being performed, what are the type of facilities necessary to complete these simulations, how much storage is needed for the massive amounts of data typically associated with large-scale three-dimensional simulations. Second, the NEAMS program will require dedicated compute resources as the IPSCs are developed and realistic simulations are performed that model the physics of interest to nuclear energy engineers and designers. Thus the program cannot rely on obtaining its cycles from other programs such as the DOE's INCITE program and must develop its own user facility.

In the section entitled "Computing Platforms and Cycles" we describe the planned ECT efforts to develop a more complete picture of these requirements. We also describe the computer facilities that are currently installed at the Idaho National Laboratory. It is desired that these resources become more broadly accessible to the NEAMS community as a User Facility, but additional investments are needed to make this happen and the highest priority items are identified. Finally, we describe the results our preliminary survey of these teams as to their anticipated requirements. This survey was informal because many of the teams were in the midst of their planning phases and were not yet ready to provide detailed information. In the section

Using ECT to Increase the Efficiency of the NEAMS Program

One of the reasons that ECT is included in the NEAMS plan is to foster efficiencies among the code teams. The ASCI and SciDAC experiences have demonstrated the value of having commonly used tools available to multiple teams. While this is not a "one-size-fits-all" prescription of what teams *must* employ, it is evident that providing sophisticated tools and capabilities tends to result in the use of those tools. This in turn results in several tangible benefits:

a) Overall, there is a reduction in the duplication of effort across the IPSC teams. This results because teams will not have to spend the time and effort identifying commercial software or publically available shareware that meets their needs; the software packages have been identified and made available to them.
b) Once a team has identified software, acquisition and deployment is often a labor-intensive and time-consuming process. Here again, having a central repository with pre-constructed "build" and installation scripts available makes the process of acquiring and installing software packages much simpler and faster.
c) All of the code teams face certain common and significant challenges, such as problem set-up, meshing, data analysis, SQA, visualization, and the like. While some of these issues can be addressed by commercial software, as in a), many of these problems are sufficiently complex that they involve fundamental research themselves. In these cases, ECT can foster and, budget permitting, assist in the development of these tools, making the resulting product available across the NEAMS community. This leverages

the research dollars spent in one part of the NEAMS community for the benefit of all, and eliminates a considerable amount of duplication of research effort that would otherwise be necessary.

d) Having ECT locating these support software tools (or causing them to come into existence), reduces the distraction of the NEAMS scientists, so that instead of spending their precious (and expensive) research time solving these problems they are free to pursue the fundamental NEAMS research.

e) One of the important benefits of ECT is that it creates a "clearing house" of ECT information
   - by building a team intimately familiar with NEAMS and the ongoing related DOE efforts (for example, ASC and SciDAC), and hence able to make technology from these other programs available to the NEAMS scientists.
   - ECT plans to deploy a web portal through which NEAMS scientists can access the tools and information to fill their ECT needs.
   - ECT staff will work directly with IPSC teams to develop customized solutions when pre-built solutions are not readily available.

ECT Milestones

The ECT program element has defined a set of milestones that will be used to measure its effectiveness at serving the NEAMS community. Milestones have been defined for the ends of Year 1 (FY09), Year 3 (FY11), and Year 5 (FY13) of the program. Naturally, the milestones will be revisited as part of the planning cycle each year to ensure that they remain relevant and the best measures of success of the element. The Milestones originally established are:

- Year 1 (FY09)
  - Gather requirements from IPSCs and set priorities.
  - Develop a SQAP (Software Quality Assurance Plan) based on the DOE Office of Nuclear Energy's AFCI guidelines and appropriate DOE regulations and requirements.
  - For each IPSC, understand the computing requirements and computing. resources available to the code teams.

- Year 3 (FY11)
  - Deploy ECT technologies available from related programs (SciDAC/ASC)
  - Develop NEAMS-related ECT as needed.
  - Facilitate SQAP deployment for the IPSCs.
  - Deploy computational resources at Idaho National Laboratory.
  - Provide advanced tools for software development processes.

- Year 5 (FY13)
    - Incorporate advanced methods for complex geometry and multi-physics applications.
    - Development of algorithms to improve usability & data representation/ understanding.

Summary FY09 Activities

Although NEAMS planning has been ongoing for a considerable period, funding for many of the program elements, including ECT, did not become available until April 2009. As a result, the FY09 activities were limited to the last six months of the fiscal year. In those six months, the ECT team worked to meet the FY09 milestones through the following activities:

- ECT team members attended meetings or interacted regularly with the other NEAMS program elements, including all of the IPSC teams, Fundamental Models and Methods, and Capability Transfer.
    - Because the Fuels and Reactors elements were pre-existing efforts, the ECT requirements of those efforts were readily describable, and the ECT team spent a fair effort cataloging and understanding those needs, as well as beginning to think about possible solutions.
    - Because the Waste Forms and Safeguards & Separations elements were new start-ups, their ECT needs are much less clearly developed. For these efforts, ECT team members are tracking the growth and organization of the element as they coalesce and put together their plans. As these plans materialize, ECT is in contact with the team IPSC leaders to capture the ECT needs as they arrive.
- ECT held discussions on SQA with several of the IPSC teams, and instituted dialog with SQA experts at the laboratories where several of the IPSC efforts are centered. In addition, ECT engaged SQA experts from Lawrence Livermore National Laboratory, with considerable experience with the growth and development of SQA in ASCI, who developed an initial draft SQAP. This document is based on software quality requirements imposed on NEAMS by DOE and the AFCI, as well as an understanding of the relevant DOE regulations and requirements.
- ECT initiated discussions with the Idaho National Laboratory Computer Center and management of INL to determine what is required to position that center to serve as a NEAMS User Facility that will provide supercomputer cycles to the NEAMS community on an as-needed basis.

Pillar 1: Tools and Libraries for the IPSC Teams

What follows is a detailed description of how the ECT element perceives its charter for each of the IPSC teams. For each IPSC, we describe:

1. *The IPSC Scope & Vision*. We briefly describe the purpose of the IPSC effort and how it fits into the "big picture" of NEAMS. We describe the vision of how the fully-functioning code or code suite serves the needs of the community.

2. *Required Technologies*. We list and describe the key technologies that will be required to achieve the vision.

3. *Current technological base*. Here we list and describe the technologies, methods, and tools currently used by the IPSC team to perform the key tasks in the topic area.

4. *Computational Technology Gaps*. We identify the *gaps* between the current enabling computational technologies in use by the IPSC and the technologies described above as being essential to achieve the full functionality of the IPSC.

5. *Candidate solutions*. In this section we lay out possible plans for acquiring, developing, or inventing enabling computational technologies, specifically tools and libraries, to close the gaps identified in item 4. Where possible, we identify and delineate
    a. Solutions crafted by acquiring, assembling, and making available *existing technologies*.
    b. Solutions that require the development and/or invention of *new technologies*. For solutions in this category, we show a possible plan for developing the required new technology.
    c. Solutions employing some combination of existing technologies and new technologies, along with a plan for assembling and creating the necessary components.

Not surprisingly, there are significant areas of overlap where the needs of the different IPSC teams are similar, if not identical. Hence, we will follow the discussions of the capabilities and needs of the individual IPSC teams with a section on "Cross-Cutting" technologies, describing these areas of commonality, the state-of-the-art currently available to the teams, the technological gaps, and the ECT plan for closing the technological gaps in the cross-cut technologies.   Discussion of the SQA practices and needs of the IPSC teams along with the compute needs are found in the sections that follow.

ECT for the Reactors IPSC

Overall Reactor Core IPSC Goal

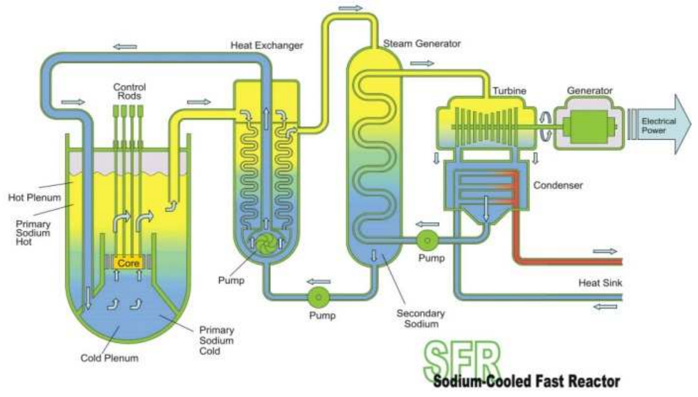


Figure 2 Conceptual diagram of a sodium-cooled fast reactor, illustrating the complexity of the systems that must be simulated.

The reactor core IPSC code team is striving to produce three-dimensional simulations of the reactor core for the design of next generation sodium cooled fast reactors and to understand their behavior in various accident scenarios and thereby ensure their safety. The primary focus is on the development of multi-physics simulation tools that couple thermal hydraulics, neutronics and structural mechanics processes into an integrated simulation tool. The developed tools should support a wide range of modeling fidelity ranging from very high fidelity ("truth") calculations to lower fidelity simulations appropriate for use in design calculations.   The high fidelity calculations will likely require capability level HPC resources to run with the corresponding requirements for compute cycles, data storage, and networking needs. The lower fidelity runs will likely run on small clusters or desktop machines. The team is currently focused on solving problems identified by the reactor design group at ANL including mixing and pressure drop in the fuel rod bundles and thermal mixing in the upper plenum. At the same time, they are working to develop their next-generation simulation framework called SHARP which supports complex geometries, high-fidelity three-dimensional simulations that currently couple thermal hydraulics using Nek5000 and neutronics using UNIC, high performance computing, and advanced visualization of resulting data.

The primary objective of the SHARP framework is to provide accurate and flexible analysis tools to nuclear reactor designers by simulating multiphysics phenomena happening in complex reactor geometries. Ideally, the coupling among different physics modules (such as neutronics, thermal-hydraulics, and structural mechanics) needs to be tight to preserve the accuracy achieved in each module. However, fast reactor cores in steady state mode represent a special case where weak coupling between neutronics and thermal-hydraulics is usually adequate. The SHARP framework design allows for both options. Another requirement for the SHARP framework has been to implement various coupling algorithms that are parallel and scalable to large scale since nuclear reactor core simulations are among the most memory and computationally intensive, requiring the use of leadership-class petascale platforms.

In the sections that follow we describe the SHARP framework and its needs along with the individual simulation components identified to date. In all cases we briefly describe their primary focus and core capabilities, and discuss the current enabling technologies used along with the goals of the component and ECT gaps. We briefly discuss the needs associated with an uncertainty quantification effort undertaken by this team along with a preliminary discussion of their software quality engineering needs and hardware requirements. We conclude by summarizing the high priority requirements for the ECT program element.

Reactor Core Simulation Components

SHARP Framework

Current capabilities: The SHARP framework balances simplicity of adding new physics modules and support for commonly needed functionality (e.g. parallel IO), on one side, with efficiency and minimization of dependencies between modules on the other. Abstractly, the framework includes the following physics components: heat transfer, neutron transport, depletion, and fuel/structural materials. Additionally, there is a complementary collection of utility modules for cross section processing, material properties, mesh generation, visualization, solution transfer between meshes, load balancing, parallel I/O, and unit/integral testing.

The SHARP framework design couples physics together and with other services through the spatial discretization or mesh. A common mesh representation, accessed through a common Application Programming Interface (API) (defined by the ITAPS SciDAC project) is used to preserve flexibility in a number of key aspects, while also providing a common focal point for the coupling activities. The SHARP framework uses the MOAB implementation of the ITAPS mesh API which provides a mesh representation which is both highly memory- and cpu time-efficient. The design preserves as much flexibility as possible, by allowing both loose and tight coupling, on the same grid or different ones.

The code system uses common services available through the Mesh API, including a data coupler to couple data from libPhys1 and libPhys2. The overall solution process and passing data between physics modules is coordinated by the coupled driver. All modules are compiled into a single executable, which then runs as a single MPI process. The top-level execution flow is controlled by the SHARP driver, which successively calls the individual modules' API functions. To date the reactor modeling team has successfully coupled two physics modules, Nek and UNIC, and is running small (32 processor) test problems on the Jazz cluster at ANL.

The team explored the use of the CUBIT mesh generation toolkit from Sandia National Labs to support two models for neutronics and fluid flow computations. The Advanced Breeder Test Reactor (ABTR) model consists of hexagon- shaped assemblies of various types (inner and outer fuel, control, reflector, and shield), each assembly consisting of a hexagonal lattice of pins inside a hexagonal duct wall. Because of the relatively straightforward layout of simple hex-shaped primitives, a geometric model for this core was developed inside CUBIT (more complex geometric models would normally be developed in a CAD tool like Pro/Engineer). Generating this model with CUBIT was a very memory-intensive process, requiring almost all of the 4GB of RAM on a 64-bit workstation even without the use of graphics. Generating a mesh for this model required over 5GB of memory, even for a moderately-sized mesh of several million elements. Other problems encountered in this effort include:

- Problems generating relatively coarse meshes for hexagonal-shaped annular surfaces, resulting in a great deal of interactive effort to tune the geometric model to avoid these problems.
- The inability to mesh a collection of stacked hexagonal prism-shaped volumes, when the meshing schemes have been specified as e.g. sweeping surface “A” to “B” but surface “B” is meshed first. This forced the generation of the mesh for each reference assembly type (fuel, control, etc.), before the assembly was copied and moved in the hexagonal lattice of assemblies. This generation method was sufficient for this core only because of the separation of assemblies by a thin sodium region.

After struggling with these problems, it was decided to use CUBIT only for surface mesh generation. A standalone tool was used to sweep this surface mesh into the third dimension.

Visualization for the SHARP framework is currently performed using the VisIt tool from LLNL. Preliminary work shows this tool is successfully displays the large data sets generated by the coupled simulations performed here and that some aspects of the ITAPS mesh data model can be supported through VisIt reader plugins.

ECT Gaps: The team requires advanced meshing capabilities that are custom tuned to the problems of interest to GNEP. Both lower-memory and fast algorithms for geometry creation and mesh generation (such as move and copy) would be of use. In addition, advanced coupling techniques that support higher order elements, different basis functions, and larger scale parallelism are a high priority. While preliminary work has been undertaken to modify VisIt to support the ITAPS mesh data model; more work needs to be done to support this more fully; particularly in the area of supporting entity sets and tagged data.

Thermal Hydraulics

Current capabilities: The thermal hydraulics component currently used in reactor core modeling is the Nek5000 software (http://nek5000.mcs.anl.gov/index.php/Main_Page). The primary focus are very high fidelity simulations to achieve a detailed understanding of mixing associated with the wire wraps in the fuel bundles and mixing in the upper plenum. Nek5000 is a computational fluid dynamics software package that uses body-conforming hexahedral elements and high-order spatial discretizations via spectral element technology. The primary fluid regimes addressed with this software include incompressible and low Mach number (variable density) flows with LES and RANS turbulence models. The Nek5000 team has invested considerable effort in producing a highly efficient code with state-of-the-art solvers and preconditioners (multigrid and scalable coarse grid solves), scalable communication kernels, low memory footprint, and scalable memory design. The code scales to over 100,000 processors on the IBM BG/L system. The code currently uses the CUBIT software (http://cubit.sandia.gov) for mesh generation, MOAB for coupling to the neutronics component, and VisIt for visualization of very large data sets.

ECT Gaps: The vision for this software is to couple it to both neutronics and structural mechanics components (e.g. Diablo, LLNL). To achieve this vision will require a mesh-to-mesh transfer capability between the fluid and mechanics codes. In addition, the Nek5000 component itself requires new mesh generation capabilities that provide high-order hexahedral meshes.

Neutronics

Current capabilities: The reactor core team is developing a new state-of-the-art unstructured neutronics solver called UNIC with the ultimate goal of enabling significantly lower uncertainty margins in the analysis of newly proposed reactor designs. UNIC is a three-dimensional deterministic neutron transport code that models the second-order form of the transport equations using finite element discretization techniques. To solve the transport problem, the Boltzmann equation can be modeled using several different techniques including both spherical harmonics (1, 2, and 3D) and discrete ordinates methods (2 and 3D). Spherical harmonics expansions fit well with the diffusive nature of problems often encountered in reactor physics, especially when homogenization approaches are used; they are also hierarchical, making them ideal for multiresolution and adaptive mesh techniques. Discrete ordinates methods have the advantage that they are comparatively cheap in memory and can more accurately treat the detailed flux distributions for heterogeneous geometries. The resulting algebraic systems are solved using the Portable, Extensible Toolkit for Scientific computing (PETSc) software (http://www.mcs.anl.gov/petsc) and typically over 80% of the total solution time is spent in the preconditioned conjugate gradient solvers. The overall performance is determined largely by the choice of preconditioner, which must be guided by the physics of the problem. UNIC uses domain decomposition-based preconditioners (for example, additive Schwarz methods) for scalability. Mixed linear tetrahedral/hexahedral meshes are generated using the CUBIT software and are partitioned by using the MeTiS package, which attempts to minimize the communication while balancing the computational work load on each processor. UNIC has been designed to scale seamlessly from desktop to very large numbers of processors (currently has been run to 80,000 processors of BG/L) which allows reactor analysts to choose the level of desired fidelity.

ECT Gaps: A number of advancements that involve enabling computational technologies are needed to improve the overall algorithmic performance of these solvers. The highest priority is the development of more efficient preconditioners that leverage the sparsity pattern in the matrices to accelerate solution. Candidate software solutions include the use of algebraic multigrid such as the solvers found in the hypre software, which can be downloaded from https://computation.llnl.gov/casc/linear_solvers/overview.html. However, due to the special form of the problems given here (namely that the first order form is not symmetric), hypre is not likely to work immediately and some development and customization is required. In addition, the mesh generation software needs to be tuned for the neutronics problem as it is currently generating too many elements for this type of analysis. The team would also like to explore alternate load balancing strategies such as those provided by the Zoltan software package (found at http://www.cs.sandia.gov/zoltan) and explore hybrid programming models (for better algorithmic convergence rates) and memory reducing algorithms for the matrix-vector products.

Fast Reactor Safety Analysis

Current capabilities: The safety analysis simulations are focused on core disruption issues, sodium boiling and dry out, cladding failure, fuel failure, protected and unprotected loss of flow, and oxide fuel deformation, disruption and material relocation. The current technology is based on the legacy code SAS4A which represents 100s of man years of effort. During many accident scenario simulations, a large number of interrelated physical phenomena occur during a relatively short time. These phenomena include transient heat transfer and hydrodynamic events, coolant boiling and fuel and cladding melting and relocation. Heat transfer in each pin is modeled with a two-dimensional (r/z) heat conduction equation. Single and two-phase coolant thermal-hydraulics are simulated with a unique, one-dimensional (axial) multiple-bubble liquid metal boiling model. The transient fuel and cladding mechanical behavior model, integrated with fission product production, release, and transport models, provides prediction of fuel element dimensional changes and cladding failure. Fuel and cladding melting and subsequent relocation are described with multiple-component fluid dynamics models, with material motions driven by pressures from coolant vaporization, fission gas liberation, and fuel and cladding vaporization. Reactivity feedbacks from fuel heating (axial expansion and Doppler), coolant heating and boiling, and fuel and cladding relocation are tracked with first order perturbation theory. Reactivity effects from reactor structural temperature changes yielding radial core expansion are modeled. Changes in reactor power level are computed with point kinetics. Numerical models used in the code modules range from semi-implicit to explicit.

The coupling of modules in time is semi-explicit within a multiple-level time step framework. It uses simple loop models for reactor coolant systems and balance of plant thermal hydraulics.

Vision: The vision for the next generation accident scenario code is to focus on higher-order, higher-resolution tools which work together in a multi-physics, multi-scale framework. Ideally high-fidelity neutronics codes will be used to model the details of the core region in three dimensions, the thermo-hydraulics will be modeled using advanced CFD and turbulence models in selected regions of the reactor along with three-dimensional structural mechanics in selected regions of the domain. Lower fidelity codes will be used to model whole-core transient behavior and will be coupled to one- or two-dimensional models in the remaining reactor regions.

ECT Gaps: The ECT gaps that must be filled to make this vision a reality include advanced meshing capabilities and design tools that can handle the conflict between the fine scale features of the wire wrap and the coarser scale needs inside the fuel pin. In addition, coupling tools are needed to join high and low fidelity models together with the flexibility to plug-and-play different models together (note: this latter is more related to the framework being developed by the capability transfer program element). Lower fidelity models need to leverage multi-core desktop compute platforms and the tools in general must port from the desktop to HPC systems such as the IBM BlueGene series.

Uncertainty Quantification for Reactor Modeling
Current capabilities: The reactor core team is currently exploring the use of stochastic finite element techniques for uncertainty quantification with particular application to heat distribution in the nuclear reactor core. The goal is to understand the normal functioning of the reactor and distance between normal operating conditions and the designed safety margins. These studies result in very large systems with high dimensions with 100,000 to millions of parameters. Such problems cannot be studied directly and so adjoint sensitivity analysis is being used here. It is currently implemented for older codes, primarily neutronics systems and less so for thermo hydraulics and structural problems.

ECT Gaps: This group would like to add derivative information at each point to reduce the number of sampling points used in the current Monte Carlo techniques. To do this, the team would like tools that set up adjoints as automatically as possible through the use of automatic differentiation tools such as ADIFOR and ADIC (http://wiki.mcs.anl.gov/autodiff).

Tools and Libraries Summary for the Reactor IPSC
The highest priority items identified by the reactor core team for ECT are:

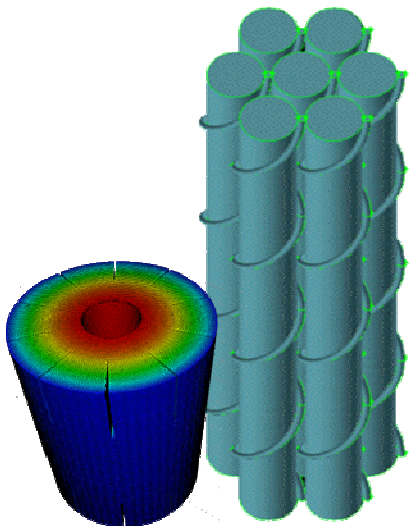
- Advanced geometry creation and mesh generation (perhaps through modifications to CUBIT if it can be released as open source)
- Mesh to mesh coupling (high order elements, different dimensions, different fidelities, different implementations of solvers)
- Visualization tools that support the reactor framework data model
- Advanced preconditioner technologies for neutronics modeling

ECT for the Fuels IPSC

Overall Fuels IPSC Goal

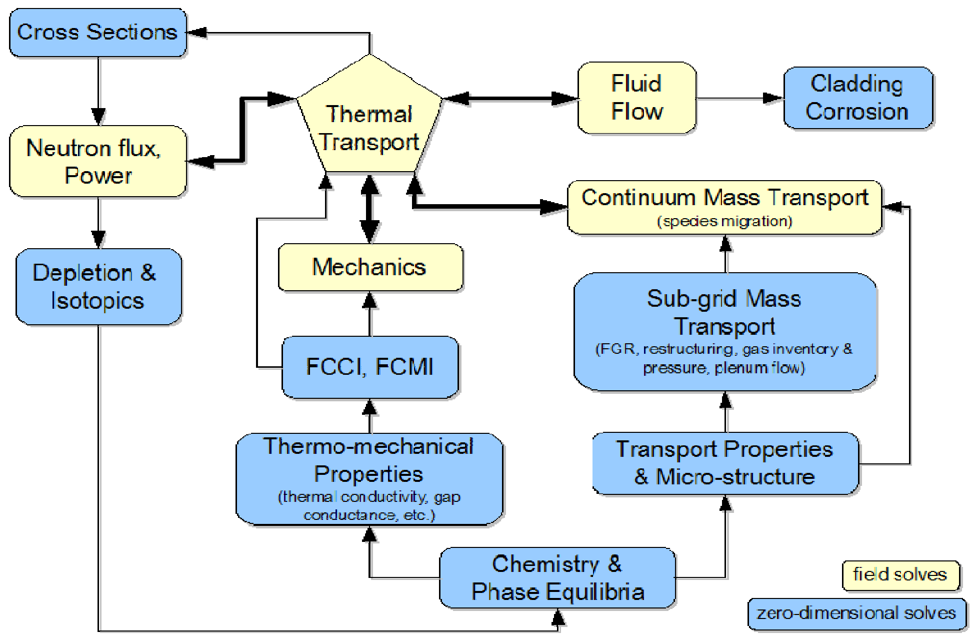
The Fuels IPSC team aims to develop a three dimensional predictive tool for the simulation of pins and assemblies applicable to existing and future design nuclear reactors in both normal and abnormal operating conditions. This tool will be used for the design, analysis and certification of new nuclear fuels; and will result in a process that drastically reduces the time required in the qualification of the proposed nuclear fuels. The new code needs to be validated for life extension problems and safety margins of future advanced fuels, simulating fuel failure phenomena such as swelling and cladding.

The Fuels IPSC will simulate the radical material and the microstructural and chemical changes that nuclear fuel undergoes both during manufacturing and irradiation. These physical phenomena are intrinsically *multiphysics*, coupling thermo-mechanics, neutronics, thermal hydraulics, and species transport, and *multiscale*, spanning many length scales (microns to meters) and time scales (minutes to days).


ages to be
uels IPSC

The current industry standard code is FRAPCON, a highly complex 1.5-dimensional code that is non-scalable and hard to use, with physics capabilities that although comprehensive are also highly simplified, material specific and empirical. The Fuels code under development will replace FRAPCON and provide an HPC alternative for the analysis of different fuel cycle scenarios: chemical forms, metal alloys, ceramics (oxides, nitrides, carbides) and mechanical forms (pellets, sphere-pack, micro-pack). This new Fuels IPSC will be science-based and validated. The team plans to release the code in three stages: (a) an initial limited version with near-term capabilities will be offered in 2010; experience with this code will inform (b) a newer version with intermediate capabilities to be delivered in 2015 (with a prototype in 2012); and (c) an advanced-capabilities final release code will follow in 2022.

As mentioned above the classes of simulations targeted by this code are multiscale: *microscale* to describe material structural changes, transmutations, fission, and neutron transport, and *macroscale* to address long range transport (power, species), nuclear power generation, and thermal mechanics. For purposes of code design, these different scales are treated as weakly coupled both at the micro/macro interface (e.g., material structural changes serve as input information to the thermal mechanics loop) and at the macro scale. Consequently, a modular approach was chosen for the design of the code: using independent multiphysics modules that interact through a simplified framework.



:ode

Some of these modules will be leveraged from existing efforts funded by other programs (e.g., ASC, SciDAC, Office of Science base program) and from legacy codes, while others will be developed as needed. For example, the team is working on thermo-mechanical-chemical, transport and flow modules and plans to reuse existing legacy codes such as the zero-dimensional models ORIGEN, MATPRO, and ChemApp. ORIGEN is a scalar code that tracks the destruction and creation of over 2000 isotopes due to fuel irradiation and material activation and provides elemental source terms for chemistry ChemApp and MATPRO modules. MATPRO is a library of fuel and cladding material properties as functions of temperature and irradiation. It is implemented as a library of functions where the functional forms are calibrated to experimental data. ChemApp is a library for the calculation of multicomponent, multiphase chemical equilibria and their associated energy balances.

The team is using NEWTRNX, a 3D deterministic code, as a placeholder while work on the neutronics package is in progress.

The design plans call for the development of a "thin" framework to facilitate the coupling of the physics modules, providing abstractions for solver interfaces and "domain state" component. This framework will likely feed from existing interdisciplinary efforts in the SciDAC (Common Component Architecture) and ASC programs, and will incorporate aspects from established framework efforts such as ADVENTURE, SHARP and MOOSE. The evaluation of these frameworks and identification of components to be adopted is an important focus of the code team at this time. A brief description of the frameworks follows:

- The ADVENTURE system is based on a hierarchical domain decomposition method (HDDM). This framework partitions the problem into non-overlapping subdomains, analyzes the subdomain problems (local problems) with direct solves and enforces the compatibility between subdomains (coarse problem) via an iterative solve. ADVENTURE assumes a finite element discretization and uses CAD model tools, TetMesh, Metis (for partitioning) and its own tools for I/O

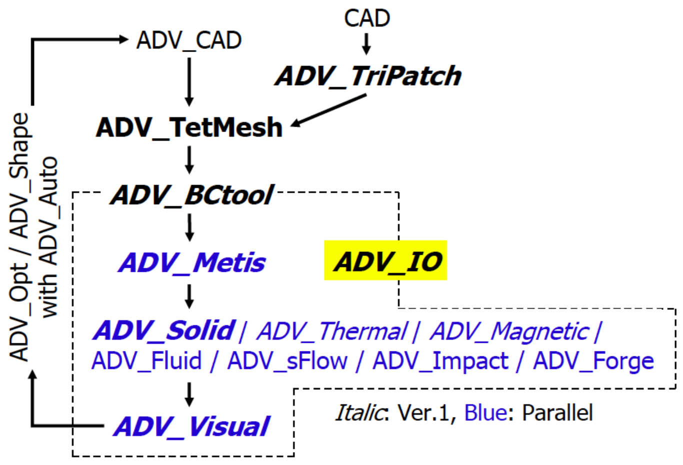

ADV_CAD
CAD
ADV_TriPatch
ADV_TetMesh
ADV_Opt / ADV_Shape
with ADV_Auto
ADV_BCtool
ADV_Metis
ADV_IO
ADV_Solid / ADV_Thermal / ADV_Magnetic /
ADV_Fluid / ADV_sFlow / ADV_Impact / ADV_Forge
ADV_Visual
Italic: Ver.1, Blue: Parallel

Figure 5 Flow diagram of the ADVENTURE framework

- MOOSE is a framework for hosting tightly coupled multiphysics applications that provides a simple development interface accessible to engineers and increases application robustness. This framework assumes that the same underlying mesh is shared throughout the application while allowing for different time/spatial scales. It takes advantage of software libraries shared by the scientific community, such as the solver libraries Petsc and Trilinos, and the meshing tool libMesh.

- SHARP was described in the prior Reactors IPSC section.

Software requirements.

Meshing is one of the more important issues to be addressed in the development of the Fuels code. Various physics modules use different types of meshes (e.g. tetrahedral versus hexahedral elements) and often require different levels of resolution. For example the structural mechanics and neutronics codes require different types of meshes of vastly different resolution: fine scale resolution for structural mechanics and flow, and a coarser mesh for neutronics (with many energy groups per mesh element).

The multiscale nature of the simulations also introduces the need for numerical algorithms and time stepping strategies that allow for a proper coupling of the scales. In particular, the codes upscaling information from sub-grid models and coupling at the meso scales must contain accurate and efficient tools for those tasks.

For the code to be truly predictive, it needs to employ Uncertainty Quantification (UQ) and Sensitivity Analysis (SA) capabilities. Module coupling and time-scale bridging complicates both UQ and SA studies.

The code needs to be customer friendly (i.e., have an “easy” user interface) in order to be effective in a community far removed from the development of the code itself.

Finally, as an overarching requirement, the code effort also needs access to programming models and programming languages that facilitate the development of multiphysics, modular simulation codes.

ECT Summary for the Fuels IPSC

We have identified several areas where the Enabling Computational Technologies component could help in the Fuels modeling effort:

- The first barrier for a potential user to adopt the fuels (or any other) code is the interface with the code itself. ECT can help the code team design and develop an easy-to-use GUI-based interface for code running and code steering.
- The Fuels code team is interested in tools that would improve productivity and position the fuels code to satisfy the software engineering requirements of the user base. ETC can provide SQA methodologies that, if followed, will satisfy the formal Program software engineering requirements. In particular, ECT can provide tools for including unit testing, regression testing, automake, software documentation, and a host of SQA functions.
- The Fuels code requires advanced meshing capabilities. In addition the coupling of multiple physics packages having diverse mesh requirements (e.g., neutronics and structural mechanics) will also require mesh to mesh transfer capabilities.

- The loose coupling (thin backplane) approach taken for the Fuels code means that each physics model would have its own data structure. This will require visualization and data analysis tools able to access a multiplicity of data structures. ECT has already started work to make sure that VisIt satisfies these requirements (supporting ITAPS mesh data model, ADVENTURE output, etc.) and can serve as the visualization tool of choice. More work needs to be done in this area.

ECT for the Waste Forms IPSC

Overall Waste Forms IPSC Goal

The objective of the NEAMS Waste Forms (WF) IPSC is to provide an integrated suite of computational capabilities for simulating the long-term performance of waste forms in the engineered environment of a waste storage or disposal repository. This suite will include first-principles codes for characterizing the properties of the waste form, high-fidelity modeling of coupled degradation and transport phenomena, and a set of efficient surrogate models of known accuracy in well-specified performance assessment regimes. The surrogate models will enable production of simulation results with which quantified predictions can be made. The ultimate goal is to support predictive simulation-based, risk-informed decision making about managing future US nuclear waste.

Three levels of simulation fidelity will be used in the WF IPSC. Together with experimental data, sub-continuum simulations of *maximum fidelity,* continuum physics models of *high-fidelity*, and, at the lowest fidelity level, *surrogate simulation components*, abstracted from the high-fidelity simulations.

The sub-continuum simulations and experiments will be used to characterize material properties and mechanistic processes significant to waste forms in repository environments. In particular, they must determine key state and transport properties needed to model the evolving isotopic composition within the waste form matrix and along repository environment pathways. Understanding of these properties is necessary to specify accurate constitutive relationships for high-fidelity continuum models. Many of the sub-continuum simulation capabilities necessary to develop this understanding are primitive or do not yet exist; it is anticipated that they will be developed and supplied by the Fundamental Methods and Models (FMM) element of the NEAMS Campaign.

Knowledge of the state and transport properties resulting from the sub-continuum work will be used to develop and verify high-fidelity macro-scale continuum physics models. These models will be used to investigate multi-physics processes that couple thermal, hydrological, chemical, and mechanical (THCM) phenomena in repository environments. The results of these simulations will be used to identify the relevant governing equations, constitutive models, and accuracies required for abstracting waste form surrogate performance assessment models.

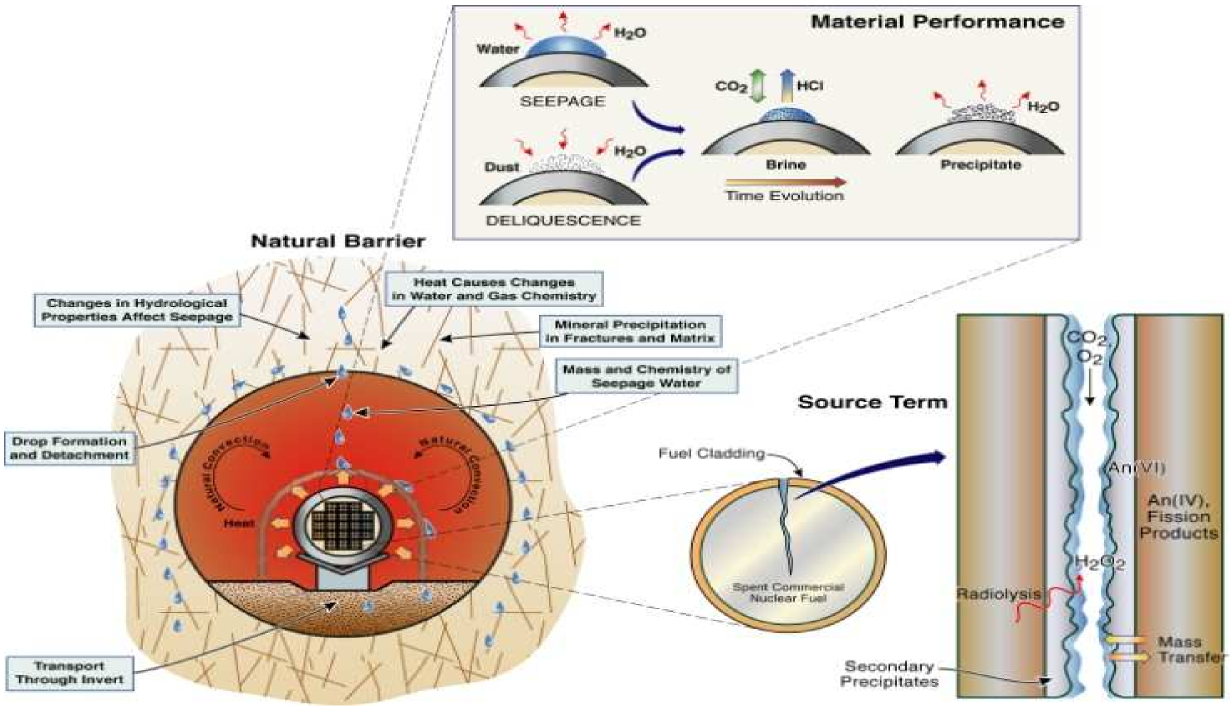


Figure 6 The Waste Forms IPSC must simulate the design of waste products and their storage environments for safety and stability over exceptionally long time spans.

At the lowest level of fidelity, *surrogate simulation components* are abstracted from the high-fidelity simulations. The surrogate simulations are intended to be “robust and fast,” and can be used for performance and design assessment analyses over large many combinations of waste forms and repository environments. Naturally, the surrogate simulations must be verified against corresponding high-fidelity simulations. The surrogate THCM performance assessment (PA) simulations are expected to employ several alternative modeling approaches, ranging from the use of high-fidelity continuum models together with reduced-dimension realizations of the waste form and environment to the use of abstracted models that are simple calibrated response-surface functions. The surrogate PA models will be self-contained modules that are flexibly linked together to simulate specified waste forms and disposal scenarios to a specified accuracy. A primary computational efficiency objective in the design of the PA codes is to have the capability to efficiently simulate hundreds or even thousands of distinct disposal scenarios.

Both the integrated high-fidelity THCM simulation codes and the surrogate PA codes will have embedded sensitivity analysis capabilities to support verification and validation (VV), uncertainty quantification (UQ), and design optimization analyses. Again, many VV and UQ functions are either primitive or not yet extant; development of the embedded sensitivity analysis capabilities and UQ tools will be in collaboration with the Verification, Validation, and Uncertainty Quantification (VU) element of the NEAMS Campaign.

A special purpose WF IPSC framework will be developed to support analysis of key phenomenological processes. The WF IPSC framework architecture will include consideration of inter-fidelity coupling, THCM multi-physics coupling, and a workflow framework. Modular simulation components will plug into the WF IPSC framework through well-defined interfaces. It will be essential for this framework to ensure transparent, traceable, reproducible, and retrievable simulation results in order to satisfy regulatory compliance requirements such as those associated with the Yucca Mountain Project (YMP) and Waste Isolation Pilot Plant (WIPP).

The WF IPSC will use software quality engineering (SQE) best-practices to develop high-confidence software components, coordinate large distributed development teams, and respond to evolving requirements.

Waste Forms Simulation Components

Sub-continuum Materials Properties Characterization

Current capabilities: The sub-continuum materials properties characterization effort must provide a host of reliable quantified descriptions (models) for an extremely broad range of sub-continuum physical phenomena, spanning chemistry, physics and materials science. The sub-continuum research tools that are required to simulate many of these phenomena are deterministic, “first principles” methods whose accuracy is fundamentally limited by the fidelity of the physical model rather than the parameterization of that model. A wide scientific community already uses an extensive collection of codes, methods, and techniques to model many of the phenomena of interest to waste forms, and the sub-continuum properties characterization effort will leverage this body of work whenever possible. Today, however, research tools at the sub-continuum scale are often in a continuous state of development, constantly implementing improvements to the physical approximations that underlie the methods. As a result, the current state of the development and application of methods to simulate sub-continuum phenomena is highly dynamic and distributed widely among many institutions. This poses special challenges for the WF IPSC, where the process requires detailed traceability with documented reproducibility and propagating verified, validated quantified uncertainties.

The FMM element of the NEAMS campaign is charged with furnishing simulation capabilities at the sub-continuum scale. It is anticipated that WF IPSC and FMM will collaborate to identify crucial gaps in sub-continuum simulation capabilities and any significant code or method development needed at these scales will be undertaken in coordination with FMM.

Moreover, many of the computer codes used in sub-continuum simulations relevant to waste forms will be commercial or otherwise proprietary codes, where the source code may not be directly accessible, or open source tools codes without formalized software quality practices, or "research" codes developed and used by individual investigators without any formal distribution. This heterogeneous software environment must be integrated into the materials properties characterization needed for the WF IPSC modeling suite, but it poses a daunting challenge for a controlled overall simulation system required to propagate uncertainties through a hierarchy of simulations. Regardless of the origins of the simulation code, all simulation results that enter the data flow of the WF IPSC process will be required to demonstrate qualification: documented verification evidence of suitability, establish reproducibility and traceability, and validation for application of the code to the computation of the quantities of interest.

High-Fidelity Continuum Models

Current capabilities: It is not possible to use sub-continuum materials models for large scale "waste form to near-field scale" simulations due to the length and time scales involved. The sub-continuum models will help provide parameterization for constitutive models used by the high-fidelity THCM models and, moreover, will identify and give improved understanding of the phenomena that are incorporated into the physical abstraction represented by a constitutive model. Hence, the fidelity of a constitutive model is dependent on both an abstracted physical model and the numerical realization of that model.

Upscaling—communicating information between different temporal and physical scales—is central to the success of the WF IPSC. But current upscaling techniques are mostly *ad hoc*, application-specific, and are generally not adequate for coupled non-linear process in heterogeneous media. New computational tools and methodologies are needed for linking different scales and representing processes to obtain high-fidelity predictions beyond the range of conditions and scale at which models and parameterizations were developed. To date, success stories are few.

The central high-performance part of the WF IPSC code suite is the high-fidelity THCM layer in the modeling hierarchy. The high-fidelity THCM code will model non-linear, highly coupled physical phenomena, and be composed of multiple interacting software components. The exceeding complexity of the phenomena and software used to simulate it and the numerical challenges of modeling a non-linear, highly coupled environment place stringent demands for detailed verification and validation. It is anticipated that much if not most of this code will need to be developed and deployed under the auspices of WF IPSC, and thereby directly benefit from use of sophisticated software quality engineering practices.

The development of the high-fidelity THCM software will follow sophisticated software engineering practices, with version control, comprehensive test suites, unit testing, regular regression testing, build tests, etc., so that to the extent possible verification will be built in. An infrastructure will be constructed for the development of the software that will enforce these practices. Typical model validation compares a model's results with experimental measurements and/or field observations. However, such measurements will be impossible to obtain for the high-fidelity THCM software and the Integrated Assessment Code at the (geologic) temporal and spatial scales of interest for post-closure repository performance. Validation for short-term WF performance will be demonstrated using all available data for short time scales, and will entail coordination with a robust experimental program. Predictions of the WF IPSC modeling suite at geological scales will be extrapolated from this foundation and will be validated using corroboration, technical review, and natural analogues.

Surrogate Performance Analysis Codes

Upscaling from high-fidelity continuum models to the surrogate models that will be used in the Integrated Assessment Code will make use of and, where appropriate, extend approaches similar to those used previously for other nuclear waste programs [5,10]. It is envisioned that these approaches can range all the way from relatively simple cases on the lower end to the more sophisticated at the higher end. At the simplest end of the range, the surrogate model may use something as straight-forward as a lookup table and interpolation of results that are provided by the Hi-Fi continuum code (e.g., the porosity surface method used by WIPP to indirectly couple mechanical closure with two-phase fluid flow calculations). At the more sophisticated end, full-up system-scale coupled continuum analyses may be performed by the surrogate model albeit with significantly coarser discretizations (and perhaps different partitioning of materials to represent the system and different solution techniques) than those used for the high-fidelity continuum model. The Integrated Assessment Code is intended to execute very quickly for the purpose for generating statistical information needed for the performance assessment and decision analysis. The surrogate models contained within it are meant to mimic the essential features of the high-fidelity THCM simulations with sufficient accuracy to satisfy certification requirements to a desired confidence.

Uncertainty Quantification

There are two approaches to UQ: a sampling-based approach and embedded approach. In a sampling based approach to UQ, simulations are used as black boxes and the calculation of response metrics of interest is based on a set of simulation response evaluations. Hence sampling-based approaches have been traditionally used when very complex and extensive applications already exist and retrofitting them to gain access to internal variables in the code is not feasible. The embedded approach requires new solvers/codes designed for the reformulated system of equations. The advantage of the embedded approach is that it finds the representation of model outputs by a one-time solution of the reformulated model. The IPSC intent is to research, evaluate, and pursue the embedded approach. A byproduct of uncertainty quantification will be sensitivities of the simulation results to the various input parameters. The sensitivity analyses will be used to identify important phenomena/parameters and help refine the PIRT, and the SA also will be used to quantify the requirements for sub-continuum scale evaluations of the quantities of interest within the constitutive models.

Tools and Libraries Summary for the Waste Forms IPSC

The Waste Forms IPSC team has identified several categories into which their software requirements fall:

- Workflow management
- Multi-physics coupling
- PDE modeling and solvers
- Pre- and post-processing
- Numerical solvers
- Sub-continuum
- SQE software
- VU tools

The IPSC has identified a set of candidate software packages to fill these needs. For example, they are considering:

- Salome (Workflow management, Multi-physics coupling, Pre- and post-processing)
- Sierra (Multi-physics coupling, PDE modeling and solvers, Pre- and post-processing)
- Trilinos (Multi-physics coupling, PDE modeling and solvers, Pre- and post-processing, numerical solvers, SQE software, VU tools)
- Dakota (VU tools)
- ITAPS (PDE modeling and solvers)
- CUBIT (PDE modeling and solvers, Pre- and post-processing)
- Common Component Architecture (Workflow management, Multi-physics coupling, PDE modeling and solvers, Pre and post-processing, Numerical solvers)

The IPSC has embarked on a “gap analysis” of their computational technology as a major FY10 undertaking, and also plan to perform a detailed comparative analysis of the tools mentioned above as candidates to fill their software needs.

ECT for the Separations and Safeguards IPSC

Overall Safeguards and Separations IPSC Goal

The vision of the Safeguards and Separations IPSC team is to develop the next generation of fuel cycle separation and waste management technologies that enable a sustainable fuel cycle with minimal processing, waste generation, and potential for material diversion. In some ways this is the IPSC with the greatest diversity of mission. While they must undertake multiphysics simulations, for example, to discover the fundamental nature of the separation processes, by which the various species of fission products can be teased out of the waste package, mitigating the harmful effects of storing highly radioactive waste long term, this code team must also develop tools to model and improve on the accounting and accountability systems that track the locations and quantities of nuclear storage; this latter work involves an entirely different mindset, and a different set of tools, than the more traditional physics-based simulations.

The Safeguards and Separations IPSC is newly begun in FY09, and hence has less history than the Reactors and Fuels teams. As a result, they are still in the process of determining their computing needs (both hardware and software tools). The fundamental approach, however, represents a dramatic change in thinking from the predecessor programs. In previous programs, computational support for project elements concerned with separations have focused on modeling the UREX (uranium extraction) process and on developing flow sheets that follow and track special nuclear material throughout the fuel cycle. This previous focus has employed testing with used fuel, employing only a limited understanding of fundamental properties.

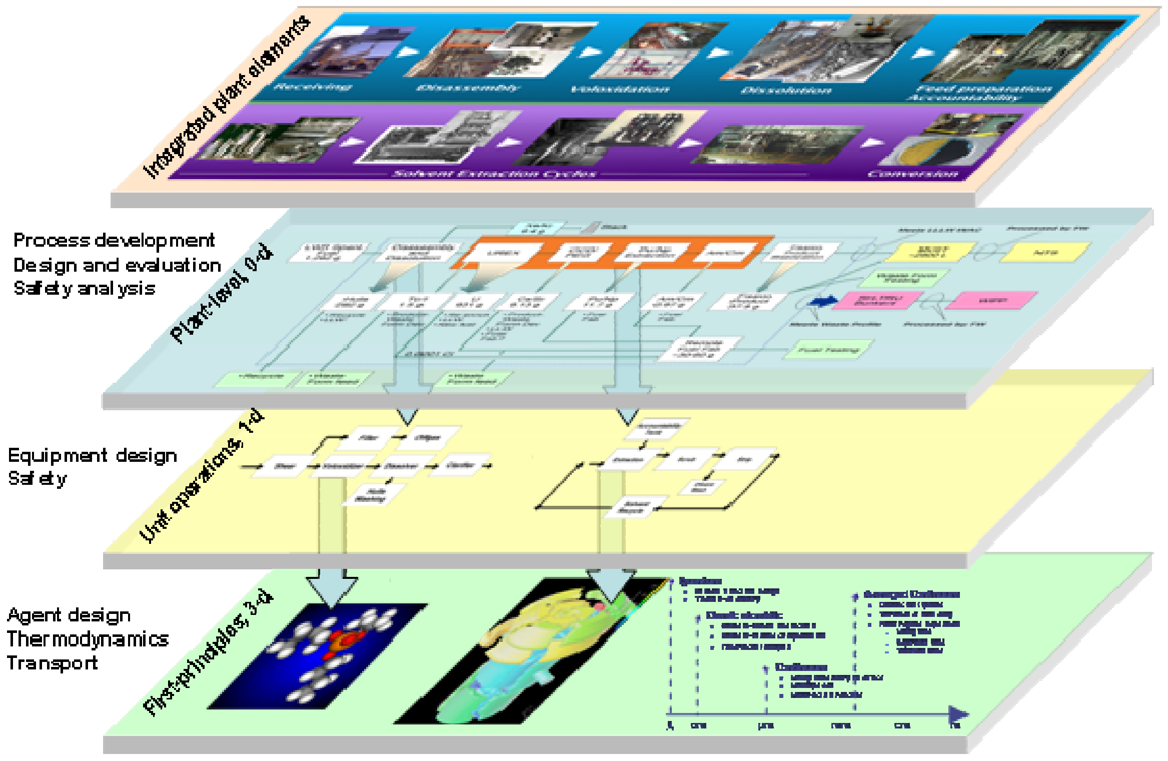


Figure 7 Conceptual schematic of part of the Safeguards and Separations responsibilities, illustrating the variety and complexity of the tasks before them.

The Safeguards and Separations IPSC is taking a new approach. While the team still needs to do flow sheet tracking of materials, e.g., the primary focus is moving from empirical examination of data to an approach based on fundamental science. The team is employing a research and development focus based on understanding the fundamental physics underlying the separations process, and have set goals to create efficient modeling routines to develop a fundamental understanding of both aqueous and electrochemical separation methods.

Modeling and Simulation Needs

The Safeguards and Separations IPSC teams have identified a host of modeling and simulation needs. A selection of the most pressing needs is listed below:

- Unit operations models
- Subscale chemical & physical models
- Flexibility to expand to new chemistries
- Dynamic process models of separation processes
- Plant simulation models for plant designs

Tools and Libraries Summary for the Safeguards and Separations IPSC

ECT needs for the Safeguards and Separations element encompass a wide variety of codes and activities. Among the most obvious ECT needs are:

Language Interoperability: Due to the wide variety of the Safeguards and Separations activities the IPSC must simulate or monitor, there are already numerous extant codes dealing with many parts of the process. But these legacy codes span a wide range of platforms, architectures, and operating environments. They are also likely to be written in any of a lengthy list of languages or wrappers. While some of the Safeguards and Separations codes are written in the traditional C and C++ languages, a good number are in other languages, including Excel, Simulink, Octave, and Fortran, among others. While many of these are written as stand-alone units, it will be increasingly important that they be linked (or at least linkable) as the team develops full end-to-end simulations. Hence, a growing need is for tools (for example, LLNL's *Babel*) that facilitate the common use of the tools in larger, overarching packages.

Graphical Interfaces. Many of the codes for Safeguards and Separations involve the user selecting from a number of different options to produce a suite of results rapidly. These codes would benefit from having a consistent set of graphical user interfaces allowing the tools to have a common "look and feel" that users can be familiarized with easily. This allows the user to quickly learn to operate a number of codes spanning a range of the proceses.

Visualizations of the simulations. Some of the simulation codes in this area are significantly complex, modeling the fundamental physics and chemistry of the separation processes. They will take considerable resources to run, and will generate fairly large data sets as their output. Analyzing the results of these simulation runs is dependent on having effective analysis tools, starting with having a good method for visualizing the results and exploring the output datasets.

Common themes among the ECT needs

One of the principal activities of the ECT team since funding commenced in April, 2009, has been gathering the information presented above regarding the goals, current capabilities, and ECT gaps of each of the IPSC teams. Not unexpectedly, a number of commonalities or similarities emerged in the ECT gaps facing the code teams. In this section we briefly identify the most obvious commonalities. In these cases, it should be feasible that the similarity of the needs would lead to common technological solutions that can be shared among the IPSC teams. In the longer view, these common solutions can form a toolkit available not only to the IPSC teams, but to the greater NEAMS community, including end users.

- **Meshing tools**. Each of the IPSC teams, in some application or another, indicated a need for meshing tools. Two types of tools were particularly in demand.
    - **Mesh generation** has special characteristics for many NEAMS applications. This is due to a number of factors, depending on which team's applications are examined. For example, the Fuels IPSC is modeling complicated systems in which the fluid dynamics of coolant over intricate geometries of bundled fuel rods wrapped in wires must be computed. The computational mesh for this modeling is extremely difficult to generate, and requires intense human intervention. Existing mesh generation tools lack the robustness to handle the special characteristics that naturally occur in several NEAMS applications.
    - **Mesh-to-mesh coupling** is necessary in several NEAMS activities for multiscale or multiphysics coupling. Easy and accurate methods of moving information computed on a grid at one scale to a grid at a different scale are important. Another very common reason this is a critical technology is the intricate geometric shapes that must be modeled in reactor cores, in fuel packages, in waste form design, or in calculating the effects of equipment used in separation technology. In these settings, it is frequently necessary to simultaneously model components of dramatically different shapes. Necessarily, the grids for these shapes are highly unlikely to be nicely aligned or compatible. Currently, there are no general tools readily available to the NEAMS community for this purpose.
- **Solver technologies**. Many of the simulations to be run by the IPSC teams require the solution of large systems of linear of nonlinear equations. Indeed, this is true of most multiphysics simulations, which often involve time-stepping algorithms with a large system of equations that must be solved at each time step. Each of the IPSC teams noted the need for high-quality, efficient solvers. However, this does not appear to be the highest priority for the teams, for two reasons. For one, high-performance solvers (especially for linear systems) exist and are easily obtained (e.g., PETSc and ***hypre***). While these solvers may not be fully effective for all the specialized problems of the IPSCs, they suffice for the present, and other ETC needs are more pressing. When those other needs have been addressed, solvers will once again take a preeminent role in the needs of the code teams.
- **User Interfaces.** Several of the IPSC teams mentioned a need for the development of user interfaces. Such interfaces are highly useful for facilitating the rapid setup of problems, easy selection of operating parameters, and efficient coupling of multiple programs. This technology development was requested rather more often than expected.
- **VisIt visualization tool**. The VisIt visualization tool is widely used and meets many of the IPSC teams needs; however, a number of extensions are required to support data

models for some applications. (While VisIt is the most widely used visualization tool, the Waste Forms IPSC indicated that they may prefer ParaView for their visualization needs. The ECT team will determine what functionality ParaView presents and how it might also be made available to the teams.)

- **Software Quality Assurance.** SQA practices vary widely among the code teams. In some IPSCs (most notably Waste Forms), SQA is being incorporated in the software development from the initial planning , while for other IPSCs SQA practices are in various states of use. The different teams requested different levels of help in the SQA area. One obvious area of concern is in trying to accomplish a NEAMS-wide common approach to SQA, resulting in a standard level of quality assurance, without attempting to prescribe the use of specific practices or tools.
- **V&V and UQ**. The NEAMS mission to provide end users with high-performance simulation capabilities that are efficient and reliable predictors of the physical systems requires that both V&V and UQ tools be incorporated into the process; hence the NEAMS VU element, whose task it is to provide such tools. It is incumbent on ECT to work closely with the VU element to determine if and how ECT can support the VU collaborations with the IPSC teams.
- **Access to Computing Hardware**. The IPSC teams require access to significant computing resources. While significant computing time is available through time grants at various user facilities, and particularly on INCITE awards, it is clear that reliance on these non-dedicated compute resources is untenable for the long term needs of the program. A major portion of the ECT effort is to be devoted to the creation of one or more NEAMS dedicated user facility.

Possible ECT Solutions from ASC and SciDAC

As described above, the apparent ECT needs of the IPSC teams show a good deal of overlap, with many of the same needs showing up for several, or even all, of the code teams. It was noted that many of the issues faced by the teams are similar to issues faced by code teams in the ASC Program, the SciDAC Program, or both. It should be no surprise, then, that there is a body of tools already available that was developed through the ASC, SciDAC, or Office of Science base programs. The ECT team has identified a number of such tools that may be employed in the NEAMS program.

Tools developed in the ASC Program include:

- Scalable linear solvers, such as the ***hypre***, or trilinos packages.
- Meshing tools, such as mesh generators, adaptive mesh refinement (AMR) packages, and mesh partitioners (e.g., Cubit, SAMRAI, Zoltan).
- Visualization tools such as VisIt or Paramesh.

- Technologies for coupling problems or meshes together, for example Sierra or Carter/Overlink.
- Performance tools, including compilers, debuggers, memory analysis tools, etc. A good example is the TotalView parallel debugger.
- Expertise in deploying user facilities.

Tools developed through the efforts of SciDAC or the Office of Science include:

- Scalable linear solvers such as PETSc or TOPS.
- Adaptive mesh refinement, front tracking, and interoperability tools, such as MOAB, ITAPS, and Babel.
- Performance tools such as PERI.
- Scalable Data Management tools, such as SDM.

The collected tools and expertise of the ASC, SciDAC, and Office of Science programs form a considerable and valuable body of work that ECT will work to integrate into NEAMS technology wherever feasible.

Plans for customizing tools

In the FY09 effort, ECT has identified a number of existing technologies that can be modified, extended, or customized to be suitable for filling the ECT gaps of the NEAMS code teams. Assuming FY10 budget permits, it is planned to support the initiation of several customization or extension efforts:

- Scalable mesh coupling technologies have been identified as a critical component in both the Reactors and the Fuels IPSCs. This includes the need for flexible tools for mesh-to-mesh transfer of field data. ECT has identified several tools that might be used as starting points, from which a mesh-to-mesh tool could be constructed. These possibilities include Carter/Overlink, MOAB coupling tools, Kuprat's coupling tools in Truches, and the SIERRA toolkit.
- Visualization is crucial to making sense of the data output by large-scale simulations. The VisIt tool, under development for some years by all three sources (ASC, SciDAC, and Office of Science) is the most widely used of the available visualization tools. For use with NEAMS certain extensions are indicated:
    - Support sets and tags for ITAPS data model
    - Complete support of Adventure plug-in for the Fuels IPSC
    - Develop in situ analysis tools
- Mesh generation tools are necessary in several applications, notably the fuels and reactors IPSCs, both of which must deal with simulations over complicated geometries. ECT has identified CUBIT as a starting point but notes that CUBIT needs modifications to

support specific needs of the NEAMS program. Furthermore, the Open Source requirement of NEAMS may pose a problem with CUBIT.

- Linear solvers are essential to many of the simulation programs. A host of linear solver technologies has been constructed over the past decade, but would require some research and modification for the NEAMS effort. One particular area of interest is the development of multigrid solvers that take advantage of operator sparsity in neutronics calculations.

Pillar 2: Software Quality Assurance for NEAMS

In this portion of the report, we detail the plan for the suggested Software Quality Assurance (SQA) approach for the NEAMS program. We begin by defining the terms for software quality processes that are used throughout this section. We highlight the key aspects of the Advanced Fuel Cycle Initiative (AFCI) graded quality rigor levels and define 9 key concepts that are suggested for NEAMS use in managing software quality practices. We conclude with a discussion of managing ongoing risk and a summary of our recommendations.

Introduction and definition of terms

Before addressing the NEAMS SQA approach it might be appropriate to review and define terminology related to SQA as used in this document.

Engineering can be defined as a discipline which applies technical and scientific knowledge to accurately predict the performance of a design, process, or system in the physical world in order to meet a desired objective or specified criteria. Engineering would be the antithesis of “trial and error”. Therefore *Software Quality Engineering* is a discipline that applies technical knowledge and computer science expertise to accurately predict the effectiveness of a software development process to produce a product or service that meets its desired objectives. Software Quality Engineering includes both the process and the product. A practitioner of this activity would be a *Software Quality Engineer*. As an example, *Software Quality Assurance* would be the activities that confirm that Software Quality Engineering is taking place within the software engineering process being used, such as an audit against a process standard or an assessment. *Software Quality Control* would assure that Software Quality Engineering is taking place on the product being developed, and would consist of activities such as inspection and testing of the product (Table 1).

Software Quality Assurance:	**Software Quality Control:**
Is Software Quality Engineering taking place in the process?	Does Software Quality Engineering exist in the product?
Error Prevention	Error Detection
Verified by Auditing	Validated by Testing

Table 1. SQA and SQC

In the commercial software industry the terms Software Quality or Software Quality Engineering or Software Quality Assurance are often used to describe practitioners or groups that primarily engage in system level testing activities. Certainly software testing is part of Software Quality Engineering if it is done using technical and scientific knowledge to predict the outcome of running a test case. The test’s outcome is usually expressed as the expected result of running the test case. The expected result is compared to the actual result to determine whether the test passes or fails. Testing has two purposes in theory, to validate that the software meets its requirements and to expose any defects (variations from the requirements or unexpected behavior). All too common in practice however, is a lack of up-to-date requirements in enough detail to be useful for building test cases. Without adequate requirements, software testing becomes more a defect exposing discipline than a requirements validation discipline. The old adage “you cannot test in quality” stems from the fact that if the software does not do what the customer requires it to do, exposing and removing most of the defects will still not allow the software to do what the customer wants it to do. Testing is a detection activity and commonly used all during the software development process (hopefully not just at the end). The creation of good tests (the fewest number of tests that cover the most features and code and find the most defects) is as complex technically as the creation of good code (the fewest lines of code that implement the most desired features and contain the fewest defects). However Software Quality Engineering encompasses many more detection and prevention activities beyond those associated with testing. Software Quality Engineering includes processes that can be used to elicit, analyze and trace requirements, architecture, design, and interface design and optimization, coding standards, inspections and reviews, templates and checklists, audits and assessments, change management, configuration management, static and dynamic code analysis, assurance of compliance to governing standards, determining customer satisfaction levels, analysis of trends and root cause analysis, measurement of the effectiveness of development processes, automation of development processes such as make/build, requirement tracking, defect tracking, design tradeoffs, inspection and review, risk assessment and risk management, as well as all aspects of unit, integration, and system level testing. Many software quality organizations such as the American Society for Quality (ASQ) would also include software project management as another software quality discipline. Certainly having capable software project managers who understand software quality engineering and are experienced developers themselves will increase the likelihood of a project’s success.

Recommended SQA Approach for NEAMS

The recommended SQA approach for NEAMS is based on experience with ASC research simulation codes. The ASC research simulation codes have the following characteristics:

- Research and Development environment
- Multi-physics simulation codes
- Staff consisting of Physicists, Material Scientists, Computer Scientists
- Languages used include C, C++, Fortran, and Python
- Platforms are parallel clusters and High Performance Computing (HPC)

Research Simulation Code is Different. The Research and Development software development environment has unique characteristics that are not found in other major software development areas such as commercial, military, MIS, outsourced, and systems. Research software development as a field is relatively small marketplace and therefore not of much interest to the mainstream software industry. Consequently relatively little emphasis is placed on this segment, in terms of historical data and tools. A major characteristic of R&D software is that it follows the scientific approach of forming a hypothesis, conducting experimentation, observation, and conclusion. It is iterative in nature and involves trial and error. From the software standpoint requirements (i.e. desired physics features) require the ability to experiment with the code to achieve an optimum design and implementation, both from a numerical precision and performance perspective. Knowledge gained experimenting with the design often changes the design approach to meet the requirements. Often a researcher is writing code to implement a design that has never before been done; this is in contrast to code written to implement an accounts payable system, the design of which is pretty well understood.

Brings Together Different Disciplines: The research environment may combine multiple physics disciplines together in a single code, for instance hydrodynamics, thermal, chemistry, EOS, and molecular dynamics. The research simulation codes typically require mesh generation, solvers, and visualization capabilities. The complexity of the codes require years of specialization and advanced degrees just to understand how to use them. They typically require large and specialized parallel platforms to execute the code in a timely fashion. This creates a unique environment where physicists, material scientists and computer scientists must work together closely to develop the codes. Oftentimes developers of the research simulation codes are also the users. Their backgrounds may be highly specialized in material sciences, but limited in computer science or software engineering. Popular languages used are C, C++, Fortran and Python, which is considerably different than main stream software which focuses on browser based Java and powerful tool kits and IDE's. The platforms used to support research simulations codes are typically Unix/Linux clusters or special purpose highly parallel super computers. Because research software is a relatively small software market, few open source and commercial tools and tool kits are available. The characteristics of research simulation codes create considerable challenges not encountered in main stream software. Recruiting of qualified staff is also difficult and getting harder when US citizenship is required. All of these characteristics make the application of SQA much more challenging than most DOD software projects or commercial software projects.

What works: The recommended SQA program for NEAMS research simulation codes will include "what worked" and minimize "what did not work" on the Advanced Simulation and Computing (ASC) program, as well as highlight ASC best practices, industry best practices, and software productivity tools that have proven useful. Leading the list of things that worked are SQA improvements that also improved development productivity, specifically automation of processes. Since these productivity enhancements improve code quality and make the developer's life easier, there is little adoption resistance; one example would be adoption of automated regression testing. Compliance simplification was also readily accepted. Being in compliance with concrete criteria based on a risk assessment is much easier to adopt than trying to remember three or more complex and overlapping government standards. Quality must go in at the beginning; it is very difficult to test enough to compensate for code that is poorly written. In some extreme cases entire code teams were replaced by new staff that was willing to follow required practices. Embedding SQEs within code teams to work as peers, rather than "audit and punish" worked well. Embedding the SQE's requires SQEs with developer skills; however the SQEs are viewed more as helpers than enforcers. As a result of embedding, software the quality culture gets adopted more readily by developers. Researchers as a group are less responsive to hierarchical edicts and more responsive to those seen as "peers with knowledge" who can be of help.

Multi-level automated regression testing works well and has improved the quality of released code, along with use of code repositories. If smoke tests take too long (longer than 30 minutes) they will be avoided, so keeping the test times short and selecting the fastest tests with the most code coverage is critical. From a support standpoint, user surveys indicated that users being able to walk down the hallway and explain to a developer a problem or ask how a feature works in a code were invaluable. Unit testing, which is not emphasized on all codes, is gaining popularity as new tools like CPPunit are maturing. Design by Contract (DBC) has also proven to be successful, as codes that use DBC tend to have lower defect rates. Whenever possible doing an activity should produce an artifact, rather than require developers to spend time documenting things that are not valuable to users of the code. Status meetings and scrums typically use applications such as Source Forge Enterprise Edition projected overhead to capture in real time comments, design tradeoffs, action items, task assignments, defect status, requirement tracking, etc. These captured artifacts are then searchable and available to the project at future dates. This technique has earned a number of noteworthy practices during independent DOE audits. Artifacts can be hierarchically or peer associated, for instance defect repairs and new features can be associated with particular releases and release notes in a repository, appearing as release notes or comments. Pydoc and Doxygen are two useful tools used to format and report embedded comments in headers or revision descriptions. Some tools also contain workflow logic which enforces a particular process. For instance before obtaining a new build number, certain actions have had to be taken on the part of the developer, such as finishing code reviews, designating a set of tests, creating a local branch, passing the regression tests on the local branch, etc.

The biggest surprise in using the risk graded approach and embedding was the reversal in code team desired risk levels. Prior to risk assessments code teams were biased to do the fewest number of practices and least rigor levels. When the risk assessment process was installed code teams in some cases actually wanted to operate at a higher risk level rigor because they felt their code was important. The avoidance tendency disappeared and major codes became expert at assuring risk level adherence in their feeder codes and library suppliers.

What Did Not Work. Equally important from retrospectives is what did not work well in the research simulation environment. Leading the list is prescriptive and heavy weight software development processes or the notion that SQA is additional work to do by the developers. The top down approach to compliance is less effective with researchers. Experience indicated that there must be a buy in at the grass roots level for processes and tools to be successful. Conducting audlets (small informal audits) using personal interviews was very effective compared to a written questionnaire. With interviews SQE can ask questions in different ways to discover information. The questionnaires were often misunderstood and avoided by developers. If the SQEs are viewed as adversarial the developers tend to isolate and keep information tightly held. Another pitfall was manually intensive make and build processes; this was especially troublesome for codes which had to run on numerous platforms. Tools to automate the make build process were used to overcome this pitfall. Lastly not version controlling feeder codes and libraries caused major heartburn. If the feeder code changed an interface without the major code knowing about it, the simulation would break and require considerable time to isolate the source of the problem. Feeders and libraries now communicate in advance all changes that may affect users and do not make changes between major releases.

Common Pitfalls. Other pitfalls along the way included the evolution of a good research idea. There were cases where a code started our being routine research and low risk. As time went on, the code became very successful, and became used for decision informing purposes. This created a situation where a QR3 code was now doing QR2 tasks, but was built using QR3 rigor. A number of mitigation steps were found helpful in this situation. These included always having code results reviewed by domain experts and use of professional judgment to validate code results, use of wrappers to limit the codes operation to well tested domains, increased use of unit testing, assertions, and design by contract. In the worst cases code refactoring was employed. Lesson learned was to consider starting at a higher rigor quality level at the outset if it is possible for the code to grow in importance over time. As mentioned earlier, not paying enough attention to the quality of the feeder and library codes caused numerous problems. Some of these codes were being supported by a part time physicist and the code kept in home directories. This of course caused problems for the major codes higher on the food chain. Pushing compliance down to the feeders and libraries eliminated many of the problems. Having the SQEs set up code repositories and conduct training, set up automated testing and defect trackers also helped. SQEs also do audlets on codes supplied by external sources, such as other labs and universities. Mitigation steps included assuring a battery of regression and acceptance tests to catch any problems in new versions of the external codes. Most external suppliers agreed to be interviewed to determine the rigor of their quality system. Understanding the size of the suppliers install base also helped in understanding the codes quality.

Balancing Discipline and Agility: One of the largest challenges facing research simulation codes such as NEAMS is the balancing of agility and discipline. On the one hand it is the desire of the developers and users to include the latest and greatest physics in the simulations and quickly incorporate new insights into the code. Some of these new physics features may or may not work well or may require design iteration to become useful. On the other hand the resulting simulations must have credibility and be built with a software process that produces code with the desired feature sets and a low defect rate. In medical, mission critical, and avionics codes the balance point must be heavily biased towards discipline. The ease of changing the code and time lag to produce a new build takes a back seat to rigorous process that assures the lowest possible defect rate. By comparison, demands are frequently made on research simulation codes that new features be added quickly, and this usually takes a back seat to a heavy weight and cumbersome development process. It is balancing the freedom for research and exploration with the dependable behavior and correctness of the code that create a unique challenge for research simulation code development and SQA. Using a risk technique found in Barry Boehm's and Richard Turner's book "Balancing Agility and Discipline" indicates a show stopper risk to research simulation codes if they use plan driven methods (table 2).

Risk Item	**Risk Rating**
Environmental risks	
Technology uncertainties	Very serious but manageable risk
Many stakeholders	Very serious but manageable risk
Complex system of systems	Very serious but manageable risk
Risks of using agile methods	
Scalability and Criticality	Serious but manageable risk
Use of simple design	Serious but manageable risk
Personnel turnover	Minimal risk
Not enough people skilled in agile methods	Minimal risk
Risks of using plan driven methods	
Rapid change	Show Stopper risk
Need for rapid results	Show Stopper risk
Emergent requirements	Show Stopper risk
Not enough people skilled in plan driven methods	Moderate risk

Table 2. Research Code Risks

The AFCI Graded Quality Rigor Levels

The NEAMS AFCI Quality Assurance Program Document (QAPD) has chosen a graded approach for quality. There are three quality rigor levels. Quality Rigor Level 3 is for routine R&D, feasibility studies, conceptual designs, exploratory tradeoffs and conceptual modeling. Quality Level Rigor 2 is for technical analysis used to inform policy reporting to congress or stakeholders, analysis for national environmental policy, and critical or controversial decisions. Quality Rigor Level 1 is for facility safety, NRC licensing related, safety basis for cat 1, 2, or 3 facilities, or benchmarking of a methodology that has potential for NRC approval.

Quality Rigor Level 3 for software requires adherence to DOE Order 414.1-C and section 1.6 and appendix B of the QAPD. Quality Rigor Level 2 for software requires adherence to DOE Order 414.1-C and section 1.6 and appendix C of the QAPD. Quality Rigor Level 1 requires adherence to DOE Order 414.1-C and section 1.6 of the QAPD and NQA – 1 2000 Requirement 3 and 11, with subpart 2.7 optional.

Quality Rigor Level	Code Usage	Required Compliance Documents
3	Routine R&D Feasibility study Conceptual designs Exploratory tradeoffs Conceptual modeling	DOE Order 414.1-C AFCI QAPD Independent Technical Review (appendix B) AFCI QAPD Section 1.6
2	Technical analysis to inform policy Reports to Congress and Key Stakeholders Analysis for National Environmental Policy Critical decisions Large expense Controversial	DOE Order 414.1-C AFCI QAPD Peer Technical Review (appendix C) AFCI QAPD Section 1.6
1	Facility safety NRC licensing Safety basis for cat 1,2,3 facilities Benchmarking of methodology that has potential for NRC approval	DOE Order 414.1-C NQA 1 – 2000 Requirement 3 and 11 (Subpart 2.7 Optional) AFCI QAPD Section 1.6

Table 3. AFCI QAPD Graded Approach

Key SQA Concepts for NEAMS

In order to meet the requirements of the AFCI QAPD and to also incorporate lesions learned and best practices for ASC scientific research codes, the remainder of this paper will recommend nine key concepts for incorporation in to the MEAMS SQA program. The goal is to balance agility with discipline.

Key Concept 1: Simplified Compliance Flow Down.

The proposed approach to the NEAMS research simulation software would create a new document called the NEAMS Software Quality Assurance Plan (SQAP). Its purpose would be to cross walk and translate for software developers which software development processes and to what level of rigor would need to be followed for QRL 1,2, and 3. The reason for this is to not have to encumber each software developer with the contents of all three governing standards and the interpretation of overlapping areas shown with arrows (see figure 8).

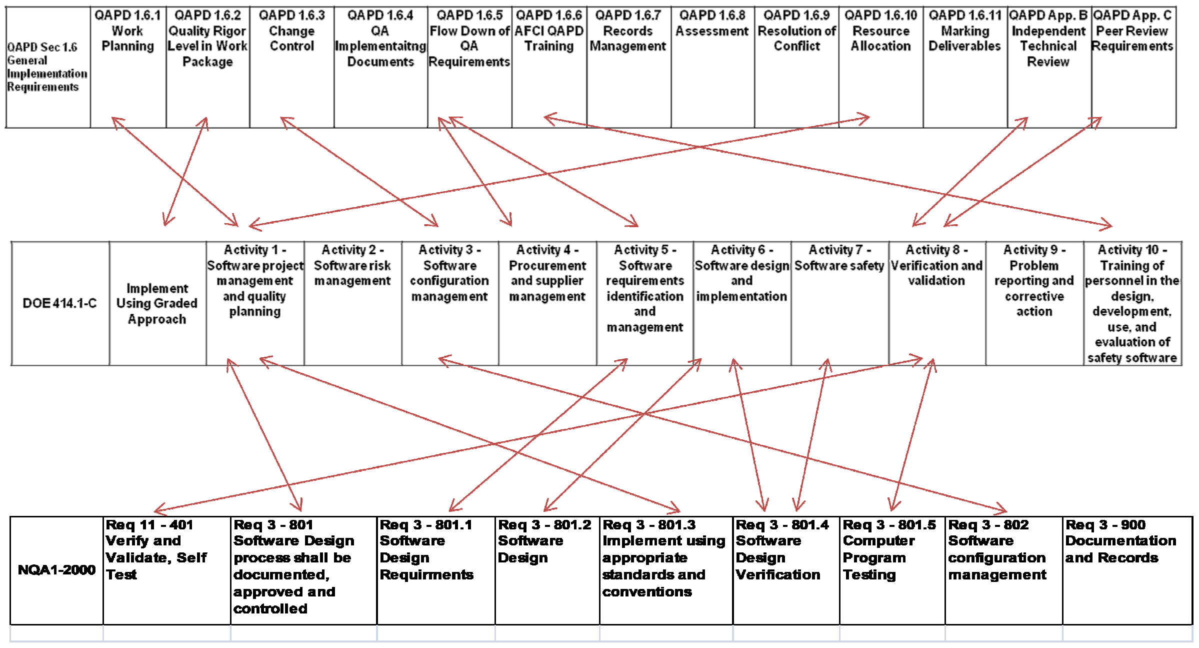

QAPD Sec 1.6 General Implementation Requirements
QAPD 1.6.1 Work Planning
QAPD 1.6.2 Quality Rigor Level in Work Package
QAPD 1.6.3 Change Control
QAPD 1.6.4 QA Implementaitng Documents
QAPD 1.6.5 Flow Down of QA Requirements
QAPD 1.6.6 AFCI QAPD Training
QAPD 1.6.7 Records Management
QAPD 1.6.8 Assessment
QAPD 1.6.9 Resolution of Conflict
QAPD 1.6.10 Resource Allocation
QAPD 1.6.11 Marking Deliverables
QAPD App. B Independent Technical Review
QAPD App. C Peer Review Requirements
DOE 414.1-C
Implement Using Graded Approach
Activity 1 - Software project management and quality planning
Activity 2 - Software risk management
Activity 3 - Software configuration management
Activity 4 - Procurement and supplier management
Activity 5 - Software requirements identification and management
Activity 6 - Software design and implementation
Activity 7 - Software safety
Activity 8 - Verification and validation
Activity 9 - Problem reporting and corrective action
Activity 10 - Training of personnel in the design, development, use, and evaluation of safety software
NQA1-2000
Req 11 - 401 Verify and Validate, Self Test
Req 3 - 801 Software Design process shall be documented, approved and controlled
Req 3 - 801.1 Software Design Requirments
Req 3 - 801.2 Software Design
Req 3 - 801.3 Implement using appropriate standards and conventions
Req 3 - 801.4 Software Design Verification
Req 3 - 801.5 Computer Program Testing
Req 3 - 802 Software configuration management
Req 3 - 900 Documentation and Records

Figure 6 NEAMS Governing Software Standards and High Level Cross Walk

The NEAMS SQAP would require a Risk Assessment done by the code team lead and certified software quality engineer. Aligning the software project with the NEAMS SQAP would assure compliance to all three governing standards (see figure 9). The NEAMS SQAP would be created by the NEAMS SQE staff. The alignment would be accomplished by a Risk Assessment. The Risk Assessment would consist of answering questions in a Risk table based on the categories given in the NEAMS QAPD.

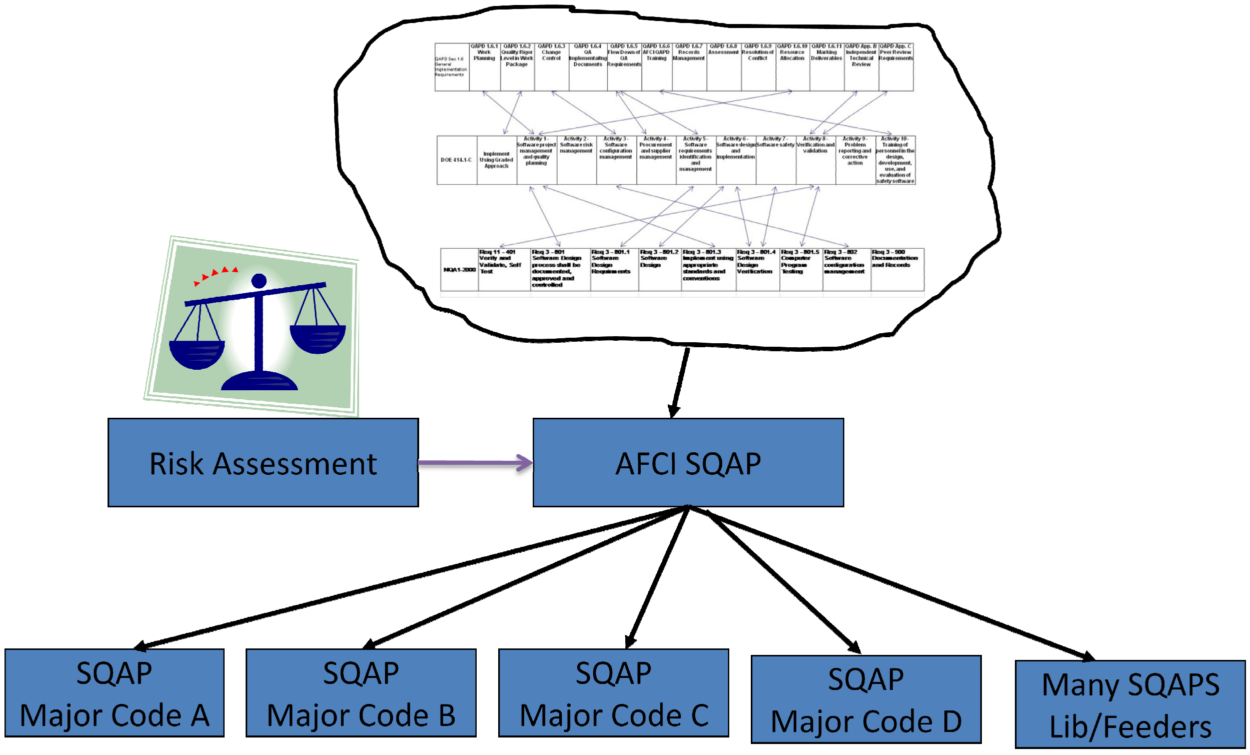


Figure 7 Simplified governing standards for software developers in the NEAMS SQAP

Key Concept 2: Risk Assessment Tool.

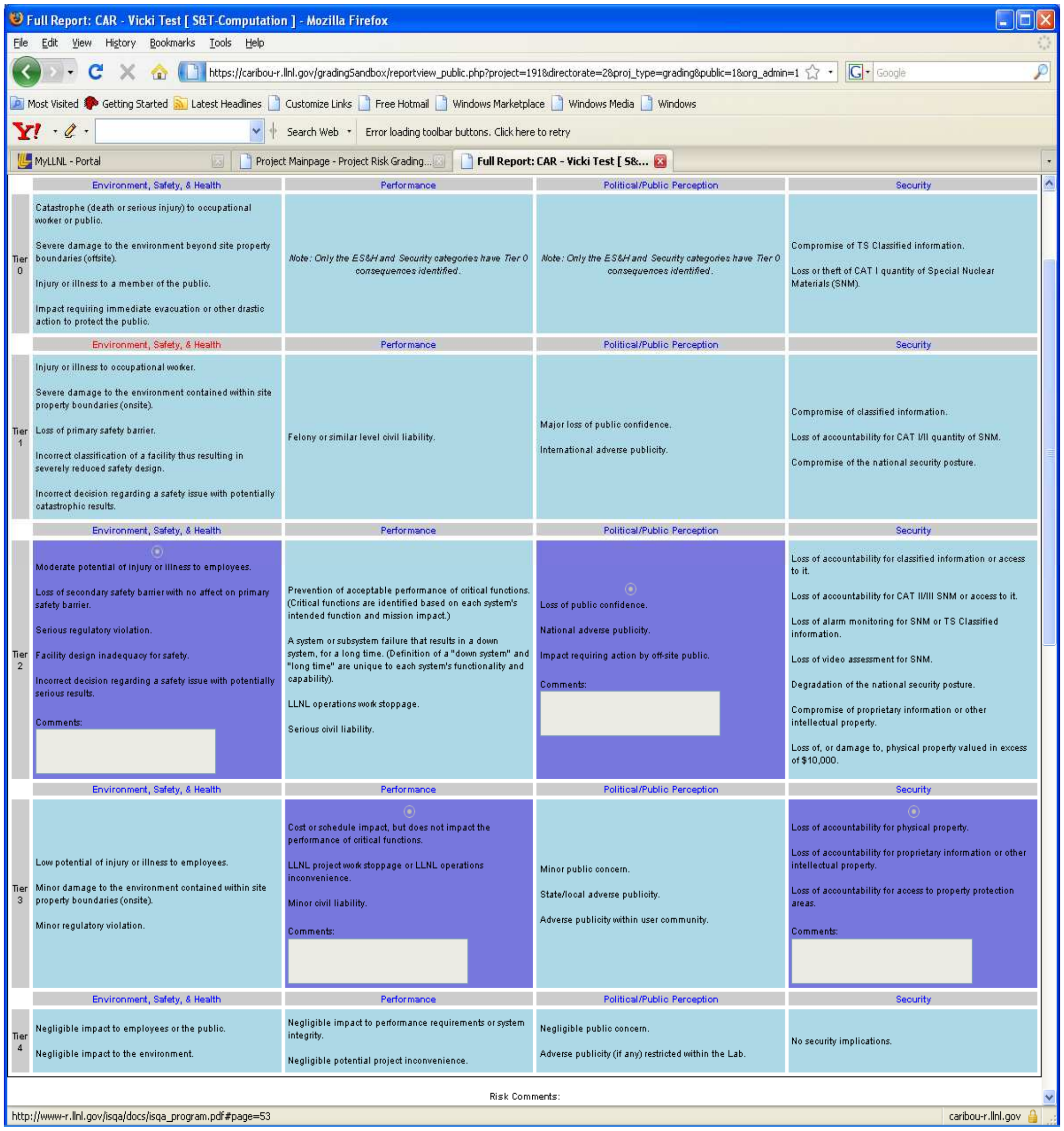
Key to the success of the proposed NEAMS SQAP will be the ability to uniformly determine the quality rigor level needed for any given code team. As an example of how to implement a risk graded approach, a description of the LLNL institutional process used on the ASC program is provided. The LLNL Risk Assessment process is aligned to DOD Order 414.1-C and QC-1 Rev 10 which does refer to NQA 1- 2000, it is similar to the NEAMS compliance requirements.

The LLNL Risk Assessment process starts by recognizing the risk consequence and likelihood of failure. The risk consequence addresses the impact to the Environment, Safety & Health, Performance, Security and Political & Public Perception if the product fails to perform as expected. Table 2a provides a general description for each risk consequence category and Table 2b is an example automated tool.

Risk Consequence Category	**Description**
Environment, Safety & Health	Risks to the operating and external environment, including: toxic release and cleanup. Risks to life and limb. Risks of regulatory liability.
Performance	Risks to meeting program requirements/goals. Risks of system downtime and work stoppage. Risks to the

	acceptable performance of critical functions, including civil liability. "Critical functions" are those important to the operation of the system or subsystem.
Political & Public Perception	Risks to governmental and public confidence and concerns.
Security	Risks to program, product and material security.

Table Table 2a. Risk Consequence Categories



Full Report: CAR - Vicki Test [S&T-Computation] - Mozilla Firefox

https://caribou-r.llnl.gov/gradingSandbox/reportview_public.php?project=191&directorate=2&proj_type=grading&public=1&org_admin=1

	Environment, Safety, & Health	Performance	Political/Public Perception	Security
Tier 0	Catastrophe (death or serious injury) to occupational worker or public. Severe damage to the environment beyond site property boundaries (offsite). Injury or illness to a member of the public. Impact requiring immediate evacuation or other drastic action to protect the public.	*Note: Only the ES&H and Security categories have Tier 0 consequences identified.*	*Note: Only the ES&H and Security categories have Tier 0 consequences identified.*	Compromise of TS Classified information. Loss or theft of CAT I quantity of Special Nuclear Materials (SNM).
Tier 1	Injury or illness to occupational worker. Severe damage to the environment contained within site property boundaries (onsite). Loss of primary safety barrier. Incorrect classification of a facility thus resulting in severely reduced safety design. Incorrect decision regarding a safety issue with potentially catastrophic results.	Felony or similar level civil liability.	Major loss of public confidence. International adverse publicity.	Compromise of classified information. Loss of accountability for CAT I/II quantity of SNM. Compromise of the national security posture.
Tier 2	Moderate potential of injury or illness to employees. Loss of secondary safety barrier with no affect on primary safety barrier. Serious regulatory violation. Facility design inadequacy for safety. Incorrect decision regarding a safety issue with potentially serious results. Comments:	Prevention of acceptable performance of critical functions. (Critical functions are identified based on each system's intended function and mission impact.) A system or subsystem failure that results in a down system, for a long time. (Definition of a "down system" and "long time" are unique to each system's functionality and capability). LLNL operations work stoppage. Serious civil liability.	Loss of public confidence. National adverse publicity. Impact requiring action by off-site public. Comments:	Loss of accountability for classified information or access to it. Loss of accountability for CAT II/III SNM or access to it. Loss of alarm monitoring for SNM or TS Classified information. Loss of video assessment for SNM. Degradation of the national security posture. Compromise of proprietary information or other intellectual property. Loss of, or damage to, physical property valued in excess of $10,000.
Tier 3	Low potential of injury or illness to employees. Minor damage to the environment contained within site property boundaries (onsite). Minor regulatory violation.	Cost or schedule impact, but does not impact the performance of critical functions. LLNL project work stoppage or LLNL operations inconvenience. Minor civil liability. Comments:	Minor public concern. State/local adverse publicity. Adverse publicity within user community.	Loss of accountability for physical property. Loss of accountability for proprietary information or other intellectual property. Loss of accountability for access to property protection areas. Comments:
Tier 4	Negligible impact to employees or the public. Negligible impact to the environment.	Negligible impact to performance requirements or system integrity. Negligible potential project inconvenience.	Negligible public concern. Adverse publicity (if any) restricted within the Lab.	No security implications.

Risk Comments:

http://www-r.llnl.gov/isqa/docs/isqa_program.pdf#page=53

Table 2b. Risk Consequence Tool

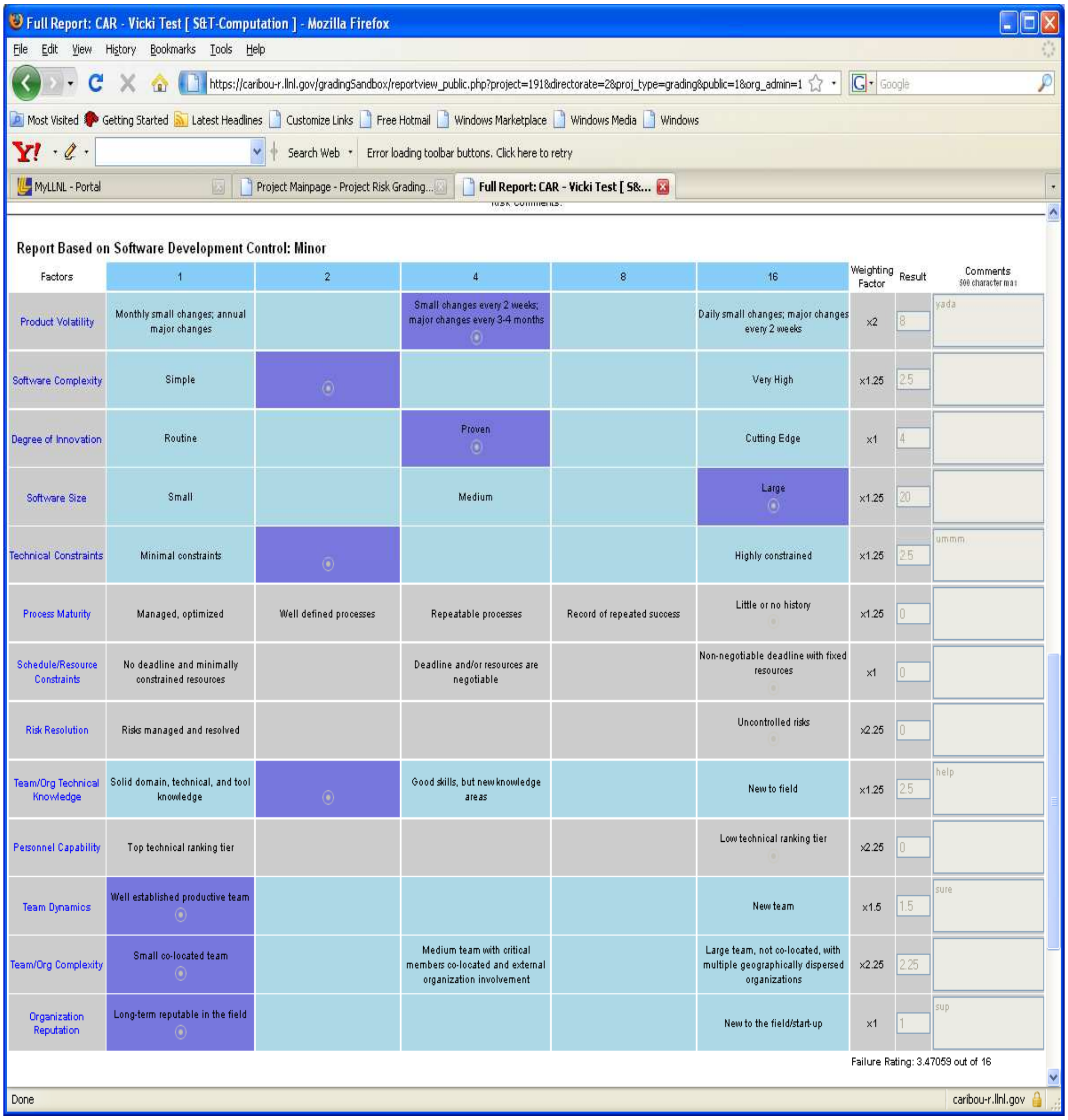
Each of these categories has 5 tiers associated with them. Tier 0 is dire consequences and Tier 4 is minimal consequences, with Tiers 1-3 being distributed between them. The SQE group, in consultation with the development team, determines the appropriate risk tier for each category. The highest level tier is used to determine the overall risk level of the product. The professional judgment needed to select the appropriate levels of risk consequence may require the SQE to work with other domain experts.

The likelihood of failure calculation takes into consideration many contributing factors of the development environment. Table 3a and 3b below shows some of the more important factors used in determining the likelihood of failure rating. The likelihood table shown was constructed from and weighted according to the COQUALMO risk factors, which trace their pedigree to 40 years of industry studies. Again, the SQE group, in consultation with the development team, determines the appropriate score for each factor. The likelihood of failure calculation is optional.

Likelihood of failure contributing factors	**Unweighted likelihood of failure score**					**Veighting**	**Likelihood of failure scores**
	1	2	4	8	16		
Product Volatility	Monthly smal changes; annua major changes		Small changes every 2 weeks major changes every 3-4 months		Daily smal changes; majo changes every 2 weeks	1.00	
Software Complexity	Simple				Very high	2.25	
Degree of Innovation	Routine		Proven		Cutting edge	1.50	
Software Size	Small		Medium		Large	2.25	
Technical Constraints	Minimal constraints				Highly constrained	1.25	
Process Maturity	Managed, optimized	Well defined processes	Repeatable processes	Record o repeated succes	Little or no history	2.25	
Schedule & Resource Constraints	No deadline anc minimally constrained resources		Deadline and/or resources are negotiable		Non-negotiable deadline with fixed resources	1.00	

Risk Resolution	Risks managed and resolved				Uncontrolled risks	1.25	
Team/Org Technica Knowledge	Solid domain technical, and tool knowledge		Good skills, but new knowledge areas		New to field	1.25	
Personnel Capability	Top technica ranking tier				Low technica ranking tier	1.25	
Team Dynamics	Well established productive team				New team	1.00	
Team/Org Complexity	Small collocated team		Medium team with critica members collocated and external organization involvement		Large team, not collocated, with multiple geographically dispersed organizations	1.25	
Organization Reputation	Long-term reputable in the field				New to the field/ start-up	1.00	
Weighted Factor Score Subtotal						18.5	
Weighted Average	Weighted Factor Score Total/Weighting Factor Total (18.5)						

Table 3a. Likelihood of Failure Table



Full Report: CAR - Vicki Test [S&T-Computation] - Mozilla Firefox

https://caribou-r.llnl.gov/gradingSandbox/reportview_public.php?project=191&directorate=2&proj_type=grading&public=1&org_admin=1

Report Based on Software Development Control: Minor

Factors	1	2	4	8	16	Weighting Factor	Result	Comments 500 character max
Product Volatility	Monthly small changes; annual major changes		Small changes every 2 weeks; major changes every 3-4 months ●		Daily small changes; major changes every 2 weeks	x2	8	yada
Software Complexity	Simple	●			Very High	x1.25	2.5	
Degree of Innovation	Routine		Proven ●		Cutting Edge	x1	4	
Software Size	Small		Medium		Large ●	x1.25	20	
Technical Constraints	Minimal constraints	●			Highly constrained	x1.25	2.5	ummm
Process Maturity	Managed, optimized	Well defined processes	Repeatable processes	Record of repeated success	Little or no history	x1.25	0	
Schedule/Resource Constraints	No deadline and minimally constrained resources		Deadline and/or resources are negotiable		Non-negotiable deadline with fixed resources	x1	0	
Risk Resolution	Risks managed and resolved				Uncontrolled risks	x2.25	0	
Team/Org Technical Knowledge	Solid domain, technical, and tool knowledge	●	Good skills, but new knowledge areas		New to field	x1.25	2.5	help
Personnel Capability	Top technical ranking tier				Low technical ranking tier	x2.25	0	
Team Dynamics	Well established productive team ●				New team	x1.5	1.5	sure
Team/Org Complexity	Small co-located team ●		Medium team with critical members co-located and external organization involvement		Large team, not co-located, with multiple geographically dispersed organizations	x2.25	2.25	
Organization Reputation	Long-term reputable in the field ●				New to the field/start-up	x1	1	sup

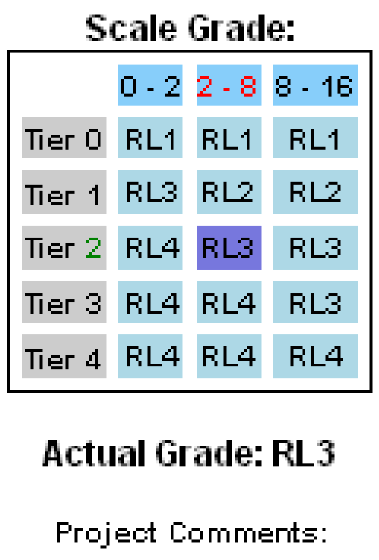
Failure Rating: 3.47059 out of 16

Table 3b. Likelihood of Failure Tool

The consequence of failure and likelihood of failure (optional) factors are then weighted and combined with the risk consequence tier to determine the overall risk level as shown in Table 4a and 4b. For instance a consequence of failure tier of 2 and a likelihood of failure score of 6 would result in an Risk Level (RL) 3 code.

Consequence of Failure Tiers			
Tier 0	RL1	RL1	RL1
Tier 1	RL3	RL2	RL2
Tier 2	RL4	RL3	RL3
Tier 3	RL5	RL4	RL3
Tier 4	RL5	RL5	RL4
	2	8	
	Likelihood of Software Failure Rating		

Table 4a. Risk Level Assessment Grading Table



Scale Grade:

	0 - 2	2 - 8	8 - 16
Tier 0	RL1	RL1	RL1
Tier 1	RL3	RL2	RL2
Tier 2	RL4	RL3	RL3
Tier 3	RL4	RL4	RL3
Tier 4	RL4	RL4	RL4

Actual Grade: RL3

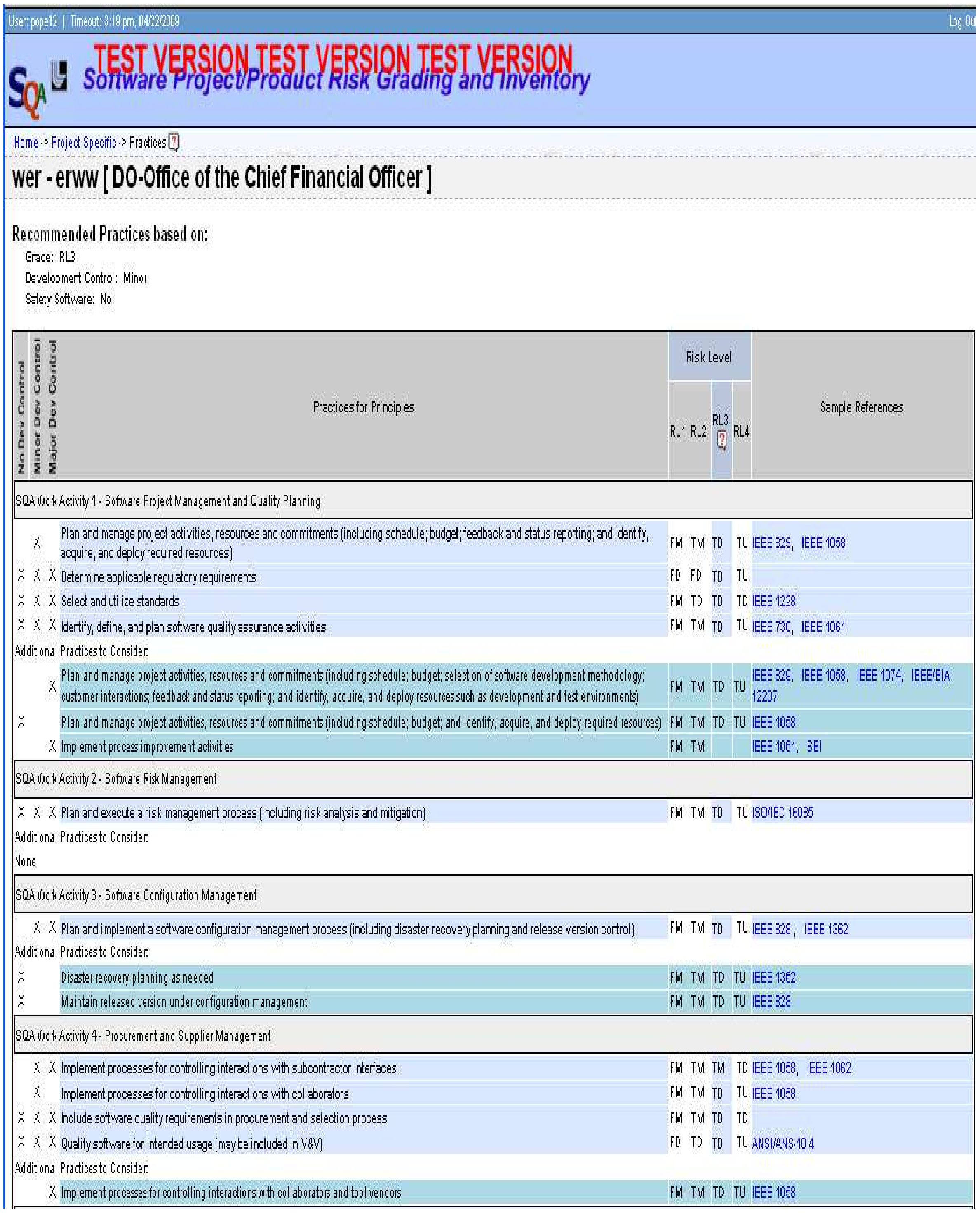
Project Comments:

Table 4b. Risk Level Assessment Grading Tool

This risk analysis results in a better understanding of the overall project risks and level of rigor needed to mitigate those risks. Using the risk assessment tool results in a uniform approach to risk assessment and creates an artifact of the risk assessment process. Risk assessment should be revisited periodically to assure that it is still valid. The risk assessment tool also has tables which delineate the software process activities:

1. Software Project Management and Quality Planning
2. Software Risk Management
3. Software Configuration Management
4. Procurement and Supplier Management
5. Software Requirements Identification and Management
6. Software Design and Implementation
7. Software Safety
8. Verification and Validation
9. Problem Reporting and Corrective Action
10. Training of Personnel in the Design, Development, Use, and Evaluation of Safety Software

The levels of rigor for the software activities span the spectrum from formally managed where software that can cause injury or death requires strict processes while a proof-of-principle code may only require understood practices. Table 4c is an example of the Risk Grading tool's recommended practices for the first four activities.



User: pope12 | Timeout: 3:19 pm, 04/22/2009 Log Ou

TEST VERSION TEST VERSION TEST VERSION

Software Project/Product Risk Grading and Inventory

Home -> Project Specific -> Practices

wer - erww [DO-Office of the Chief Financial Officer]

Recommended Practices based on:

Grade: RL3

Development Control: Minor

Safety Software: No

No Dev Control	Minor Dev Control	Major Dev Control	Practices for Principles	Risk Level				Sample References
				RL1	RL2	RL3	RL4	
SQA Work Activity 1 - Software Project Management and Quality Planning								
	X		Plan and manage project activities, resources and commitments (including schedule; budget; feedback and status reporting; and identify, acquire, and deploy required resources)	FM	TM	TD	TU	IEEE 829, IEEE 1058
X	X	X	Determine applicable regulatory requirements	FD	FD	TD	TU	
X	X	X	Select and utilize standards	FM	TD	TD	TD	IEEE 1228
X	X	X	Identify, define, and plan software quality assurance activities	FM	TM	TD	TU	IEEE 730, IEEE 1061
Additional Practices to Consider:								
		X	Plan and manage project activities, resources and commitments (including schedule; budget; selection of software development methodology; customer interactions; feedback and status reporting; and identify, acquire, and deploy resources such as development and test environments)	FM	TM	TD	TU	IEEE 829, IEEE 1058, IEEE 1074, IEEE/EIA 12207
X			Plan and manage project activities, resources and commitments (including schedule; budget; and identify, acquire, and deploy required resources)	FM	TM	TD	TU	IEEE 1058
		X	Implement process improvement activities	FM	TM			IEEE 1061, SEI
SQA Work Activity 2 - Software Risk Management								
X	X	X	Plan and execute a risk management process (including risk analysis and mitigation)	FM	TM	TD	TU	ISO/IEC 16085
Additional Practices to Consider:								
None								
SQA Work Activity 3 - Software Configuration Management								
	X	X	Plan and implement a software configuration management process (including disaster recovery planning and release version control)	FM	TM	TD	TU	IEEE 828 , IEEE 1362
Additional Practices to Consider:								
X			Disaster recovery planning as needed	FM	TM	TD	TU	IEEE 1362
X			Maintain released version under configuration management	FM	TM	TD	TU	IEEE 828
SQA Work Activity 4 - Procurement and Supplier Management								
	X	X	Implement processes for controlling interactions with subcontractor interfaces	FM	TM	TM	TD	IEEE 1058, IEEE 1062
	X		Implement processes for controlling interactions with collaborators	FM	TM	TD	TU	IEEE 1058
X	X	X	Include software quality requirements in procurement and selection process	FM	TM	TD	TD	
X	X	X	Qualify software for intended usage (may be included in V&V)	FD	TD	TD	TU	ANSI/ANS-10.4
Additional Practices to Consider:								
		X	Implement processes for controlling interactions with collaborators and tool vendors	FM	TM	TD	TU	IEEE 1058

Table 4c. Recommended Practices

It should be recognized that doing an initial risk analysis to determine the level of rigor for a software project does not take the place of hazard analysis for safety related codes. The risk analysis would instead establish that a software project needs to do hazard analysis because the initial risk assessment has determined that the Consequence of Failure (Severity) and other development risk factors warrant it. Notice also that the process rigor left hand columns (Table 4c) indicate the amount of control an enterprise has over the software project. This can vary from none to major control depending on whether the project is being done in house or by a supplier. Obviously the amount of control over the project is related to the ability to control activities. For each software development activity, the amount of rigor can vary from managed (M) to documented (D) to understood (U). The activities also can be formal (F) or tailored (T). For instance on a RL 1 project, activity 4 "procurement and supplier management", sub activity "qualify software for intended usage" would have to be a formally managed activity, on a RL 4 project the same activity and sub activity could be tailored and understood. After discussions with software safety experts, future versions of this assessment tool will change terminology to not be confused with more formal risk analysis techniques. The terms "consequence of failure" will be changed to "severity", "likelihood of failure" will be changed to "development environment risks" and "risk levels" changed to "quality levels". Allowing the rigor of a software process to be appropriate for the level of risk facilitates the use of a graded approach to an enterprise's software quality engineering process and avoids the challenges and lack of support that is encountered with the "one size fits all" approach.

Key Concept 3: Flow Down to Feeders and Libraries.

It is important to note in the proposed NEAMS SQA approach that feeder codes and libraries are also required to have their own code team SQAP. This was an important lesson learned on the ASC codes. NEAMS may also find over time that the major codes are quite dependent on many feeder codes and libraries. For instance the codes that do the meshing or the display of results, or libraries that contain the equations of state for the materials used. If there is a lack of quality in the data or feeder codes it will feed up to the main codes. In the best of cases these errors in the feeder codes or data may show up as conspicuous errors when the major codes are tested, but in the worst case they will simply skew the answers and not be easily detectable. To exacerbate matters the major codes may make changes to correct the errors coming from the feeder codes or data. This error masking phenomenon can create a situation where the simulation produces the correct answer, but for the wrong reason. The proposed NEAMS SQE approach will drive the software quality initiative down to the lowest levels to decrease the uncertainly level in the final result. This does not mean that all feeder and data codes must be at the same Quality Rigor Level as the main codes. Mitigation steps (such as specific acceptance testing) may be taken so that a QRL3 display code could be used with a QRL2 main code. Figure 10 shows a typical assortment of feeder and data codes. Note that codes that collect and condition data stored in libraries must be audited, also open source codes, university codes, and COTS and GOTS codes.

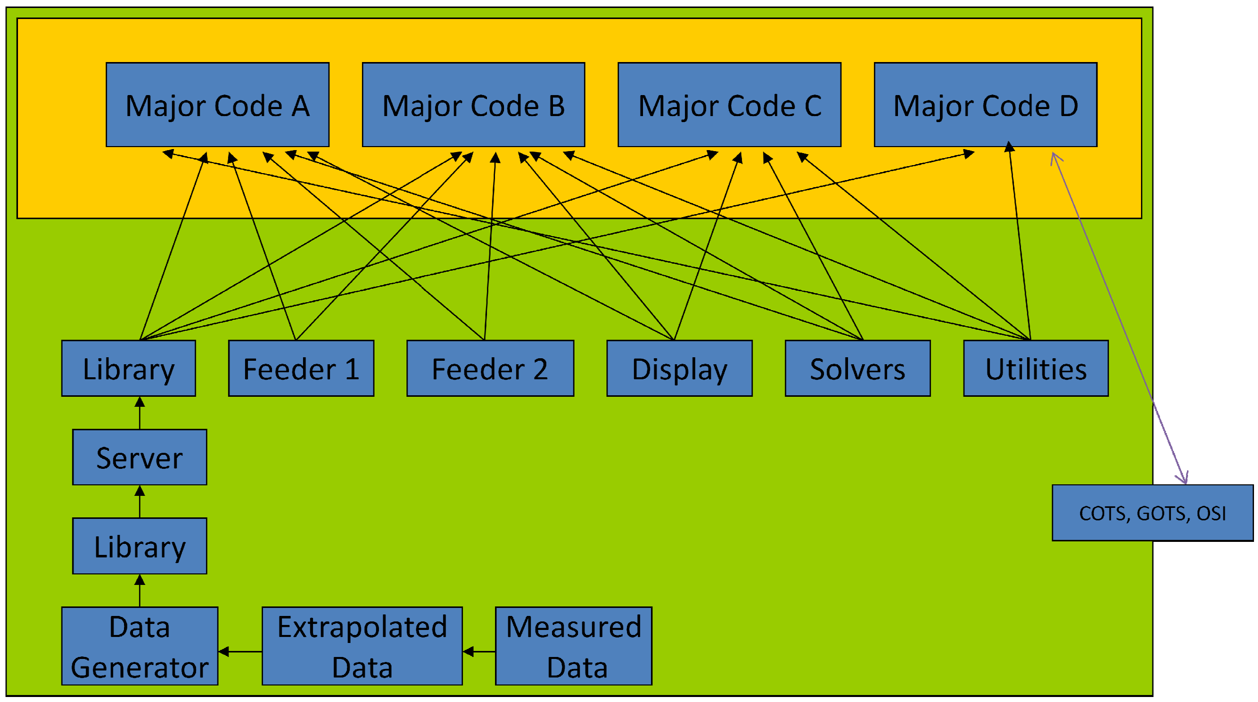
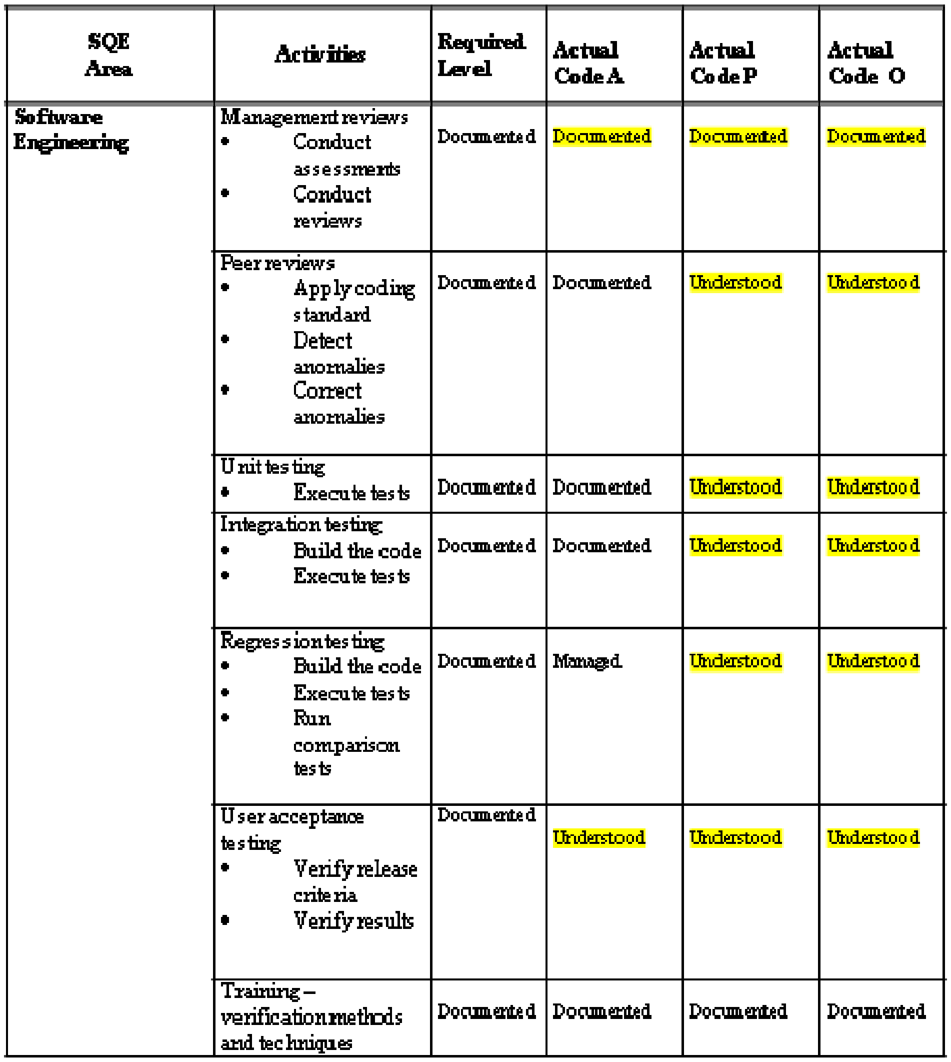


Figure 8 Feeders and library codes

Key Concept 4: Actual versus Gap analysis.

In the proposed NEAMS SQE approach, each software project team would also have a SQAP written by an SQE that compares the required level of rigor to be in compliance with the actual level of rigor that the team is using. The actual rigor is determined by the SQE conducting an audlet (small informal audit) of the actual practices being used by the code team. For each required software development activity, a table is created showing the level of rigor necessary to be in compliance and the level of rigor the team is currently using (or plans to use). The purpose of this table is to identify gaps between what is required to be in NEAMS compliance and what is actually being done. Table 5 is an example of a compliance table from a SQAP. In this table Codes A, P, and O are all in compliance with the Software Engineering Management Review criteria of documented management reviews.



SQE Area	Activities	Required Level	Actual Code A	Actual Code P	Actual Code O
Software Engineering	Management reviews • Conduct assessments • Conduct reviews	Documented	Documented	Documented	Documented
	Peer reviews • Apply coding standard • Detect anomalies • Correct anomalies	Documented	Documented	Understood	Understood
	Unit testing • Execute tests	Documented	Documented	Understood	Understood
	Integration testing • Build the code • Execute tests	Documented	Documented	Understood	Understood
	Regression testing • Build the code • Execute tests • Run comparison tests	Documented	Managed	Understood	Understood
	User acceptance testing • Verify release criteria • Verify results	Documented	Understood	Understood	Understood
	Training – verification methods and techniques	Documented	Documented	Documented	Documented

Table 5. Compliance Table

Key Concept 5: Process Improvement to Close Gaps.

Any gaps between what is required by NEAMS and what is being done by the code teams will be noted, and a process improvement plan will be required for areas where the actual level of rigor falls short of the required level. In table 5 under the Peer Review activities Code P and O are required to have a documented level of rigor, however they are operating at an understood level of rigor, which creates a gap. The gap would need to be addressed with process improvement. For instance a way to archive the invitation and findings of peer reviews. This could be accomplished by using an automated tool.

The actual level of rigor may exceed the required level, this will be noted but not required to be corrected. If the code team is new, then the audlet would be used to determine the level of rigor to be implemented. The project team's SQAP allow each team to uniquely show how they meet the required level of rigor. The assigned embedded SQE can offer suggestions on how to do this. Experience has shown that often times code teams are doing the required process, but may use different terminology for the process or not think about the practice in the same way. The SQE would assure these vocabulary issues are cleared up. The other benefit of this concept is that it focuses process improvement activities on the areas where they are most needed.

Key Concept 6: Automation and Tools.

A description of some of the tools found useful in the ASC research code effort is listed below:

- **Dynamic analyzers** – (Coverage) Examples: LCOV, GCOV, Purify, Intel. These tools can be enabled during regression testing to measure how much of the code is being covered by the regression tests. The coverage report shows statements and functions that are missed by the regression tests and provides good insights into how to improve regression tests. The code coverage can be measured at the statement or function level. Many of the ASC code teams also inserted hooks into their code to be able to measure feature coverage. Experience on ASC indicates that statement coverage of 60% or above is a reasonable goal. The reason 100% is not practical is the time it would take to check every error path, and the fact that recently added code may not yet have a full set of regression tests. Function coverage should run about 70% or above. Feature testing should be much higher. For features that are commonly used, 100% regression test coverage is the most common goal.

- **Static Analyzers** - (defect finding) Examples: Klocwork, Covarity, McCabe Battlemap. These tools statically analyze the code and look for over a thousand different weak coding practices. Some examples of weak coding practices would be not initializing variables before using, overflowing a string field or loop index, dereferencing a null pointer or stale pointer, not releasing memory (memory leak), etc. Many of the problems these tools find elude peer reviews and regression tests. Static analyzers check every line of code and every path and decision branch. Reports from the tools are used by developer or SQEs to repair the code. The false positive rates of these two static analyzers are under 15% making them much more useful than LINT tools. Klocwork also has an architectural tool that allows a graphical viewing of the interactions between modules and suggests ways to improving interfaces. McCabe Battlemap also has a similar feature, allowing a graphical representation of the calling hierarchy. Both tools have extensive code metrics, including object oriented metrics which can be useful in refactoring.

- **Trackers** - (Defect and Requirement) Examples: SFEE, Bugzilla, Round Up, Clearquest. Tracking requirements and defects is accomplished using tracking tools. Requirements need to be assigned to developers for design/implementation and also assigned test cases for subsequent validation. Defects found by users or developers other than the author need to be opened in a defect tracker and tracked until they are closed. Closing a defect usually requires fixing it and regression testing the code. Aging reports are created by tracking tools and help the code team decide which defects are not getting fixed in a timely manner or which requirements are not being implemented.

- **Code repositories** – Examples: Perforce, Subversion, CVS, Clearcase. Configuration management of code (and related artifacts) is critical in a team development environment. Repositories allow check out of code, creation of test branches, orderly merging of check-ins, and a history of each change to the code between versions. Perforce has a more powerful merging capability than CVS or subversion and is better for larger code teams. Subversion maybe sufficient for smaller code teams. Repositories should also be backed up on a regular basis automatically and stored at another location for disaster recovery. Each version of code released to users must also be in the repository so that when questions arise from users, developers can duplicate the same version to explore the concern.

- **Automated build and regression testing** – Examples: ATS, Tapestry, Autoconf, Quick Test Pro, Home Brew. Testing tools allow testing to proceed in an automated fashion on a check in, nightly, and pre-release basis. Testing tools are usually specific to the platforms (operating systems and schedulers) that are going to support the testing environment. The automated testing tools must be able to find the executable (or source) code to be tested, locate the tests (or inputs to the code to be tested), locate the baselines (expected results of running the tests), run the tests and produce reports to inform developers if the test passed or failed. The tools usually have a number of options about how the tests are to be configured, how many processors are to be used, whether to run in batch or interactive mode, how to make the comparisons between baselines and test results, tolerances, curve fitting, numerical comparisons, or to make corrections to the baseline. The tools may also inform developers of fialed test using e-mail reporting. Typically these are written in a scripting language such as Perl or Python. The make and build process should also be automated to reduce the possibility of error in configuring the code for a particular platform. Tools such as autoconf or gmake are useful for this function and may work in conjunction with the automated testing tool. There are also a number of COTS gui test tools (such as HP Quick Test Pro), which are useful for testing codes that use gui's. Most of the research simulation codes on the ASC program do not use guis for input. There are also open source test tools such as FIT or Fitness that allow table driven testing from spreadsheets or tables using fixtures. The ASC program at LLNL is in the process of building a general purpose system testing tool called Nazar, which combines the best features of Tapestry and ATS.

- **Combination tools** – Examples: Source Forge Enterprise Edition, Rational Rose. Tools exist that are a framework of multiple tools which can seamlessly interact to support process automation. Source Forge Enterprise Edition has been found to be very useful on the ASC program. This tool allows code team tasks to be tracked and interfaces to Microsoft Project, defects and requirements to be tracked, contains a code repository, a document repository, a wiki, a forum, and the ability to associate tasks, trackers and artifacts. SFEE also allows the embedding of workflow in the tool to enforce process. Tools such as Rational Rose implement entire development methodologies, such as the Rational Unified Process. They also allow Unified Modeling Language to be used to describe sue cases and translate them into code. The LLNL ASC experience found the RUP process to be heavyweight for research simulation codes.

- **Tool-smithing** – Examples: Python, PERL. A number of useful reports can be generated to aid developers using a scripting language such as Python and open source utilities such as Google FLOT for charting. Scripts have been created to show user activity by version, compiler warning levels by version over time, last changes to code before tests began failing, etc. suing scripting in conjunction with automated tools.

It should be noted however that tools themselves are not a silver bullet. The quality culture and quality aware project management must also be in place. Commercial tools have non-trivial licensing costs, and all tools, even open source tools require continual support and a learning curve. However tools such as those discussed in this section have made large improvements in quality and productivity by eliminating redundant and error prone manual processes. Many of the tasks associated with rolling out tools to developer groups can be facilitated by the SQEs.

Key Concept 7: Embedded SQE and Independent Reporting Organization.

The approach of having the SQEs work closely with the code teams to conduct audlits, author SQAPS, and help install productivity enhancing tools is called an embedded approach. An advantage of this approach is it establishes gaps at the outset, positions the SQE as a helper rather than an adversary, and creates an easy to audit flow down of NEAMS SQA requirements. The code team level SQAPs will also identify tools used to meet or enforce rigor levels and any artifacts generated as the result of engaging in their software process.

The SQE organization can be a part of the Validation and Verification group. It is important that the SQE group have an independent reporting path to the NEAMS program manager. Staffing of the proposed NEAMS SQA program would require assignment some of SQE's to specific code teams, in most cases shared between more than on code team, and other SQE's to focus of tools, feeder codes, libraries, GOTS, COTS and OSI codes. Qualifications of the SQE's would include prior developer experience and certification by an organization such as ASQ.

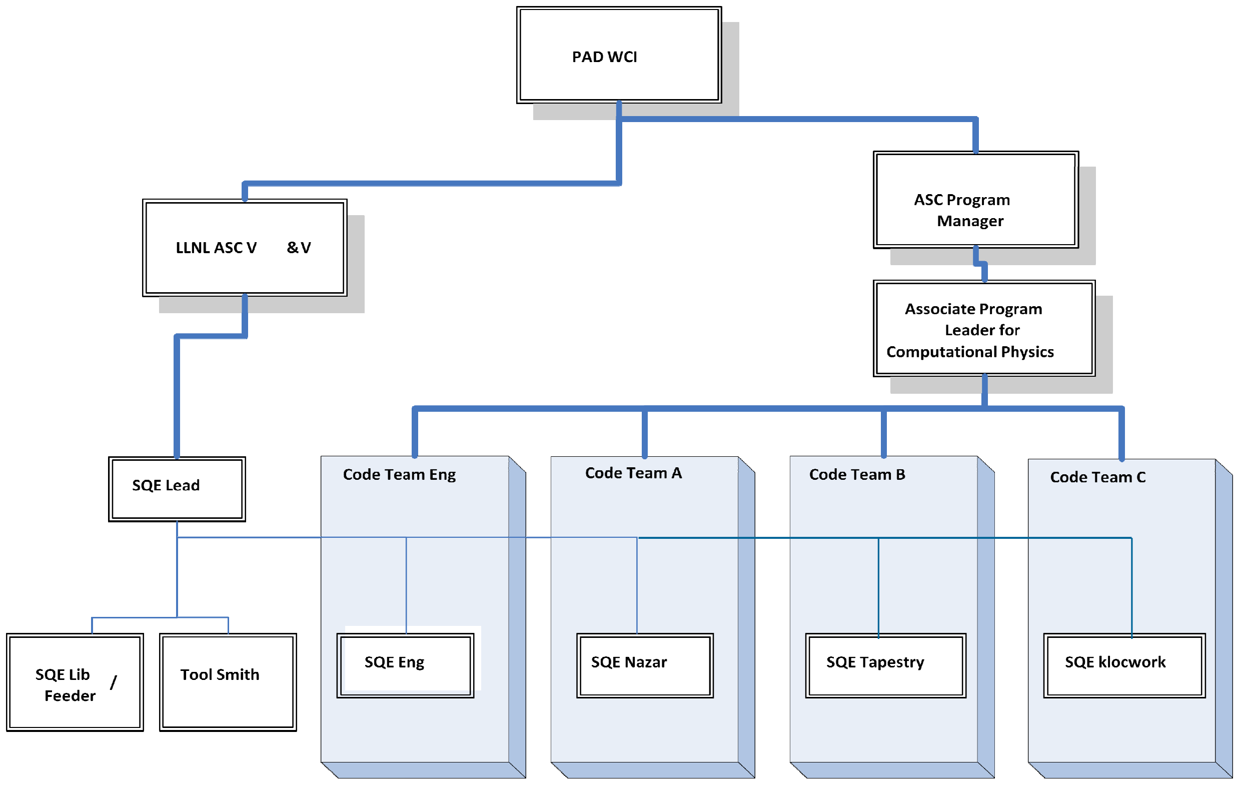


Figure 9 Embedded SQE and independent reporting organization

The code team SQAP template can be in IEEE (730-2002) or a common NEAMS template. The important part is that the ten listed activities in DOD Order 414.1.-C (plus any others) are listed along with levels of rigor required and actual are traceable to the code teams. The ten software activities in DOE Order 414.1-c are listed below:

1. Software project management and quality planning;
2. Software risk management;
3. Software configuration management;
4. Procurement and supplier management;
5. Software requirements identification and management;
6. Software design and implementation;
7. Software safety;
8. Verification and validation;
9. Problem reporting and corrective action;
10. Training of personnel in the design, development, use, and evaluation of safety software.

The proposed NEAMS SQA embedded approach requires a considerable amount of up front using the SQE's to risk assess the code teams and write a SQAP for each which includes a gap analysis and possible process improvement plan. Additional documents may also be required for the larger codes. On the ASC program at LLNL, large codes must also have a Configuration Management Plan (CMP), Software Test Plan (STP), and Disaster Recovery Plan (DRP). Smaller codes can include these plans in their SQAP. Advantages of the embedded approach are audits are done initially, the SQAP is understandable in the context of the individual code team's environment, and gaps surface early and are closed early in the project. The SQE is viewed as a value added member of the code team, helping the team with suggestions and tools to close any gaps. This approach has proven effective on the LLNL ASC program over the past five years for research simulation codes.

Key Concept 8: On Going Best Practices Forums.

Best practice forums are periodic meetings that bring developers together to discuss what is working and not working within their software projects. At LLNL best practice meetings are conducted as local meetings, but could also be implemented on NEAMS in an unclassified environment with collaboration software, such as "Go To Meeting" or Webex to bring together teams that are geographically dispersed. Management is not allowed in best practice meetings so that failures as well as success can comfortably be discussed. The best practice meetings have done much more than share best practices, the meetings opened up communication between groups that had remained isolated in prior years. New processes and tools are much more readily accepted by code teams if they are recommended by their peers rather than prescribed top down. The information for the meeting is captured and put on the Best Practice wiki so that it can be accessed later and by a larger audience. An additional wiki site was added to list industry best practices information as well for System, Military, Outsourced, MIS, and Commercial software categories.

Best practices that have been found successful on the ASC research simulation codes include the following:

- Best Practices Forums
- Multi-level automated regression testing
- Code repositories
- Defect/Requirement trackers
- Static code analysis
- Code coverage measurement (feature, function, statement)
- Embedded SQEs
- Source Forge Enterprise Edition
- Risk based graded approach
- Process and Product SQE metrics

Multi-level automated regression testing has been adopted by most all code teams as a best practice. The regression tests consist primarily of designed simulations with verified or validated answers known a priori. There are differing levels of regression tests. A smoke test is the smallest collection of regression tests, they run automatically whenever new code or modified code is checked in. They take 15 to 30 minutes to run. Nightly regression tests are run each evening and take 2-3 hours and contain a more comprehensive test set. Prior to a new release, a full set of regression tests are run, these take 2 or 3 days and include the full set of tests, both optimized and instrumented modes with numerous combinations of node/processor allocations. The automation of the regression tests is accomplished by test tools that have been written by the ASC code teams. The ASC SQEs are currently in the process of combining these two test tools into a single test tool that has all the features that each code team requires.

Each code team uses code repositories for configuration management of code, Perforce and Subversion being the most popular. Trackers are used for requirement and defect tracking, the most popular being Source Forge Enterprise Edition tracker, Bugzilla, and Round Up. Static analysis is done on the codes using the Klocwork tool. To date thousands of potential defects have been removed that were not detectable in code reviews, compilation, or testing. The quality of the testing is measured using dynamic analyzers to measure code coverage. Code coverage is accomplished at three levels, feature, function, and statement coverage. This best practice will indicate areas where tests are not exercising the code and defects could go undetected. Coverage tools used are GCOV, LCOV, Intel, and Purify. Source Forge Enterprise Edition is used for project task management, documentation archives, meeting notes, functional specifications, code branch tracking, project wikis, and forums. The embedded approach allows the SQEs to work shoulder to shoulder with developers to implement improved practices and tools. These are a few of the numerous best practices that have emerged from the best practices forums.

Minutes from best practices meetings are made available through a searchable wiki based on meeting date or topic. Also, industry best practices have been added to the best practices wiki thanks to the permission of Capers Jones. These best practices are his findings studying commercial, MIS, military, outsourced, and system software projects.

Key Concept 9: Use of Natural Metrics.

The Embedded SQE gather natural metrics about the code teams and codes and this data is also tracked for trends on the SQA project web site. Natural metrics imply that the developers are not asked to do extra work to determine and track these metrics, most of measurement and collection work is done by the SQEs. The list of natural metrics tracked on each code team is for:

- Compliance documentation (Boolean)
- Compliance (SQE internal, 1-10)
- Code metrics (Name, Lead, Size, FP, #Users, Risk)
- Coverage metrics (feature, function, statement)
- Estimated Fault Density
- Estimated CMM Level
- Static analyzed (lines, errors, error rate, fixed)
- Cyclomatic Complexity

The natural metrics are posted to the SQE web site which is set up as a collaborative documentation repository using Source Forge Enterprise Edition. Code teams themselves may keep many additional metrics of interest, such as which versions of code are used most frequently, warning levels on compile, aging reports for defects, etc.

Compliance metrics keep track of required compliance flow down documentation, requiring it be complete and current. Complete means contains required risk assessment, actual practice, plan to close any gaps, reviewed, signed, and in the ASC repository. Compliance ratings are the result of the audlet, which is an interview with the code team lead and/or members. Code teams are given a rating from 0 to 100% for each of the ten activities in DoD Order 414.1-C. The percent approximates the percentage of compliance in that activity. This metric is used to prioritize process improvements for code teams and activities within the code team. For instance if a code team scores 30% on design compliance, then an SQE may work with the team to install a tool to capture design trade off notes, functional specifications, or in the case of legacy code, flow chart the as built design (using a tool such as Visustin), run a calling hierarchy static analyzer (using a tool such as McCabe Battlemap), or look for circular relationships (using a tool like Klocwork architect tool).

Code metrics list the code teams, the code team lead, the number of lines of code (excluding comments), number of function points (usually backfired), number of users of the code, and assigned risk level of the code from the risk assessment tool. Estimated fault density is calculated by the SQE using a spreadsheet tool derived from the COQUALMO method. This method uses the same risk factors used in COCOMOII to estimate the fault density of the code. Fault density is expressed as defects per thousand lines of code (ksloc). The ECMM metric is also used, this metric uses a spreadsheet tool that the SQE fills out to estimate the CMM level of the software group, 1 to 5. One is the least mature process, 5 is the most mature. The results of the COQUALMO and ECMM are correlated, as empirical fault density data exists for different CMM levels. The COQUALMO and ECMM metrics are used to prioritize process improvements for the code teams. The estimated fault density is also compared against the actual discovered fault density measured by the Klocwork static analyzer tool.

Results of static analysis are also kept as metrics. This includes the number of lines of code analyzed, the number of potential errors found, the error rate (per ksloc), and the number of errors fixed. On the ASC program at LLNL about 6 million lines of code have been analyzed and the number of defects found and fixed is over 8,000. These defects had not been previously detected in code reviews, unit testing, and regression testing. Average cyclomatic complexity is also tracked and also used to prioritize process improvements. The average cyclomatic complexity is the average number of paths through each module of code analyzed. Lower numbers would theoretically indicate easier to maintain code, 10 is a suggested maximum.

On-Going Risk Management

An on-going risk management process is conducted by the SQE team. This includes identifying risks, assessing the risk, analyzing and tracking the risk, and using the information to manage the SQE project. Risks that have been managed include understaffing, and underfunding. This is a consequence of other government programs receiving higher priorities. As the funding reductions continue the work load has not reduced, resulting in understaffing. Use of automation and estimation tools help to mitigate this risk. Another related risk is the retirement of domain experts and the lack of qualified US citizens in the sciences. Platform volatility is a constant risk, with the simulation codes needing to run on a number of different and specialized platform types. Research codes tend to sprawl out instead of being designed with specific architectures. Refactoring and tools that allow insights into the calling hierarchies and architecture of the codes are helpful here. Dead code is also a problem, created by a new experimental feature that has been abandoned; the old code however was left in. Static Analyzers find this code and help with its safe removal.

Simulation codes assume that the users are very knowledgeable about the inputs they are providing to the codes. Therefore there is minimal input checking functionality in the codes adding risk that a user error could go undetected. To mitigate this risk tests have been derived from the user guide documents to exercise the input over ranges of legal and erroneous values. In research simulation development physicists and developers focus on the physics and complex algorithms and input checking does not receive as much emphasis. An error in the input that is not detected however could certainly skew simulation results, especially if is small enough to give a reasonable looking answer. In research codes small errors are much more serious than large conspicuous errors that core dump the simulation.

The ASC program uses natural metrics to compare to other industries and to gage that the level of quality is appropriate. The quality of research simulation codes is not going to reach the level required for military mission critical software, nor should it. The timeliness of the answers and need to continually improve and add features to simulations would not be achievable with an overly rigid quality program. Benchmarking fault density to other industries indicated that the quality of the ASC research simulation codes does exceed that of most commercial software products and web businesses.

SQA Plan Summary

In conclusion, the proposed NEAMS Software Quality Assurance program is based on the unique requirements for research simulation codes. Based on experiences of what worked and what did not, the following nine key concepts are proposed for NEAMS SQA:

1. Simplified Compliance Flow Down
2. Risk Assessment Tools
3. Flow Down to Feeders and Libraries
4. Required/Actual Gap Analysis
5. Process Improvements to Close Gaps
6. Automation and Tools Support
7. Embedded SQE's and Independent Reporting Organization
8. Shared Best Practices Approach
9. Natural Metrics to Prioritize Improvements

The application of these proven research simulation key concepts has helped institutionalize a quality culture at the grass roots level on the ASC program at LLNL. The proposed approach addresses the challenge of balancing agility and discipline while encouraging the continuous improvement of SQA practice.

Current SQA practices and needs of the IPSC teams

In this section we highlight the discussions held with each of the IPSC teams on the current practices they are using or plan to use to ensure a high quality software product. These discussions were informal; in FY10 we plan to map the current practices to the nine key concepts listed above and help each of the teams create a more robust SQAP.

Reactor Core Modeling

The software provided by this team is divided into components and managed in an svn repository. Regression testing software is currently being evaluated with the leading candidate solution being 'FLASH Test'; a open source custom tool developed at the University of Chicago for building a suite of tests; examining individual models across many different platforms and providing an integrated view of the results. The team has not yet selected their documentation tools. Doxygen is a leading candidate for software documentation, but the lack of support for f90 is problematic.

The team is interested in tools that can express the formalities of the coupling (interfaces, data flow, etc) and would like to consider the Rational software design tools from IBM. They are also interested in considering other regression testing software packages and documentation systems.

Fuels Modeling

The Fuels team has several SQA tools in place including software documentation tools, repository tools, and unit testing practices and is interested in working with ECT to evaluate their current practices and identify new tools that would improve productivity.

Waste Forms Modeling

High-level requirements and plans for the software engineering environment for the Waste Forms IPSC are based upon Sandia National Laboratory's rigorous experience implementing SQE within numerous software development projects, especially those within the Advanced Simulation and Computing (ASC) program. It is expected that eventually some Waste Forms codes will be required to satisfy Quality Rigor Level 1 Requirements defined in the AFCI requirements; practices and tools are planned to enable development at this rigor level. For other codes in the suite, however, development at lower levels of rigor will also be supported. The software practices in the Waste Forms IPSC will emphasize efficient development of quality software, which must be reliable, usable, efficient, and portable, as well as maintainable and flexible when incorporating new components. Further, the IPSC will adopt practices requiring the use of appropriate processes and practices to develop and maintain the code at a high level of software quality from the instant that it is created- that is, the software will be "born assessed.

Safeguards and Separations

No specific discussions were held with the SafeSep team regarding software quality assurance in FY09. We will pursue specific discussions on this issue in FY10.

Pillar 3*: Computing Platforms and Cycles*

Part of the charter of ECT is to ensure that the IPSC teams have access to adequate computing hardware to carry out their portions of the NEAMS mission. To begin fulfilling this portion of the ECT operation the team devoted effort to learning what the hardware needs of the IPSC teams are and determining what resources can be made available to them in later years of the program.

Preliminary Compute Needs of the IPSC Teams

In this section we highlight the preliminary discussions that we have ad with the IPSC team regarding their needs for compute cycles, memory and storage. The discussions, because they are informal, have had different levels of detail. This will be unified to a consistent level of understanding in FY10 using the questionnaire mentioned in the previous section.

Reactor Core Modeling

FLOPS: The team currently relies on INCITE awards on the Office of Science Leadership Class Computing Facilities at ANL and ORNL for the resources needed for their largest scale runs. They foresee a near-term need for intermediate resources to support a large number of runs with 100-500 processors and would like to see a dedicated capacity resource dedicated to NEAMS (e.g. 5000 processor cluster).

Memory: The neutronics calculations have very large memory requirements (e.g. 2.4 terabytes/group or 24 petabytes per 10000 groups) needed for subsections of the reactor. Architectures targeting NEAMS applications will need to address this requirement.

Storage and post processing: A typical 217 pin run of the thermal hydraulics calculation will generate a petabyte of data which implies the file system will need to operate at 100 Gbytes/s to move data between machines. It is anticipated that post-processing will be done as the run proceeds (on BG system) to get statistics. A parallel data analysis machine (approximately 10% of the computing system) is required to visualize analyze the post-processing data.

Fuels Modeling

No specific discussions on hardware needs were held with the fuels team in FY09. We will pursue specific discussions on this issue in FY10.

Waste Forms Modeling

The IPSC envisions little immediate need for outside resources, as they are in early development stages of the framework and identifying component technologies. However, they envision having a “major need” for computing cycles beginning in late FY10 or early FY11. They will need *capability computing* for the high-fidelity models and *capacity computing* for surrogate assessment codes, which will have *many* environment scenarios to test.

Safeguards and Separations

The wide range of programs created and employed in the Safeguards and Separations IPSC indicate that a wide variety of hardware needs are also to be encountered here. Many of the component codes are small, employing only a few processors (or just one processor). These codes are likely to be run on desktop machines; the main hardware need in these cases is for fast interconnects allowing the data being produced to be moved from place-to-place rapidly and efficiently.

At the other extreme, some of the physical separation simulations involve highly sophisticated multiphysics packages, such as molecular dynamics, interfacial flows, or densitometer simulations. These codes require large-scale discretizations, run for considerable lengths of time even on highly parallel machines, and produce large quantities of data. These simulations will require access to large multicore, parallel machines.

While most of the design codes are designed as “what if” codes, allowing the designer to modify and monitor the effects of changes to the design parameters, and do not typically require truly high-performance computing. One exception to this, however, is, for example, HostDesigner, a design code that can examine cascaded designs and employ multiple linked runs. For example, HostDesigner was employed and examined 72 million designs in 7 minutes on an Apple desktop. For the most part, the IPSC codes require capacity, rather than capability processing.

In at least one novel application, team members are creating an interactive system for modeling separation plant safeguard scenarios using a package with the “look & feel” of a video game, with an investigator moving through the plant encountering various states of the safeguard system. This simulation is performed employing fast graphics processing units (GPU).

In summary, while the Safeguards and Separations IPSC is at the beginning of their planning, it is already possible to foresee a host of ECT needs, in language interoperability, user interfaces, visualization, and computing hardware. It is entirely likely that as the IPSC gears up more fully other needs will be identified, making it crucial that the ECT team continue working closely with the Safeguards and Separations management.

Surveys of the IPSC team compute needs

In FY09 we conducted an informal survey of the IPSC teams to determine how much computer time and storage are needed to conduct their mission; these results are reported in earlier sections. Essentially, they indicate that a wide variety of platform needs is envisioned, ranging from stand-alone desktops to time on supercomputers.

As the teams begin to develop their products, it will become increasingly important to gather accurate descriptions of the computing cycle & storage requirements from all IPSCs. The goal in this area is to identify:

- the *type* of computing resources required by the program;
- the *size* (i.e., capacity and capability) of the required computing resources; and
- *where* to find and/or site the required computing resources.

To accurately amass these computing requirements, in FY09 we began to devise a questionnaire that will provide the answers to a common set of questions regarding their codes and resources:

What are the characteristics of the codes and simulations?

- Continuum, discrete, other?
- Are they compute-bound or communication-bound?
- How many CPUs does a typical simulation require?
- What is the average size of the output data generated by a typical simulation?
- What are the visualization needs?
- Described the type of activity the simulations support (code development (testing, debugging), code verification and UQ, large-scale parallel computing, production runs
- How many runs or ensemble of runs will be done per year?

What resources are currently available?

- Are they currently sufficient?
- What are the expected future needs?
- Can the team's computational needs be satisfied by a computing facility at a remote site?

In answering these questions, the ECT team will ask each program element to classify the needs as short term (1-3 years), medium term (4-6 years), and long term (7-10 years).

Developing a NEAMS computer facility

Ensuring that supercomputer time is available to the code teams requires identifying available machines at the participating laboratories, determining access requirements, and discovering the mechanisms by which the code teams fan fulfill the access requirements. For the most part, this aspect, "local supercomputing" is better handled by the code teams themselves, as they are generally already familiar with the requirements at their home laboratories.

A second approach, the “user facility” approach, is to create at one of the laboratories a virtual “NEAMS User Facility” in which time on one or more supercomputers is reserved by NEAMS specifically for use by the NEAMS community.

The Computing Facility at Idaho National Laboratory, operated by the Center for Advanced Modeling and Simulation (CAMS) is intended to fulfill just such a role. The center is equipped with a 2048 processor SGI Altix shared-memory Linux cluster with an Infiniband interconnect, running at a peak of 20 teraflops. With its shared memory and Intel processors, the system has been used successfully fuels related problems, demonstrating the utility of the system for NEAMS work. The Center also has a PowerWall display system, enabling virtual exploration of datasets and performance of important analyses. They are purchasing an immersive visualization system, as well. The Center is in the process of purchasing a sizeable persistent mass storage system, significantly increasing the size of simulations that could be run.

The primary use of the INL computing center to date has essentially been internal INL use only. One of the primary bottlenecks in using this facility as an external user facility is the low bandwidth on connections to external networks, which significantly limits the effectiveness of remote use. A relatively small investment of $300K would provide the resources necessary to allow for a connection to the ESnet backbone; alleviating this restriction. Furthermore, the Center must further develop its systems for managing user accounts and system resources for offsite users.

FY10 Planned Activity

The ECT team has a comprehensive set of activities planned, beginning in FY10, to address the needs identified in the previous sections. While the scope of FY10 activities is naturally dependent on the size of the ECT budget, the intent is to work primarily in four basic areas: requirements gathering, software quality assurance, constructing the ECT clearing house, and assisting the creation of a user facility at the Idaho computing center. More specifically, the activities planned are:

- **Requirements Gathering.** ECT will continue to gather the requirements and priorities of the IPSCs and VU, FMM, and CT elements of NEAMS.
    - Because they are less fully developed than the Fuels and Reactors IPSCs, most of the requirements gathering focus will be on the Waste Forms and Safeguards & Separations IPSCs.
    - ECT will maintain contact and work to understand how we can support the VU, FMM, and CT efforts.
- **Software quality assurance.** The SQA activities of the ECT team will be focused on providing an overarching SQA philosophy, selling that philosophy to the program elements, and providing tools and techniques by which the philosophy can be incorporated into the code efforts. The specific FY10 activities envisioned include:
    - Collecting and deploying tools for a graduated risk-based assurance approach.
    - Working closely with IPSC teams to evaluate and, as needed, improve or replace current approaches (e.g., implement a test system for reactor team).
    - Collaborate with CT on consistent SQA deployment across NEAMS technologies.
- **ECT Clearing House.** The team intends to build a “clearing house” of ECT information and tools, making them available easily to the NEAMS program elements. The most likely mechanism for accomplishing this is to mount a Web Portal, through which NEAMS researchers, developers, and practitioners can easily download the tools themselves along with easily implemented scripts for installation, high-quality documentation of the tools, and information about best practices and philosophies. In order to leverage the vast body of work that has been previously amassed, ECT will focus on the SQA tools developed by and available through the SciDAC and ASC programs.
- **Idaho computing center.** ECT considers the establishment of one or more dedicated user facilities to be of critical importance to the success of NEAMS; it is essential that the program have a known, reliable source of computing cycles and not be dependent on the generosity of local computer centers or on the competitive awarding of computer time. To that end, ECT will focus on aiding in the development of the Idaho National Laboratory as a NEAMS user facility. This will include providing support for staffing to handle user accounts and data management, to develop high-speed connections to the internet, and to insure

that other infrastructure necessary for effective use by remote users be emplaced. ECT will also encourage and facilitate the remote use by a few friendly, external NEAMS users, in order to a) prove the concept of INL as a user facility, and b) identify obstacles that must be overcome for effective use.

Conclusion

Following on the successes of ASCI, ASC, and SciDAC over the past two decades, it has become clear that computational simulation really has taken a place as a peer to theory and experiment in a new paradigm of science. Simulation has been accepted as a fundamental tool that can be used to predict and interpret the results of experiment as well as to guide the design of experiments. NEAMS is intended to deliver to its ultimate customers a comprehensive, integrated capability for performing large-scale multiphysics simulation to be used as a crucial tool in the design, engineering, licensing, and operation of the next-generation nuclear power system. To accomplish this ambitious goal, NEAMS was founded around four code teams whose charter is to provide advanced simulation capabilities in reactors, fuels, waste forms, and separations & safeguards. But learning from the ASCI experience, NEAMS management recognized that all the code teams are faced with some common software and hardware needs. Enabling Computational Technologies is a NEAMS activity whose task it is to determine the commonalities of need among the code teams and ensure that the necessary software and hardware resources are acquired or built, and are made available to the NEAMS scientists in timely manner.

In FY09, the ECT team made a concerted effort to understand the NEAMS program and the particular needs of the Integrated Performance and Safety codes in the areas of tools and libraries, software quality assurance, and computer facilities and resources. While this effort is not complete, several emerging themes became clear: model set up and mesh-to-mesh coupling tools, visualization technologies, help with software quality assurance tools and user interfaces are high priority items for the ECT program element. We also analyzed the needs of the AFCI program in software quality assurance and provided a recommendation for a risk-based graded approach. Our approach contains nine key concepts that balance the need for a disciplined yet flexible approach to software development in the NEAMS research environment. Finally, we initiated discussions with the IPSC teams to understand their current and envisioned compute resource needs (FLOPS and storage). We also engaged the Idaho National Laboratory computer center to understand what facilities they can provide to the broader NEAMS community and what is needed to ensure successful deployment as an external user facility. In FY10, several activities are planned to continue this work including further information and requirements gathering, building a web portal clearinghouse for ECT information, initiating the deployment of SQA tools to the IPSC teams, and strengthening Idaho’s ability to support external users. Funding permitting, the ECT program element will also engage in the development of customization of key tools and libraries such as mesh-to-mesh coupling tools and expansion of VisIt visualization capabilities.